



GumNaaM_Helpers



Mega File for Final Term

All Files in ONE FILE

(Handouts_waqar_moaz_junaaid_vu toper_orange monkey)

الحمد لله گمنام ہیلپرز ان تمام کتابوں میں مدد کرتے ہیں جن کے لنکس نیچے دیئے گئے ہیں
لیذا ہم سے صرف انہی کتابوں کی مدد کا سوال کیا جائے جن کے لنکس نیچے موجود ہیں

Groups Join Krty hi Description Read krna na bhooolyn

جزاک اللہ خیرا و احسن الجزاء

GumNaaM_Helpers All What's App Group Links

Cs101 https://chat.whatsapp.com/KCQ3jK6FcoYEvDYTF1gnBS	Eng101 https://chat.whatsapp.com/Cwp4Ag9TG6ZKxyRgThqM6S
Cs201 https://chat.whatsapp.com/DrYV1FAXmPpImVZFw6V2R	Eng201 https://chat.whatsapp.com/EvvxcPtmrM22VTZh5NCTw
Cs301 https://chat.whatsapp.com/EPLvH2f8EB7aH8e1AtidV	Eco401 https://chat.whatsapp.com/K3NLvyue6BdK0sVMQDFoU2
Cs302 https://chat.whatsapp.com/BaYfW0P67vNIsI2n36otZM	Mcm301 https://chat.whatsapp.com/GUyzfdHzu8r43TW16vhZWF
Cs304 https://chat.whatsapp.com/C0mk0wdo0BU46dlhc67IQ	Mgt101 https://chat.whatsapp.com/BmrYaQYulBOKguHOIT6t2Y
Cs401 https://chat.whatsapp.com/Jz9WDdHX84IKgkrMPAV0Mf	Mgt201 https://chat.whatsapp.com/DZwztUyxV50D6AshnvWCD
Cs402 https://chat.whatsapp.com/DW9AZLb4i1P9la4XmcCsRz	Mgt211 https://chat.whatsapp.com/EmX9xloZnbl12pX9m2DZwa
Cs403 https://chat.whatsapp.com/HXV25Lc6MnLKZgrA7xOqjF	Mgt301 https://chat.whatsapp.com/G62QTipycOXFrP0ZxfDIBK
Cs408 https://chat.whatsapp.com/EFy2mrYJodx5d85W1jtII7	Mgt501 https://chat.whatsapp.com/H0dTUINbesw0GS4LLSt0W4
Cs411 https://chat.whatsapp.com/lrOaF8Q3tkJKzeVTHapHR9	Mgt502 https://chat.whatsapp.com/L3GSMFgwDl7DAsu9r0vAxl
Cs501 https://chat.whatsapp.com/Eggx3X0SH2LhOysM57SIi	Mgt503 https://chat.whatsapp.com/G7BNxqyLYwP48jftOfkbj
Cs502 https://chat.whatsapp.com/FbLU96Xhmr2F199fnb5Yw	Mgt602 https://chat.whatsapp.com/HXRpOrEL18vD7lb91nLHTB
Cs504 https://chat.whatsapp.com/Gn7w0UzNCQ2FzRK7Zy4Io	Mgt610 https://chat.whatsapp.com/BxMcR1ouiao0wuqLLlJ5dB
Cs506 https://chat.whatsapp.com/BL4FFmDKf2S3WIoLSAD2JK	Mth101 https://chat.whatsapp.com/LHsbl65FbwGH2SNFF9ipSb
Cs507 https://chat.whatsapp.com/EINDZKv4e4zGLGqL8imozR	Mth202 https://chat.whatsapp.com/GevRebusRaxFh6dgnzRJuM
Cs508 https://chat.whatsapp.com/HTMAwnJ4Bic2ky8MbT3PuW	Mth301 https://chat.whatsapp.com/EFm472nc2woDProEqnY1lj
Cs601 https://chat.whatsapp.com/KYoaAutdXRINDS0Mm5XfzP	Mth401 https://chat.whatsapp.com/EP7tlcc17TzLtaNaOgD4AcA
Cs602 https://chat.whatsapp.com/FpJkLAFNqpZHRyi6vhl5n8	Mth501 https://chat.whatsapp.com/Dy7XUUVFqqD5LLw4i5X4Xzi
Cs604 https://chat.whatsapp.com/KjDhHjxawtU43Veo9Eawa4	Mth601 https://chat.whatsapp.com/BoD3akuXGK25MFB4UEZcRr
Cs605 https://chat.whatsapp.com/BoF34LtlMIYK1L4nQaEO2Ce	isL201 https://chat.whatsapp.com/BtoqaMbbELW99MLLTQMgWc
Cs610 https://chat.whatsapp.com/FcNcztKyYfd9szsJElino2	Pak301 https://chat.whatsapp.com/KuR9Y7YWeA32enRwKeZJaR
Cs615 https://chat.whatsapp.com/HLVjBNcorFN9bwZhGekFI3	Phy101 https://chat.whatsapp.com/GjC6e47vF1zHwRs5iOc13j
Cs619 https://chat.whatsapp.com/FEklYr700ui0ihxqnk55A6	Phy301 https://chat.whatsapp.com/lBrOdznEHDi92fGa76Dqmr
	Sta301 https://chat.whatsapp.com/BchodAnjbydKXLuxFfkGxR

...Paid Service Available for LMS Activities...

Only for those students who are so busy in office or work or job that they can't watch their LMS due to their Busyness

FOR MORE INFO. CONTACT US ONLY WHAT'S APP 0302-1400084

Data Structures

Table of Contents

Data Structures.....	1
Lecture No. 01.....	3
Lecture No. 02.....	12
Lecture No. 03.....	20
Lecture No. 04.....	32
Lecture No. 05.....	46
Lecture No. 06.....	56
Lecture No. 07.....	63
Lecture No. 08.....	71
Lecture No. 09.....	82
Lecture No. 10.....	93
Lecture No. 11.....	105
Lecture No. 12.....	123
Lecture No. 13.....	135
Lecture No. 14.....	146
Lecture No. 15.....	158
Lecture No. 16.....	170
Lecture No. 17.....	189
Lecture No. 18.....	203
Lecture No. 19.....	211
Lecture No. 20.....	220
Lecture No. 21.....	231
Lecture No. 22.....	240
Lecture No. 23.....	257
Lecture No. 24.....	273
Lecture No. 25.....	284
Lecture No. 26.....	296
Lecture No. 27.....	307
Lecture No. 28.....	321
Lecture No. 29.....	333
Lecture No. 30.....	348
Lecture No. 31.....	360
Lecture No. 32.....	370
Lecture No. 33.....	379
Lecture No. 34.....	386
Lecture No. 35.....	393
Lecture No. 36.....	404
Lecture No. 37.....	416
Lecture No. 38.....	424
Lecture No. 39.....	430
Lecture No. 40.....	441
Lecture No. 41.....	449
Lecture No. 42.....	459
Lecture No. 43.....	468
Lecture No. 44.....	474
Lecture No. 45.....	485

Data Structures

Lecture No. 23

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 4
4.4.1, 4.4.2

Summary

- Single Right Rotation
- Single Left Rotation
- Double Right-Left Rotation
- Double Left- Right Rotation
- Deletion in AVL Tree
- Cases of Deletion in AVL Tree

We demonstrated how to write the insert procedure for an AVL tree in the previous lecture. The single and double rotation call methods were also discussed at length. While inserting a node if we see that the tree is going to be unbalanced, it means that the AVL condition is being violated. We also carried out the balancing of the tree by calling single or double rotation. Now let's see the code of these routines i.e. single and double rotation.

Single Right Rotation

At first, we will discuss the code of SingleRightRotation function. Its argument is `TreeNode`, which is given the name `k2`. The reason of using `k2` is that it alongwith `k1` etc was used in the earlier discussion. After the rotation, this function will return a pointer, as tree will be re-organized. Resultantly, its root will be changed. Following is the code of this SingleRightRotation function.

```
TreeNode<int>* singleRightRotation(TreeNode<int>* k2)
{
    if( k2 == NULL ) return NULL;
    // k1 (first node in k2's left subtree) will be the new root
    TreeNode<int>* k1 = k2->getLeft();

    // Y moves from k1's right to k2's left
    k2->setLeft( k1->getRight() );
    k1->setRight(k2);

    // reassign heights. First k2
    int h = Max(height(k2->getLeft()), height(k2->getRight()));
```



```

k2->setHeight( h+1 );
// k2 is now k1's right subtree
h = Max( height(k1->getLeft()), k2->getHeight());
k1->setHeight( h+1 );
return k1;
}

```

In the function, at first, we check whether k2 is NULL. If it is NULL, the function is exited by returning NULL. If it is not NULL, the rotation process starts. The figure below depicts the single right rotation process. In this diagram, we see that k2 has been shown as the root node to this function. We see that k1 is the left subtree of k2.

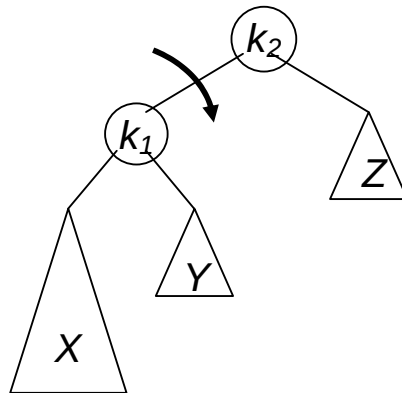


Fig 23.1: Single Right Rotation

We are going to apply single right rotation on the link between k1 and k2. The node k1 will be the new root node after rotation. So we get its value by the following statement

```
TreeNode <int>* k1 = k2 -> getLeft() ;
```

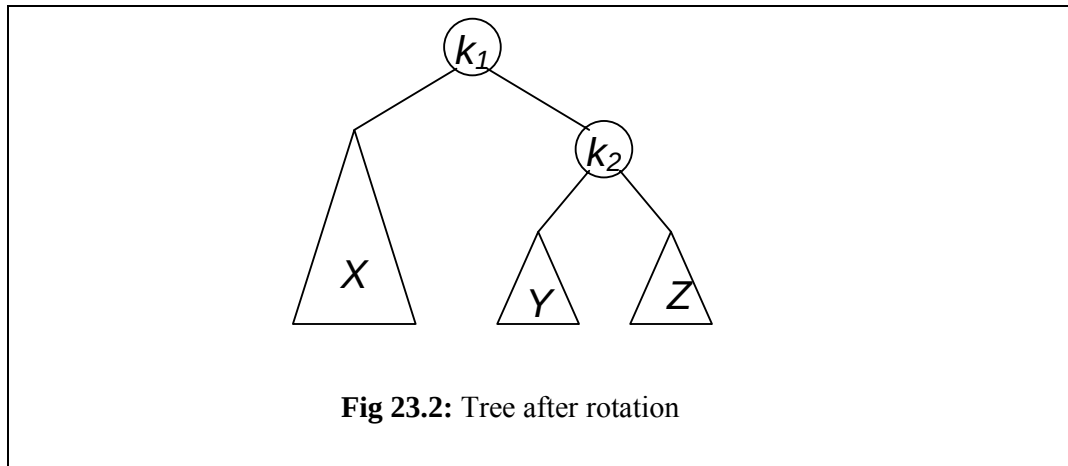
Due to the single right rotation, k2 has come down, resulting in the upward movement of k1. The tree Y has to find its new place. The place of the trees X and Z remains intact. The change of place of Y is written in the code as below.

```
k2->setLeft( k1->getRight() );
```

In the above statement, we get the right child of k1 (i.e. k1 -> getRight()) i.e. Y and pass it to setLeft function of k2. Thus it becomes the left child of k2. By the statement

```
k1 -> setRight( k2 ) ;
```

We set the node k2 as right child of k1. Now after these three statements, the tree has been transformed into the following figure.



From the above figure, it is reflected that k_1 is now the root of the tree while k_2 is its right child. The Y, earlier the right subtree of k_1 (see fig 23.1), has now become the right subtree of k_2 . We see that the inorder traversal of this tree is X k_1 Y k_2 Z. It is the same as of the tree before rotation in fig 23.1 i.e. X k_1 Y k_2 Z.

Now we set the heights of k_1 and k_2 in this re-arranged tree. In the code, to set the height of k_2 , we have written

```
int h = Max(height(k2->getLeft()), height(k2->getRight()));
k2->setHeight( h+1 );
```

Here we take an integer h and assign it a value. This value is the maximum height among the two subtrees i.e. height of left subtree of k_2 and height of right subtree of k_2 . Then we add 1 to this value and set this value as the height of k_2 . We add 1 to h as the height of root is one more than its child.

Similarly we set the height of k_1 and get the heights of its left and right subtrees before finding the higher value of these two by the Max function. Afterwards, we add 1 to this value and set this value as the height of k_1 . The following two statements perform this task.

```
h = Max( height(k1->getLeft()), k2->getHeight());
k1->setHeight( h+1 );
```

In the end, we return the root node i.e. k_1 . Thus the right single rotation is completed. In this routine, we saw that this routine takes root node of a tree as an argument and then rearranges it, resets the heights and returns the new root node.

Height Function

In the above SingleRightRotation, we used the *height* routine. This routine calls the *getHeight* function for the argument passed to it and returns the value got from that function. If there is an empty tree, then by definition its height will be -1. So in this routine, we return the value -1 if the argument passed to it is NULL. Following is the code of this routine.

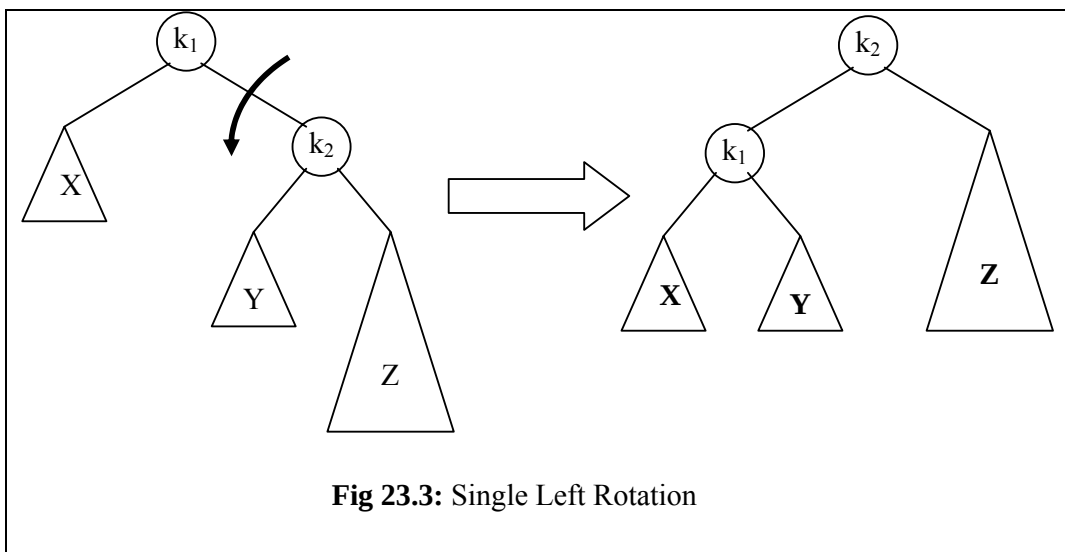
```

int height( TreeNode<int>* node )
{
    if( node != NULL ) return node->getHeight();
    return -1;
}

```

Single Left Rotation

This rotation is almost similar to the single right rotation. We start from k_1 . k_1 is the root while k_2 is its right child. Due to the left rotation, the right child k_2 of the root k_1 will become the root. The k_1 will go down to the left child of k_2 . The following figure shows that we have to change the positions of some pointers. The condition of the tree before and after single left rotation is also seen.



Following is the code of the function performing the single left rotation.

```

TreeNode<int>* singleLeftRotation( TreeNode<int>* k1 )
{
    if( k1 == NULL ) return NULL;

    // k2 is now the new root
    TreeNode<int>* k2 = k1->getRight();
    k1->setRight( k2->getLeft() ); // Y
    k2->setLeft( k1 );

    // reassign heights. First k1 (demoted)
    int h = Max(height(k1->getLeft()), height(k1->getRight()));
    k1->setHeight( h+1 );

    // k1 is now k2's left subtree
    h = Max( height(k2->getRight()), k1->getHeight());
    k2->setHeight( h+1 );
    return k2;
}

```

```
}

```

In the above code, it is evident that if the node passed to the function is not NULL (that means k1 is not NULL), we assign to k2 the value of k1 -> getRight (). Now in the rotation, Y tree i.e. the left subtree of k2, becomes the right subtree of k1. We write it in the next statement as.

```
k1->setRight( k2->getLeft() );
```

In this statement, we get the left child of k2 i.e. Y and set it as the right child of k1. Afterwards, k1 is set as the left child of k2. The following statement reflects this process.

```
k2->setLeft( k1 );
```

After single left rotation, the k2 becomes the root node after going up. However, k1 goes down as the left child of k2. The tree Y, earlier the left child of k2 has, now become the right child of k1. X and Z continue to remain at the previous positions. The tree has been transformed with changes into the heights of k1 and k2. We reassign the heights to these nodes in line with the process adopted in the right single rotation. At first, we adjust the height of k1. We get the heights of left and right subtrees of k1. The greater of these is taken and assigned to an int h. Then we add 1 to this value and set it as the height of k1. The following two lines of the code execute this task.

```
int h = Max(height(k1->getLeft()), height(k1->getRight()));
k1->setHeight( h+1 );
```

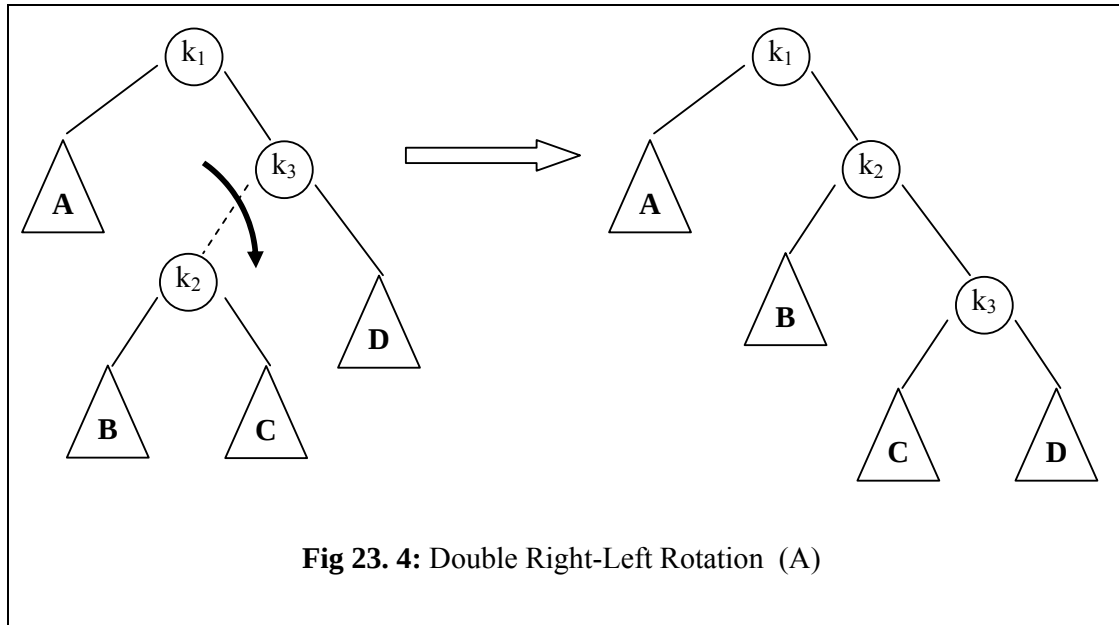
Similarly, we adjust the height of k2 by getting the greater one among the heights of its right subtree and its left subtree. Taking k1 as its left subtree, we get the height of k1. We add 1 and set it as the height of k2. Following are the statements that perform this task.

```
h = Max( height(k2->getRight()), k1->getHeight());
k2->setHeight( h+1 );
```

Finally, k2 is returned as the root of the tree.

Double Right-Left Rotation

As obvious from the nomenclature, in the double rotation, we at first carry out the right rotation before going ahead with the left rotation. Let's say that we pass it the node k1. Look at the following figure to see what are the nodes k1, k2 and k3. The figure 23.4 shows the first step (A) of double right-left rotation that is the single right rotation. It shows the rearranged tree on the right side in the figure.



In the left portion of the above figure, we see that k_1 is the root of the tree. k_3 is the right child of k_1 while k_2 is the left child of k_3 . A, B, C and D are trees. We carry out the right rotation between the link of k_3 and k_2 . In the code, it is shown that if k_1 is not NULL, we go ahead and perform the rotation. The code of this double rotation is given below.

```
TreeNode<int>* doubleRightLeftRotation(TreeNode<int>* k1)
{
    if( k1 == NULL ) return NULL;

    // single right rotate with k3 (k1's right child)
    k1->setRight( singleRightRotation(k1->getRight()));

    // now single left rotate with k1 as the root
    return singleLeftRotation(k1);
}
```

We perform the right rotation with the help of k_3 . Now think about the single right rotation. We are at k_3 and its left child k_2 will be involved in the rotation. In the code, for the single right rotation, we have written

$k1 \rightarrow \text{setRight}(\text{singleRightRotation}(k1 \rightarrow \text{getRight()}));$

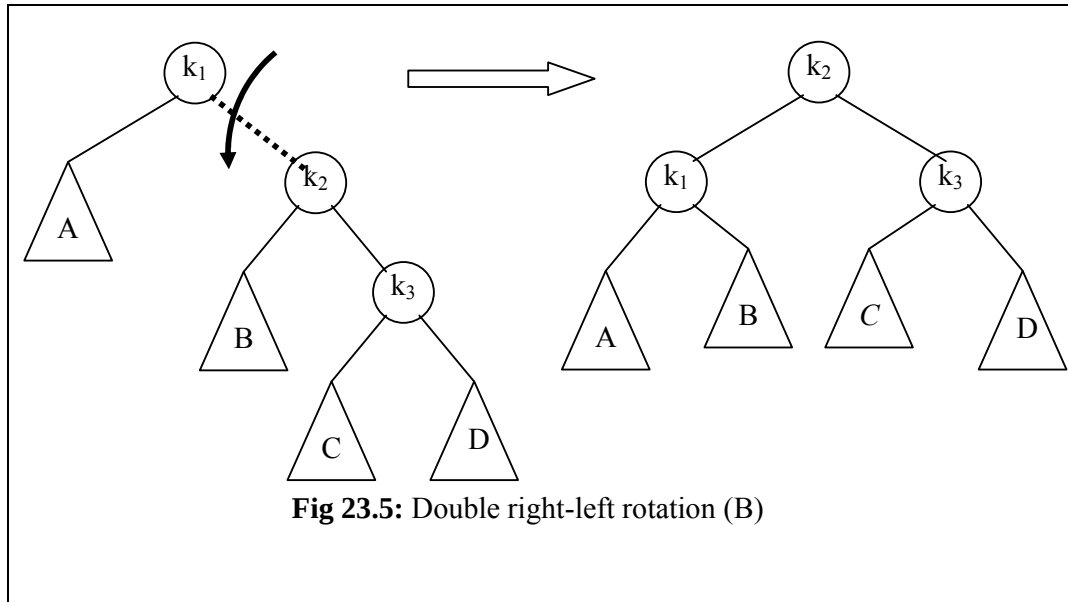
Here, we are passing the right child of k_1 (i.e. k_3) to the single right rotation. The function `singleRightRotation` itself will find the left child of k_3 to perform the single right rotation. The `singleRightRotation` will take k_3 downward and bring up its left child i.e. k_2 . The left subtree of k_2 i.e. C, now becomes the right subtree of k_3 . Due to this single right rotation, the tree will be transformed into the one, shown in the right

part of the figure 23.4.

After this we apply the single left rotation on this tree that is got from the single right rotation. This single left rotation is done by passing it k1. In this rotation, k1 goes down and k2 i.e. the right child of k1, becomes the root. We return the result of single left rotation. The following statement in the above code performs this task.

```
return singleLeftRotation(k1);
```

Thus the tree gets its final form i.e. after the double right-left rotation. The following figure shows the single left rotation and the final form of the tree.



In the above figure, it was witnessed that the double rotation consists of few statements. That's why, we have had written the routines for single right and left rotations. In the double right-left rotation, we just carried out a single right rotation and the single left rotation to complete the double right-left rotation.

Double Left-Right Rotation

While performing the double left-right rotation, we simply carry out the single left rotation at first. It is followed by the single right rotation. Following is the code of the double Left-Right rotation.

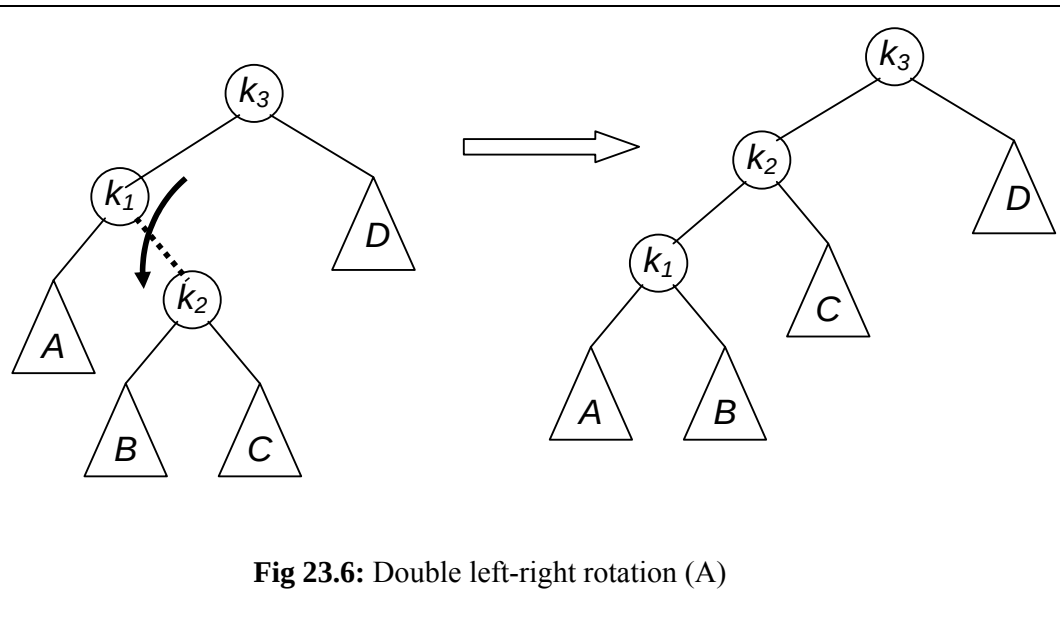
```
TreeNode<int>* doubleLeftRightRotation(TreeNode<int>* k3)
{
    if( k3 == NULL ) return NULL;

    // single left rotate with k1 (k3's left child)
    k3->setLeft( singleLeftRotation(k3->getLeft()));

    // now single right rotate with k3 as the root
    return singleRightRotation(k3);
}
```

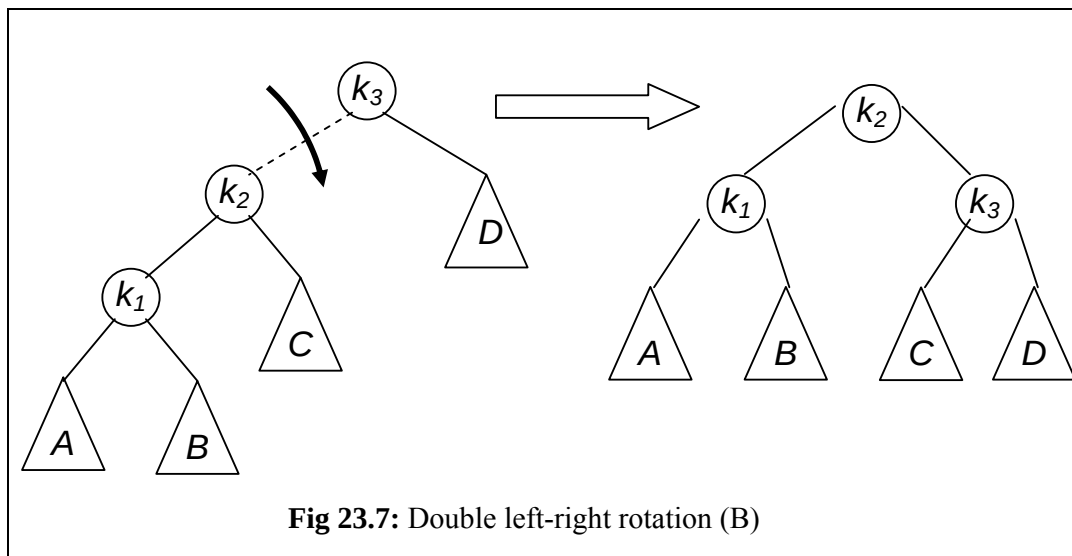
}

In the above code, we receive the argument node as k_3 . As we have to do the left rotation first, it is applied to the left child of k_3 . From the figure below (Fig 23.6), we see that the left child of k_3 is k_1 . However, when we send k_1 to the singleLeftRotation it will take the right child of k_1 i.e. k_2 . This process will rotate the link between k_1 and k_2 . The tree formed as a result of this single left rotation, is shown at the right side in the figure 23.6.



In this new rearranged tree, k_2 comes up while k_1 goes down, becoming the left subtree of k_2 . The subtree B, earlier left subtree of k_2 , now becomes the right subtree of k_1 .

The second step in the double left-right rotation is to apply the single right rotation on this new tree. We will carry out the single right rotation on k_3 . The pictorial representation of this single right rotation on k_3 is given in the figure below.



In the code, we wrote the statement for this as below.

```
return singleRightRotation(k3);
```

We pass $k3$ to the `singleRightRotation`. It is, internally applied to the left child of $k3$ i.e. $k2$. Now due to the right rotation, $k2$ will come up and $k3$ will become its right child. The node $k1$ will remain the left child of $k2$. The subtree C that was the right subtree of $k2$, will now be the left subtree of $k3$.

By now, the discussion on the insert routine has been completed. We have written the insert routine so that data item could be passed to it and inserted at a proper position in the tree. After inserting a node, the routine checks the balance or height factors and does the left or right rotation if needed and re-balances the tree.

Note that a property of the AVL tree is that while inserting a node in an AVL tree if we have to balance the tree then we have to do only one single right or left rotation or one double rotation to balance the tree. There is no such situation that we have to do a number of rotations. We do the one single or one double rotation at the node whose balance has become 2 after inserting a node.

Deletion in AVL Tree

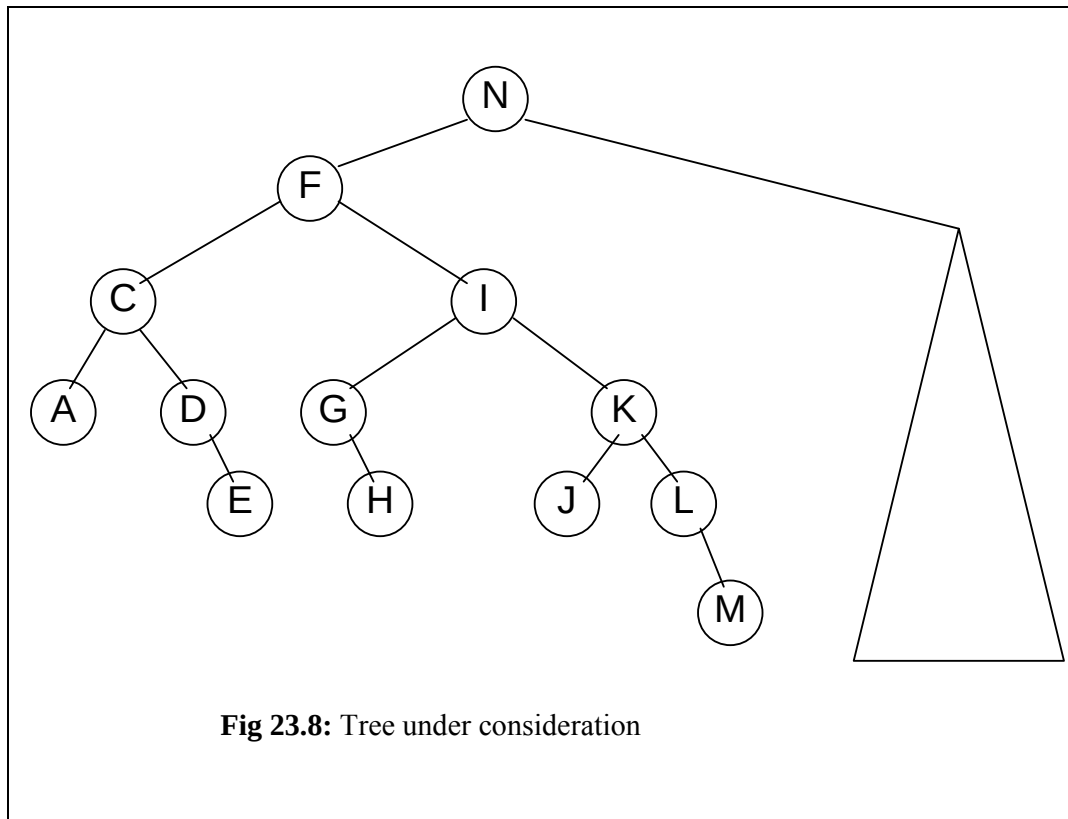
The deletion of a data item from a data structure has always been difficult whatever data structure we use. The deletion is the inverse of insertion. In deletion there is a given value x and an AVL tree T . We delete the node containing the value x and rebalance the tree if it becomes unbalance after deleting the node. We will see that the deletion of a node is considerably more complex than the insertion of a node. It is complex in the sense that we may have to do more than one rotations to rebalance the tree after deleting a node. We will discuss the deletion case by case and will see that about what points we have to take care in the deletion process.

We are not going to write the code for deletion here. If we have to use AVL tree routines or class, the deletion and insertion routines of AVL tree are available in standard library. We can use these routines in our program. We have no need to write these routines. But here we discuss these to understand their functionality.

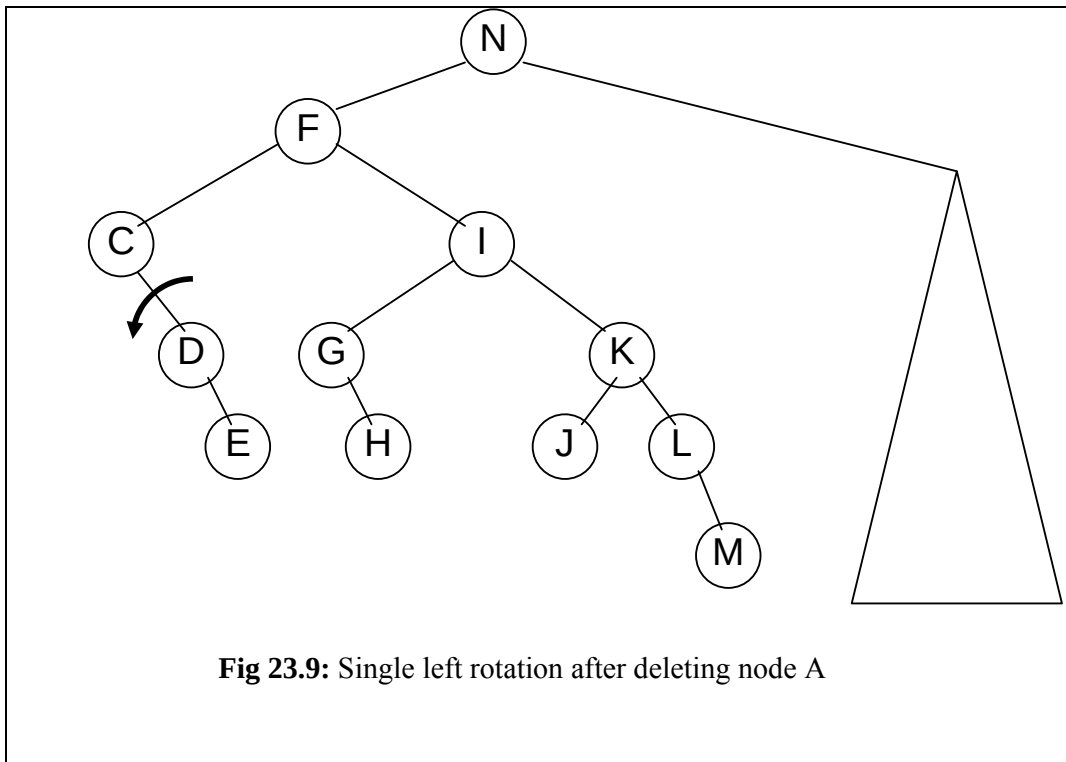
We know that insertion in a height-balanced tree requires at most one single rotation or one double rotation. We do this rotation at the node whose balance violates the AVL condition. We can use rotations to restore the balance when we do a deletion. If the tree becomes unbalance after deleting a node then we use rotations to rebalance it. We may have to do a rotation at every level of the tree. Thus in the worst case of deletion we have to do $\log_2 N$ rotations. As $\log_2 N$ is the number of levels of a tree of N nodes.

Let's consider an example of deleting a node from a tree. In this example, we will discuss the worst case of deletion that is we have to do rotation at each level after deleting a node. Look at the following figure i.e. Fig 23.8. In this figure the root of the tree is node N and we have expanded only the left subtree of this node. The right subtree of it is indicated by a triangle. We focus on the left subtree of N . The balance of each non-leaf node of the left subtree of N is -1 . This means that at every non-leaf node the depth/height of left subtree is one shorter than the height of right subtree. For example look at the node C . The left subtree of C is one level deep where as it's right subtree is two levels deep. So balance of it is $1 - 2 = -1$. If we look at node I its left

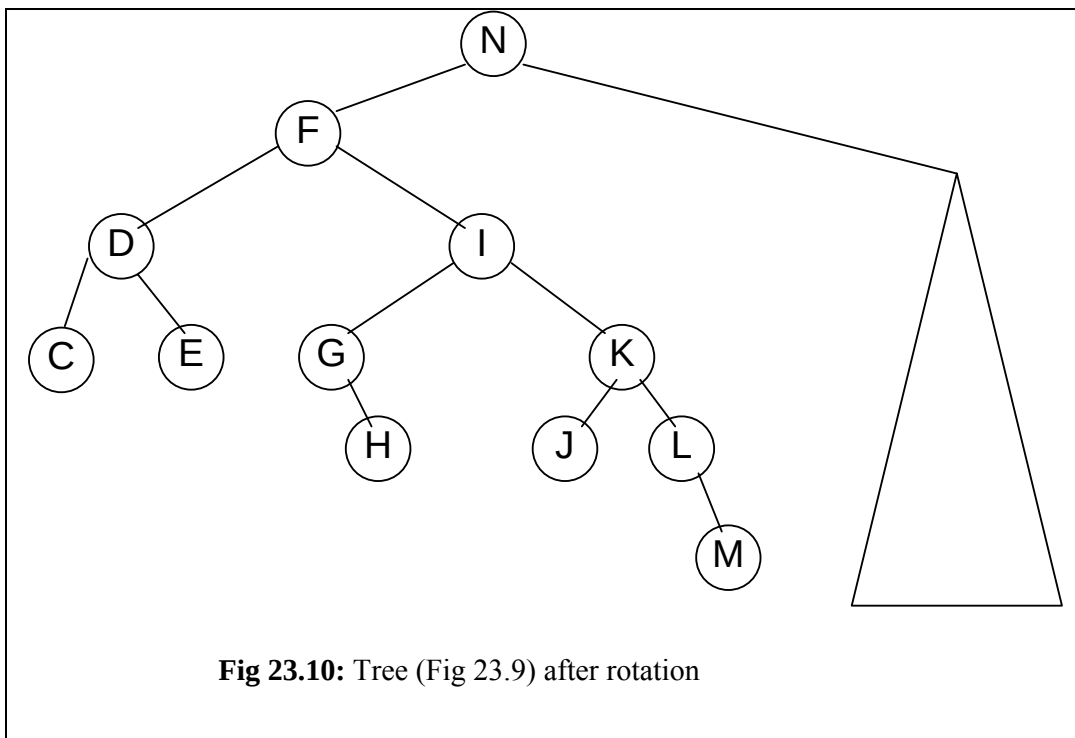
subtree has height 2 as there are two levels where nodes G and H exists. The right subtree of I has number of levels (i.e. height) 3 where exists the nodes K, L and M respectively. Thus the balance of I is $2 - 3 = -1$. Similarly we can see that other nodes also have the balance -1 . This tree is shown in the following figure.



Here in this tree, the deletion of node A from the left subtree causes the worst case of deletion in the sense that we have to do a large number of rotations. Now we delete the node A from the tree. The effect of this deletion is that if we consider the node C the height of its left subtree is zero now. The height of the right subtree of C is 2. Thus the balance of C is 2 now. This makes the tree unbalance. Now we will do a rotation to make the tree balanced. We rotate the right subtree of C that means the link between C and D so that D comes up and C goes down. This is mentioned in the figure below.



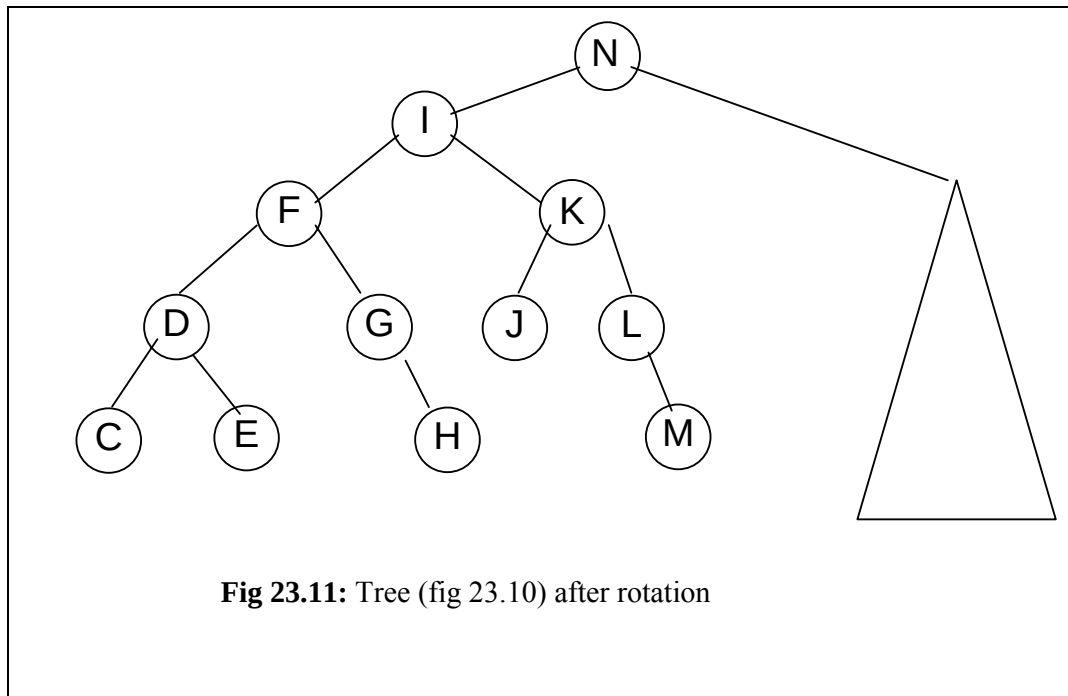
After this rotation the tree has transformed into the following figure. Now D becomes the left child of F and C becomes the left child of D.



By looking at the inorder traversal of the tree, we notice that it is preserved. The inorder traversal of the tree before rotation (i.e. fig 23.8) is C D E F G H I J K L M N.

Now if we traverse the tree after rotation (i.e. fig 23.9) by inorder traversal we get C D E F G H I J K L M N, which is the same as it was before rotation.

After this rotation we see that the tree having D as root is balanced. But if we see the node F we notice that the height of its left subtree is 2 and height of its right subtree is 4. Thus the balance of F is -2 (or 2 if we take the absolute value) now. Thus the tree becomes unbalance. So we have to do rotation again to balance the tree. The whole left subtree of F is shorter so we do the left rotation on the link of F and I (in the tree in fig 23.9) to bring F down and I up so that the difference of height could be less. After rotation the tree gets the new form that is shown in the figure below.



Here we see that the nodes G and H, which were in the left subtree of I, now become the right subtree of F. We see that the tree with I as root is balanced now. Now we consider the node N. We have not expanded the right subtree of N. Although we have not shown but there may be nodes in the right subtree of N. If the difference of heights of left and right subtree of N is greater than 1 then we have to do rotation on N node to balance the tree.

Thus we see that there may be such a tree that if we delete a node from it we have to do rotation at each level of the tree. We notice that we have to do more rotations in deletion as compared to insertion. In deletion when we delete a node we have to check the balance at each level up to the root. We do rotation if any node at any level violates the AVL condition. If nodes at a level do not violate AVL condition then we do not stop here we check the nodes at each level and go up to the root. We know that a binary tree has $\log_2 N$ levels (where N is total number of nodes) thus we have to do $\log_2 N$ rotations. We have to identify the required rotations that mean we have to identify that which one rotation out of the four rotations (i.e. single left rotation, single right rotation, double right-left rotation and double left-right rotation) we have to do. We have to identify this at each level.

We can summarize the deletion procedure as under.

Delete the node as in binary search tree. We have seen in the discussion of deleting a node from a BST that we have three cases, which we discussed as follows

Case I: The node to be deleted is the leaf node i.e. it has no right or left child. It is very simple to delete such node. We make the pointer in the parent node pointing to this node as NULL. If the memory for the node has been dynamically allocated, we will release it.

Case II: The node to be deleted has either left child (subtree) or right child (subtree). In this case we bypass the node in such a way that we find the inorder successor of this node and then link the parent of the node to be deleted to this successor node. Thus the node was deleted.

Case III: The node to be deleted has both the left and right children (subtree). This is the most difficult case. In this case we find the inorder successor of the node. The left most node of the right subtree will be the inorder successor of it. We put the value of that inorder successor node into the node to be deleted. After it we delete the inorder successor node recursively.

In deletion in AVL tree, we delete the node as we delete it in a BST. In third case of deletion in BST we note that the node deleted will be either a leaf or have just one subtree (that will be the right subtree as node deleted is the left most subtree so it cannot have a left subtree). Now we are talking about deletion in an AVL tree. Since this is an AVL tree so if the deleted node has one subtree that subtree contains only one node. Why it is so? Think about its reason and answer.

After deleting the node we traverse up the tree from the deleted node checking the balance of each node at each level up to the root. Thus the deletion in AVL tree is like the deletion in BST except that here in AVL tree we have to rebalance the tree using rotations.

Cases of Deletion in AVL Tree

Now let's consider the cases of deletion that will help to identify what rotation will be applied at what point.

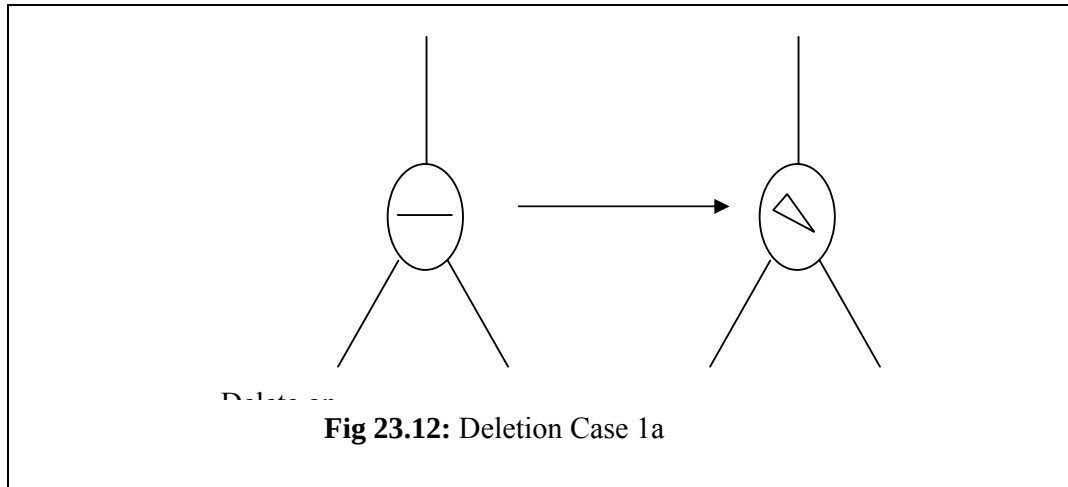
There are 5 cases to consider. Let's go through the cases graphically and determine what actions are needed to be taken. We will not develop the C++ code for deleteNode in AVL tree. This is left as an exercise.

Case 1a:

The first case is that the parent of the deleted node had a balance of 0 and the node was deleted in the parent's *left* subtree.

In the following figure (Fig 23.12) the left portion shows the parent node with a horizontal line in it. This horizontal line indicates that the left and right subtrees of this node have the same heights and thus the balance of this node is 0. When we delete a node from its left subtree then height of its right subtree becomes larger than the left subtree. The right portion in the figure shows this. We are showing a symbol instead of balance of node inside the node. The notation (symbol) in the node

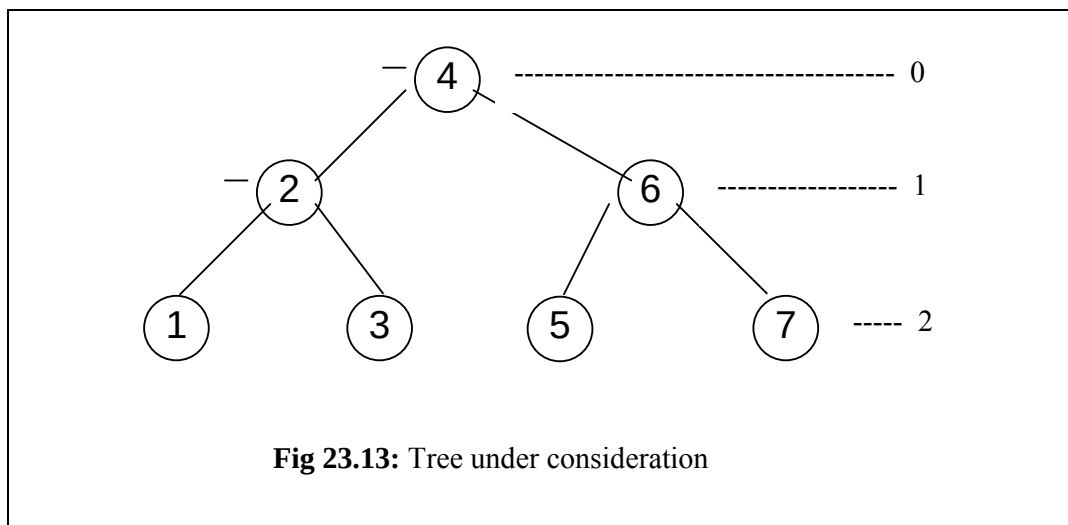
indicates that the height of left subtree is shorter than the height of right subtree of the node.



Now the action that will be taken in this case is that, change the balance of the parent node and stop. No further effect on balance of any higher node. There is no need of rotation in this case. Thus it is the easiest case of deletion.

Let's consider an example to demonstrate the above case.

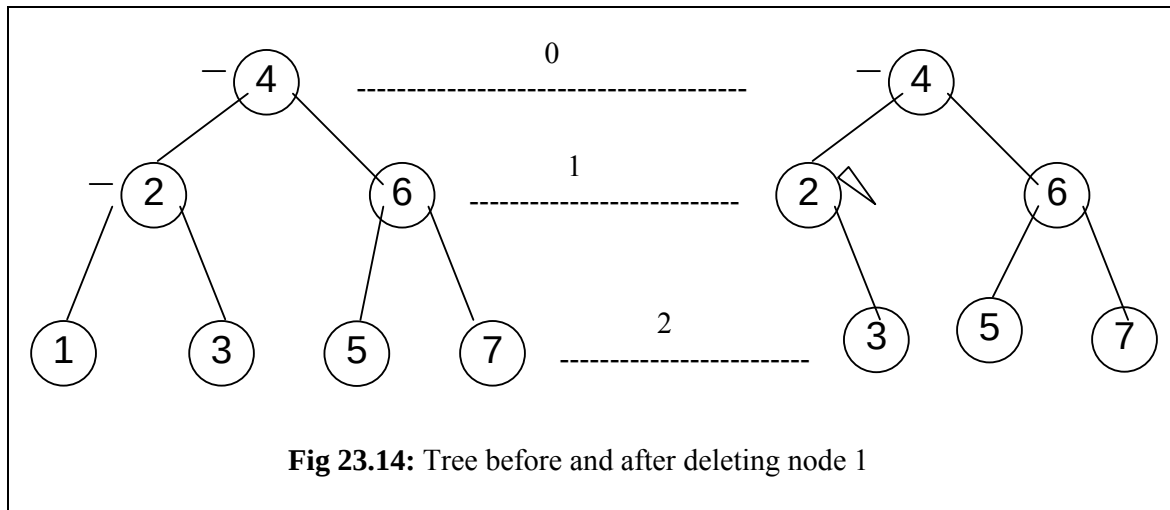
Consider the tree shown in the following figure i.e. Fig 23.13. This is a perfectly balanced tree. The root of this tree is 4 and nodes 1, 2 and 3 are in its left subtree. The nodes 5, 6 and 7 are in the right subtree.



Consider the node 2. We have shown the balance of this node with a horizontal line, which indicates that the height of its left subtree is equal to that of its right subtree. Similarly we have shown the balance of node 4.

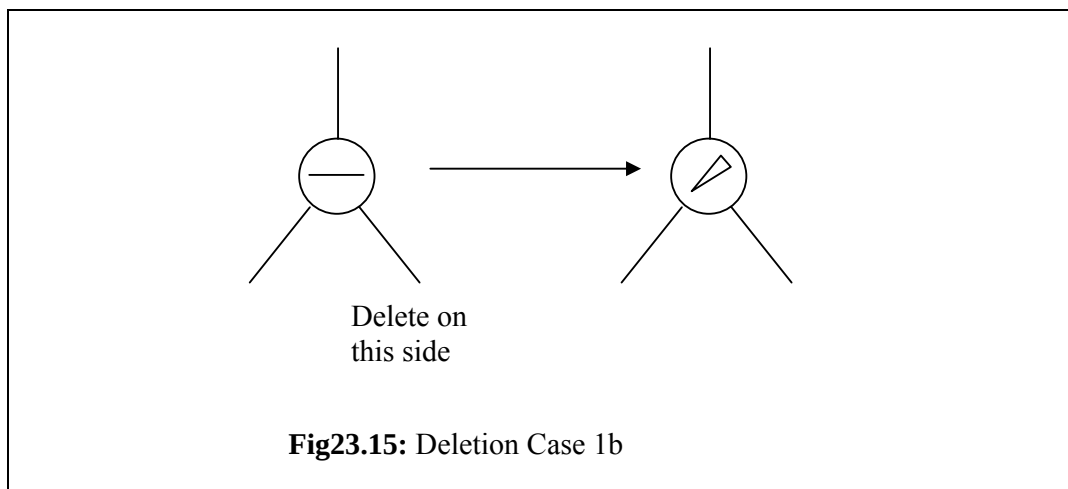
Now we remove the node 1 which is the left subtree of node 2. After removing the left child (subtree) of 2 the height of left subtree of 2 is 0. The height of right subtree of 2 is 1 so the balance of node 2 becomes -1 . This is shown in the figure by placing a sign that is down ward to right side, which indicates that height of right subtree is greater

than left subtree. Now we check the node 4. Here the height of left subtree of it is still 2. The height of its right subtree is also 2. So balance of the node 4 is 0 that is indicated by the small horizontal line (minus sign) in the figure below. Here we don't need to change the balance of 4.



Case 1b:

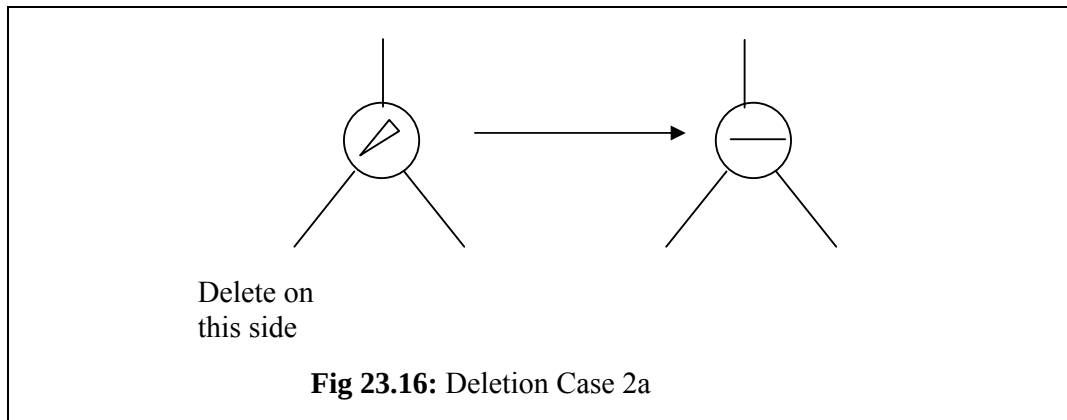
This case is symmetric to case 1a. In this case, the parent of the deleted node had a balance of 0 and the node was deleted in the parent's *right* subtree. The following figure shows that the balance of the node was zero as left and right subtree of it have the same heights.



After removing the right child the balance of it changes and it becomes 1, as the height of its left subtree is 1 greater than the height of right subtree. The action performed in this case is the same as that in case 1a. That is change the balance of the parent node and stop. No further effect on balance of any higher node. The previous example can be done for this case only with the change that remove the right child of node 2 i.e. 3.

Case 2a:

This is the case where the parent of the deleted node had a balance of 1 and the node was deleted in the parent's *left* subtree. This means that the height of left subtree of the parent node is 1 greater than the height of its right subtree. It is shown in the left portion of the figure below.



Now if we remove a node from the left subtree of this node then the height of left subtree will decrease by 1 and get equal to the height of right subtree. Thus the balance of this node will be zero. In this case, the action that we will perform to balance the tree is that change the balance of the parent node. This deletion of node may have caused imbalance in higher nodes. So it is advised to continue up to the root of the tree. We will do rotation wherever the balance of the node violates the AVL condition.

Data Structures

Lecture No. 24

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 4

4.4

Summary

- Deletion in AVL Tree
- Other Uses of Binary Trees

Deletion in AVL Tree

At the end of last lecture, we were discussing about deleting a node from an AVL tree. There are five cases to consider while deleting a node of an AVL tree. When a node is deleted, the tree can become unbalanced. We calculate the *balance factor* of each node and perform rotation for unbalanced nodes. But this rotation can prolong to the *root* node. In case of insertion, only one node's balance was adjusted as we saw in previous lectures but in case of deletion, this process of rotation may expand to the *root* node. However, there may also be cases when we delete a node and perform no or one rotation only.

Now, we will see the five cases of deletion. A side note is that we are not going to implement these cases in C++ in this lecture, you can do it yourself as an exercise with the help of the code given inside your text book. In this lecture, the emphasis will be on the deletion process and what necessary actions we take when a node is required to be deleted from an AVL tree. Actually, there are two kinds of actions taken here, one is deletion and the other one is the rotation of the nodes.

Case 1a: The parent of the deleted node had a balance of 0 and a node was deleted in the parent's left subtree.

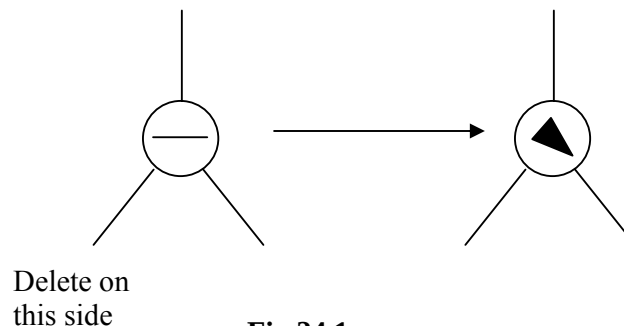


Fig 24.1

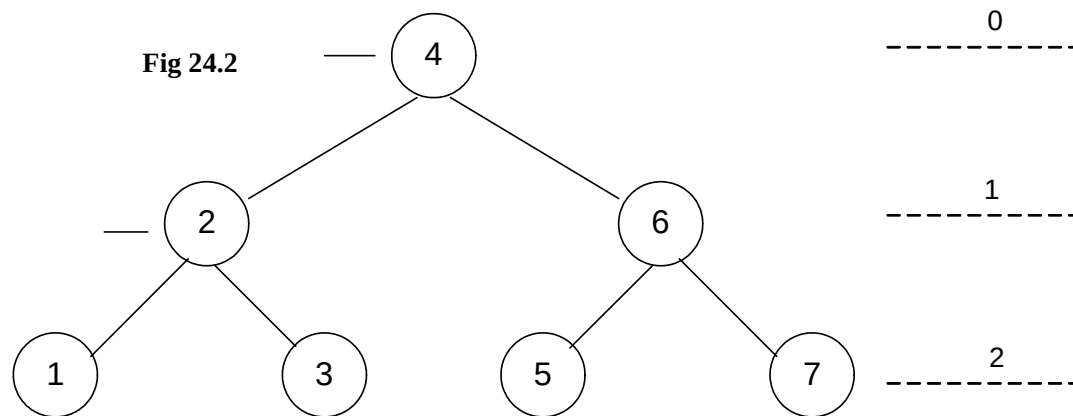
In the left tree in the *Fig 24.1*, the horizontal line inside the tree node indicates that the balance is 0, the *right* and *left* subtrees of the node are of equal levels. Now, when a node is deleted from *the left* subtree of this node, this may reduce by one level and cause the balance of the *right* subtree of the node to increase by 1 relatively. The

balance of the node in favor of the *right* subtree is shown by a triangular knob tilted towards right. Now, the action required in this case to make the tree balanced again is:

Change the balance of the parent node and stop. There is no further effect on balance of any higher node.

In this case, the balance of the tree is changed from 0 to -1 , which is within the defined limits of AVL tree, therefore, no rotation is performed in this case.

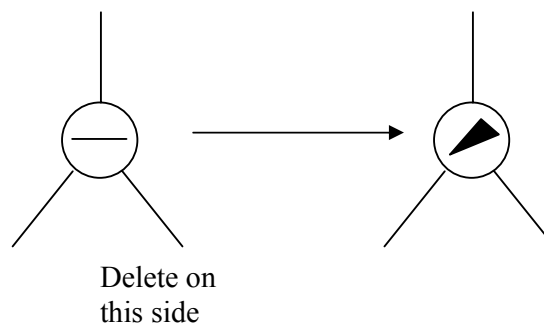
Below is a tree in which the height of the *left* subtree does not change after deleting one node from it.



The node 4 is the root node, nodes 2 and 6 are on level 1 and nodes 1, 3, 5, 7 are shown on level 2. Now, if we delete the node 1, the balance of the node 2 is tilted towards right, it is -1 . The balance of the *root* node 4 is unchanged as there is no change in the number of levels within *right* and *left* subtrees of it. Similarly, there is no change in the balances of other nodes. So we don't need to perform any rotation operation in this case.

Let's see the second case.

Case 1b: the parent of the deleted node had a balance of 0 and the node was deleted in the parent's right subtree.



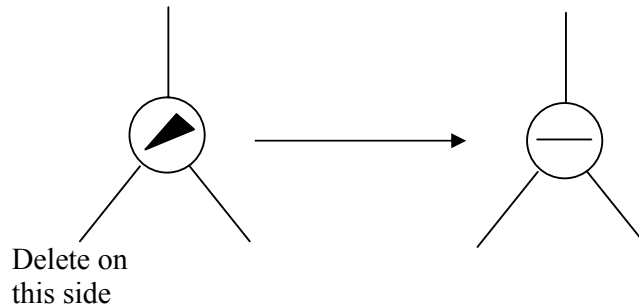
On the left of *Fig 24.3*, the **Fig 24.3** after a node is deleted from the *right* subtree of it. The balance of the tree is tilted towards left as shown in the right tree show in the *Fig 24.3*. Now, we see what action will be required to make the tree balanced again.

Change the balance of the parent node and stop. No further effect on balance of any higher node (same as 1a).

So in this case also, we don't need to perform rotation as the tree is still an AVL (as

we saw in the *Case 1a*). It is important to note that in both of the cases above, the balance of the parent node was 0. Now, we will see the cases when the balance of the parent node is not 0 previously.

Case 2a: The parent of the deleted node had a balance of 1 and the node was deleted in the parent's left subtree.



In the *Fig 24.4* **Fig 24.4** balance factor as 1, which means that the *left* subtree of the parent node is one level more than the number of levels in the *right* subtree of it. When we delete one node from the *left* subtree of the node, the height of the *left* subtree is changed and the balance becomes 0 as shown in the right side tree of *Fig 24.4*. But it is very important understand that this change of levels may cause the change of balance of higher nodes in the tree i.e.

Change the balance of the parent node. May have caused imbalance in higher nodes so continue up the tree.

So in order to ensure that the upper nodes are balanced, we calculate their *balance factors* for all nodes in higher levels and rotate them when required.

Case 2b: The parent of the deleted node had a balance of -1 and the node was deleted in the parent's right subtree.

Similar to the *Case 2a*, we will do the following action:

Change the balance of the parent node. May have caused imbalance in higher nodes so continue up the tree.

Now, we see another case.

Case 3a: The parent had balance of -1 and the node was deleted in the parent's left subtree, right subtree was balanced.

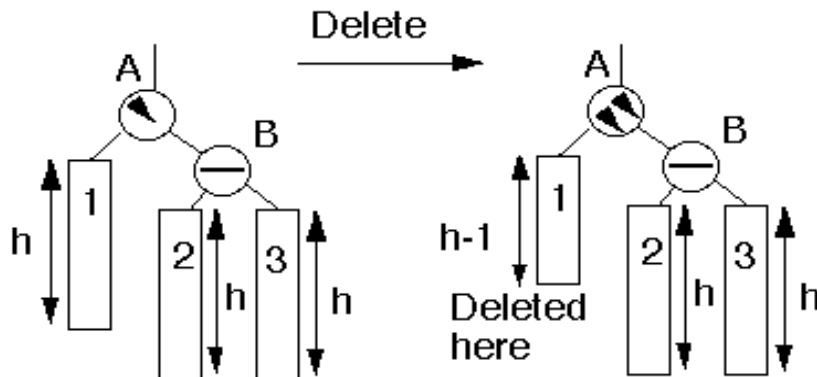
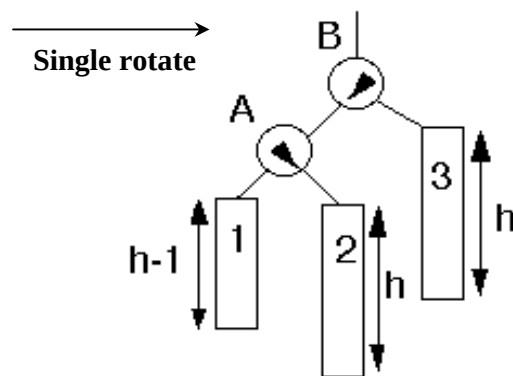


Fig 24.5

As shown in the *left* tree in Fig 24.5, the node A is tilted towards right but the *right* subtree of A (node B above) is balanced. The deleted node lies in the *left* subtree of the node A. After deletion, the height of the *left* subtree is changed to $h-1$ as depicted in the right tree of above figure. In this situation, we will do the following action:

Perform single rotation, adjust balance. No effect on balance of higher nodes so stop here.



Node A has become the *left* subtree of node B and node 2 *left* subtree of node B has become the *right* subtree of node A. The balance of node B is tilted towards left and balance of node A is tilted towards right but somehow, both are within AVL limits. Hence, after a single rotation, the balance of the tree is restored in this case.

Case 4a: Parent had balance of -1 and the node was deleted in the parent's left subtree, right subtree was unbalanced.

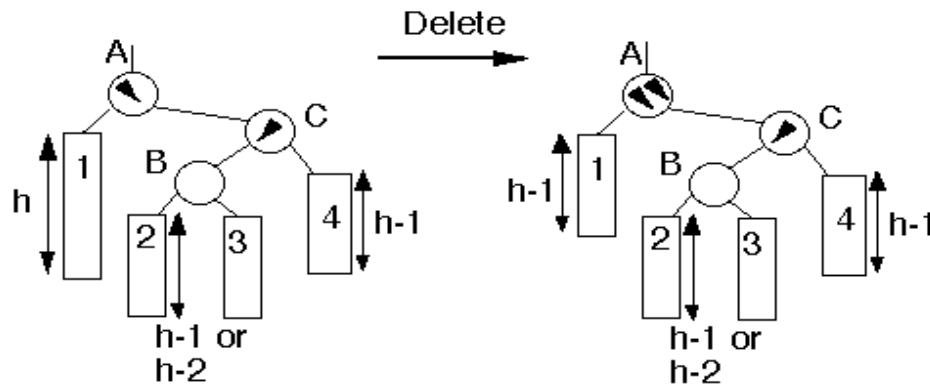


Fig 24.7

In the last *case 3a*, the *right* subtree of node A was balanced. But in this case, as shown in the figure above, the node C is tilted towards left. The node to be deleted lies in the *left* subtree of node A. After deleting the node the height of the *left* subtree of node A has become $h-1$. The balance of the node A is shown tilted towards right by showing two triangular knobs inside node A. So what is the action here.

Double rotation at B. May have affected the balance of higher nodes, so continue up the tree.

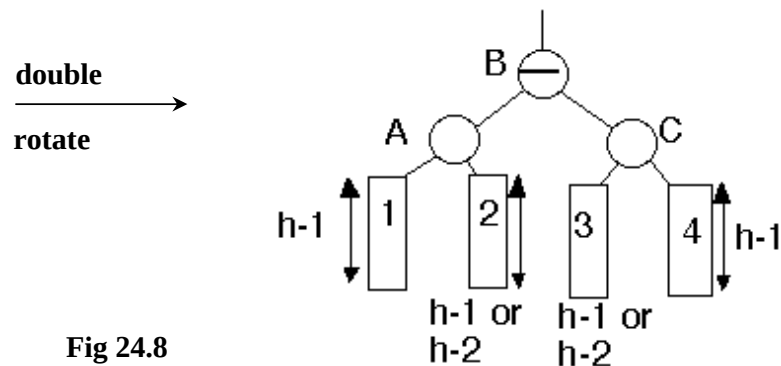


Fig 24.8

Node A, which was the *root* node previously, has become the *left* child of the new *root* node B. Node C, which was the *right* child of the *root* node C has now become the *right* child of the new *root* node B.

Case 5a: The parent had balance of -1 and the node was deleted in the parent's left subtree, right subtree was unbalanced.

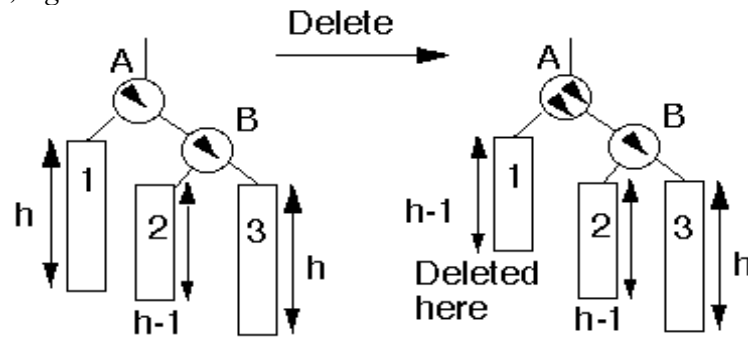


Fig 24.9

In the figure above, the *right* tree of the node B has a height of $h-1$ while the *right* subtree is of height h . When we remove a node from the *left* subtree of node A, the new tree is shown on the right side of Fig 24.9. The subtree 1 has height $h-1$ now, while subtrees 2 and 3 have the same heights. So the action we will do in this case is:

Single rotation at B. May have effected the balance of higher nodes, so continue up the tree **single**

rotate →

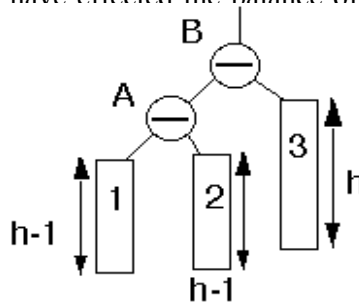


Fig 24.10

These were the five cases of deletion of a node from an AVL tree. Until now, we are trying to understand the concept using the figures. You might have noticed the phrase '*continue up the tree*' in the actions above. How will we do it? One way is to maintain the pointer of the parent node inside each node. But often the easiest method when we go in downward direction and then upward is *recursion*. In recursion, the work to be done later is pushed on to the stack and we keep on moving forward until at a certain point we back track and do remaining work present in the stack. We delete a node when we reach at the desired location and then while traversing back, do the rotation operation on our way to the *root* node.

Symmetrical to case 2b, we may also have cases 3b, 4b and 5b. This should not be a problem in doing it yourself.

Other Uses of Binary Trees

A characteristic of binary trees is that the values inside nodes on the left of a node are smaller than the value in the node. And the values inside the nodes on the right of a node are greater than the value in the node. This is the way a binary tree is constructed.

Whatever is the size of the tree, the search is performed after traversing upto $\log_2 n$ levels maximum.

We have observed that the binary tree becomes a linked list and it can become shallow. The AVL conditions came into picture to control the height balance of a binary tree. While searching in an AVL tree, in the worst case scenario we have to search $1.44 \log_2 n$ levels. For searches, binary and AVL trees are the most efficient but we do have some other kinds of trees that will be studied later.

Lets see what could be some other uses of binary trees, we start our discussion with *Expression Trees*.

Expression Trees

Expression trees, the more general parse trees and abstract syntax trees are significant components of compilers.

We already know about compilers that whenever we write our program or code in some computer language like C++ or Java. These programs are compiled into assembly language or byte code (in case of Java). This assembly language code is translated converted into machine language and an executable file is made by another program called the assembler.

By the way, if you have seen your syllabus, you might have seen that there is a dedicated subject on compilers. We study in detail about compilers in that course. For this course, we will see expression or parse trees.

We will take some examples of expression trees and we will not delve into much depth of it rather that would be an introduction to expression trees.

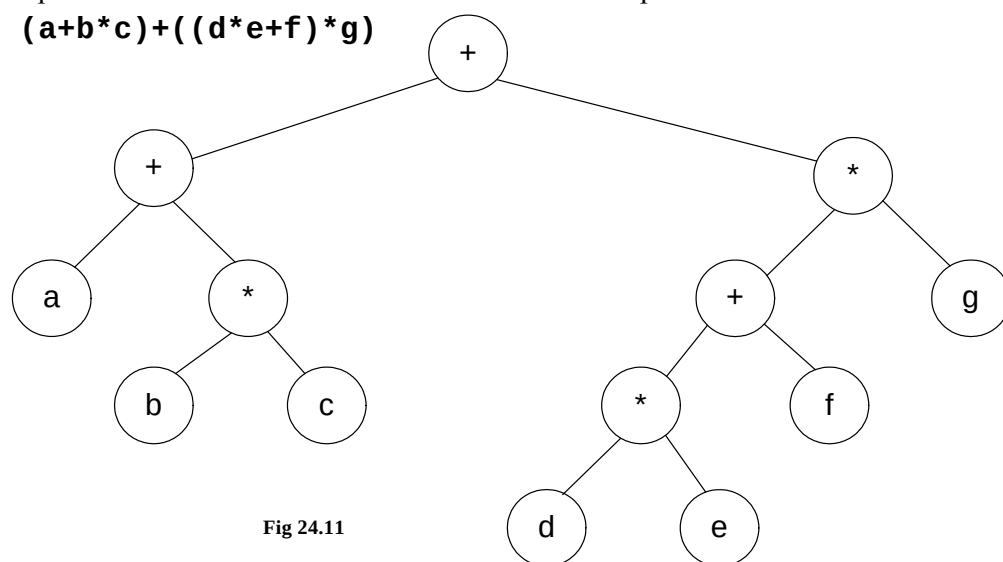


Fig 24.11

You can see the infix expression above $(a + b * c) + ((d * e + f) * g)$, it is represented in the tree form also.

You can see from bottom of the tree that the nodes b and c in the nodes are present at the same level and their parent node is multiplication (*) symbol. From the expression also, we see that the b and c are being multiplied. The parent node of a is + and *right* subtree of + is $b * c$. You might have understood already that this subtree is depicting $a + b * c$. On the right side, node d and e are connected to the parent *. Symbol + is the parent of * and node f. The expression of the subtree at node + is $d * e + f$. The parent of node + is * node, its *right* subtree is g. So expression of the subtree at this node * is $(d * e + f) * g$. The root node of the tree is +.

These expression trees are useful in compilers and in spreadsheets also, they are sometimes called parse trees.

Parse Tree in Compilers

See the expression tree of expression $A := A + B * C$ below. We are adding B and C, adding the resultant in A and then finally assigning the resultant to A.

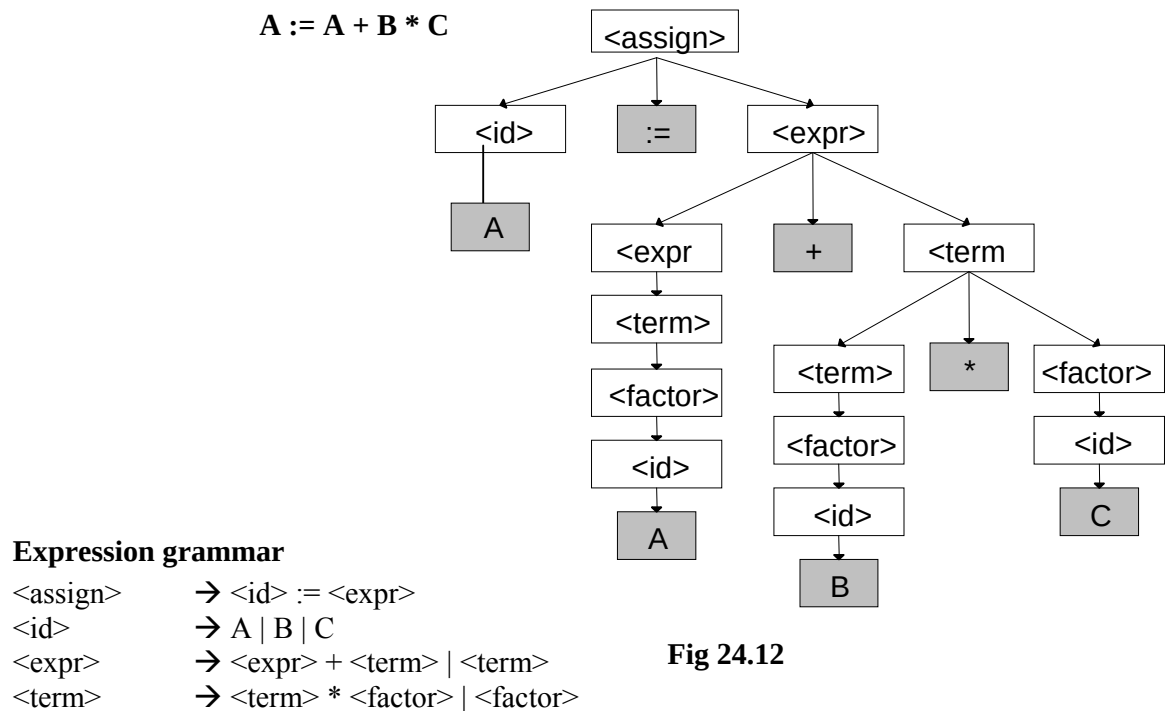


Fig 24.12

The *root* node in the parse tree shown above is <assign>.

The assignment statement (<assign>) has three parts. On the left of it, there is always an identifier (single or an array reference) called *l-value*. The *l-value* shown in the tree above is <id> and the identifier is A in the expression above. The second part of assignment statement is assignment operator (= or :=). On the right of assignment operator lies the third part of assignment statement, which is an expression. In the expression $A := A + B * C$ above, the expression after assignment operator is $A + B * C$. In the tree, it is represented by the node <expr>. The node <expr> has three subnodes: <expr>, + and <term>. <expr>'s further *left* subtree is <expr>, <term>, <factor>, <id> and then finally is B. The *right* subchild <term> has further subnodes as <term>, * and <factor>. <factor> has <id> as subchild and <id> has C.

Note the nodes in gray shade in the tree above form $A = A + B * C$.

Compiler creates these parse trees. We will see how to make these trees, when we will parse any language tree like C++. Parsing mean to read and then extract the required structure. A compiler parses a computer language like C++ to form parse trees. Similarly, when we do speech recognition. Each sentence is broken down into a certain structure in form of a tree. Hand writing recognition also involves this. The tablet PCs these days has lot of implementation of parse trees.

Parse Tree for an SQL Query

Let's see another example of parse trees inside databases. The parse trees are used in query processing. The queries are questions to the databases to see a particular kind of data. Consider you have a database for a video store that contains data of movies and artists etc. You are querying that database to find the titles of movies with stars born in 1960. The language used to query from the database is called SQL (Structured Query Language), this language will be dealt in depth in the databases course. The tables lying in this movies database are:

StarsIn(title, year, starName)

MovieStar(name, address, gender, birthdate)

The following SQL query is used to retrieve information from this database:

SELECT title

FROM StarsIn, MovieStar

WHERE starName = name AND birthdate LIKE '%1960' ;

This query is asking to retrieve the titles of those movies from *StarsIn* and *MovieStar* tables where the birthdate of an actor is 1960. We see in query processing a tree is formed as shown in the figure below:

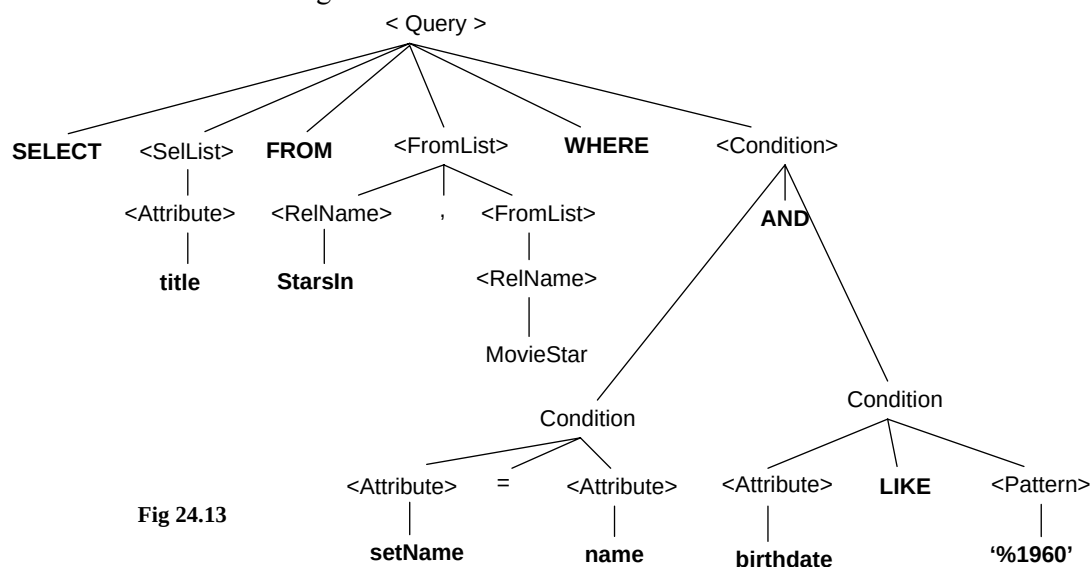


Fig 24.13

The *root* node is Query. This node is further divided into SELECT, < SelList >, FROM, < FromList >, WHERE and < Condition > subnodes. < SelList > will be an *Attribute* and finally a *title* is reached. Observe the tree figure above, how the tree is expanded when we go in the downward direction. When the database engine does the query process, it makes these trees. The database engine also performs query

optimization using these trees.

Compiler Optimization

Let's see another expression tree here:

Common subexpression:

$(f+d*e) + ((d*e+f)*g)$

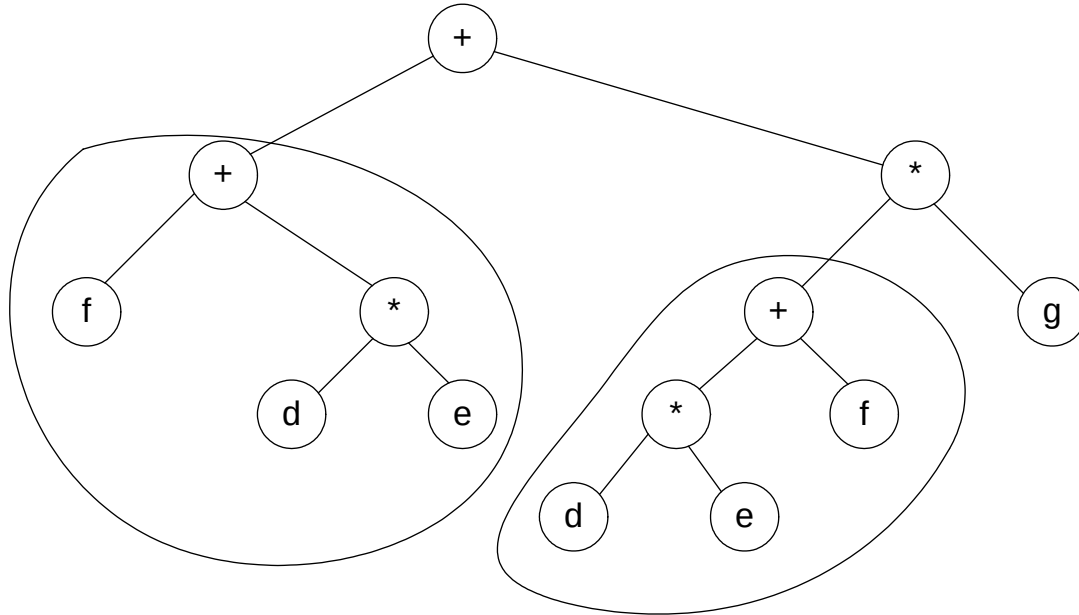


Fig 24.14

The *root* node is $+$, *left* subtree is capturing the $f+d*e$ expression while the right subtree is capturing $(d*e+f)*g$.

Normally compilers has intelligence to look at the common parts inside parse trees. For example in the tree above, the expressions $f+d*e$ and $d*e+f$ are same basically. These common subtrees are called *common subexpressions*. To gain efficiency, instead of calculating the *common subexpressions* again, the compilers calculates them once and use them at other places. The part of the compiler that is responsible to do this kind of optimization is called *optimizer*.

See the figure below, the optimizer (part of compiler) will create the following graph while performing the optimization. Because both subtrees were equivalent, it has taken out one subtree and connected the link from node $*$ to node $+$.

(Common Subexpression:
 $(f+d*e) + ((d*e+f)*g)$)

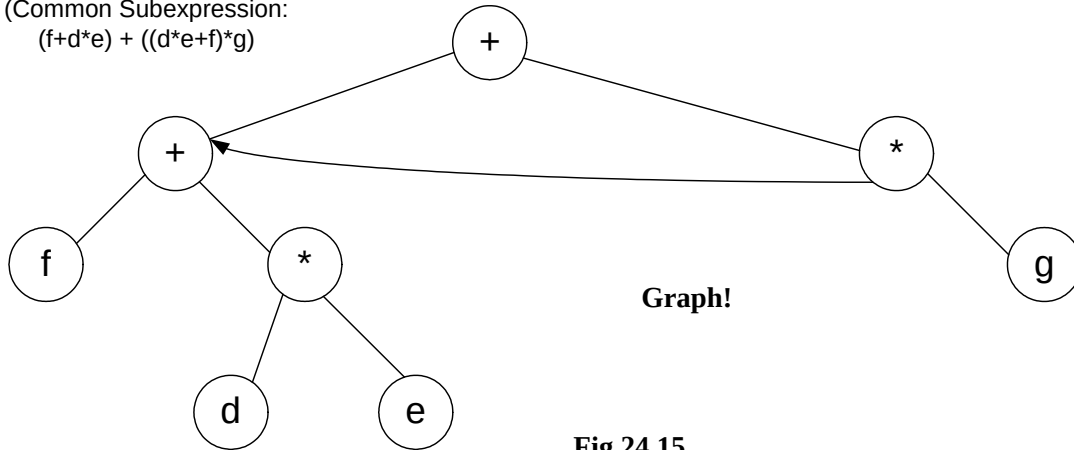


Fig 24.15

This figure is not a tree now because it has two or more different paths to reach a node. Therefore, this has become a graph. The new connection is containing a directed edge, which is there in graphs.

Optimizer uses the expressions trees and converts them to graphs for efficiency purposes. You read out from your book, how the expression trees are formed, what are the different methods of creating them.

Data Structures

Lecture No. 25

Reading Material

Data Structures and Algorithm Analysis in C++

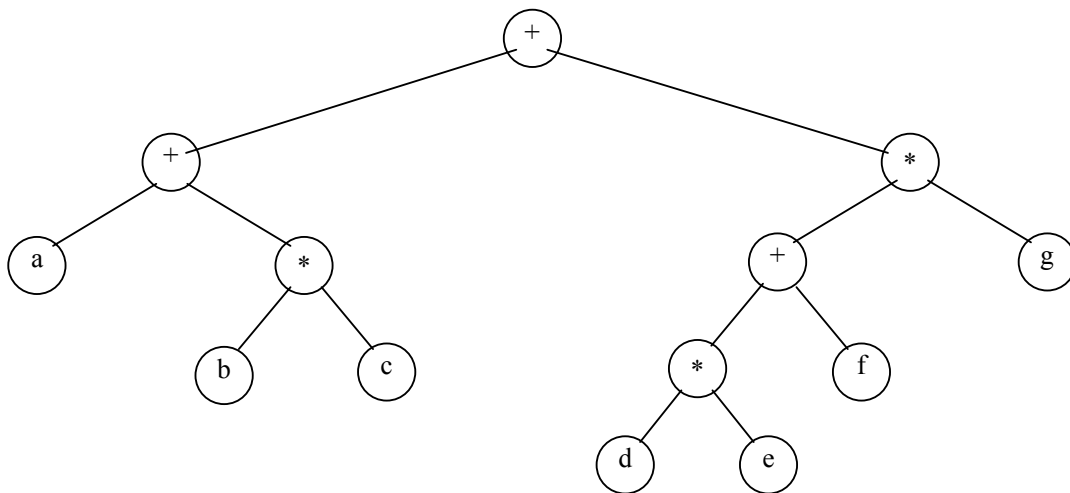
Chapter. 4, 10
4.2.2, 10.1.2

Summary

- Expression tree
- Huffman Encoding

Expression tree

We discussed the concept of expression trees in detail in the previous lecture. Trees are used in many other ways in the computer science. Compilers and database are two major examples in this regard. In case of compilers, when the languages are translated into machine language, tree-like structures are used. We have also seen an example of expression tree comprising the mathematical expression. Let's have more discussion on the expression trees. We will see what are the benefits of expression trees and how can we build an expression tree. Following is the figure of an expression tree.



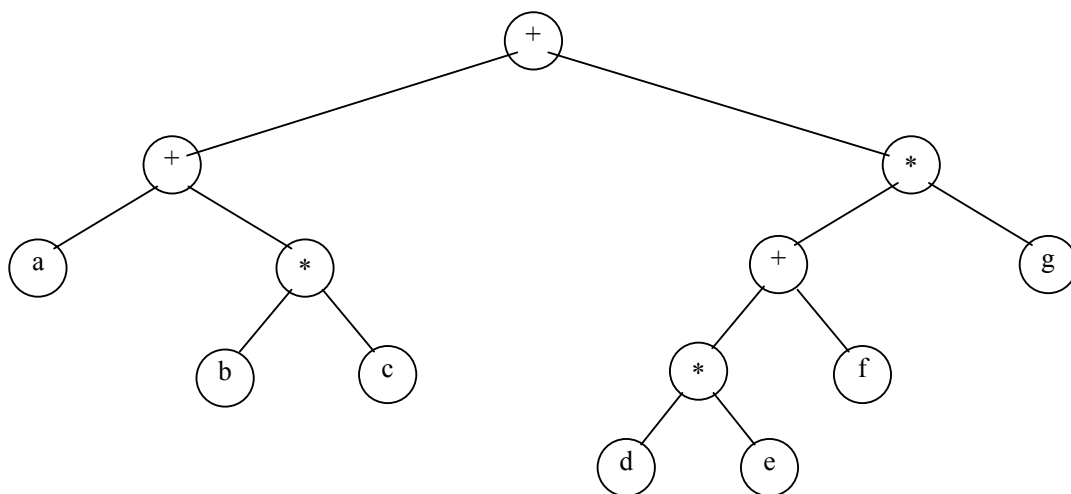
In the above tree, the expression on the left side is $a + b * c$ while on the right side, we have $d * e + f * g$. If you look at the figure, it becomes evident that the inner nodes contain operators while leaf nodes have operands. We know that there are two types of nodes in the tree i.e. inner nodes and leaf nodes. The leaf nodes are such nodes which have left and right subtrees as null. You will find these at the bottom level of the tree. The leaf nodes are connected with the inner nodes. So in trees, we have some inner nodes and some leaf nodes.

In the above diagram, all the inner nodes (the nodes which have either left or right

child or both) have operators. In this case, we have + or * as operators. Whereas leaf nodes contain operands only i.e. a, b, c, d, e, f, g . This tree is binary as the operators are binary. We have discussed the evaluation of postfix and infix expressions and have seen that the binary operators need two operands. In the infix expressions, one operand is on the left side of the operator and the other is on the right side. Suppose, if we have + operator, it will be written as $2 + 4$. However, in case of multiplication, we will write as $5 * 6$. We may have unary operators like negation (-) or in Boolean expression we have NOT. In this example, there are all the binary operators. Therefore, this tree is a binary tree. This is not the Binary Search Tree. In BST, the values on the left side of the nodes are smaller and the values on the right side are greater than the node. Therefore, this is not a BST. Here we have an expression tree with no sorting process involved.

This is not necessary that expression tree is always binary tree. Suppose we have a unary operator like negation. In this case, we have a node which has (-) in it and there is only one leaf node under it. It means just negate that operand.

Let's talk about the traversal of the expression tree. The inorder traversal may be executed here.



Inorder traversal yields: $a + b * c + d * e + f * g$

We use the inorder routine and give the root of this tree. The inorder traversal will be $a + b * c + d * e + f * g$. You might have noted that there is no parenthesis. In such expressions when there is addition and multiplication together, we have to decide which operation should be performed first. At that time, we have talked about the operator precedence. While converting infix expression into postfix expression, we have written a routine, which tells about the precedence of the operators like multiplication has higher precedence than the addition. In case of the expression $2 + 3 * 4$, we first evaluate $3 * 4$ before adding 2 to the result. We are used to solve such expressions and know how to evaluate such expressions. But in the computers, we have to set the precedence in our functions.

We have an expression tree and perform inorder traversal on it. We get the infix form of the expression with the inorder traversal but without parenthesis. To have the

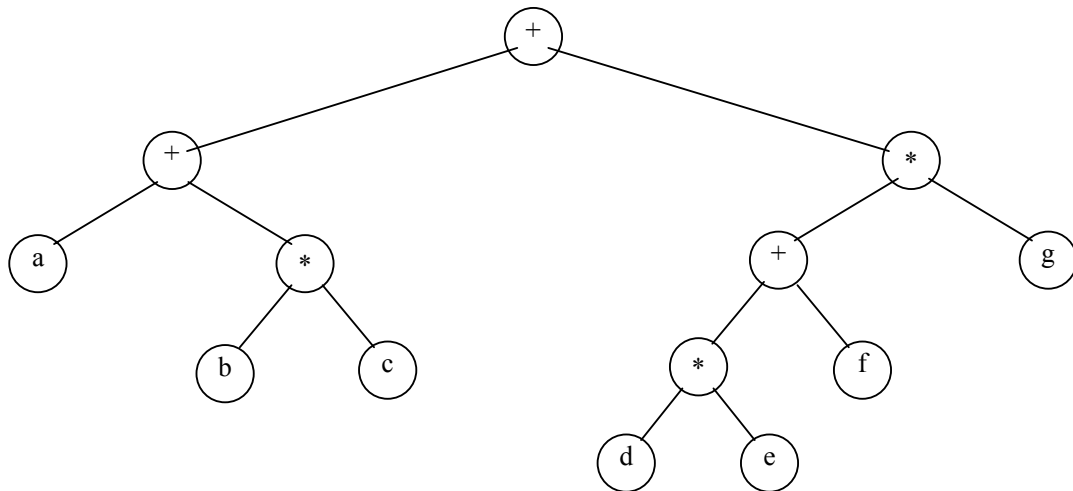
parenthesis also in the expressions, we will have to do some extra work.

Here is the code of the inorder routine which puts parenthesis in the expression.

```
/* inorder traversal routine using the parenthesis */  
  
void inorder(TreeNode<int>* treeNode)  
{  
    if( treeNode != NULL ){  
        cout << "(";  
        inorder(treeNode->getLeft());  
        cout << ")";  
        cout << *(treeNode->getInfo());  
        cout << "(";  
        inorder(treeNode->getRight());  
        cout << ")";  
    }  
}
```

This is the same inorder routine used by us earlier. It takes the root of the tree to be traversed. First of all, we check that the root node is not null. In the previous routine after the check, we have a recursive call to inorder passing it the left node, print the *info* and then call the inorder for the right node. Here we have included parenthesis using the *cout* statements. We print out the opening parenthesis '(' before the recursive call to inorder. After this, we close the parenthesis. Then we print the *info* of the node and again have opening parenthesis and recursive call to inorder with the right node before having closing parenthesis in the end. You must have understood that we are using the parenthesis in a special order. We want to put the opening parenthesis before the start of the expression or sub expression of the left node. Then we close the parenthesis. So inside the parenthesis, there is a sub expression.

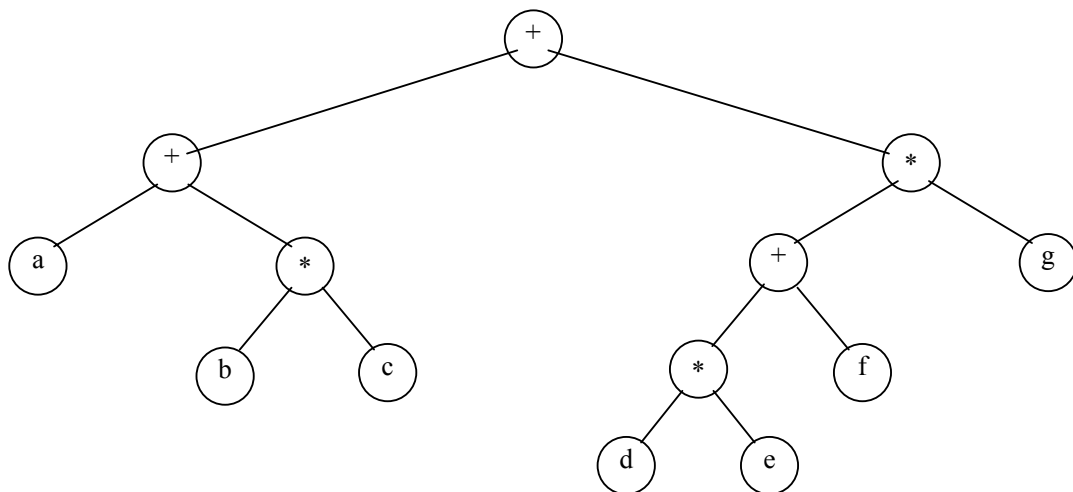
On executing this inorder routine, we have the expression as $(a + (b * c)) + (((d * e) + f) * g)$.



Inorder: $(a + (b * c)) + (((d * e) + f) * g)$

We put an opening parenthesis and start from the root and reach at the node 'a'. After reaching at plus (+), we have a recursive call for the subtree *. Before this recursive call, there is an opening parenthesis. After the call, we have a closing parenthesis. Therefore the expression becomes as $(a + (b * c))$. Similarly we recursively call the right node of the tree. Whenever, we have a recursive call, there is an opening parenthesis. When the call ends, we have a closing parenthesis. As a result, we have an expression with parenthesis, which saves a programmer from any problem of precedence now.

Here we have used the inorder traversal. If we traverse this tree using the postorder mode, then what expression we will have? As a result of postorder traversal, there will be postorder expression.



Postorder traversal: $a\ b\ c\ *\ +\ d\ e\ *\ f\ +\ g\ *\ +$

This is the same tree as seen by us earlier. Here we are performing postorder traversal. In the postorder, we print left, right and then the parent. At first, we will print a .

Instead of printing $+$, we will go for b and print it. This way, we will get the postorder traversal of this tree and the postfix expression of the left side is $a b c * +$ while on the right side, the postfix expression is $d e * f + g * +$. The complete postfix expression is $a b c * + d e * f + g * +$. The expression undergoes an alteration with the change in the traversal order. If we have some expression tree, there may be the infix, prefix and postfix expression just by traversing the same tree. Also note that in the postfix form, we do not need parenthesis.

Let's see how we can build this tree. We have some mathematical expressions while having binary operators. We want to develop an algorithm to convert postfix expression into an expression tree. This means that the expression should be in the postfix form. In the start of this course, we have seen how to convert an infix expression into postfix expression. Suppose someone is using a spreadsheet program and typed a mathematical expression in the infix form. We will first convert the infix expression into the postfix expression before building expression tree with this postfix expression.

We already have an expression to convert an infix expression into postfix. So we get the postfix expression. In the next step, we will read a symbol from the postfix expression. If the symbol is an operand, put it in a tree node and push it on the stack. In the postfix expression, we have either operators or operands. We will start reading the expression from left to right. If the symbol is operand, make a tree node and push it on the stack. How can we make a tree node? Try to memorize the *treeNode* class. We pass it some data and it returns a *treeNode* object. We insert it into the tree. A programmer can also use the *insert* routine to create a tree node and put it in the tree.

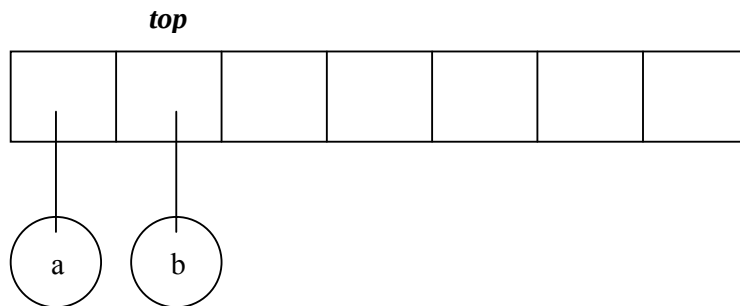
Here we will create a tree node of the operand and push it on the stack. We have been using templates in the stack examples. We have used different data types for stacks like numbers, characters etc. Now we are pushing *treeNode* on the stack. With the help of templates, any kind of data can be pushed on the stack. Here the data type of the stack will be *treeNode*. We will push and pop elements of type *treeNode* on the stack. We will use the same stack routines.

If symbol is an operator, pop two trees from the stack, form a new tree with operator as the root and $T1$ and $T2$ as left and right subtrees and push this tree on the stack. We are pushing operands on the stacks. After getting an operator, we will pop two operands from the stack. As our operators are binary, so it will be advisable to pop two operands. Now we will link these two nodes with a parent node. Thus, we have the binary operator in the parent node.

Let's see an example to understand it. We have a postfix expression as $a b + c d e + *$. If you are asked to evaluate it, it can be done with the help of old routine. Here we want to build an expression tree with this expression. Suppose that we have an empty stack. We are not concerned with the internal implementation of stack. It may be an array or link list.

First of all, we have the symbol a which is an operand. We made a tree node and push it on the stack. The next symbol is b . We made a tree node and pushed it on the stack. In the below diagram, stack is shown.

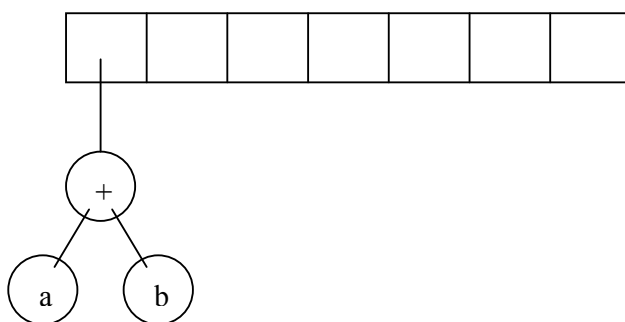
a b + c d e + * *



If symbol is an operand, put it in a one node tree and push it on a stack.

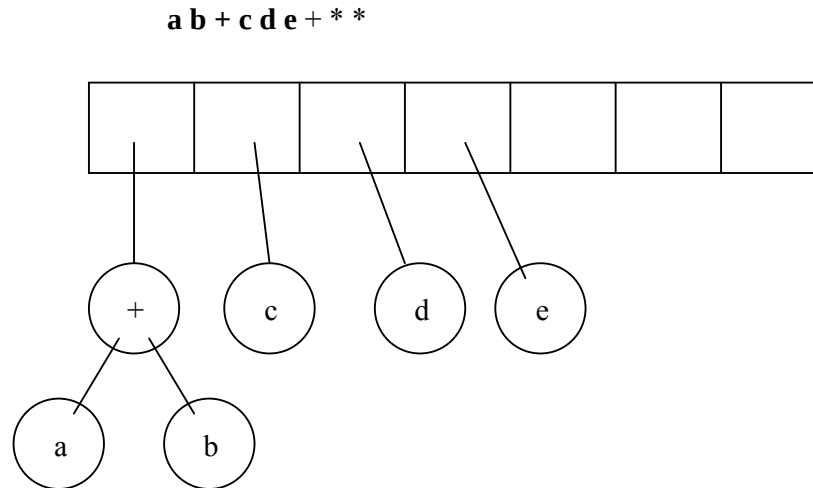
Our stack is growing from left to right. The top is moving towards right. Now we have two nodes in the stack. Go back and read the expression, the next symbol is + which is an operator. When we have an operator, then according to the algorithm, two operands are popped. Therefore we pop two operands from the stack i.e. a and b . We made a tree node of +. Please note that we are making tree nodes of operands as well as operators. We made the + node parent of the nodes a and b . The left link of the node + is pointing to a while right link is pointing to b . We push the + node in the stack.

a b + c d e + * *

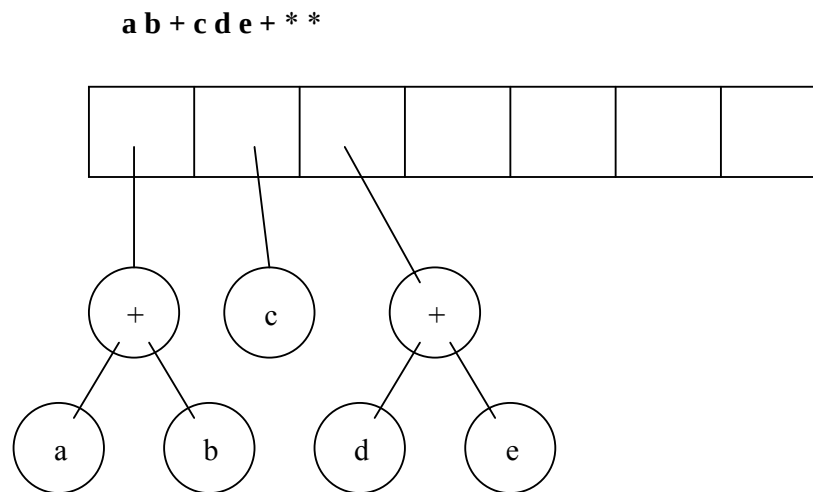


If symbol is an operator, pop two trees from the stack, form a new tree with operator as the root and T_1 and T_2 as left and right subtrees and push this tree on the stack.

Actually, we push this subtree in the stack. Next three symbols are c , d , and e . We made three nodes of these and push these on the stack.

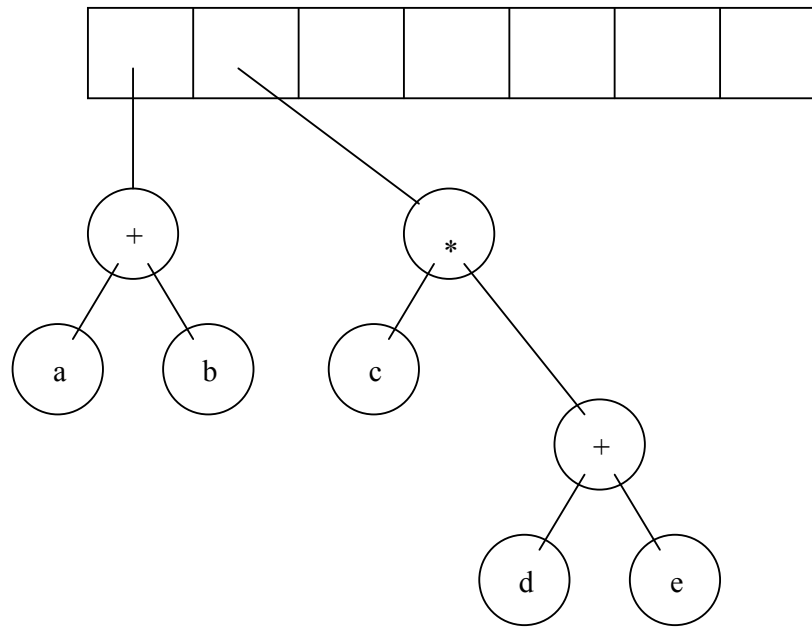


Next we have an operator symbol as $+$. We popped two elements i.e. d and e and linked the $+$ node with d and e before pushing it on the stack. Now we have three nodes in the stack, first $+$ node under which there are a and b . The second node is c while the third node is again $+$ node with d and e as left and right nodes.



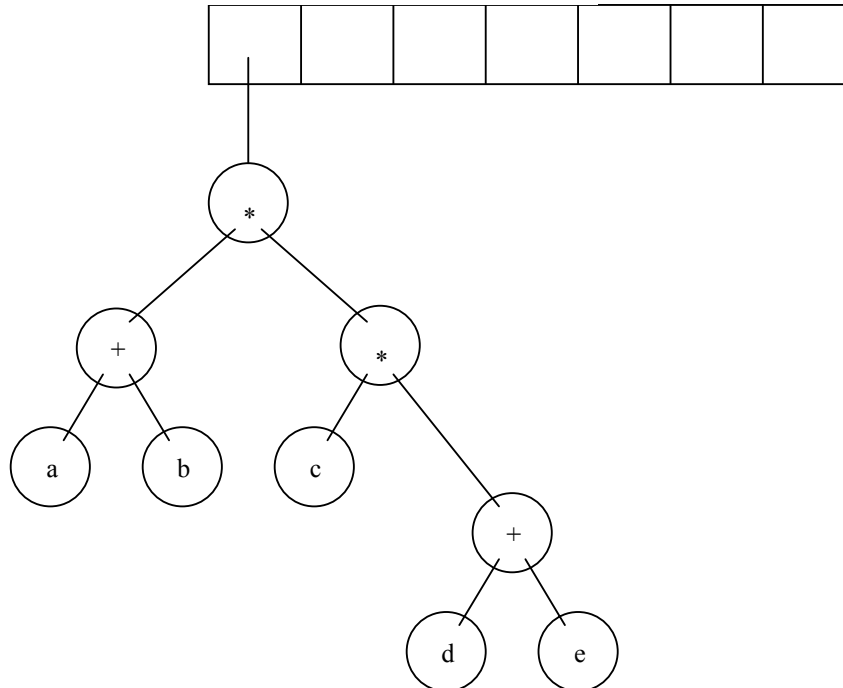
The next symbol is $*$ which is multiplication operator. We popped two nodes i.e. a subtree of $+$ node having d and e as child nodes and the c node. We made a node of $*$ and linked it with the two popped nodes. The node c is on the left side of the $*$ node and the node $+$ with subtree is on the right side.

a b + c d e + * *



The last symbol is the * which is an operator. The final shape of the stack is as under:

a b + c d e + * *



In the above figure, there is a complete expression tree. Now try to traverse this tree in the inorder. We will get the infix form which is $a + b * c * d + e$. We don't have parenthesis here but can put these as discussed earlier.

This is the way to build an expression tree. We have used the algorithm to convert the infix form into postfix form. We have also used stack data structure. With the help of templates, we can insert any type of data in the stack. We have used the expression tree algorithm and very easily built the expression tree.

In the computer science, trees like structures are used very often especially in compilers and processing of different languages.

Huffman Encoding

There are other uses of binary trees. One of these is in the compression of data. This is known as Huffman Encoding. Data compression plays a significant role in computer networks. To transmit data to its destination faster, it is necessary to either increase the data rate of the transmission media or simply send less data. The data compression is used in computer networks. To make the computer networks faster, we have two options i.e. one is to somehow increase the data rate of transmission or somehow send the less data. But it does not mean that less information should be sent or transmitted. Information must be complete at any cost.

Suppose you want to send some data to some other computer. We usually compress the file (using winzip) before sending. The receiver of the file decompresses the data before making its use. The other way is to increase the bandwidth. We may want to use the fiber cables or replace the slow modem with a faster one to increase the network speed. This way, we try to change the media of transmission to make the network faster. Now changing the media is the field of electrical or communication engineers. Nowadays, fiber optics is used to increase the transmission rate of data.

With the help of compression utilities, we can compress up to 70% of the data. How can we compress our file without losing the information? Suppose our file is of size 1 Mb and after compression the size will be just 300Kb. If it takes ten minutes to transmit the 1 Mb data, the compressed file will take 3 minutes for its transmission. You have also used the gif images, jpeg images and mpeg movie files. All of these standards are used to compress the data to reduce their transmission time. Compression methods are used for text, images, voice and other types of data.

We will not cover all of these compression algorithms here. You will study about algorithms and compression in the course of algorithm. Here, we will discuss a special compression algorithm that uses binary tree for this purpose. This is very simple and efficient method. This is also used in jpg standard. We use modems to connect to the internet. The modems perform the live data compression. When data comes to the modem, it compresses it and sends to other modem. At different points, compression is automatically performed. Let's discuss Huffman Encoding algorithm.

Huffman code is method for the compression of standard text documents. It makes use of a binary tree to develop codes of varying lengths for the letters used in the original message. Huffman code is also part of the JPEG image compression scheme. The algorithm was introduced by David Huffman in 1952 as part of a course assignment at MIT.

Now we will see how Huffman Encoding make use of the binary tree. We will take a

simple example to understand it. The example is to encode the 33-character phrase:

"traversing threaded binary trees"

In the phrase we have four words including spaces. There is a new line character in the end. The total characters in the phrase are 33 including the space. We have to send this phrase to some other computer. You know that binary codes are used for alphabets and other language characters. For English alphabets, we use ASCII codes. It normally consists of eight bits. We can represent lower case alphabets, upper case alphabets, 0,1,...9 and special symbols like \$, ! etc while using ASCII codes. Internally, it consists of eight bits. If you have eight bits, how many different patterns you can have? You can have 256 different patterns. In English, you have 26 lower case and 26 upper case alphabets. If you have seen the ASCII table, there are some printable characters and some unprintable characters in it. There are some graphic characters also in the ASCII table.

In our example, we have 33 characters. Of these, 29 characters are alphabets, 3 spaces and one new line character. The ASCII code for space is 32 and ASCII code for new line character is 10. The ASCII value for 'a' is 95 and the value of A is 65. How many bits we need to send 33 characters? As every character is of 8 bits, therefore for 33 characters, we need $33 * 8 = 264$. But the use of Huffman algorithm can help send the same message with only 116 bits. So we can save around 40% using the Huffman algorithm.

Let's discuss how the Huffman algorithm works. The algorithm is as:

- List all the letters used, including the "space" character, along with the frequency with which they occur in the message.
- Consider each of these (character, frequency) pairs to be nodes; they are actually leaf nodes, as we will see.
- Pick the two nodes with the lowest frequency. If there is a tie, pick randomly amongst those with equal frequencies.
- Make a new node out of these two, and turn two nodes into its children.
- This new node is assigned the sum of the frequencies of its children.
- Continue the process of combining the two nodes of lowest frequency till the time only one node, the root is left.

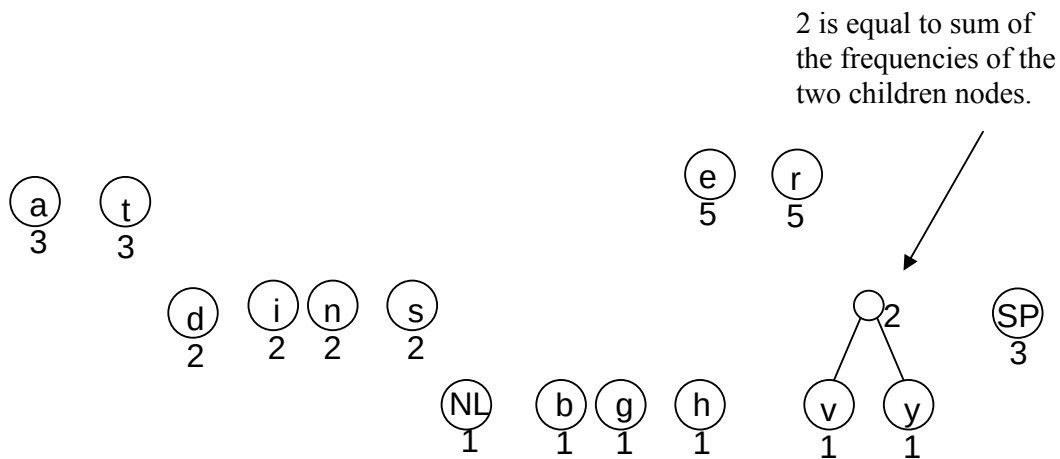
Let's apply this algorithm on our sample phrase. In the table below, we have all the characters used in the phrase and their frequencies.

Original text:		
<i>traversing threaded binary trees</i>		
size: 33 characters (space and newline)		
Letters	:	Frequency
NL	:	1
SP	:	3
a	:	3

b	:	1
d	:	2
e	:	5
g	:	1
h	:	1

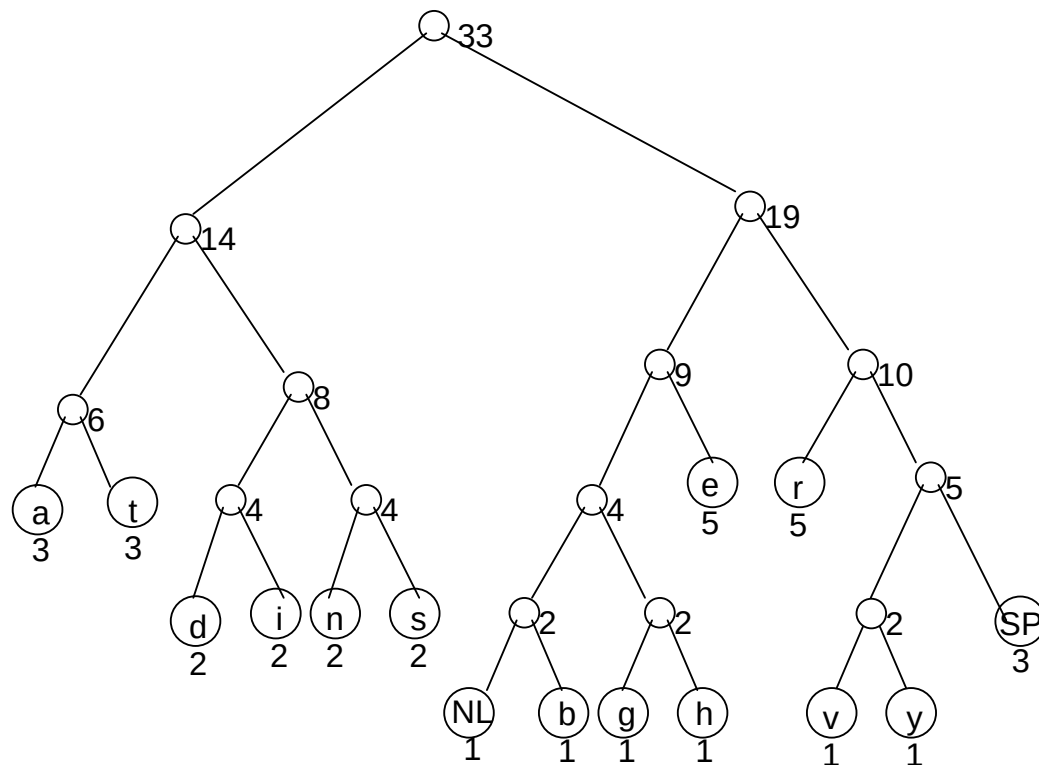
The new line occurs only once and is represented in the above table as NL. Then white space occurs three times. We counted the alphabets that occur in different words. The letter *a* occurs three times, letter *b* occurs just once and so on.

Now we will make tree with these alphabets and their frequencies.



We have created nodes of all the characters and written their frequencies along with the nodes. The letters with less frequency are written little below. We are doing this for the sake of understandings and need not to do this in the programming. Now we will make binary tree with these nodes. We have combined letter *v* and letter *y* nodes with a parent node. The frequency of the parent node is 2. This frequency is calculated by the addition of the frequencies of both the children.

In the next step, we will combine the nodes *g* and *h*. Then the nodes *NL* and *b* are combined. Then the nodes *d* and *i* are combined and the frequency of their parent is the combined frequency of *d* and *i* i.e. 4. Later, *n* and *s* are combined with a parent node. Then we combine nodes *a* and *t* and the frequency of their parent is 6. We continue with the combining of the subtree with nodes *v* and *y* to the node *SP*. This way, different nodes are combined together and their combined frequency becomes the frequency of their parent. With the help of this, subtrees are getting connected to each other. Finally, we get a tree with a root node of frequency 33.



We get a binary tree with character nodes. There are different numbers with these nodes that represent the frequency of the character. So far, we have learnt how to make a tree with the letters and their frequency. In the next lecture, we will discuss the ways to compress the data with the help of Huffman Encoding algorithm.

Data Structures

Lecture No. 26

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 4

4.4.2

Summary

- Huffman Encoding
- Mathematical Properties of Binary Trees

Huffman Encoding

We will continue our discussion on the Huffman encoding in this lecture. In the previous lecture, we talked about the situation where the data structure binary tree was built. Huffman encoding is used in data compression. Compression technique is employed while transferring the data. Suppose there is a word-document (text file) that we want to send on the network. If the file is, say, of one MB, there will be a lot of time required to send this file. However, in case of reduction of size by half through compression, the network transmission time also get halved. After this example, it will be quite easy to understand the Huffman encoding to compress a text file.

We know that Huffman code is a method for the compression of standard text documents. It makes use of a binary tree to develop codes of varying lengths for the letters used in the original message. Huffman code is also a part of the JPEG image compression scheme. David Huffman introduced this algorithm in the year 1952 as part of a course assignment at MIT.

In the previous lecture, we had started discussing a simple example to understand Huffman encoding. In that example, we were encoding the 32-character phrase: "*traversing threaded binary trees*". If this phrase were sent as a message in a network using standard 8-bit ASCII codes, we would have to send $8 \times 32 = 256$ bits. However, the Huffman algorithm can help cut down the size of the message to 116 bits.

In the Huffman encoding, following steps are involved:

1. List all the letters used, including the "space" character, along with the frequency with which they occur in the message.
2. Consider each of these (character, frequency) pairs as nodes; these are actually leaf nodes, as we will see later.
3. Pick two nodes with the lowest frequency. If there is a tie, pick randomly amongst those with equal frequencies
4. Make a new node out of these two and develop two nodes as its children.
5. This new node is assigned the sum of the frequencies of its children.
6. Continue the process of combining the two nodes of lowest frequency till the time, only one node, the root, is left.

In the first step, we make a list of all letters (characters) including space and end line character and find out the number of occurrences of each letter/character. For example we ascertain how many times the letter 'a' is found in the file and how many times 'b' occurs and so on. Thus we find the number of occurrences (i.e. frequency) of each letter in the text file.

In the step 2, we consider the pair (i.e. letter and its frequency) as a node. We will consider these as leaf nodes. Afterwards, we pick two nodes with the lowest frequency in the list. If there are more than one pairs of same frequency, we will choose a pair randomly amongst those with equal frequencies.

Suppose, in a file, the letter 'a' occurs 50 times and 'b' and 'c' five times each. Here, 'b' and 'c' have the lowest frequency. We will take these two letters as leaf nodes and build the tree from these ones. As fourth step states, we make a new node as the parent of these two nodes. The 'b' and 'c' are its children. In the fifth step, we assign to this new node the frequency equal to the sum of the frequencies of its children. Thus a three-node tree comes into existence. This is shown in the following figure.

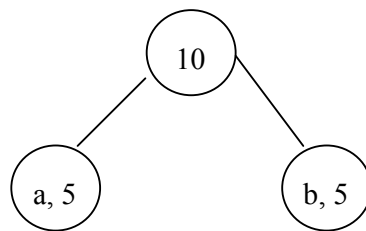


Fig 26.1:

We continue this process of combining the two nodes of lowest frequency till the time, only one node i.e. the root is left.

Now we come back to our example. In this example, there is a text string as written below.

traversing threaded binary trees

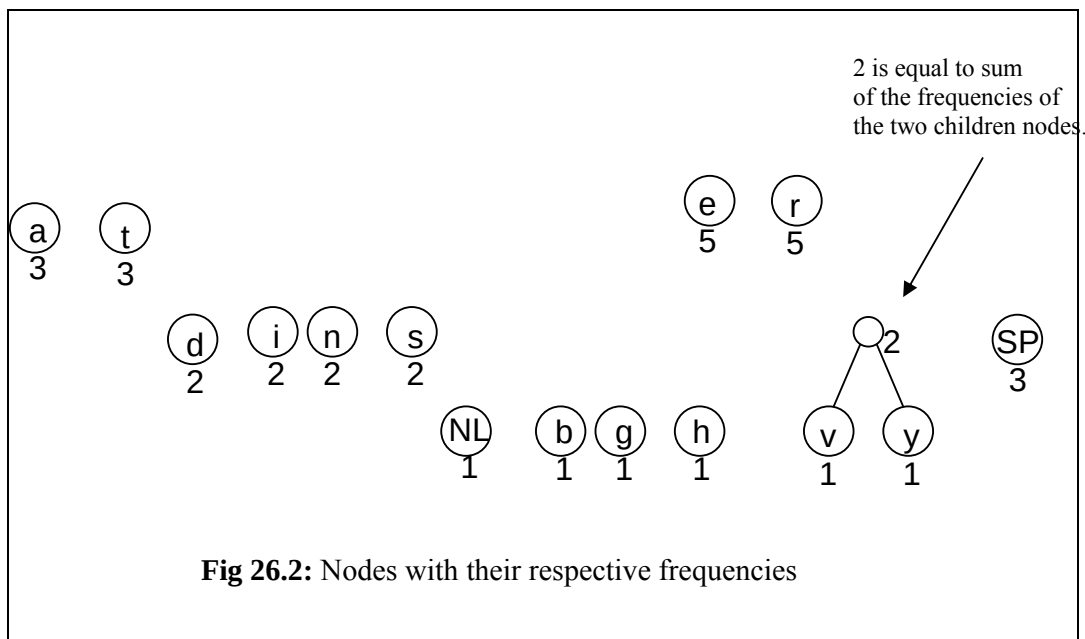
The size of this character string is 33 (it includes 3 space characters and one new line character). In the first step, we perform the counting of different characters in the string manually. We do not assign a fake or zero frequency to a letter that is not present in the string. A programmer may be concerned only with the characters/letters that are present in the text. We see that the letters and their frequencies in the above text is as given below.

Character	frequency	character	frequency
NL	1	I	2
SP	3	n	2
A	3	r	5
B	1	s	2
D	2	t	3
E	5	v	3
G	1	y	1
H	1		

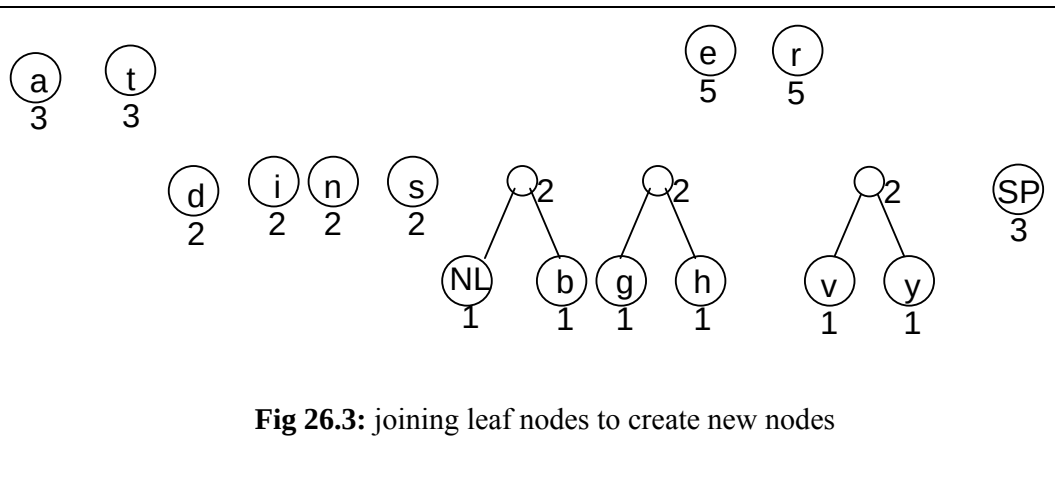
Table 1: Frequency table

In the second step, we make nodes of these pairs of letters and frequencies. The following figure (fig 26.2) depicts the letters as nodes. We have written the frequency of each letter with the node. The nodes have been categorized with respect to the frequencies for simplicity. We are going to build the tree from downside i.e. from the lowest frequency.

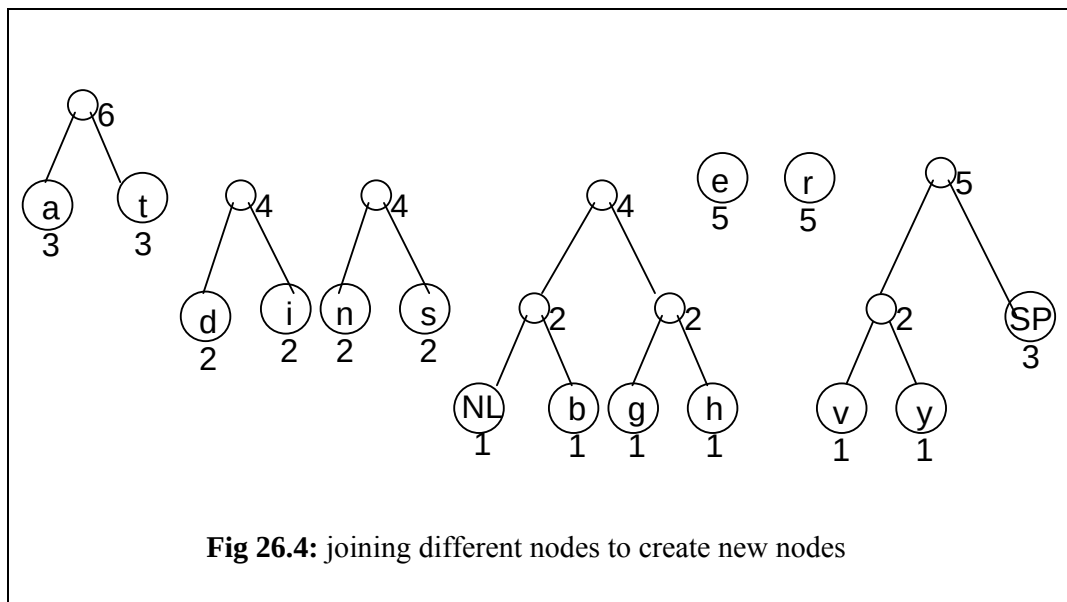
Now according to third step, two nodes of lowest frequency are picked up. We see that nodes NL, b, g, h, v and y have the frequency 1. We randomly choose the nodes v and y. now, as the fourth step, we make a new node and join the leaf nodes v and y to it as its children. We assign the frequency to this new (parent) node equal to the sum of the frequencies of its children i.e. v and y. Thus in the fifth step; the frequency of this new node is 2. We have written no letter in this node as shown in the figure below.



Now we continue this process with other nodes. Now we join the nodes g and h as children of a new node. The frequency of this node is 2 i.e. the sum of frequencies of g and h. After this, we join the nodes NL and b. This also makes a new node of frequency 2. Thus the nodes having frequency 1 have joined to the respective parent nodes. This process is shown in the following figure (Fig 26.3).

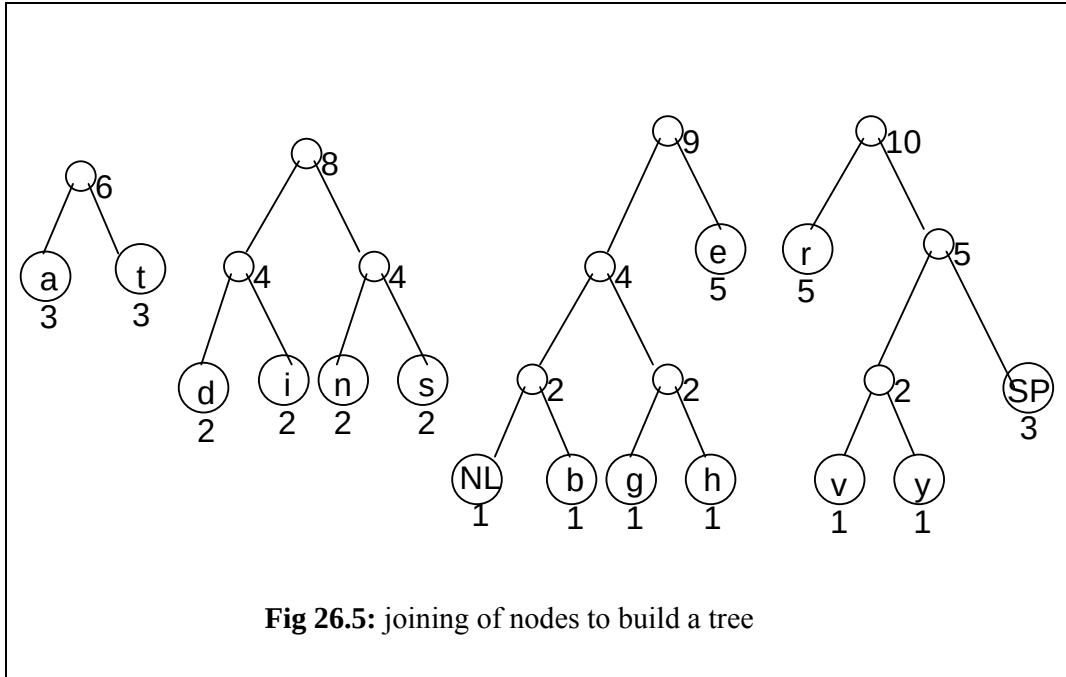


Now we come to the nodes with a frequency 2. Here we join the pair of nodes d and i and also the pair n and s. Resultantly, the two nodes coming into being after joining have the frequency 4. This frequency is the sum of the frequencies of their children. Now, we will bring the nodes, a and t together. The parent node of a and t has the frequency 6 i.e. the sum of a and t. These new nodes are shown in the following figure.

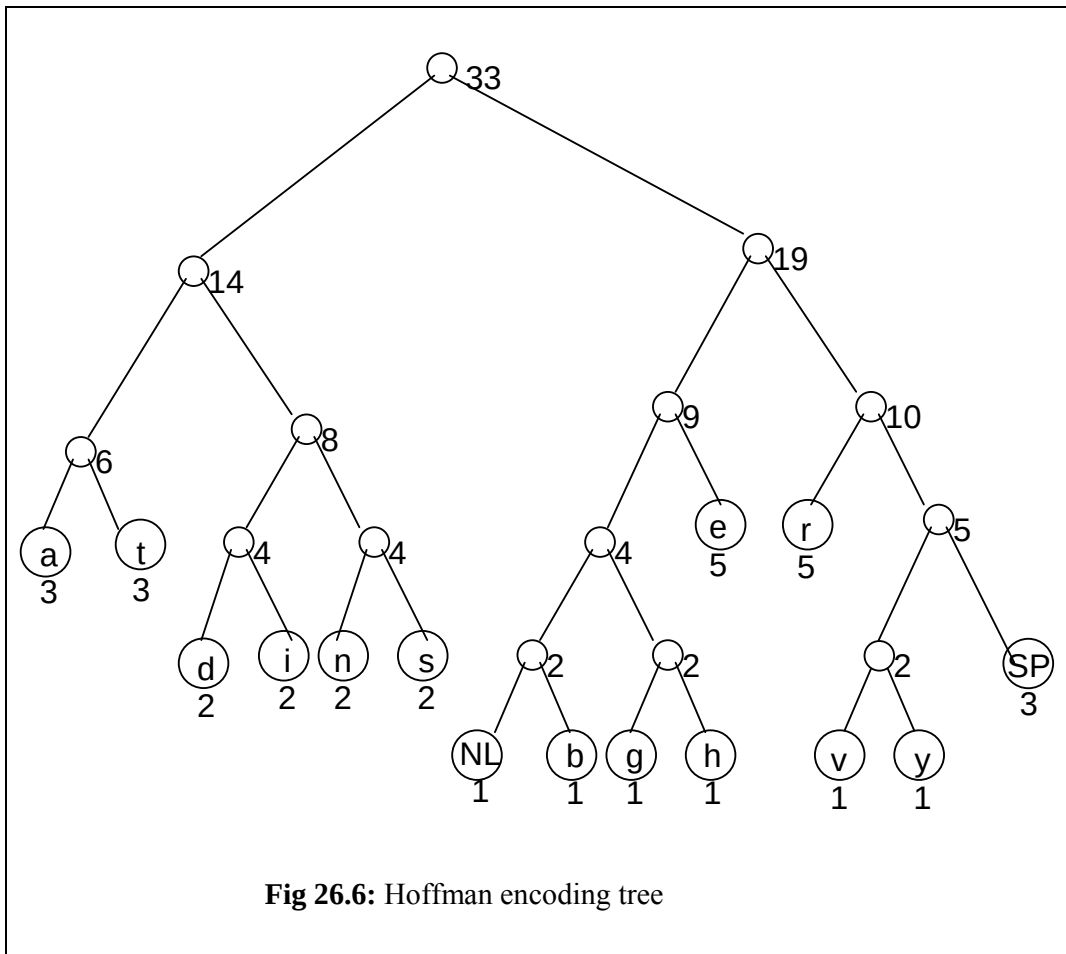


Now we consider these new nodes as inner nodes and build the tree upward towards the root. Now we take the node SP and join it with the node that is the parent of v and y. The resultant node has frequency 5 as the frequencies of its children are 2 and 3 respectively. Now we join these nodes of higher frequencies. In other words, the node r is joined with the newly created node of frequency 5 that is on the left side of node r in the figure 26.5. Thus a new node of frequency 10 is created. We join the node e and the node of frequency 4 on its right side. Resultantly, a new node of frequency 9 comes into existence. Then we join the nodes having frequency 4 and create a new

node of frequency 8. The following figure shows the nodes created so far.



Now we will join the nodes of frequency 6 and 8 to create the node of frequency 14 and join the nodes of frequency of 9 and 10 to develop a new node of frequency of 19. At the end, we make the root node with the frequency 33 and it comprises nodes of frequency 14 and 19. Thus the tree is completed and shown in the following figure.

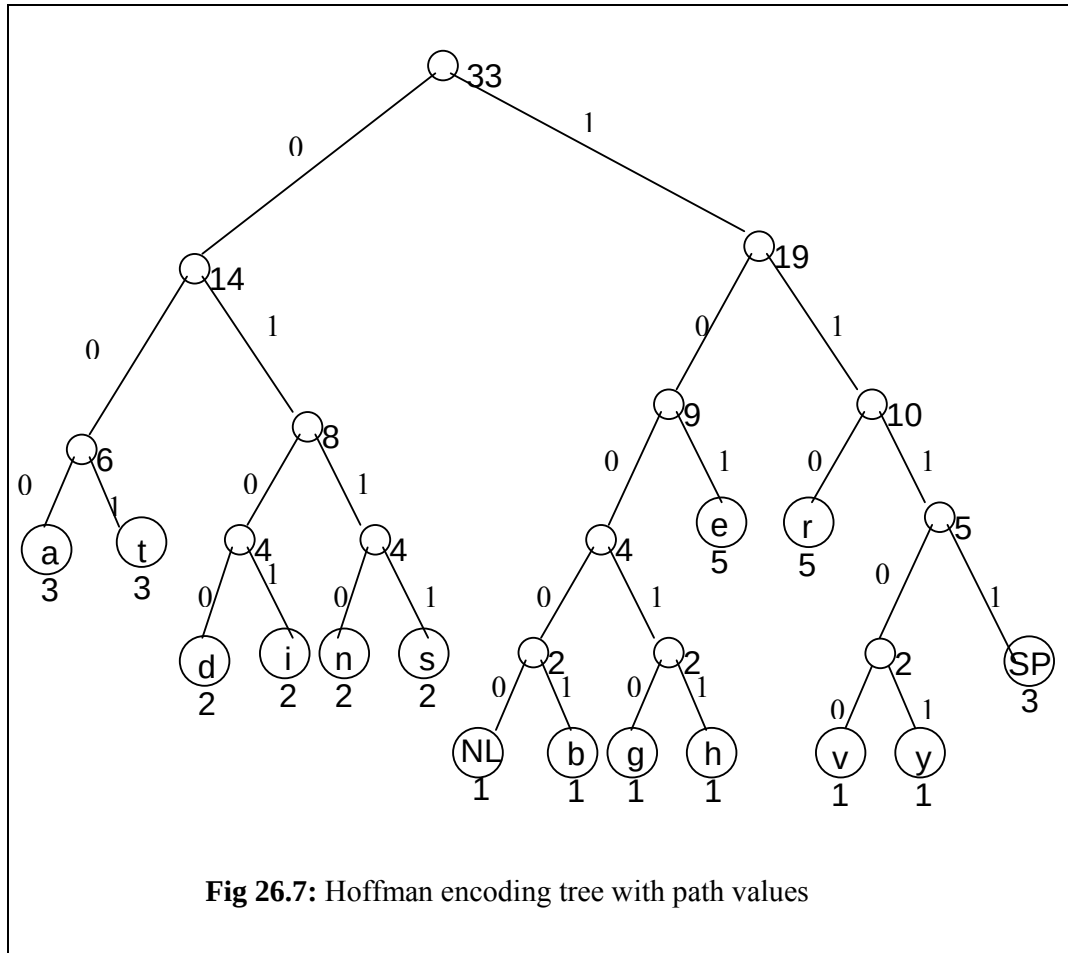


Now we will perform other steps of Huffman encoding and develop character-encoding scheme needed for data compression.

To go ahead, we have to do the following steps.

- Start at the root. Assign 0 to left branch and 1 to the right branch.
- Repeat the process down the left and right subtrees.
- To get the code for a character, traverse the tree from the root to the character leaf node and read off the 0 and 1 along the path.

We start from the root node of the tree and assign the value 0 to the left branch and 1 to the right branch. Afterwards, we repeat this value assigning at nodes in left and right subtrees. Thus all the paths have a value 0 or 1 as shown in the following figure. We will develop the code with the help of these path values.



In the last step, we get the code for the characters. To get the code for a character, there is need of traversing the tree from the root to the character leaf node and read off the 0 and 1 along the path. We start from the root and go to the letter of leaf node following the edges. 0 and 1 are written down in the order in which they come in this traversal to leaf node. For example, we want the code of letter d. To reach d from the root; we have to go through the nodes of frequency 14. This path has value 0. Here, 0 will be the first digit in the code of d. From 14, we go to node of frequency 8. This link (from 14 to 8) has value 1. Thus the second digit in code is 1. From node of frequency 8, we go to node of frequency 4 to its left side. This side has value 0, meaning that 0 is the third digit of code. From 4, we finally go to d and in this link we get the value 0. Thus we see that to reach at d from the root, we have gone through the branches 0, 1, 0 and 0. Thus, the code of letter d is 0100. Similarly the code of i is 0101. The same way, we find the code for each letter in the tree. The following table shows the letters and their correspondent codes.

character	code	character	code
NL	10000	i	0101
SP	1111	n	0110
a	000	r	110

b	10001	s	0111
d	0100	t	001
e	101	v	11100
g	10010	y	11101
h	10011		

Table 2: Hoffman code table

We know that every character is stored in the computer in binary format. Each character has a code, called ASCII code. The ASCII code of a character consists of ones and zeros i.e. it is in binary form. ASCII code is of eight bits. Normally, we remember the decimal value of the characters. For example, letter 'A' has decimal value 65. We can easily convert the decimal value into binary one in bit pattern. We need not to remember these values. ASCII value of any character can be found from the ASCII table. The ASCII code of each character is represented in eight bits with different bit patterns.

Here in the example, we assign a code of our own (i.e. Hoffman code) to the letters that are in the message text. This code also consists of ones and zeros. The above table shows the characters and their Hoffman code. Now we come back to the message text from which we developed the tree and assigned codes to the letters in the text.

Look at the table (Table 2) shown above. Here we notice that the code of letters is of variable length. We see that letters with higher frequency have shorter code. There are some codes with a length five that are the codes of NL, b, g, h, v and y. Similarly we see that the letters SP, d, i, n and s have codes of length four. The codes of the remaining letters have length three. If we look at the frequency table (Table 1) of these letters, we see that the letters with some higher frequency like a, e, r, and t, have the code of shorter length, whereas the letters of lower frequency (like NL, b, g, h, v and y) have codes of larger length.

We see in the table of the Hoffman codes of the letters that there will be need of 5, 4 or in some codes only 3 bits to represent a character, whereas in ASCII code, we need 8 bits for each character. Now we replace the letters in the text message with these codes. In the code format i.e. in the form of ones and zeros, the message becomes as under.

Our original message was

traversing threaded binary trees

The encoded form of the message is as under

t	r	a	v	e	r	s	i	n	g	t
001	110	000	11100	101	1100	111	0101	0110	10010	1111001
1001	11110	1010	10000	10010	10100	11111	1000	10101	110000	
1101	11011	111100	11101	10110	10111	10000				

We see that there are only ones and zeros in the code of message. Some letters have been shown with their corresponding code. The first three bits 001 are for letter 't'. The next three bits 110 are for letter 'r'. After this the letter 'a' has three bits i.e. 000.

Next to it is the letter 'v' that has five bits. These bits are 11100 (shaded in the figure). Similarly we have replaced all the letters of the message with their corresponding Huffman code. The encoded message is shown above.

Let's compare this Huffman encoded message with the encoding of message with ASCII code. The ASCII code encoding means that if we have encoded this message with 8 bits per character, the total length of message would be 264. As there are 33 characters, so the total length is $33 \times 8 = 264$. But in the Huffman encoded message (written in above table), there are only 120 bits. This number of bits is 54% less than the number of bits in ASCII encoding form. Thus we can send the message with almost half of the original length to a receiver.

Here the Huffman encoding process comes to an end. In this process, we took a message, did the frequency count of its characters and built a tree from these frequencies. From this tree, we made codes of letters consisting of ones and zeros only. Then with the help of these codes, we did the data compression. In this process we saw that less data is required for the same message.

Now an important thing to be noted is regarding the tree building. We have built the tree with our choices of nodes with same and different frequencies. Now if we have chosen the nodes to join in different way, the tree built would be in a different form. This results in the different Huffman code for the letters. These will be in ones and zeros but with different pattern. Thus the encoded message would have different codes for the letters. Now the question arises how does the receiver come to know that what code is used for what letter? The answer is very simple that the sender has to tell the receiver that he is using such codes for letters. One way to tackle this problem is that the sender sends the tree built through the use of frequencies, to the receiver to decode the message. The receiver keeps a copy of this tree. Afterwards, when the sender sends a message, the receiver will match it with respect to bit pattern with the tree to see that what letter the sender has sent. The receiver will find the letter for the first 2, 3, 4 or what numbers of bits match to a letter that it has in the tree as the receiver also has a copy of the tree. We know that a tree has a root, inner nodes and leaf node(s). The leaf node is a node whose left and right links is NULL. An inner node has a left or right or both children. Now consider that the receiver has the same tree data structure that the sender used to construct the codes. The sender has sent the message that we have discussed above. The sender sends the 122 bits encoded message to the receiver. The receiver will take the first bit of message and being on the root of the tree, it will decide that on what side this bit will go. If the bit is zero, it will go to the left of the root. However, if the bit is one, it will go to the right side of the root before reaching to the child node. The next bit will go to the next level of the tree to the left or right side depending on zero or one. Thus the traversal will go one level down on receiving a bit. If we (the receiver) are on a path of the tree where the leaf node is at level 6, it cannot reach the leaf node unless the receiver receives 6 bits. We consider the case of letter 'e' whose code was of three bits. In this case, we go to the first level by first bit and the second bit takes us to the third level. Finally on reaching the third level with the third bit, we get the leaf node. We know that this node is letter 'e' as the sender has sent us (as receiver) the whole tree with the characters in the nodes. Thus after traversing the three bits, the receiver has confirmed that it has received the letter 'e'. Now on receiving the fourth bit, the receiver will go back to the root and continue to choose the path i.e. on zero it will go to the left and on one it will go to right. This way, it will reach the leaf node. The character at that node will be the next character of the message. The bit pattern that was comprised of

following these links in the tree is extracted from the message. On the next bit, the receiver again goes to the root node and the previous procedure is repeated to find the next character. This way, it decodes the whole message.

The compression is very useful technique especially for communication purposes. Suppose that we have to send a file of one Mb. Here each line in the file can be compressed to 50 or 60 percent. Thus the file of one MB will be compressed to half MB and can be sent more easily.

There is one more thing about this tree. When we started to build the tree from leaf nodes i.e. bottom-up build of tree, we saw that there were choices for us to choose any two leaf nodes to join. In our example, we chose them at random. The other way to do it is the priority queue. We have seen the example of bank simulation. There we used a priority queue for the events. We know that in a priority queue, the elements do not follow the FIFO (first in first out) rule. But the elements have their position in the queue with respect to a priority. In the example of bank simulation, in the Event Queue, we remove the element from the queue that was going to occur first in the future. We can use priority queue here in such a way that we put the letters in the queue with respect to their frequencies. We will put and remove letters from the queue with respect to their frequency. In this priority queue, the character with lowest frequency is at the start of the queue. If two characters have the same frequency, these will be one after the other in the queue. The character with the larger frequency will be in the last of the queue. Now we take two frequencies from the queue and join them to make a new node. Suppose that the nodes that we joined have frequency 1 and 2 respectively. So the frequency of the new node will be 3. We put this new node in the queue. It takes its position in the queue with respect to its frequency as we are using the priority queue. It is evident in procedure that we proceed to take two nodes from the queue. These are removed and their parent node goes back to the queue with a new frequency. This procedure goes on till the time the queue is empty. The last node that becomes in the queue in the result of this procedure is the root node. This root node has frequency 33 if we apply this procedure to our previous example.

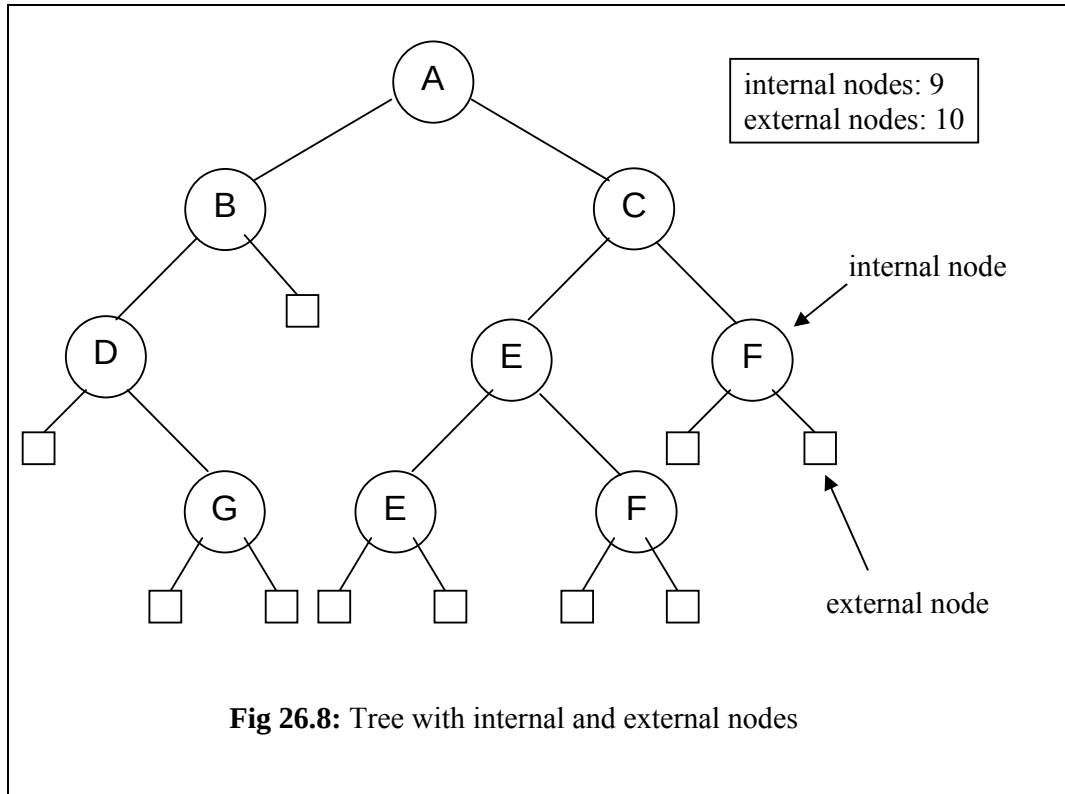
Let's talk about some general things related to Hoffman encoding. We use modems to connect to Internet. These modems do the compression. The modem has a compression feature. The modem has a chip that performs the task of compression. When we give a sentence of say 80 characters to the modem to send, the modem makes a Hoffman encoded tree of these characters. Then it will make codes from this tree. We know that there is also a modem on the other end that will decode this message. The sender modem will send the tree structure to the receiver. Now the sender modem will send the data in compressed form. The receiving modem will decode this compressed data by using the Hoffman encoded tree. Now the question arises, will these codes be useful for other messages? In our example message the letter 'y' has lower frequency and code of five bits. It may happen that in some messages 'y' has higher frequency, needing code of less number of bits. To solve this problem, the modems (sender and receiver) revise the codes after a certain time period or after a particular number of bytes. In this revision, they build a new Hoffman tree and exchange it to use it for communication for the next time period or for the next fixed number of bytes.

There is some other compression techniques/algorithms for compression. The zip routines like winzip and the jpeg, mpeg and other image formatting routines use different algorithms for compression. We will read the compression algorithm in detail in the course of Algorithms.

Mathematical Properties of Binary Trees

There are some mathematical properties of binary trees, which are actually theorems. We will not prove these here. Most of these properties will be studied in some other courses where we will prove them as theorem. Here we are going to talk about some properties, much needed in the next topic about trees.

The first property is that a binary tree of N internal nodes has $N+1$ external nodes. We are familiar with the term binary tree and internal node. The term external node is a new one. To understand the external nodes, look at the following figure.



In this figure, the nodes with value i.e. A, B, C, D, E, F and G are the internal nodes. Note that the leaf nodes are also included in internal nodes. In the figure, we see that the right pointer of B is NULL. Similarly, the left pointer of D is NULL. The square nodes in the figure are the NULL nodes. There is no data in these nodes as these are NULL pointers. However these are the positions where the nodes can exist. These square nodes are the external nodes. Now we see in the figure that the internal nodes (leaf nodes are also included) are 9 and the external nodes (NULL nodes indicated by squares) are 10 (i.e. $9 + 1$). Hence it is the property that we have stated. We will see the usage of this property in the upcoming lectures.

Data Structures

Lecture No. 27

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 4
4.3

Summary

- Properties of Binary Tree
- Threaded Binary Trees
- Adding Threads During Insert
- Where is Inorder Successor?
- Inorder Traversal

Properties of Binary Tree

By the end of the last lecture, we were having a discussion about the properties of the binary trees. Let us recall, that I told you about a property of the binary trees regarding relationship between internal nodes and external nodes i.e. If the number of internal nodes is N , the number of external nodes will be $N+1$. Today I am going to discuss another property of the binary trees, which together with the previous lecture, will give us a start into a new topic. Let me have your attention to the second property of the binary trees.

Property

A binary tree with N internal nodes has $2N$ links, $N-1$ links to internal nodes and $N+1$ links to external nodes.

Please recall that the first property dealt with the relationship between internal and external nodes. This property is dealing with the relationship of links to the internal nodes.

Now, what is a link? As you might already have understood, a link is that line, which we draw between two nodes in a tree. Internally we use pointers in C++ to realize links. In pictorial sketches, however, we use a line to show a link between the two nodes. The property defines, that if you have a binary tree with N nodes, how many links, it will have between the internal nodes (remember, it includes the leaf nodes), and how many links it will have between the external nodes. We have not been showing any links between the external nodes in the diagrams. These are, in fact, null pointers. That means, these are the links, which we will show with the help of the square nodes. Let us see a binary tree, on the basis of which, we will further explore this property. In the following figure, the binary tree is shown again, which, in addition to the normal links between the internal nodes, also contains external nodes as squares and the external links as lines going to those squares.

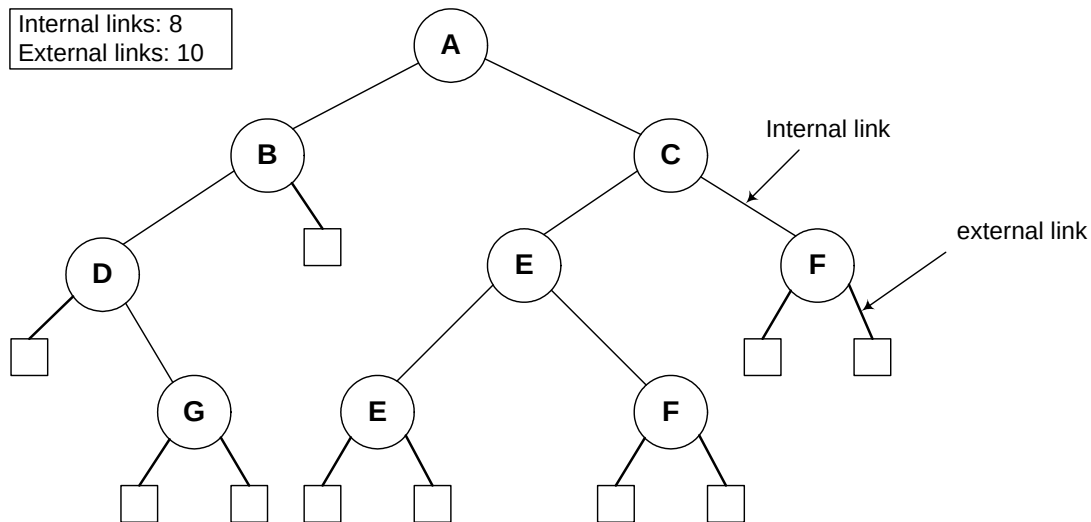


Fig 27.1

Now if you count the total number of links in the diagram between internal and external nodes, it will be $2N$. Remember, we are talking about links and not nodes. In this tree, we have 9 nodes marked with capital letters, 8 internal links and 10 external links. Adding the both kinds of links, we get 18, which is exactly 2×9 .

As discussed already that these properties are mathematical theorems and can therefore be proven mathematically. Let us now prove this property as to how do we get $2N$ links in a binary tree with N internal nodes.

Property

A binary tree with N internal nodes has $2N$ links, $N-1$ links to internal nodes and $N+1$ links to external nodes.

- In every rooted tree, each node, except the root, has a unique parent.
- Every link connects a node to its parents, so there are $N-1$ links connecting internal nodes.
- Similarly each of the $N+1$ external nodes has one link to its parents.
- Thus $N-1+N+1=2N$ links.

In the previous lectures, I told you about the important property of the trees, that they contain only one link between the two nodes. I had also shown you some structures, which did not follow this property and I told you, that those were graphs.

Threaded Binary Trees

- In many applications binary tree traversals are carried out repeatedly.
- The overhead of stack operations during recursive calls can be costly.
- The same would true if we use a non-recursive but stack-driven traversal procedure
- It would be useful to modify the tree data structure which represents the binary

tree so as to speed up, say, the inorder traversal process: make it "stack-free".

You must be remembering that there were four traversing methods of binary trees: *preorder*, *inorder*, *postorder* and *levelorder*. First three *preorder*, *inorder* and *postorder* were implemented using recursion. Those recursive routines were very small, 3 to 4 lines of code and they could be employed to traverse a tree of any size.

We also traversed BST in inorder to retrieve the information in sorted order. We employed stacks in recursive implementations. Although, recursive routines are of few lines but when recursion is in action, recursive stack is formed that contains the function calls. We also explicitly used stack for inorder non-recursive traversal. When the calling pattern of recursive and non-recursive stack based routines were compared, the calling pattern of both of the routines were similar.

Suppose that we have a BST that is traversed again and again for some operations of find or print. Due to lot of recursive operations, the stack size keeps on growing. As a result, the performance is affected. To overcome this performance bottleneck, we can use non-recursive method but stack-driven traversal will again be an issue. The *push* and *pop* operations of stack for insertion and retrieval will again take time. So is there a way to do traversal without using a stack neither of implicit function call stack nor explicit. The same idea is presented in the last bullets above that leads to threaded binary trees:

- *It would be useful to modify the tree data structure which represents the binary tree so as to speed up, say, the inorder traversal process: make it "stack-free".*

The idea in the above statement is to modify the tree data structure to speed up and make it stack-free. Now, we see what kind of modification is required in the binary trees.

- *Oddly, most of the pointer fields in our representation of binary trees are NULL!*
- *Since every node (except the root) is pointed to, there are only N-1 non-NULL pointers out of a possible 2N (for an N node tree), so that N+1 pointers are NULL.*

We know that all the leaf node pointers are NULL. Each node of the tree contains the data part, two pointer variables for left and right nodes links. But these pointer variables are used when the node has further child nodes. We know that in a binary tree the total number of links are $2N$ including both internal and external and the number of NULL pointers is $N+1$.

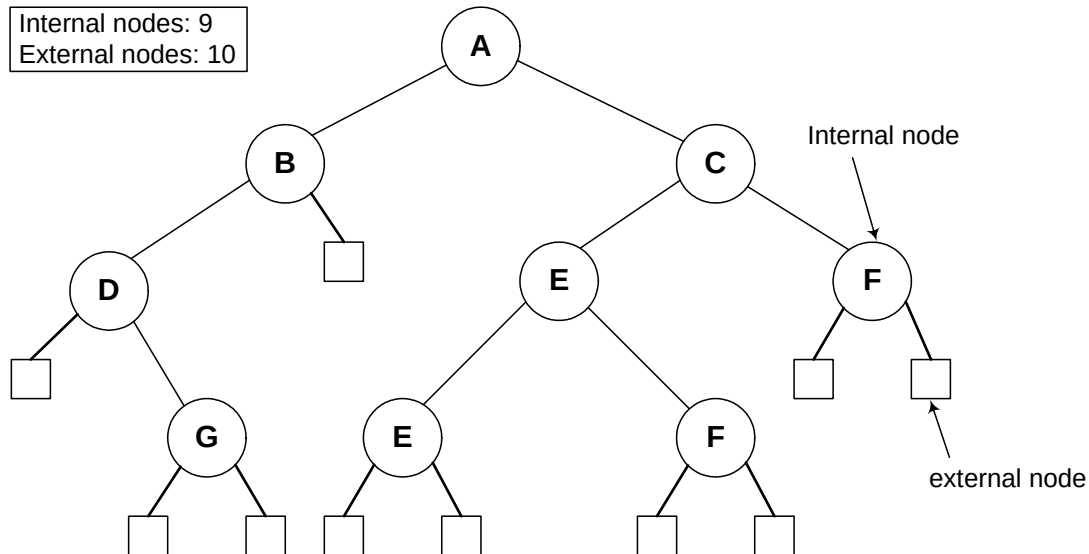


Fig 27.2

In the figure above, the tree is the same as shown in Fig 27.1. The square nodes shown in this figure are external nodes. Thinking in terms of pointers all the pointers of these nodes are NULL or in other words they are available to be used later. We recognize these nodes as *leaf* nodes. Besides that, what can we achieve using them is going to be covered in Threaded Binary Trees.

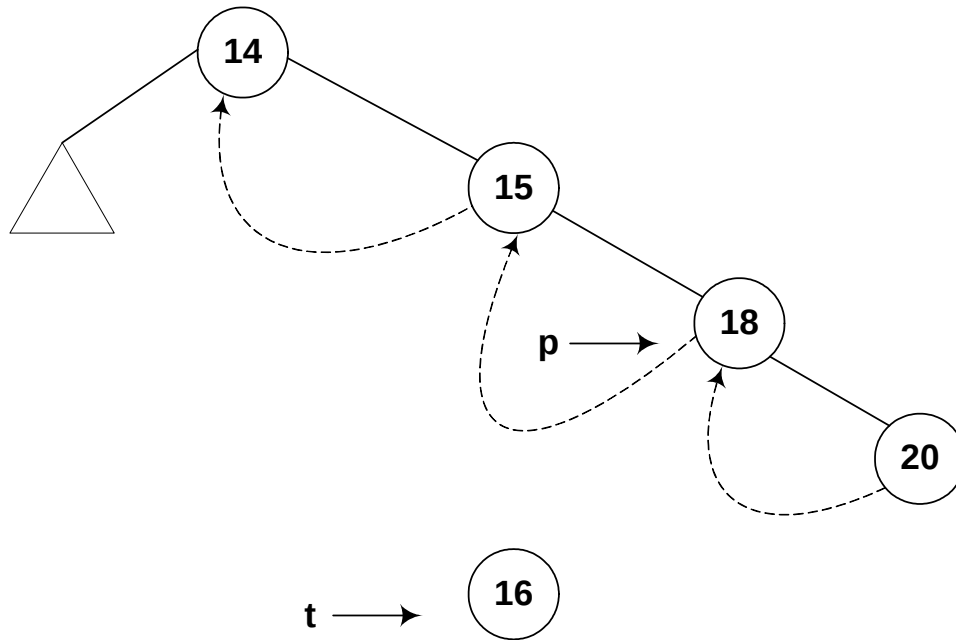
- *The threaded tree data structure will replace these NULL pointers with pointers to the inorder successor (predecessor) of a node as appropriate.*

We are creating a new data structure inside the tree and when the tree will be constructed, it will be called a threaded binary tree. The NULL pointers are replaced by the inorder successor or predecessor. That means while visiting a node, we can tell which nodes will be printed before and after that node.

- *We'll need to know whenever formerly NULL pointers have been replaced by non NULL pointers to successor/predecessor nodes, since otherwise there's no way to distinguish those pointers from the customary pointers to children.*

This is an important point as we need to modify our previous logic of identifying leaf nodes. Previously the node with left and right nodes as NULL was considered as the leaf node but after this change the leaf node will contain pointers to predecessor and successor. So in order to identify that the pointers has been modified to point to their inorder successor and predecessor, two flags will be required in the node. One flag will be used for successor and other for predecessor. If both the pointers were NULL, left pointer variable will be used to point inorder predecessor, the flag for this will be turned on and the right pointer variable will be used to keep inorder successor and the flag will be turned on once the successor address is assigned.

Adding Threads During Insert

**Fig 27.3**

If we print the above tree in inorder we will get the following output:

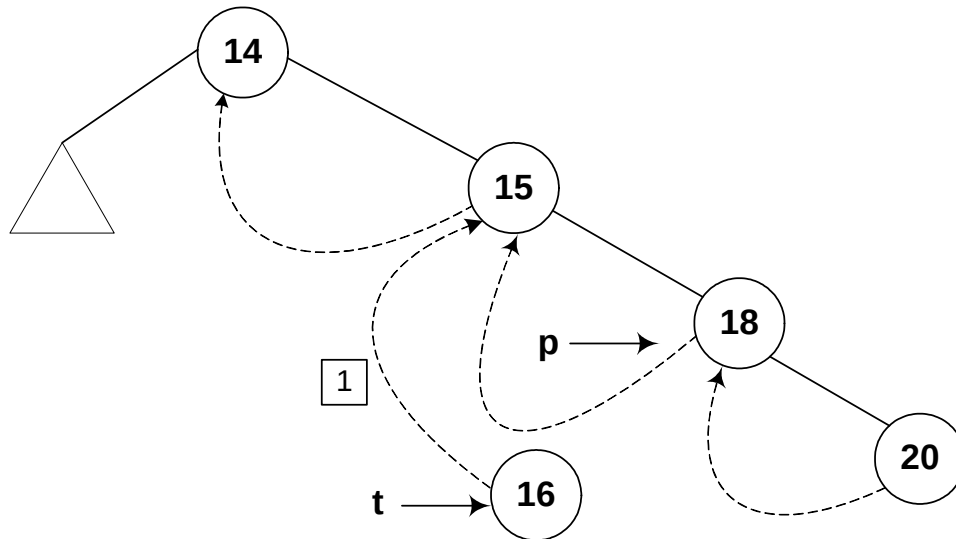
14 15 18 20

In the above figure, the node 14 contains both left and right links. The left pointer is pointing to a subtree while the right subtree is pointing to the node 15. The node 15's right link is towards 18 but the left link is NULL but we have indicated it with a rounded dotted line towards 14. This indicates that the left pointer points to the predecessor of the node.

Below is the code snippet for this logic.

```
t->L = p->L; // copy the thread
t->LTH = thread;
t->R = p; // *p is successor of *t
t->RTH = thread; p->L = t; // attach the new leaf
p->LTH = child;
```

Let's insert a new node in the tree shown in the above figure. The Fig 27.4 indicates this new insertion.

**Fig 27.4**

The new node 16 is shown in the tree. The left and right pointers of this new node are NULL. As node 16 has been created, it should be pointed to by some variable. The name of that variable is *t*. Next, we see the location in the tree where this new node with number 16 can be inserted. Clearly this will be after the node 15 but before node 18. As a first step to insert this node in the tree as the left child of the node 18, we did the following:

1. $t \rightarrow L = p \rightarrow L$; // copy the thread
2. $t \rightarrow LTH = \text{thread}$;
3. $t \rightarrow R = p$; // *p is successor of *t
4. $t \rightarrow RTH = \text{thread}$;
5. $p \rightarrow L = t$; // attach the new leaf
6. $p \rightarrow LTH = \text{child}$;

As the current predecessor of node 18 is 15. After node 16 will be inserted in the tree, it will become the inorder predecessor of 18, therefore, in the first line of the code $t \rightarrow L = p \rightarrow L$, left pointer of node 18 (pointed to by pointer *p*) is assigned to the left pointer of node 16 (pointer to by pointer *t*).

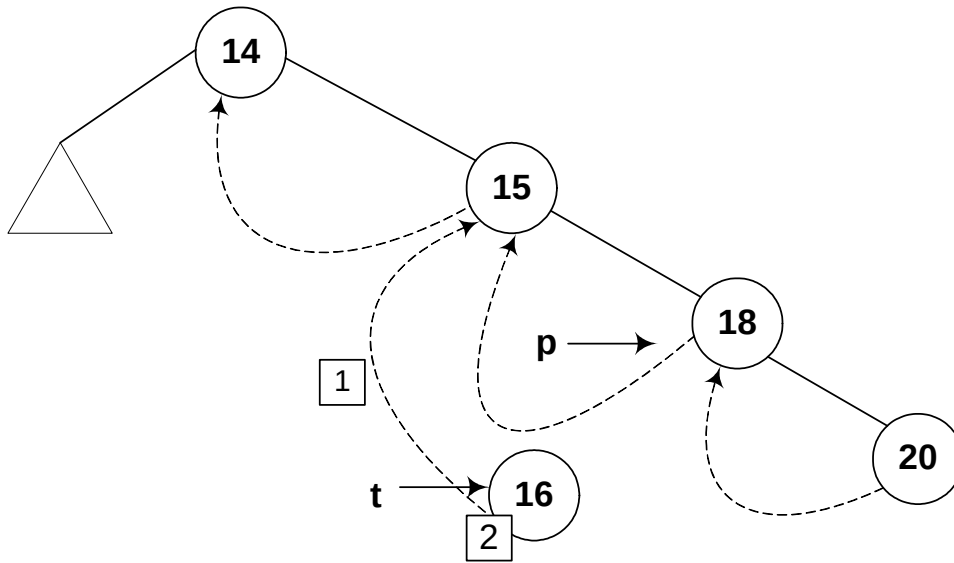


Fig 27.5

In the next line of code $t \rightarrow LTH = thread$, the left flag is assigned a variable *thread* that is used to indicate that it is on.

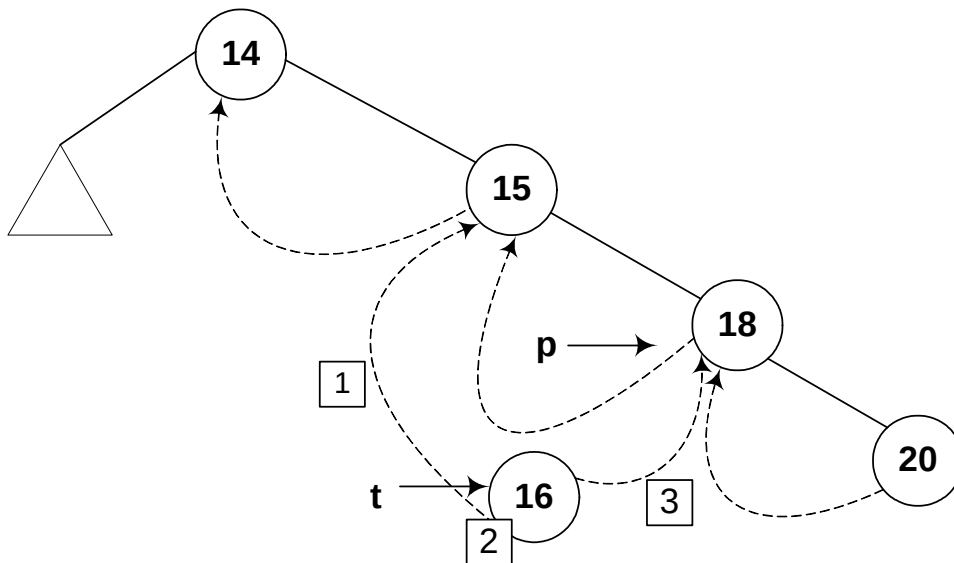


Fig 27.6

In the third line of code, $t \rightarrow R = p$, 18 being the successor of node 16, its pointer *p* is assigned to the right pointer ($t \rightarrow R$) of node 16.

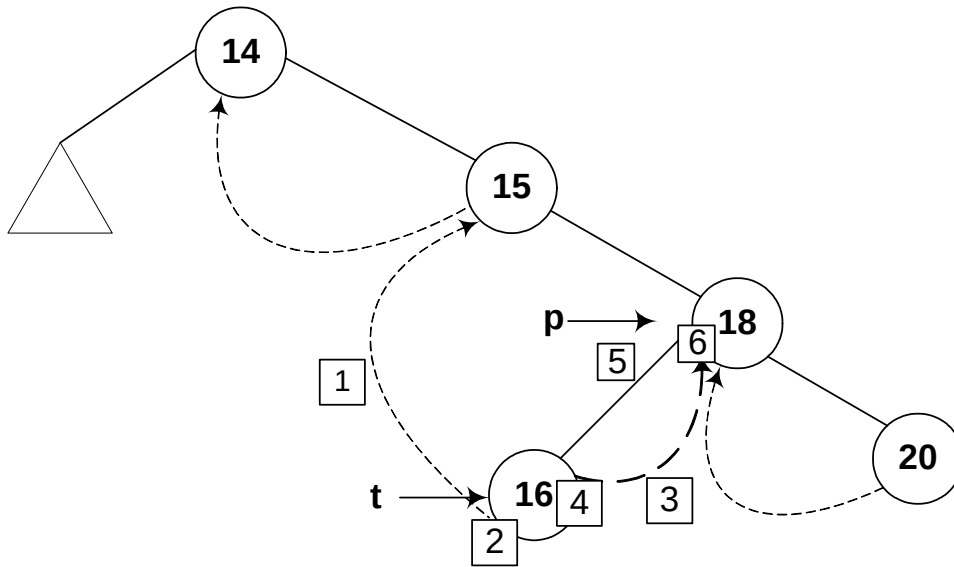


Fig 27.9

The flag is truned on in the last line, $p \rightarrow LTH = child$.

If we insert few more nodes in the tree, we have the tree as given below:

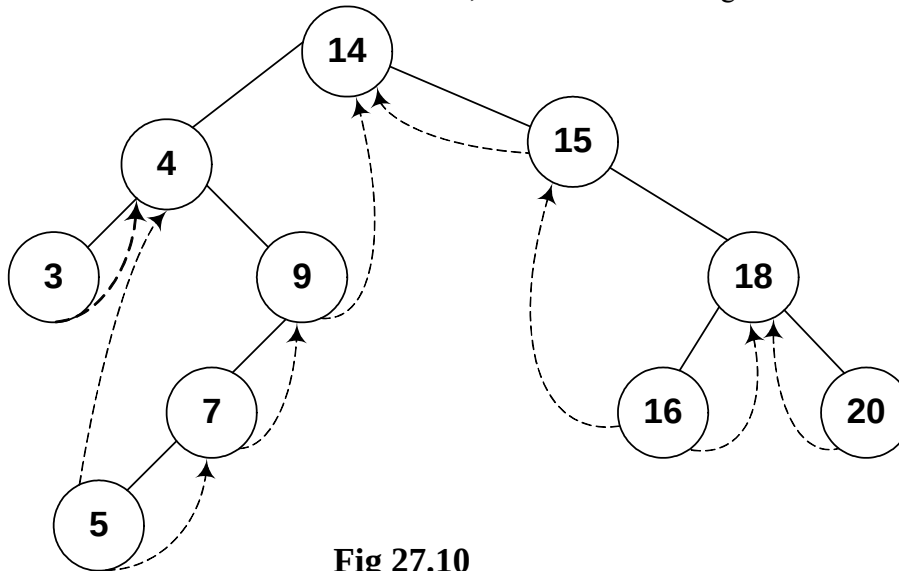


Fig 27.10

Above given is a BST and you have seen many BSTs before, which are not thread binary trees. Without the threads, it is clear from the figure that there are number of links present in the tree that are NULL. We have converted the NULLs to threads in this tree.

Let's do inorder non-recursive traversal of the tree. We started at 14 then following the left link came to 4 and after it to 3. If we use recursion then after the call for node 3 is finished (after printing 3), it returns to node 4 call and then 4 is printed using the recursive call stack. Here we will print 3 but will not stop. As we have used threads, we see the right pointer of node 3 that is not NULL and pointing to its successor node

4, we go to 4 and print it. Now which node is inorder successor of node 4. It is node 5. From node 4, we traversed to right child of it node 9. From node 9, we went to node 7 and then finally node 5. Now, this node 5 is a leaf node. Previously, without using threads, we could identify leaf nodes, whose both pointers left and right were NULL. In this case, using threads, as discussed above, we set the pointers and turn the flags on when a pointer left or right is set to its predecessor or successor. After printing node 5, we traverse its right thread and go to node 7. In this fashion, whole of the tree can be traversed without recursion.

Now, let's see some code:

```
TreeNode* nextInorder(TreeNode* p)
{
    if(p->RTH == thread)
        return(p->R);
    else {
        p = p->R;
        while(p->LTH == child)
            p = p->L;
        return p;
    }
}
```

Above given is a routine *nextInorder*, which gives the inorder successor of a node passed in parameter pointer *p*. Now what it does is:

If the *RTH* flag of the *p* node (the node passed in the parameter) is *thread* then it will return the node being pointed by the right thread (*p->R*), which would be its inorder successor. Otherwise, it does the following.

It goes to the right node of *p* and starts pointing it using the same *p* pointer. From there it keeps on moving (in a loop fashion) towards the left of the node as long as the statement *p->LTH == child* is true. After this loop is terminated, the node *p* is returned.

Next, we see this pictorially.

Where is Inorder Successor?

Inorder successor of 4.

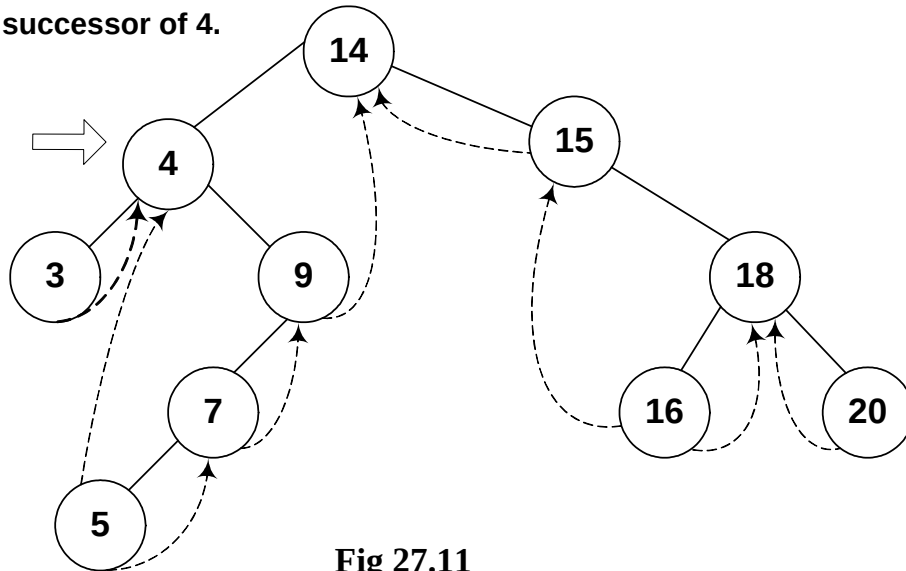


Fig 27.11

We are at node 4 and want to find its inorder successor. If you remember the *delete* operation discussed in the previous lecture, where we searched for the inorder successor and found it to be the *left-most node in the right subtree* of the node.

Inorder successor of 4.

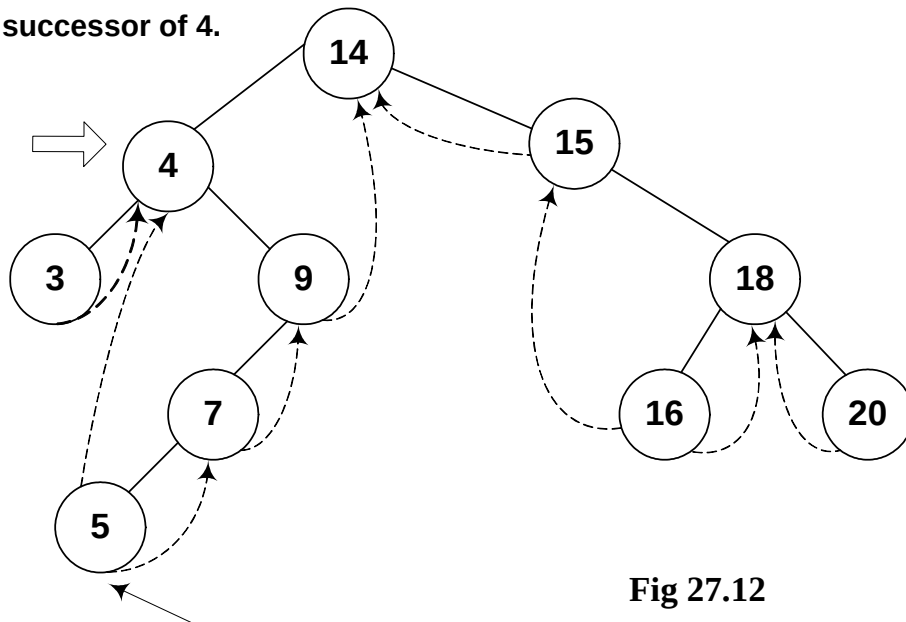


Fig 27.12

Left most node in right subtree of 4

In this figure, the right subtree of 4 is starting from node 9 and ending at node 5. Node 5 is the left most node of it and this is also the inorder successor of node 4. We cannot go to node 5 directly from node 4, we go to node 9 first then node 7 and finally to node 5.

Inorder successor of 9.

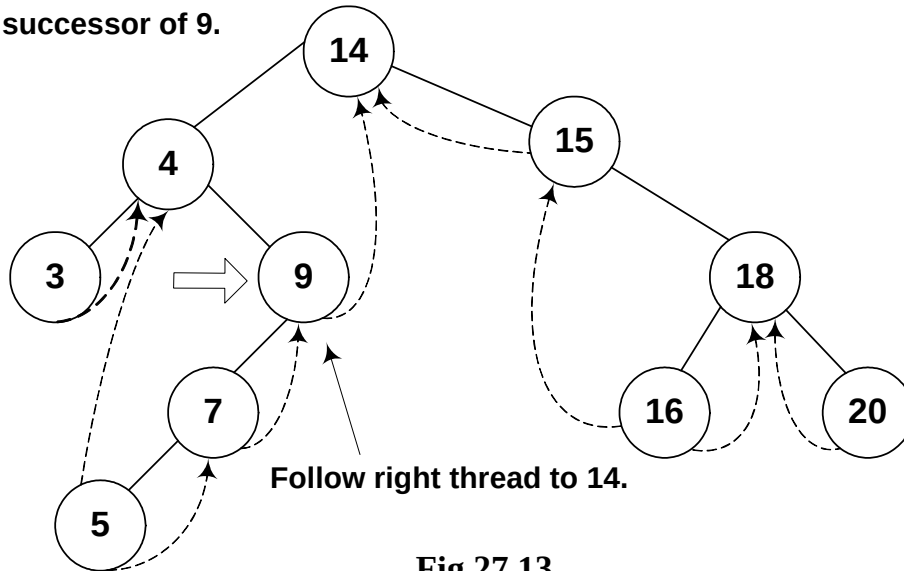


Fig 27.13

We move from node 9 to node 5 following the normal tree link and not thread. As long as the normal left tree link is there of a node, we have set the *LTH* flag to *child*. When we reach at node 5, the left link is a thread and it is indicated with a flag. See the while loop given in the above routine again:

```
while(p->LTH == child)
    p = p->L;
return p;
```

Inorder Traversal

Now by using this routine, we try to make our inorder traversal procedure that is non-recursive and totally stack free.

- *If we can get things started correctly, we can simply call `nextInorder` repeatedly (in a simple loop) and move rapidly around the tree inorder printing node labels (say) - without a stack.*

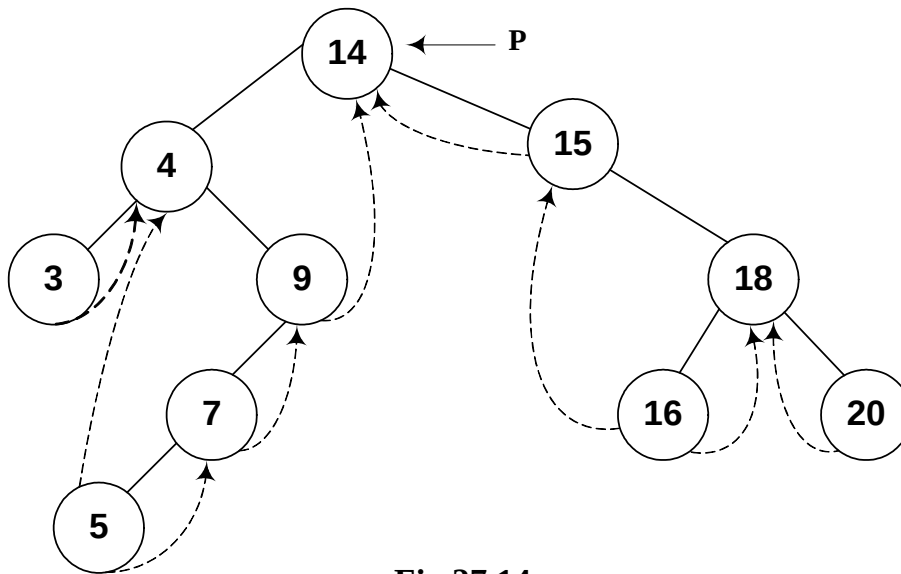


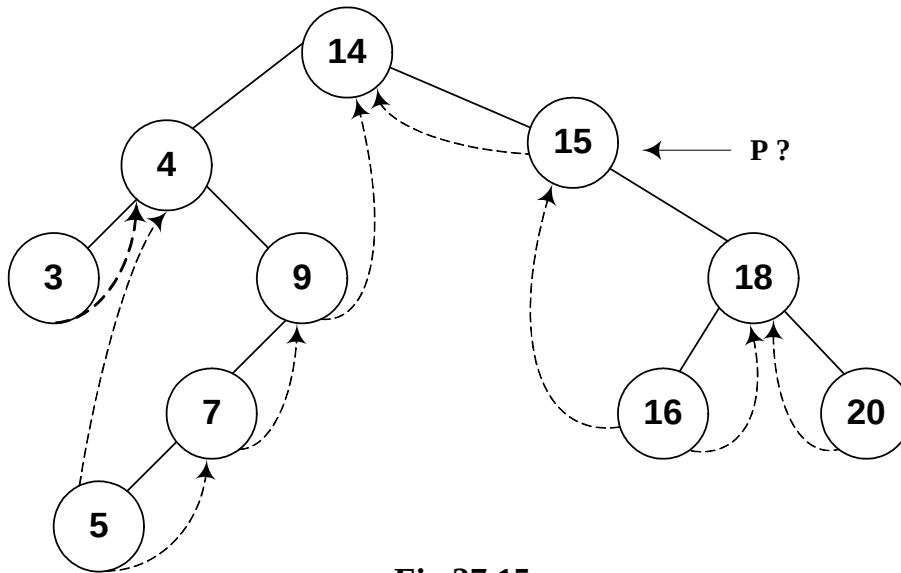
Fig 27.14

The pointer *p* is pointing to the *root* node of the tree. If we start traversing from this node and pass this *root* node pointer to our routine *nextInorder* above, it will create a problem.

We see the routine again to see the problem area clearly:

```
TreeNode* nextInorder(TreeNode* p)
{
    if(p->RTH == thread)
        return(p->R);
    else {
        p = p->R;
        while(p->LTH == child)
            p = p->L;
        return p;
    }
}
```

In the first part, it is checking for the RTH flag to be set to *thread*, which is not the case for the root node. The control will be passed to the else part of the routine. In else part, in the very first step, we are moving towards right of root that is to node 15.

**Fig 27.15**

- *If we call nextInorder with the root of the binary tree, we're going to have some difficulty. The code won't work at all the way we want.*

Note that in this tree inorder traversal, the first number we should print is 3 but now we have reached to node 15 and we don't know how can we reach node 3. This has created a problem. In the lecture, we will make a small change in this routine to cater to this situation i.e. when a *root* node pointer is passed to it as a parameter. After that change the routine will work properly in case of *root* node also and it will be non-recursive, stack free routine to traverse the tree.

Data Structures

Lecture No. 28

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 6
6.3.1

Summary

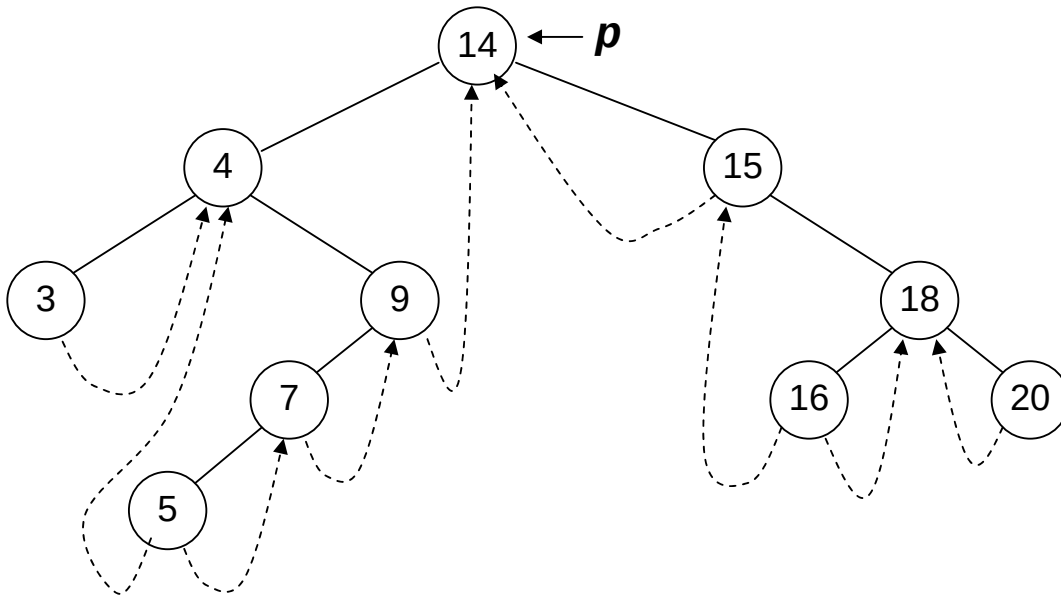
- Inorder traversal in threaded trees
- Complete Binary Tree

Inorder traversal in threaded trees

Discussion on the inorder traversal of the threaded binary tree will continue in this lecture. We have introduced the threads in the tree and have written the *nextInorder* routine. It is sure that the provision of the root can help this routine perform the inorder routine properly. It will go to the left most node before following the threads to find the inorder successors. The code of the routine is given below:

```
/* The inorder routine for threaded binary tree */  
  
TreeNode* nextInorder(TreeNode* p){  
  
    if(p->RTH == thread) return(p->R);  
    else {  
        p = p->R;  
        while(p->LTH == child)  
            p = p->L;  
        return p;  
    }  
}
```

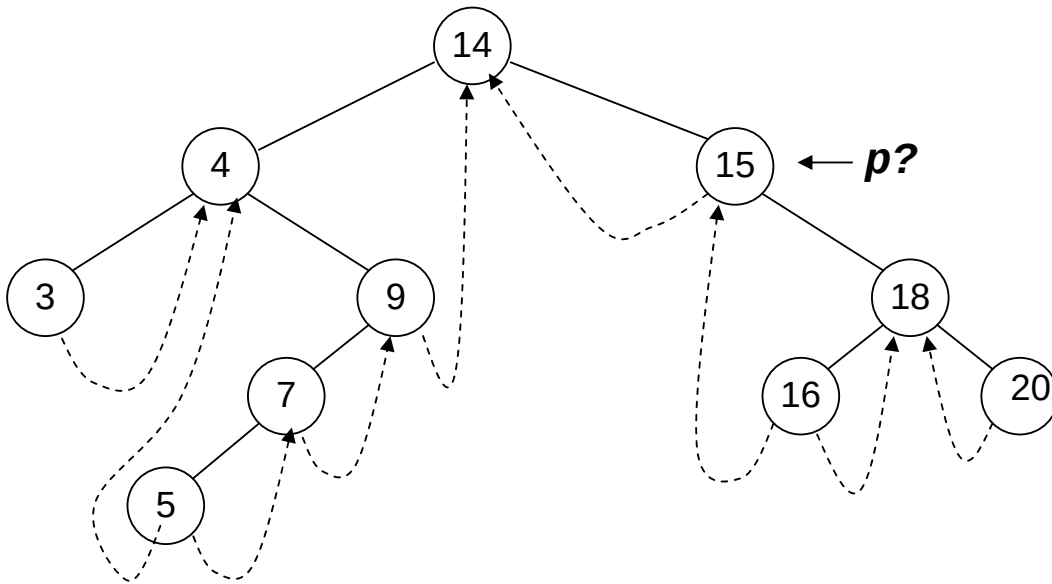
When we apply this routine on the sample tree, it does not work properly because the pointer that points to the node goes in the wrong direction. How can we fix this problem? Let's review the threaded binary tree again:



In the above figure, we have a binary search tree. Threads are also seen in it. These threads point to the successor and predecessor.

Our *nextInorder* routine, first of all checks that the right pointer of the node is thread. It means that it does not point to any tree node. In this case, we will return the right pointer of the node as it is pointing to the inorder successor of that node. Otherwise, we will go to some other part. Here we will change the value of pointer p to its right before running a while loop as long as the left pointer is the node. That means the left child is not a thread. We move to the left of the pointer p and keep on doing so till the time the left pointer becomes a thread.

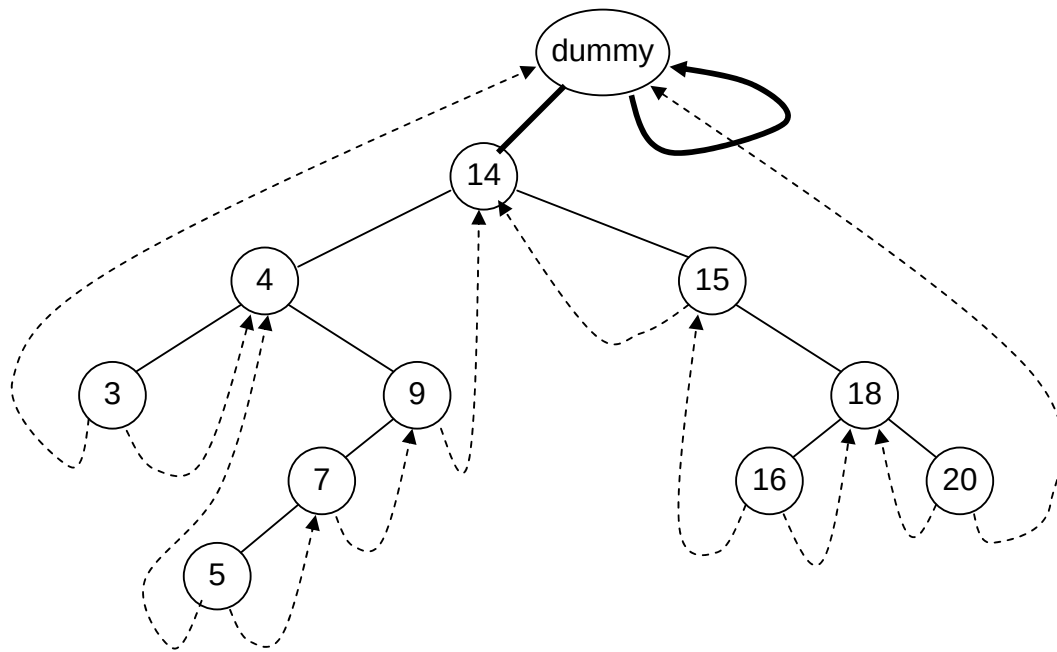
We will pass the root of the tree to the *nextInorder* routine. The pointer p is pointing to the node 14 i.e. the root node. As the right pointer of the node 14 is not a thread, so the pointer p will move to the node 15 as shown below:



Here we want the inorder traversal. It is obvious from the above figure that 15 is not the first value. The first value should be 3. This means that we have moved in the wrong direction. How this problem can be overcome? We may want to implement some logic that in case of the root node, it is better not to go towards the right side. Rather, the left side movement will be appropriate. If this is not the root node, do as usual. It may lend complexities to our code. Is there any other way to fix it? Here we will use a programming trick to fix it.

We will make this routine as a private member function of the class so other classes do not have access to it. Now what is the trick? We will insert a new node in the tree. With the help of this node, it will be easy to find out whether we are on the root node or not. This way, the pointer p will move in the correct direction.

Let's see this trick. We will insert an extra node in the binary tree and call it as a *dummy* node. This is well reflected in the diagram of the tree with the *dummy* node. We will see where that dummy node has been inserted.



This dummy node has either no value or some dummy value. The left pointer of this node is pointing to the root node of the tree while the right pointer is seen pointing itself i.e. to *dummy* node. There is no problem in doing all these things. We have put the address of *dummy* node in its right pointer and pointed the left thread of the left most node towards the *dummy* node. Similarly the right thread of the right-most node is pointing to the *dummy* node. Now we have some extra pointers whose help will make the *nextInorder* routine function properly.

Following is a routine *fastInorder* that can be in the public interface of the class.

```
/* This routine will traverse the binary search tree */

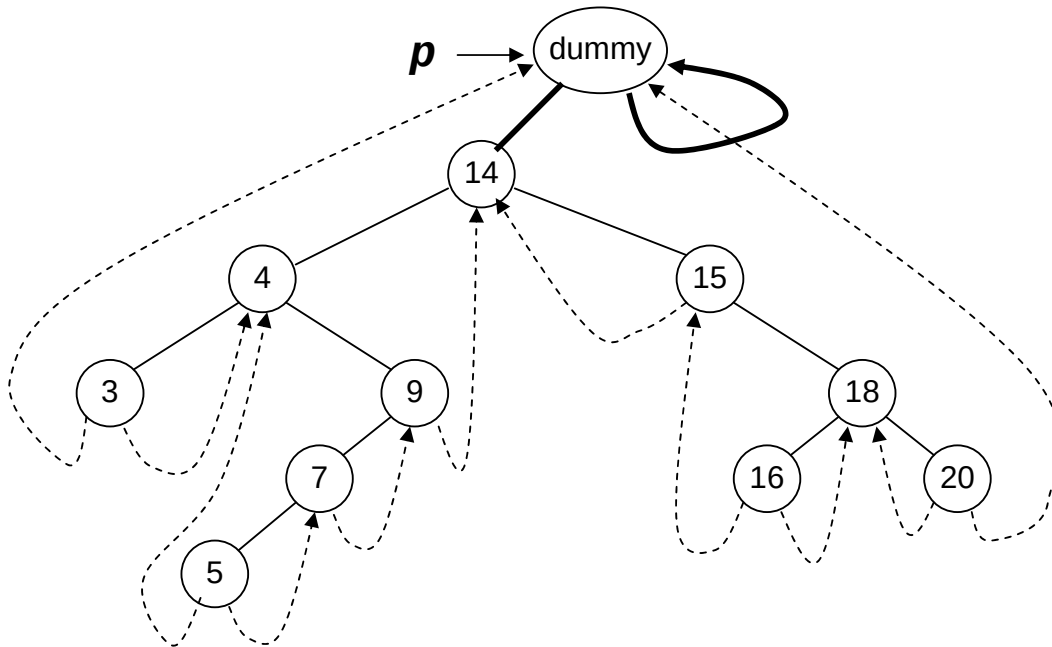
void fastInorder(TreeNode* p)
{
    while((p=nexInorder(p)) != dummy) cout << p->getInfo();
}
```

This routine takes a *TreeNode* as an argument that make it pass through the root of the tree. In the while loop, we are calling the *nextInorder* routine and pass it *p*. The pointer returned from this routine is then assigned to *p*. This is a programming style of C. We are performing two tasks in a single statement i.e. we call the *nextInorder* by passing it *p* and the value returned by this routine is saved in *p*. Then we check that the value returned by the *nextInorder* routine that is now actually saved in *p*, is not a *dummy* node. Then we print the *info* of the node. This function is called as:

```
fastInorder(dummy);
```

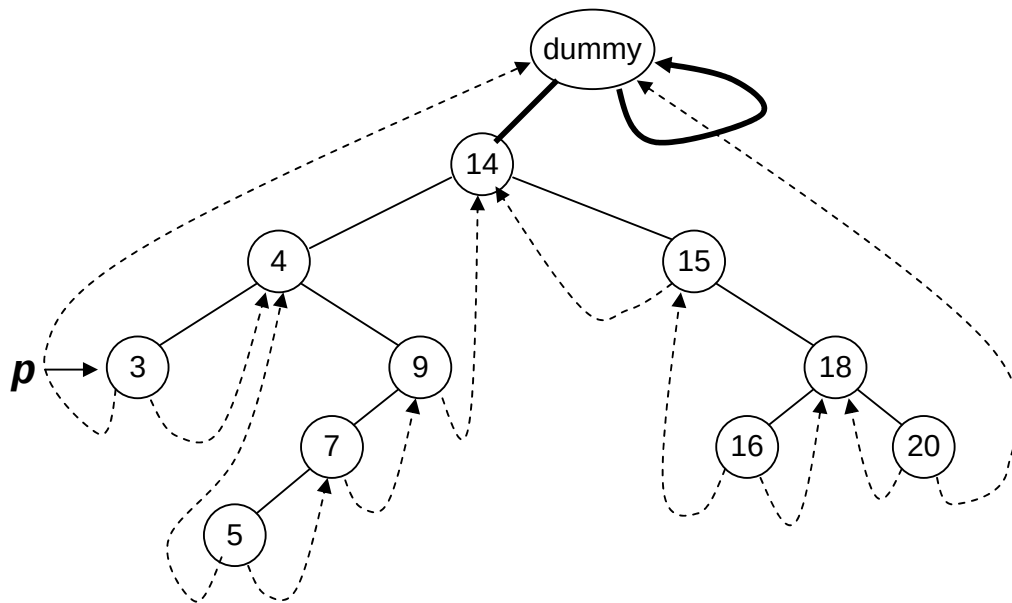
We are not passing it the root of the tree but the *dummy* node. Now we will get the correct values and see in the diagrams below that *p* is now moving in the right direction. Let's try to understand this with the help of diagrams.

First of all we call the *nextInorder* routine passing it the dummy node.



The pointer p is pointing to the *dummy* node. Now we will check whether the right pointer of this node is not thread. If so, then it is advisable to move the pointer towards the right pointer of the node. Now we will go to the while loop and start moving on the left of the node till the time we get a node with the left pointer as thread. The pointer p will move from dummy to node 14. As the left pointer of node 14 is not thread so p will move to node 4. Again the p will move to node 3. As the left pointer of p is thread, the while loop will finish here. This value will be returned that is pointing to node 3. The node 3 should be printed first of all regarding the inorder traversal. So with the help of our trick, we get the right information.

Now the while loop in the *fastInorder* will again call the *nextInorder* routine. We have updated the value of p in the *fastInorder* that is now pointing to the node 3. This is shown in the figure below:



According to the code, we have to follow the right thread of the node 3 that is pointing to the node 4. Therefore p is now pointing to the node 4. Here 4 is inorder successor of 3. So the pointer p has moved to the correct node for inorder traversal.

As the right pointer of the node 4 is a link, p will move to node 9. Later, we will go on the left of nodes and reach at the node 5. Looking at the tree, we know that the inorder successor of the node 4 is node 5. In the next step, we will get the node 7 and so on. With the help of threads and links, we are successful in getting the correct inorder traversal. No recursive call has been made so far. Therefore stack is not used. This inorder traversal will be faster than the recursive inorder traversal. When other classes use this routine, it will be faster. We have not used any additional memory for this routine. We are using the null links and putting the values of thread in it. This routine is very simple to understand. In the recursive routines, we have to stop the recursion at some condition. Otherwise, it will keep on executing and lead to the aborting of our program.

Complete Binary Tree

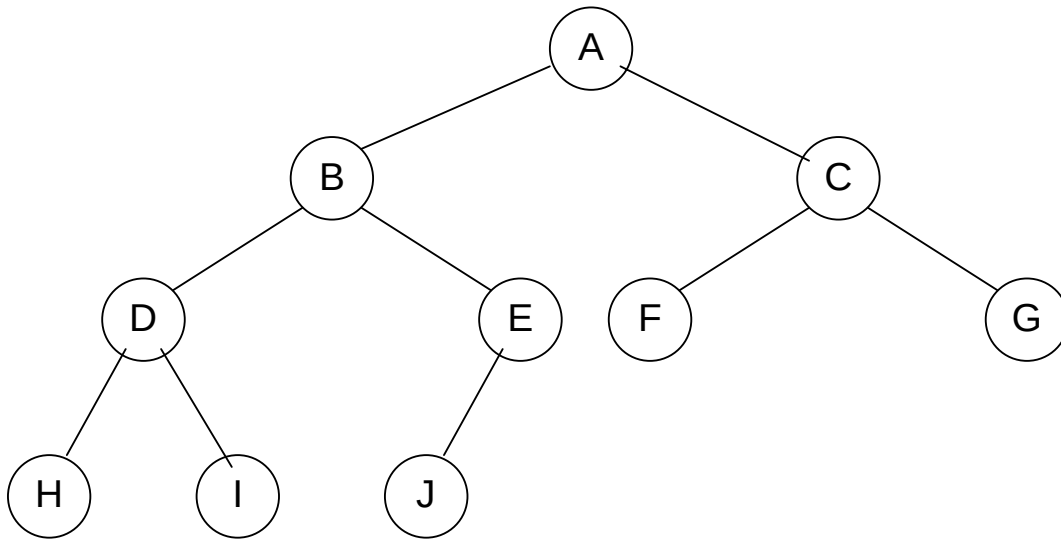
We have earlier discussed the properties of the binary trees besides talking about the internal and external nodes' theorem. Now we will discuss another property of binary trees that is related to its storage before dilating upon the complete binary tree and the heap abstract data type.

Here is the definition of a complete binary tree:

- A complete binary tree is a tree that is completely filled, with the possible exception of the bottom level.
- The bottom level is filled from left to right.

You may find the definition of complete binary tree in the books little bit different from this. A perfectly complete binary tree has all the leaf nodes. In the complete binary tree, all the nodes have left and right child nodes except the bottom level. At the bottom level, you will find the nodes from left to right. The bottom level may not

be completely filled, depicting that the tree is not a perfectly complete one. Let's see a complete binary tree in the figure below:



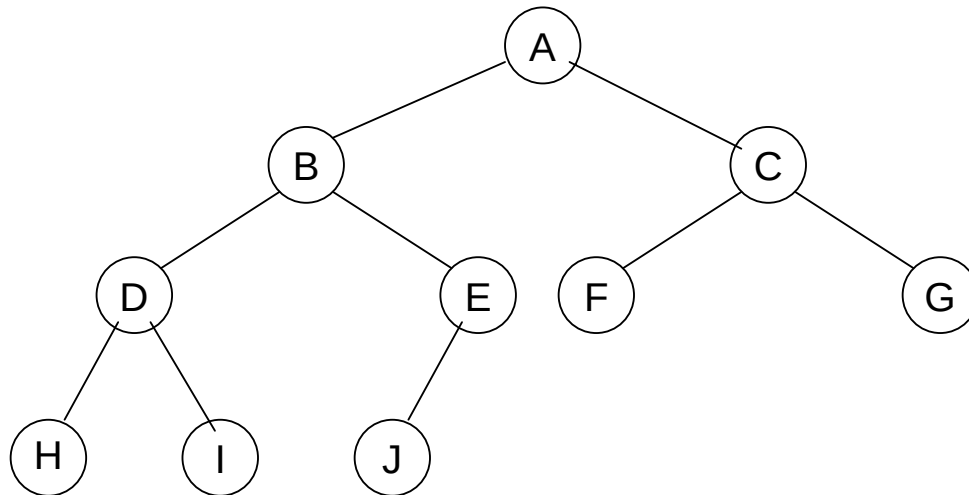
In the above tree, we have nodes as A, B, C, D, E, F, G, H, I, J. The node D has two children at the lowest level whereas node E has only left child at the lowest level that is J. The right child of the node E is missing. Similarly node F and G also lack child nodes. This is a complete binary tree according to the definition given above. At the lowest level, leaf nodes are present from left to right but all the inner nodes have both children. Let's recap some of the properties of complete binary tree.

- A complete binary tree of height h has between 2^h to $2^{h+1} - 1$ nodes.
- The height of such a tree is $\lfloor \log_2 N \rfloor$ where N is the number of nodes in the tree.
- Because the tree is so regular, it can be stored in an *array*. No pointers are necessary.

We have taken the floor of the $\log_2 N$. If the answer is not an integer, we will take the next smaller integer. So far, we have been using the pointers for the implementation of trees. The *TreeNode* class has left and right pointers. We have pointers in the balance tree also. In the threaded trees, these pointers were used in a different way. But now we can say that an array can be stored in a complete binary tree without needing the help of any pointer.

Now we will try to remember the characteristics of the tree. 1) The data element can be numbers, strings, name or some other data type. The information is stored in the node. We may retrieve, change or delete it. 2) We link these nodes in a special way i.e. a node can have left or right subtree or both. Now we will see why the pointers are being used. We just started using these. If we have some other structure in which trees can be stored and information may be searched, then these may be used. There should be reason for choosing that structure or pointer for the manipulation of the trees. If we have a complete binary tree, it can be stored in an array easily.

The following example can help understand this process. Consider the above tree again.



	A	B	C	D	E	F	G	H	I	J				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14

We have seen an array of size 15 in which the data elements A, B, C, D, E, F, G, H, I, J have been stored, starting from position 1. The question arises why we have stored the data element this way and what is justification of storing the element at the 1st position of the array instead of 0th position? You will get the answers of these very shortly.

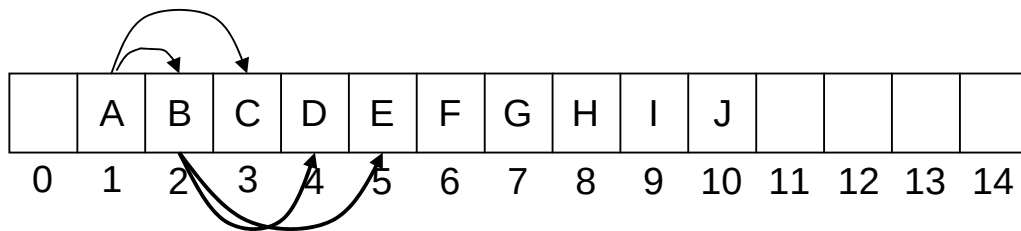
The root node of this tree is A and the left and right children of A are B and C. Now look at the array. While storing elements in the array, we follow a rule given below:

- For any array element at position i , the left child is at $2i$, the right child is at $(2i + 1)$ and the parent is at $\text{floor}(i/2)$.

In the tree, we have links between the parent node and the children nodes. In case of having a node with left and right children, stored at position i in the array, the left child will be at position $2i$ and the right child will be at $2i+1$ position. If the value of i is 2, the parent will be at position 2 and the left child will be at position $2i$ i.e. 4. The right child will be at position $2i+1$ i.e. 5. You must be aware that we have not started from the 0th position. It is simply due to the fact if the position is 0, $2i$ will also become 0. So we will start from the 1st position, ignoring the 0th.

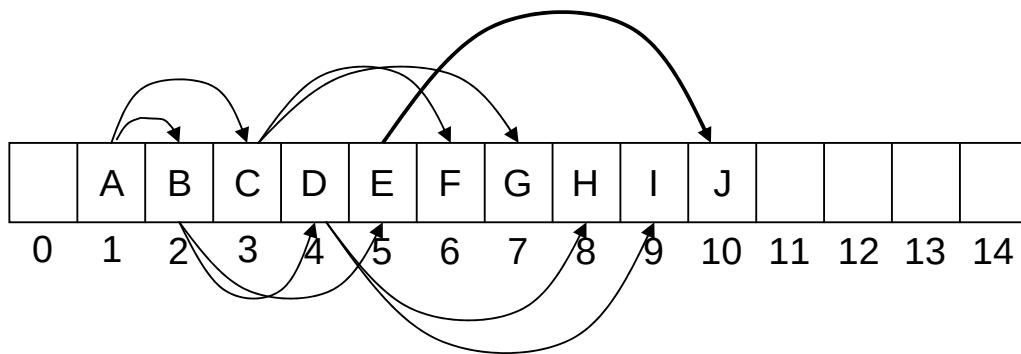
Lets see this formula on the above array. We have A at the first position and it has two children B and C. According to the formula the B will be at the $2i$ i.e. 2nd position and C will be at $2i+1$ i.e. 3rd position. Take the 2nd element i.e. B, it has two children D and E. The position of B is 2 i.e. the value of i is 2. Its left child D will be at position $2i$ i.e. 4th position and its right child E will be at position $2i+1$ i.e. 5. This is shown in the

figure below:

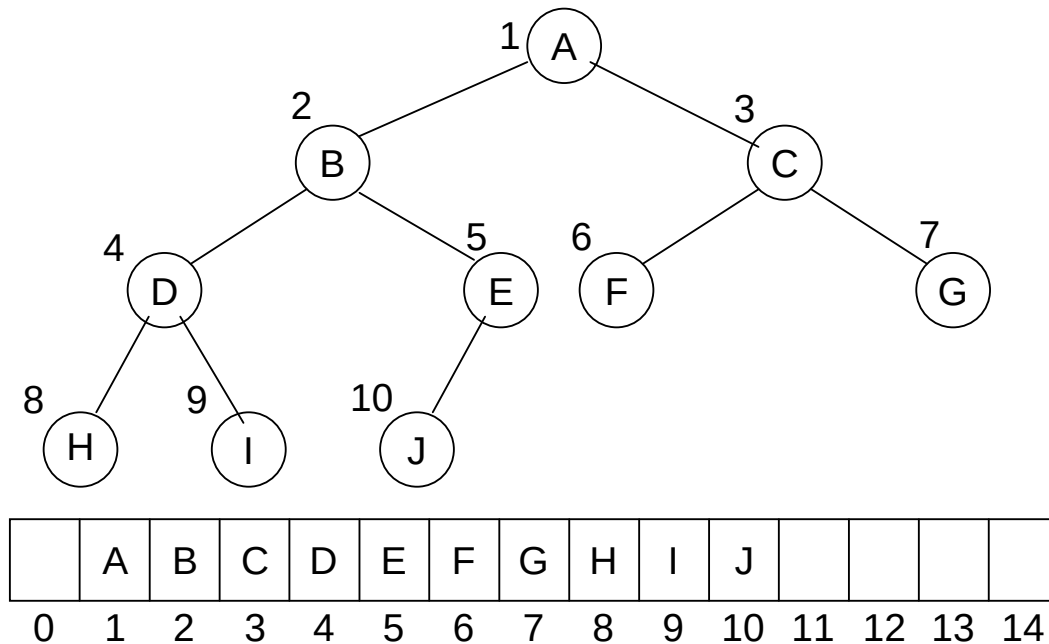


If we want to keep the tree's data in the array, the children of *B* should be at the position 4 and 5. This is true. We can apply this formula on the remaining nodes also. Now you have understood how to store tree's data in an array. In one respect, we are using pointers here. These are not C++ pointers. In other words, we have implicit pointers in the array. These pointers are hidden. With the help of the formula, we can obtain the left and right children of the nodes i.e. if the node is at the *i*th position, its children will be at $2i$ and $2i+1$ position. Let's see the position of other nodes in the array.

As the node *C* is at position 3, its children should be at $2*3$ i.e. 6th position and $2*3+1$ i.e. 7th position. The children of *C* are *F* and *G* which should be at 6th and 7th position. Look at the node *D*. It is at position 4. Its children should be at position 8 and 9. *E* is at position 5 so its children should be at 10 and 11 positions. All the nodes have been stored in the array. As the node *E* does not have a right child, the position 11 is empty in the array.



You can see that there is only one array going out of *E*. There is a link between the parent node and the child node. In this array, we can find the children of a node with the help of the formula i.e. if the parent node is at *i*th position, its children will be at $2i$ and $2i+1$ position. Similarly there is a link between the child and the parent. A child can get its parent with the help of formula i.e. if a node is at *i*th position, its parent will be at $\text{floor}(i/2)$ position. Let's check this fact in our sample tree. See the diagram below:



Level Order Numbers & Array index

Consider the node *J* at position is 10. According to the formula, its parent should be at $\text{floor}(10/2)$ i.e. 5 which is true. As the node *I* is at position 9, its parent should be at $\text{floor}(9/2)$ i.e. 4. The result of $9/2$ is 4.5. But due to the floor, we will round it down and the result will be 4. We can see that the parent of *I* is *D* which is at position 4. Similarly the parent of *H* will be at $\text{floor}(8/2)$. It means that it will be at 4. Thus we see that *D* is its parent. The links shown in the figure depict that *D* has two children *H* and *I*. We can easily prove this formula for the other nodes.

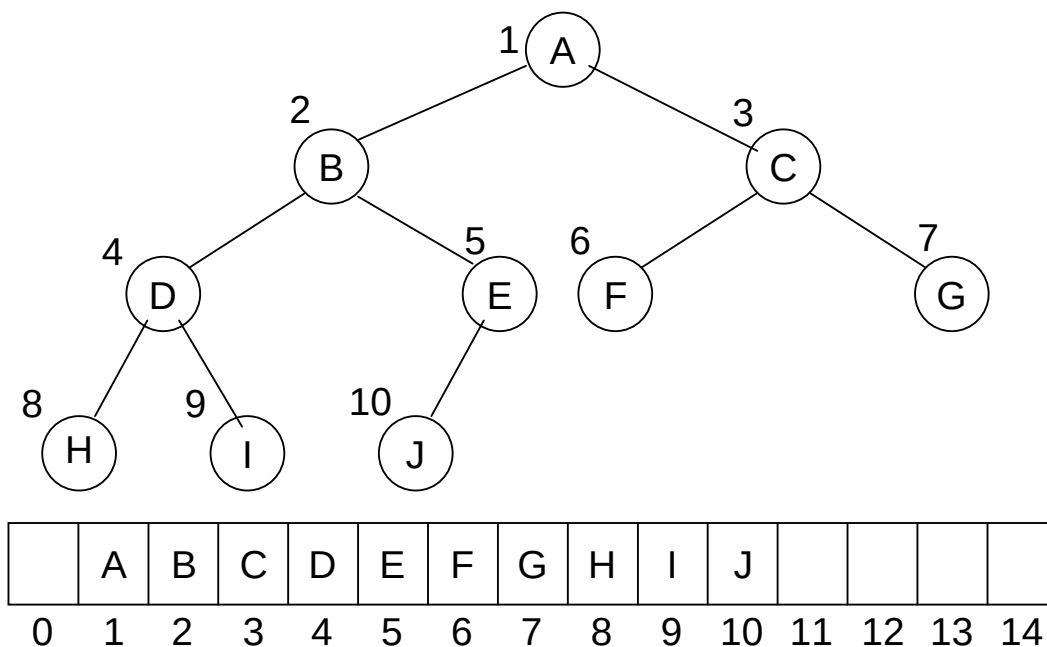
From the above discussion we note three things. 1) We have a complete binary tree, which stores some information. It may or may not be a binary search tree. This tree can be stored in an array. We use $2i$ and $2i+1$ indexing scheme to put the nodes in the array. Now we can apply the algorithms of tree structure on this array structure, if needed.

Now let's talk about the usage of pointers and array. We have read that while implementing data structures, the use of array makes it easy and fast to add and remove data from arrays. In an array, we can directly locate a required position with the help of a single index, where we want to add or remove data. Array is so important that it is a part of the language. Whereas the data structures like tree, stack and queue are not the part of C or C++ language as a language construct. However we can write our classes for these data structures. As these data structures are not a part of the language, a programmer can not declare them directly. We can not declare a tree or a stack in a program. Whereas we can declare an array directly as `int x [ ]`; The array data type is so efficient and is of so common use that most of the languages support it. The compiler of a language handles the array and the programmer has to do nothing for declaring and using an array.

We have built the binary trees with pointers. The use of pointers in the memory requires some time. In compilers or operating system course, we will read that when a program is stored in the memory and becomes a process, the executable code does not come in the memory. There is a term paging or virtual memory. When a program executes, some part of it comes in the memory. If we are using pointers to go to different parts of the program, some part of the code of program will be coming (loading) to memory while some other may be removed (unloading) from the memory. This loading and unloading of program code is executed by a mechanism, called paging. In Windows operating system, for this virtual memory (paging mechanism), a file is used, called page file. With the use of pointers, this process of paging may increase. Due to this, the program may execute slowly. In the course of Operating System and Compilers, you will read in detail that the usage of pointers can cause many problems.

So we should use arrays where ever it can fulfill our requirements. The array is a very fast and efficient data structure and is supported by the compiler. There are some situations where the use of pointers is beneficial. The balancing of AVL tree is an example in this regard. Here pointers are more efficient. If we are using array, there will be need of moving a large data here and there to balance the tree.

From the discussion on use of pointers and array, we conclude that the use of array should be made whenever it is required. Now it is clear that binary tree is an important data structure. Now we see that whether we can store it in an array or not. We can surely use the array. The functions of tree are possible with help of array. Now consider the previous example of binary tree. In this tree, the order of the nodes that we maintained was for the indexing purpose of the array. Moreover we know the level-order traversal of the tree. We used queue for the level-order of a tree. If we do level-order traversal of the tree, the order of nodes visited is shown with numbers in the following figure.



In the above figure, we see that the number of node *A* is 1. The node *B* is on number 2 and *C* is on number 3. At the next level, the number of nodes *D*, *E*, *F* and *G* are 4, 5, 6 and 7 respectively. At the lowest level, the numbers 8, 9 and 10 are written with nodes *H*, *I* and *J* respectively. This is the level-order traversal. You must remember that in the example where we did the preorder, inorder and post order traversal with recursion by using stack. We can do the level-order traversal by using a queue. Now after the level-order traversal, let's look at the array shown in the lower portion of the above figure. In this array, we see that the numbers of *A*, *B*, *C* and other nodes are the same as in the level-order traversal. Thus, if we use the numbers of level-order traversal as index, the values are precisely stored at that numbers in the array. It is easy for us to store a given tree in an array. We simply traverse the tree by level-order and use the order number of nodes as index to store the values of nodes in the array. A programmer can do the level-order traversal with queue as we had carried out in an example before. We preserve the number of nodes in the queue before traversing the queue for nodes and putting the nodes in the array. We do not carry out this process, as it is unnecessarily long and very time consuming. However, we see that the level-order traversal directly gives us the index of the array depending upon which data can be stored.

Data Structures

Lecture No. 29

Reading Material

Data Structures and Algorithm Analysis in C++

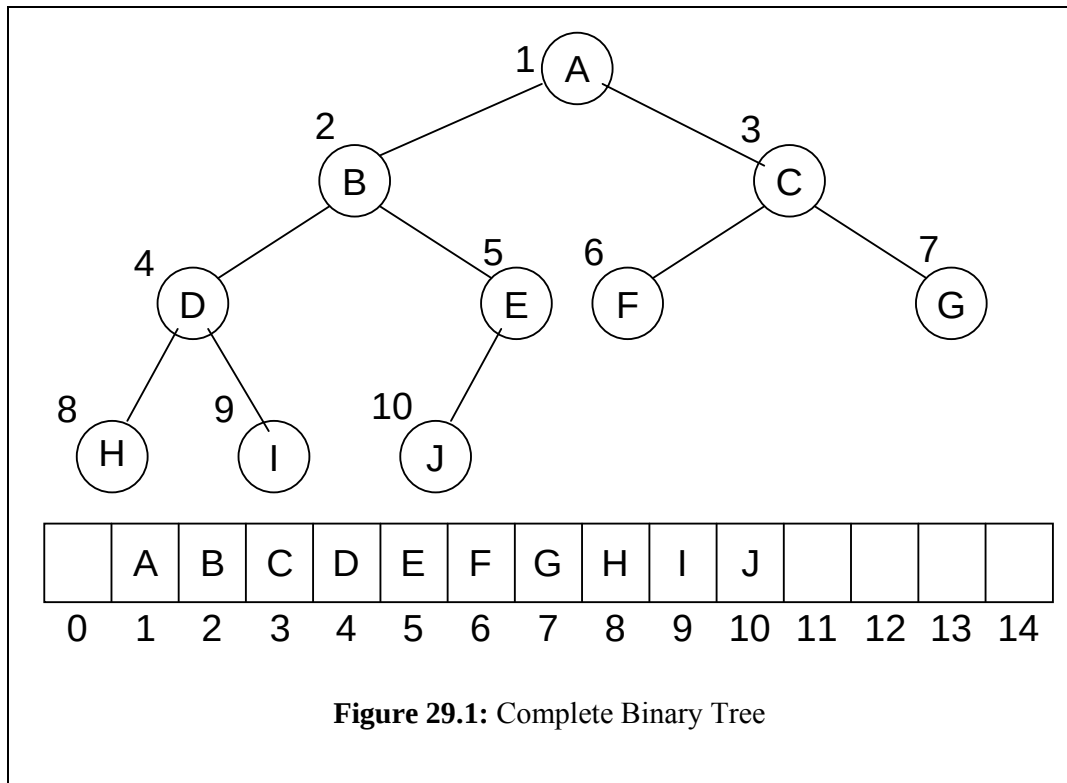
Chapter. 6
6.3

Summary

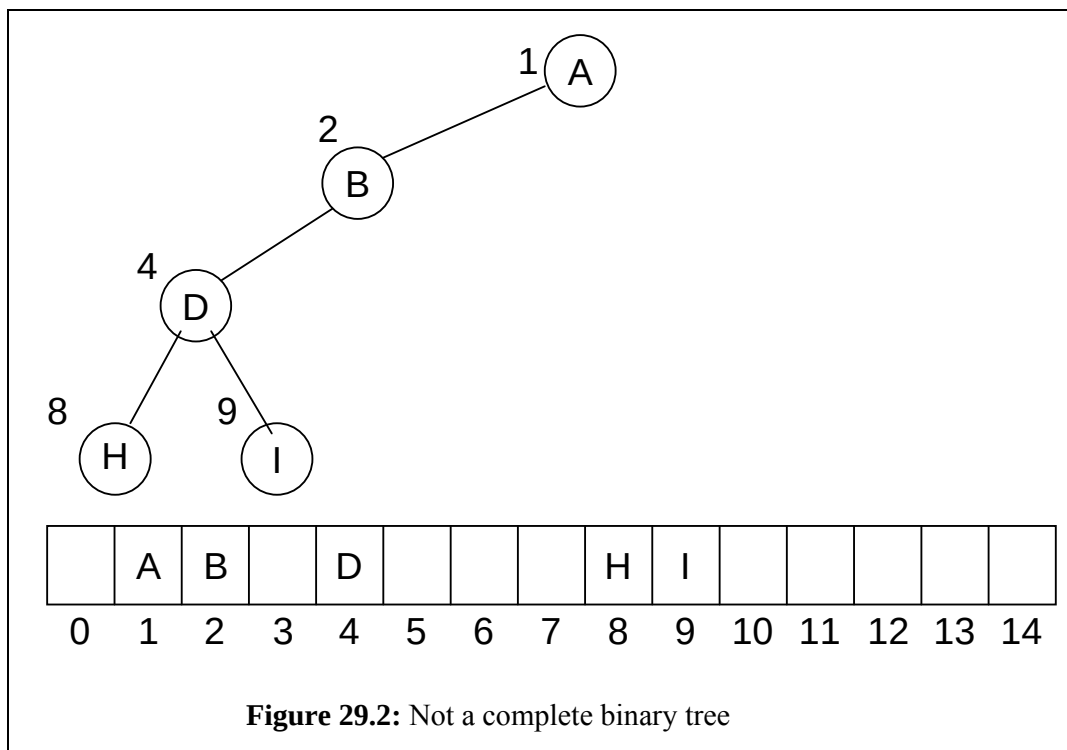
- Complete Binary Tree
- Heap
- Max Heap
- Insertion in a Heap

Complete Binary Tree

In the previous lecture, we talked about the ways to store a complete binary tree in an array. The $2i$ and $2i+1$ scheme were employed to store the link of the parent to children and link of children to parent. Through this link, a programmer can go to the children of a node. We know that array is a very efficient data structure. In a number of languages, it is found as a built-in data type. Now the question arises if we can store the binary tree in an array, why there should be the use of pointers? It is very simple that an array is used when the tree is a complete binary tree. Array can also be used for the trees that are not complete binary trees. But there will be a problem in this case. The size of the array will be with respect to the deepest level of the tree according to the traversal order. Due to incomplete binary tree, there will be holes in the array that means that there will be some positions in the array with no value of data. We can understand it with a simple example. Look at the following figure where we store a complete binary tree in an array by using $2i$ and $2i+1$ scheme. Here we stored the nodes from A to J in the array at index 1 to 10 respectively.



Suppose that this tree is not complete. In other words, B has no right subtree that means E and J are not there. Similarly we suppose that there is no right subtree of A. Now the tree will be in the form as shown in the following figure (29.2).



In this case, the effort to store this tree in the array will be of no use as the $2i$ and $2i+1$ scheme cannot be applied to it. To store this tree, it may be supposed that there are nodes at the positions of C, F, G, E and J (that were there in previous figure). Thus we transform it into a complete binary tree. Now we store the tree in the array by using $2i$ and $2i+1$ scheme. Afterwards, the data is removed from the array at the positions of the imaginary nodes (in this example, the nodes are C, F, G, E and J). Thus we notice that the nodes A, B and H etc are at the positions, depicting the presence of a complete binary tree. The locations of C, f, G, E and J in the array are empty as shown in the following figure.

	A	B		D				H	I					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Now imagine that an incomplete binary tree is very deep. We can store this tree in the array that needs to be of large size. There will be holes in the array. This is the wastage of memory. Due to this reason, it is thought that if a tree is not completely binary, it is not good to store it into an array. Rather, a programmer will prefer to use pointers for the storage.

Remember that two things are kept into view while constructing a data structure that is memory and time. There should such a data structure that could ensure the running of the programs in a fast manner. Secondly, a data structure should not use a lot of memory so that a large part of memory occupied by it does not go waste. To manage the memory in an efficient way, we use dynamic memory with the help of pointers. With the use of pointers only the required amount of memory is occupied.

We also use pointers for complex operations with data structure as witnessed in the deletion operation in AVL tree. One of the problems with arrays is that the memory becomes useless in case of too many empty positions in the array. We cannot free it and use in other programs as the memory of an array is contiguous. It is difficult to free the memory of locations from 50 to 100 in an array of 200 locations. To manage the memory in a better way, we have to use pointers.

Now we come to a new data structure, called 'heap'.

Heap

Heap is a data structure of big use and benefit. It is used in priority queue. Recall the example of bank simulation. In that case, we used event-based queues. We put the events that were going to happen in a special queue i.e. priority queue. This priority queue does not follow the FIFO rule. We put the elements in the queue at the end but later got the element with respect to its priority. We get the element that is going to occur first in the future. In that example, we implemented the priority queue with arrays. It was seen that when we insert an element in the queue, the internally used data was sorted in the array. Thus the event with minimum time of occurrence becomes at first position in the sorted array. We get the event with minimum time first. After the removal of the element, a programmer shifts the array elements to left. When we insert a new element, the array is sorted again. As there, in the bank example, were five or six events, the use of array fulfilled our requirement. We need not to have other data type that makes the queue efficient. The array is efficient but sorting is an expensive procedure. It may be complex and time-consuming. Secondly, we have to shift elements while adding or removing them from the array. Thus the

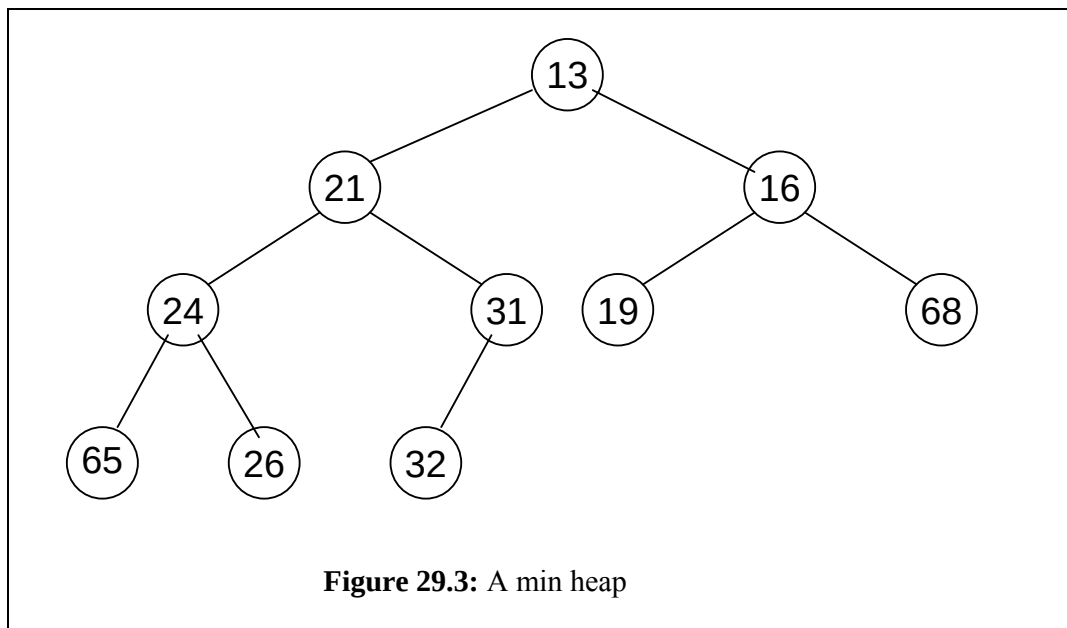
implementation of priority queue with an array is not efficient in terms of time. There is an alternate that is called heap. The use of priority queue with the help of heap is a major application. The priority queue is itself a major data structure, much-used in operating systems. Similarly priority queue data structure is used in network devices especially in routers. Heap data structure is also employed in sorting algorithms. There are also other uses of heap.

Let's discuss the heap data structure with special reference to its definition,

“The definition of heap is that it is a complete binary tree that conforms to the heap order”.

In this definition there is a term ‘heap order’. The heap order is a property that states that in a (min) heap for every node X, the key in the parent is smaller than (or equal to) the key in X. In other words, the parent node has key smaller than or equal to both of its children nodes. This means that the value in the parent node is less than the value on its children nodes. It is evident from the definition that we implement the heap by complete binary tree. It can also be implemented by some other method. But normally, we implement heap with complete binary tree. We know that in a binary search tree, the value of a node is greater than the values in its left subtree and less than the values in its right subtree. The heap has a variation to it. In heap, the value of a node is less than the values of its left and right children. The values in left and right children may be more or less with each other but these will be greater than the value in their parent node.

Consider the tree shown in the following figure.



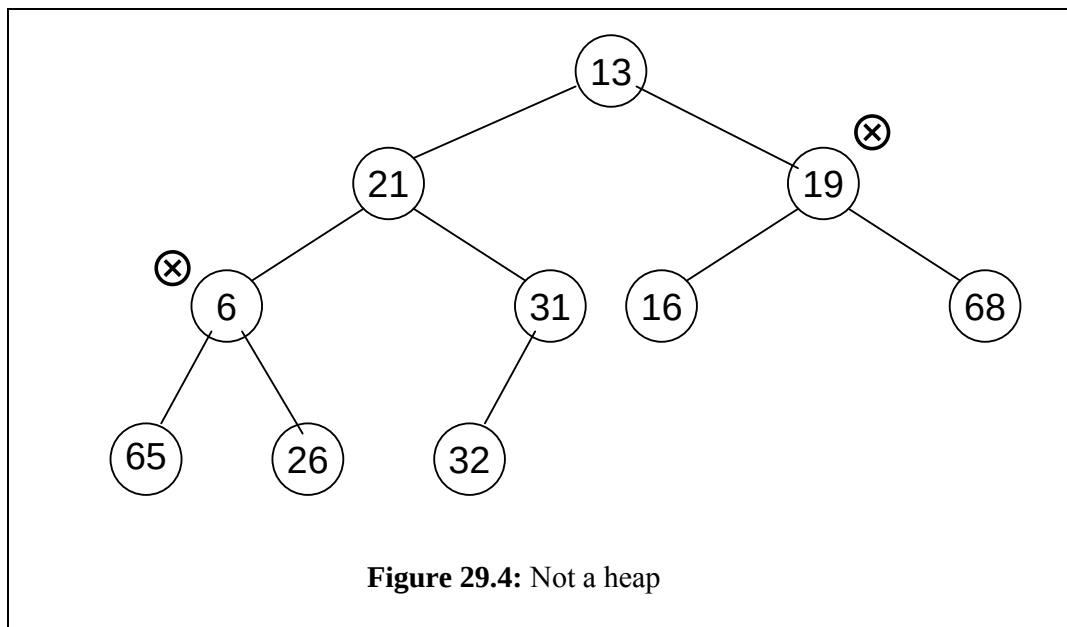
This is a complete binary tree. Now consider the values (numbers) in the nodes. This is not a binary search tree. If we carry out the inorder traversal, the result will be 65, 24, 26, 21, 32, 31, 13, 19, 16, 68.

We can see that it is not in a sorted order as got while traversing a binary search tree. So it is not a binary search tree. It's simply a complete binary tree.

Now we see the heap order in this tree. We start from node having value 13. Its left

child has value 21 while the value possessed by the right child is 16. So this is in line with the heap order property. Now we come to node 21. The values in its left and right child are 24 and 31 respectively. These values are greater than 21. Thus, it also fulfills the heap order property. Now we consider the node having value 16. The value in left child node of it is 19 and value in right child node is 68. So the value of parent node (i.e. 16) is less than the values of its children nodes. Similarly we can see other nodes. The node 24 is less than its children that are 65 and 26 respectively. And the node 31 is less than its child i.e. 32. Now for this tree, three things have been proved. First, this is a binary tree. Secondly it is a complete binary tree. Thirdly it fulfills the heap order property i.e. the value of the parent node is less than that of its left and right child nodes. This property is valid for every node of the tree. The leaf nodes have no child so there is no question of heap property. The heap shown in the figure above is a min heap. In the min heap, the value of parent node is less than the values in its child nodes. Thus in a min heap, the value of the root node is the smallest value in the tree.

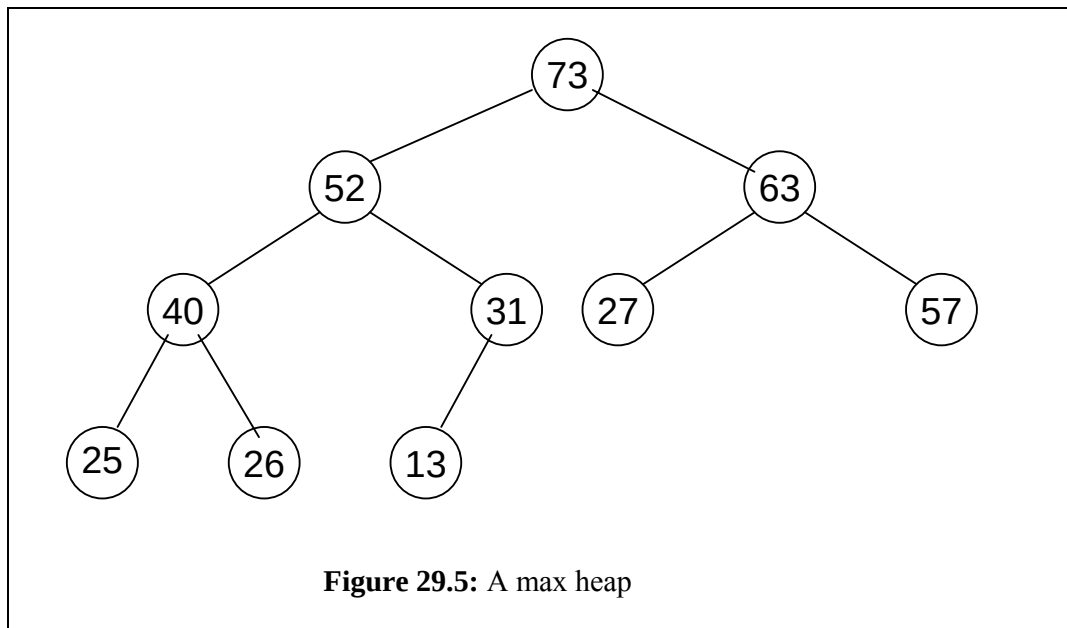
Now consider the tree in the following figure. This is a complete binary tree. This is neither a search tree, nor a heap.



Look at the node having value 19. The values in its left and right child nodes are 16 and 68 respectively. Thus the value of left child (i.e. 16) is less than that of the parent. So it is not a heap. If it were a heap or min heap, the value 16 should have been parent and the value 19 should have its child. Due to this violation, (the value of child is less than that of the parent) it is not a heap (min heap).

Max Heap

We can also make a max heap. In max heap, each node has a value greater than the value of its left and right child nodes. Moreover, in this case, the value of the root node will be largest and will become lesser at downward levels. The following figure shows a max heap.

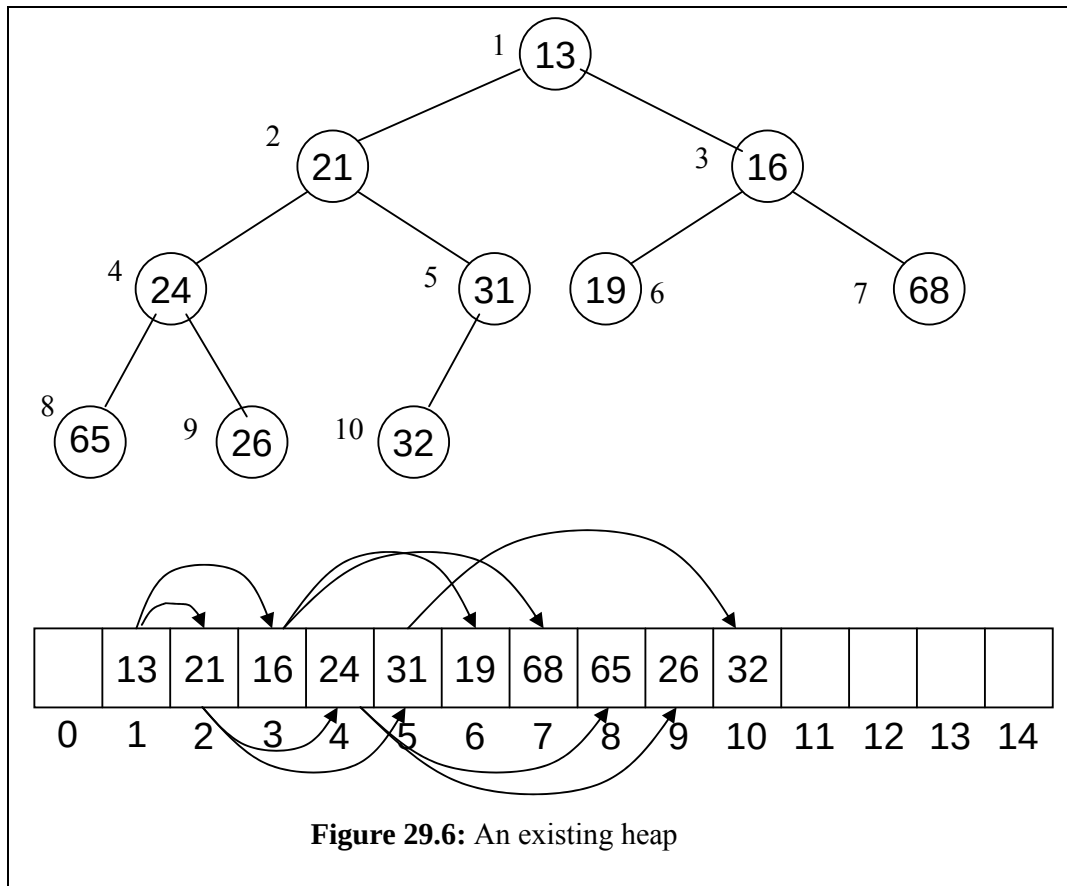


Consider the min heap again. By removing the root from the min heap, we get the smallest value. Now if the remaining values adjust themselves to again form a heap, the minimum value among these remaining values will get on the top. And if we remove this value, there will be the second minimum value of the heap. The remaining values again form a heap and there will be the smallest number among these on the top. Thus we will get the third minimum number. If we continue this process of getting the root node and forming a heap of remaining values, the numbers will get in ascending order. Thus we get the sorted data. By putting data in heap and getting it in a particular way, we achieve a sorting procedure. This way, a programmer gets sorted data.

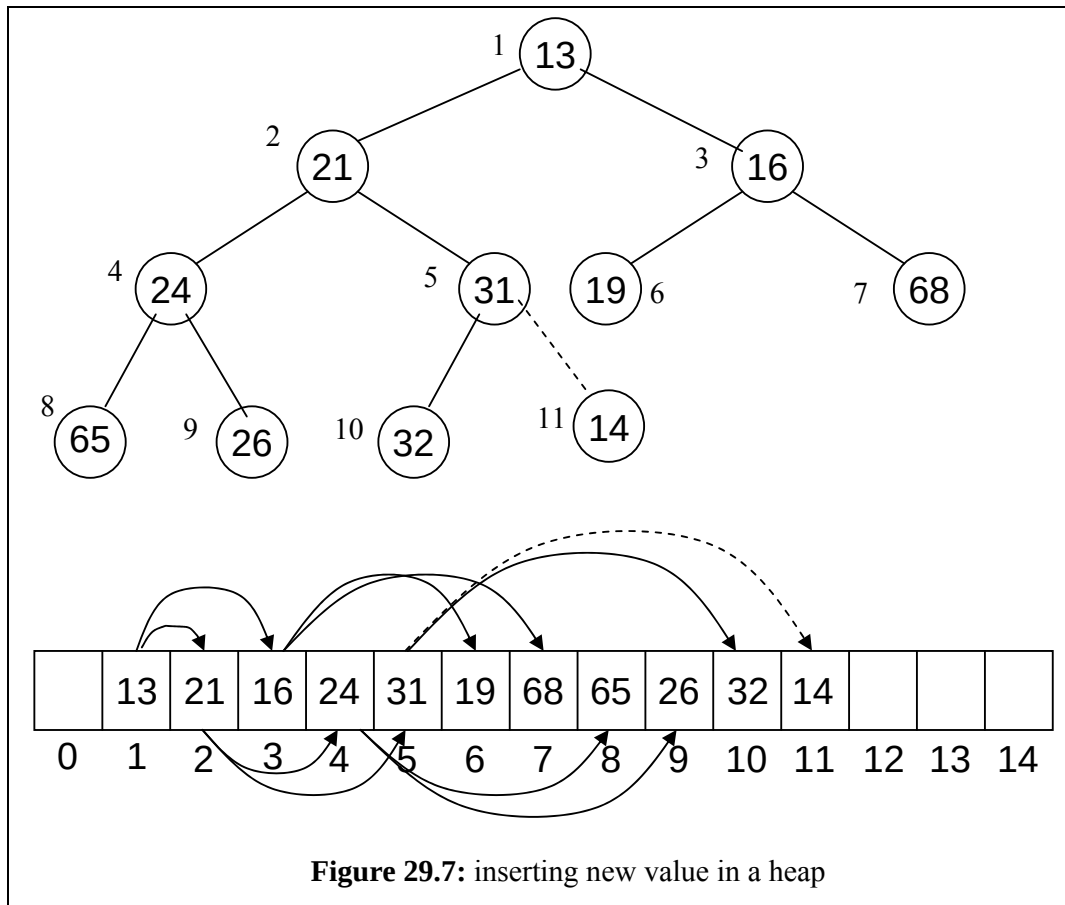
Now suppose that the numbers in heap are priorities or events. If we build the max heap, the number with greater priority will get on the top. Now if we get data from the heap, the data on top will be gotten first. As it is max heap, the data on top will be the largest value.

Insertion in a Heap

Now let's discuss the insertion of a value in a min or max heap. We insert a value in a min heap and rearrange it in such a way that the smallest value gets on the top. Similarly, if we are inserting a value in a max heap, the largest value goes on the top. To discuss insertion in min heap, consider the following existing heap. This is the same heap discussed earlier. As it is a complete binary tree, we keep it in an array. In the figure, the level order of numbers of the nodes is also seen that describes the position of nodes (index number) in the array.

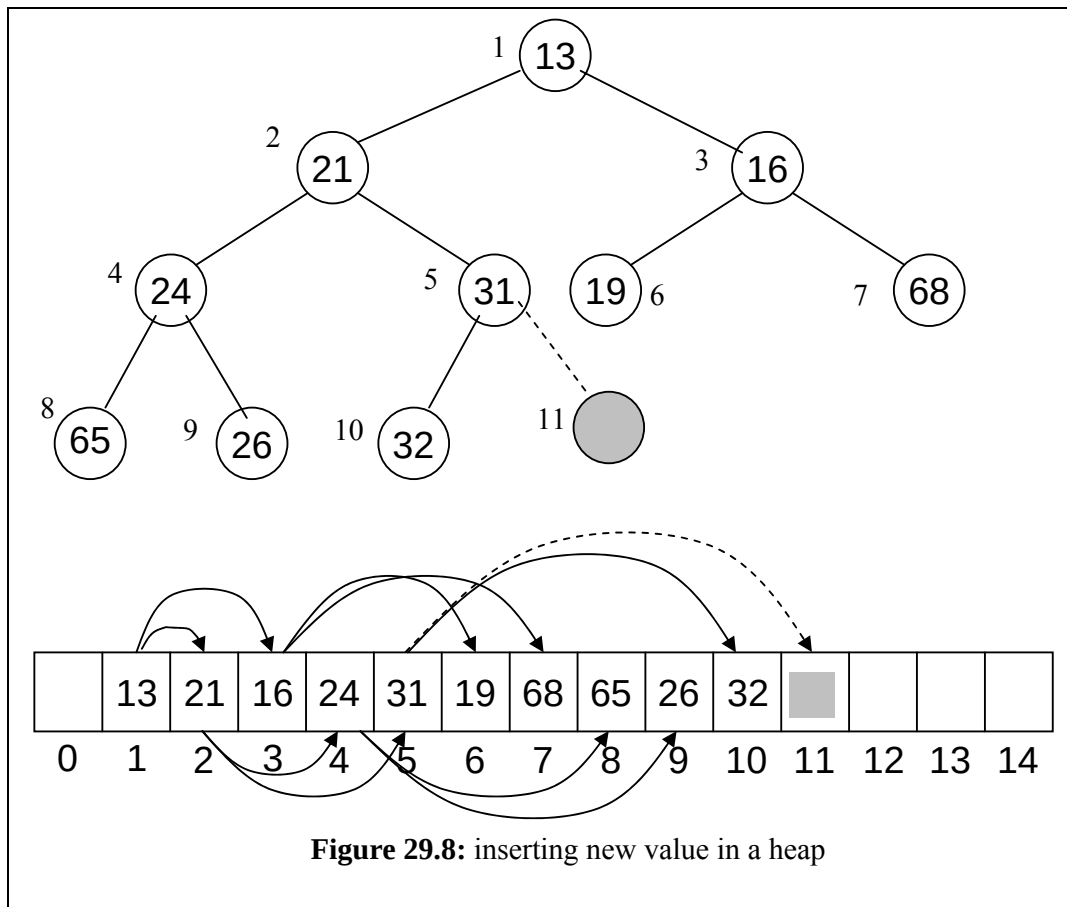


Now we add a new node to this existing heap. Suppose the node that we want to add has a value 14. We will add this node at a position in such a way that the tree should remain complete binary one. Following this guideline, this node will be added on right side of node 31. In the array, the node 31 is presently at position 5. Now according to the formula of $2i$ and $2i+1$, the left child of 31 will be at positions 10 that already exists. The right child of 31 (that is 14 now) will be at position 11 in the array. This addition of node in the tree and value in the array is shown in the following figure. We have shown the node link and position in the array with dotted lines. By this, it is shown that the node of value 14 may become here. We did not actually put it there. We have to put it at such position so that the heap should remain intact.

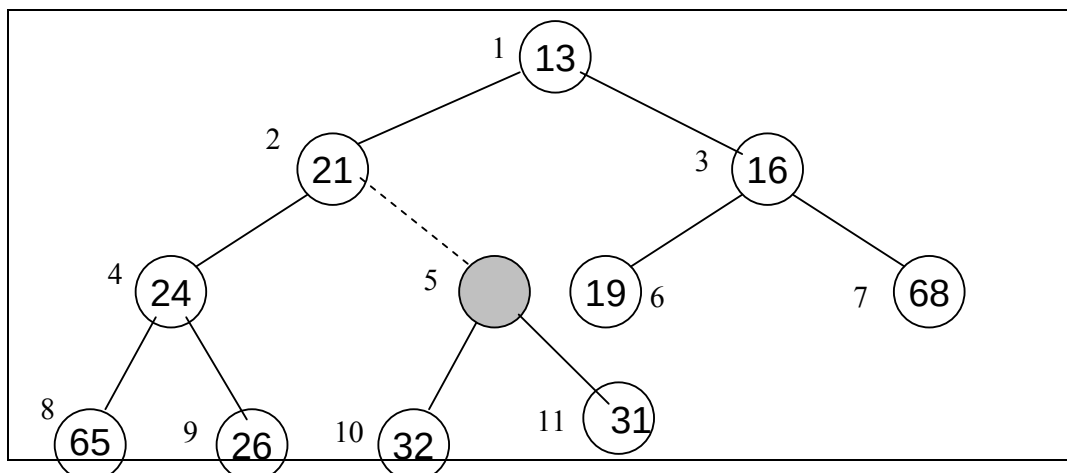


With the addition of new node, we see that this is still a complete binary tree and can be stored in an array. There are no holes in the array, leading to no wastage of the positions.

Now preservation of the heap order is needed. We are discussing the min heap in this example. In the min heap, the smallest element is at the top (root). And at each level, every node has a smaller value than that of its children nodes. Now we have to put the node 14 at its proper position with respect to heap order. Thus heap is being implemented with complete binary tree with the help of an array. We have to find the final position of node 14 in the tree and array preserving the heap order. A look on the tree in the previous figure shows that if we put node 14 (shown with dotted line) at that position, the heap order will not be preserved. By considering the node 31, we see that the value of its right child (i.e. 14) is less than it. Thus heap property is violated. To preserve heap property, the node 14 should be up and node 31 should be down. If node 14 is at position of 31 and it is compared with its parent that is 21, we come to know that node 21 should also be at down position. How do we take node 14 to its proper position? To achieve this objective, we have to take node 21 and 31 down and move the node 14 up. Now let's see how we can do this. In the following figure, the position is seen where a new node can be added with an empty circle (hole). The value of this node is not shown. However, the new inserted value may go to that position. Similarly in the array we show that there will be some value at position 11.



Now we compare the new value to be inserted (that is 14) at position 11 with its parent node. The parent node of newly inserted node is 31 that is greater than the new value. This is against the (min) heap property. According to the heap property, the node 14 may be the parent of 31 but not its child. So we move the node 31 to the position 11 and the new node to the position 5 that was earlier the position of 31. This technique is also employed in the array. Thus we get the array and tree in the form as shown in the following figure.



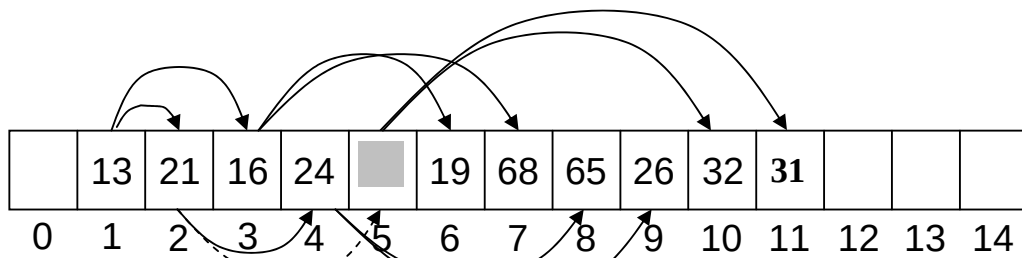


Figure 29.9: inserting new value in a heap

Now if the new node comes at the position 5, its parent (that is node 21 at position 2) is again greater than it. This again violates the heap property because the parent (i.e. 21) is greater than the child whose value is 14. So we bring the node 21 down and take the new node up. Thus the node 21 goes to the position 5 and the new node attains the position 2 in the tree and array as shown in the following figure.

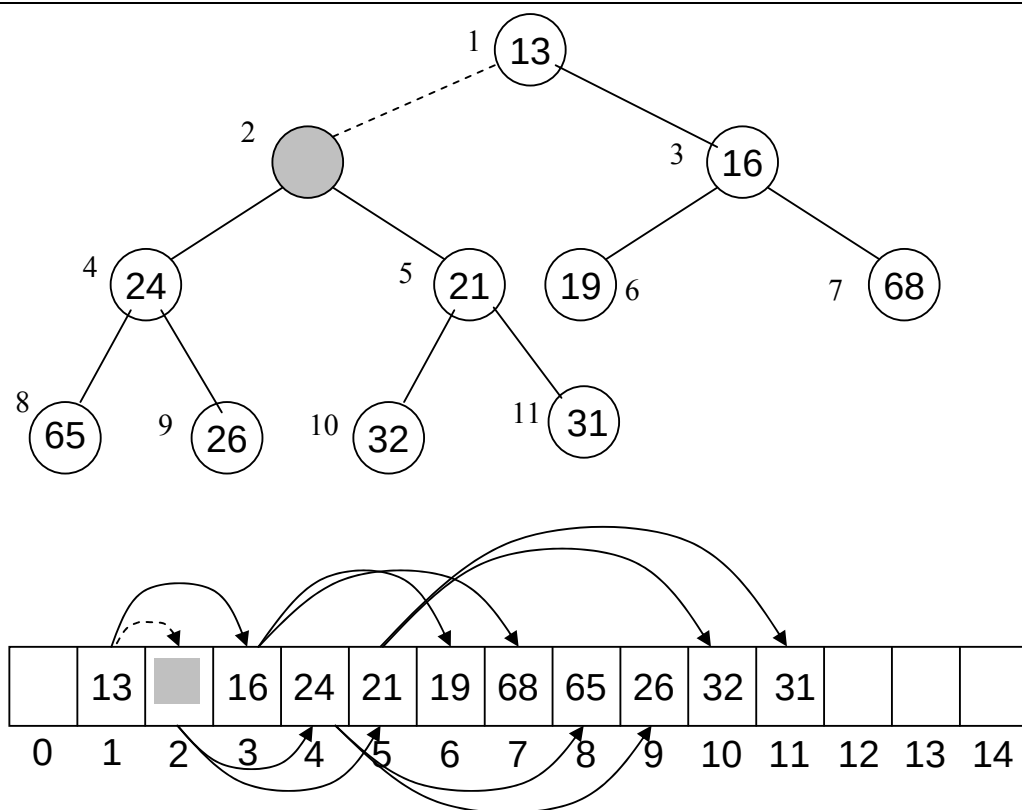
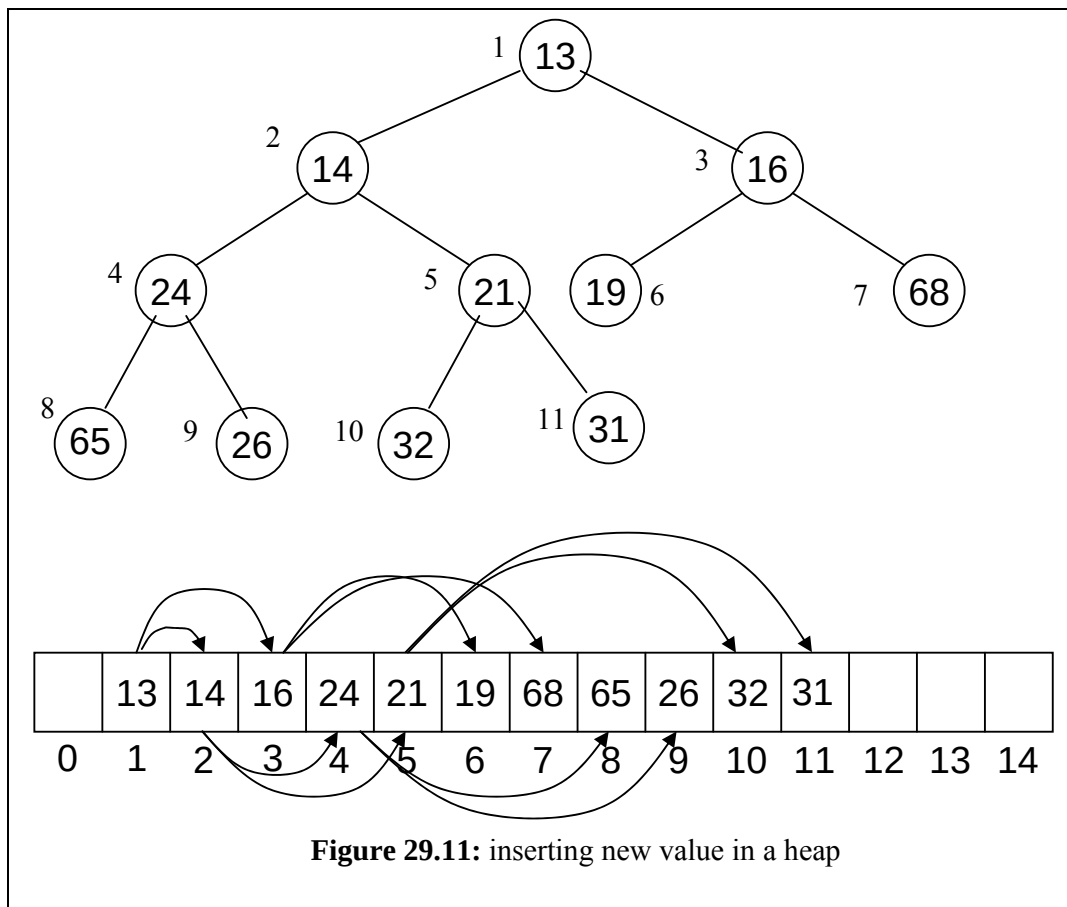


Figure 29.10: inserting new value in a heap

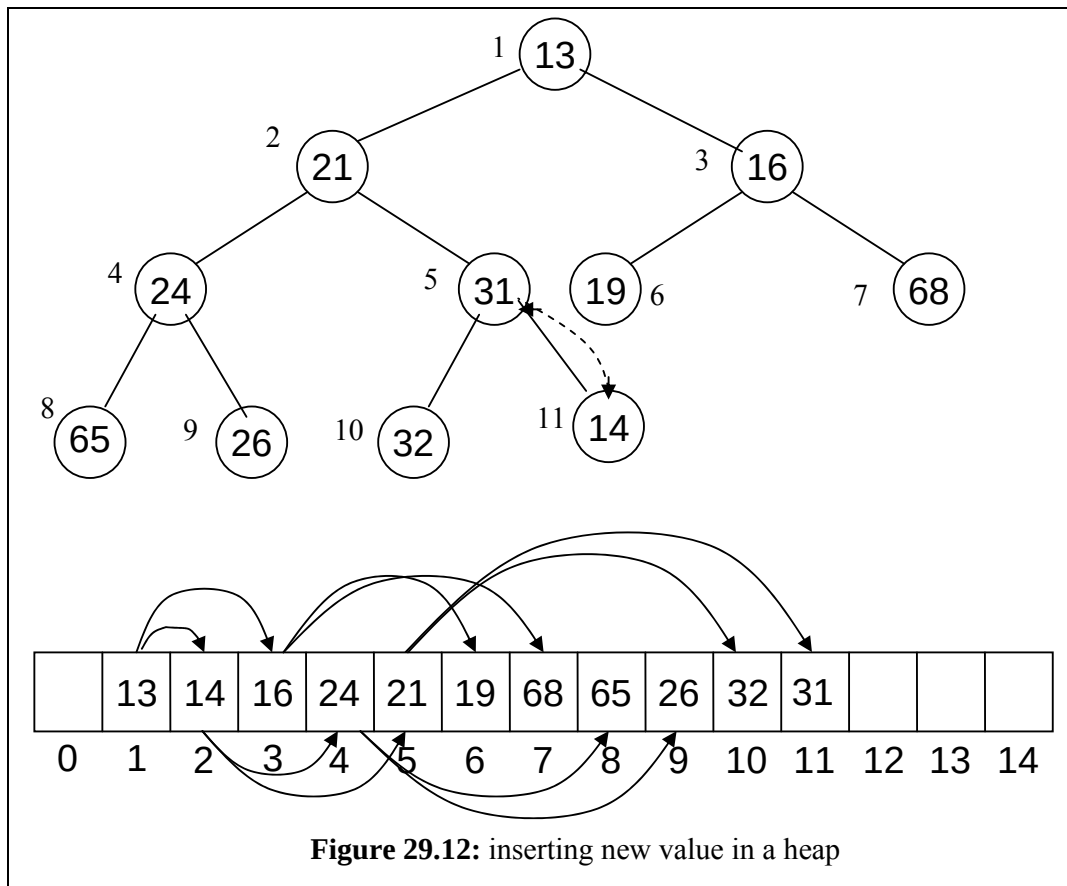
To interchange the positions of values in the array, only these values are swapped which is very easy process. We can do this easily in arrays. Now if the new value 14 comes at the position 2, the heap property will be preserved due to the fact that the

parent of this node i.e. the node of value 13 is less than its child (the new node that is 14). Thus the final position of new node 14 is determined and we put it here i.e. at position 2. This is shown in the following figure.

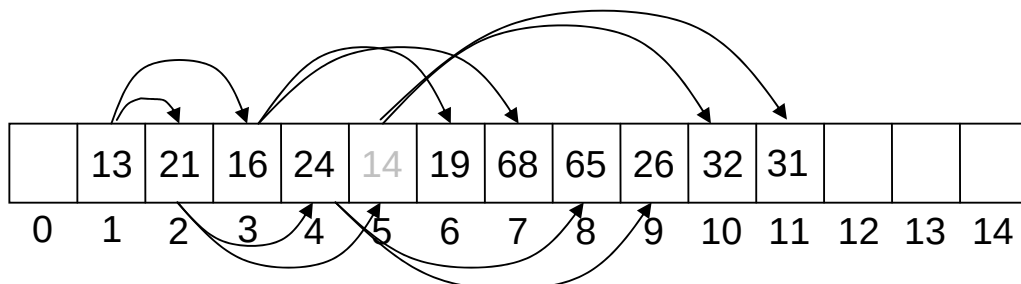


It is clear by now that the tree after insertion of a new value follows the heap property. We see that there is an algorithm for inserting a new value in the heap. If we have a heap stored in an array as a complete binary tree, a new value is put in the array at a position so that it can hold preserving the property of complete binary tree. In our example, it is the position 11 where new value may be put. We know that the parent of a node is at position $\text{floor}(i / 2)$. Thus we find the position of parent node of the new node and compare this new value with that. If the parent node has a value greater than this new value (i.e. its child), the heap property is violated. To maintain this property, we exchange these values. Now at the new position we find the parent of that position and compare the value with the value at that position. Thus the array is traversed by level-order. After a comparison, we go one level up in the tree. And we go to upward by looking at the parent node of the newly inserted node level by level. We stop at the position where the value of the parent node is less than the value of its child i.e. the node to be inserted. To insert a node, it is necessary to find the position of the node before putting the value there. This is efficient and fast way as actual values are not exchange in this case. We simply move the data. Under the second method, it can also be done with exchanges. Let's do this for our previous heap. Look at the following figure. Here we put the new node 14 at its position. It is the right child of node 31. We have already seen that this is the position where a new node can

be added so that the tree remains complete binary tree. In the array, it is at position 11.

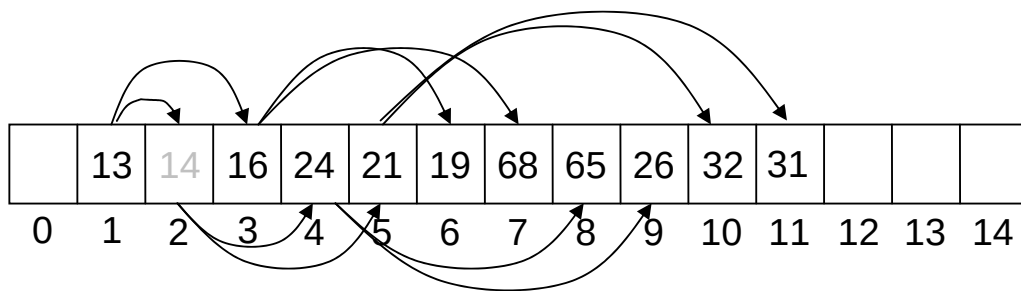


Now we check the heap order and compare node 14 with its parent that is node 31. It is seen that this child node i.e. 14 is less than its parent i.e. 31. Thus the heap order property is violated. We exchange the node 14 and 31 due to which the node 14 becomes the parent and node 31 turns into its child. That is why, the node 31 is now at position 11 and node 14 is at position 5. After this exchange, the tree remains complete binary one as we only have exchanged the values of nodes. The following array representation shows this.

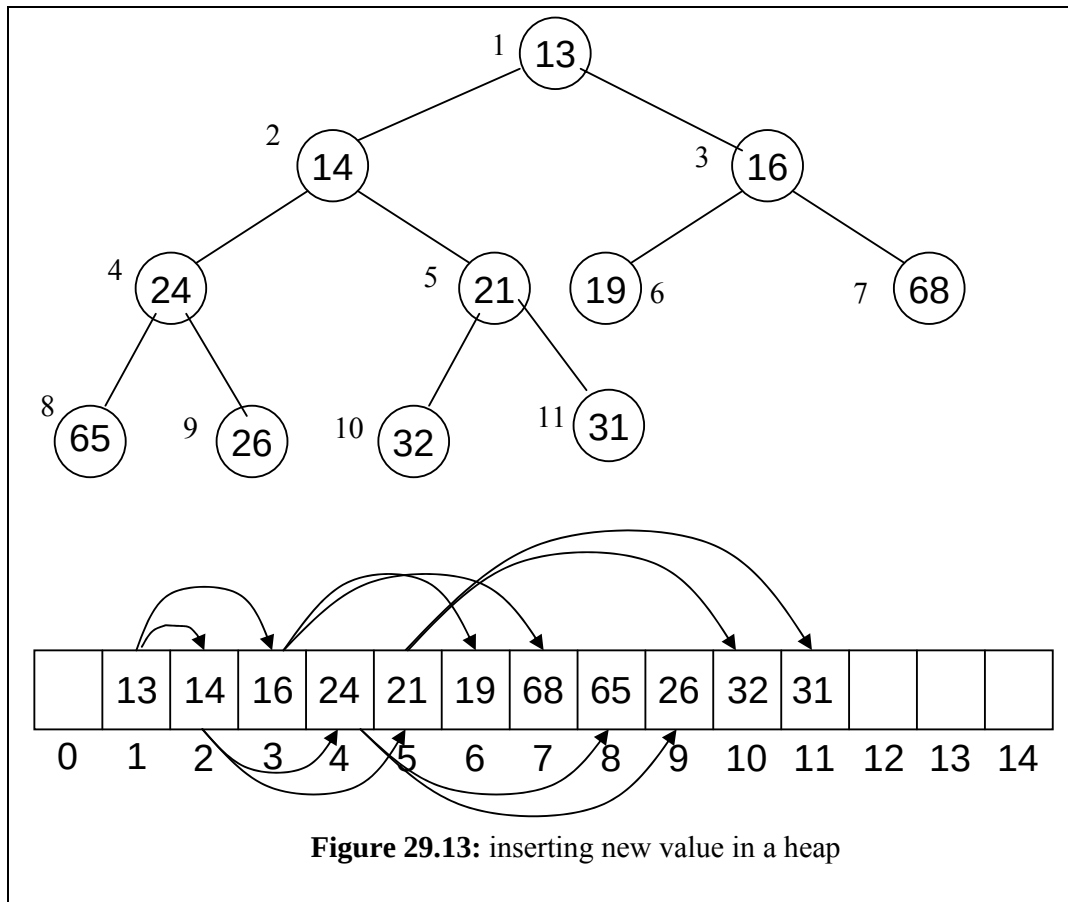


After this we compare the node 14 with its new parent. The new parent of node 14 can be found by the formula of $\text{floor}(i/2)$. The parent node of 14 will be at position $\text{floor}(5/2)$ that is position 2. We can see that the node at position 2 is 16. Thus 16 is greater than its child i.e. 14, violating the heap property. So we again exchange these values. The node 14 now becomes the parent of 16 and 16 gets its child. In the array, the

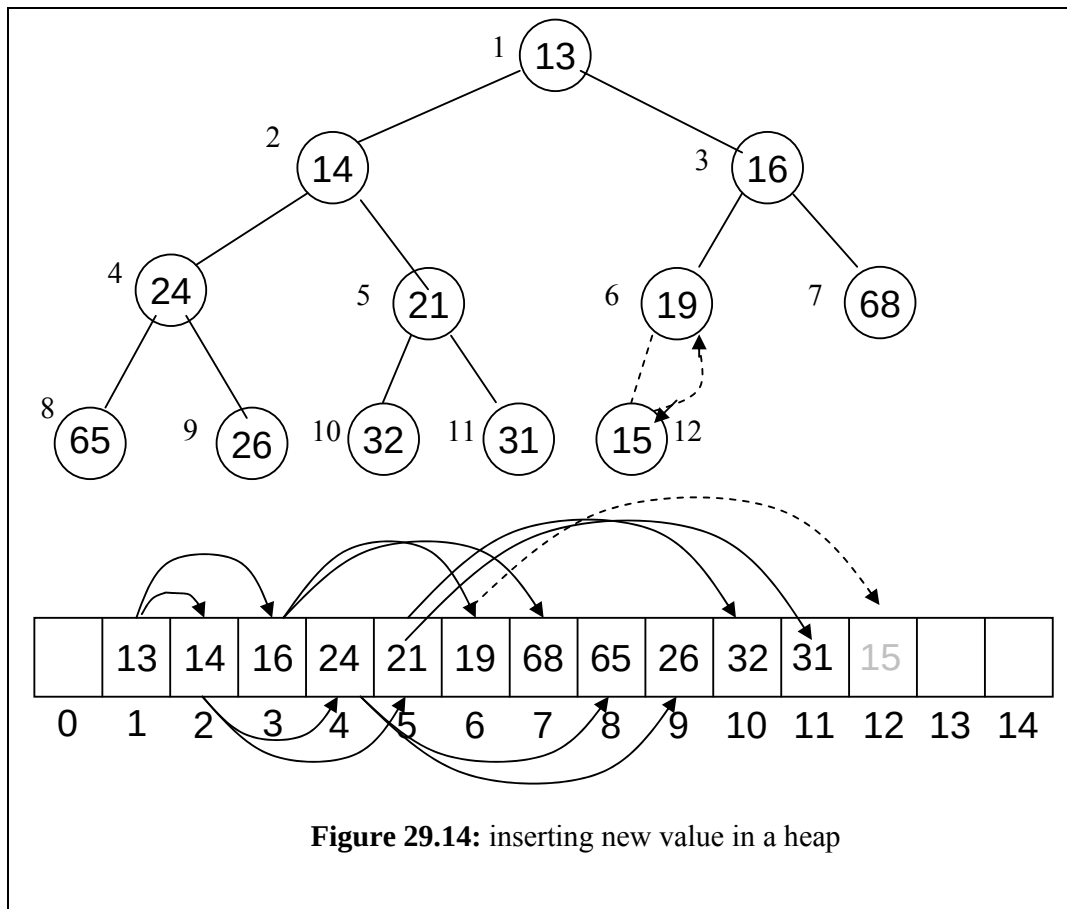
nodes 14 and 21 are at positions 2 and 5 respectively. The array representation of it is as below.



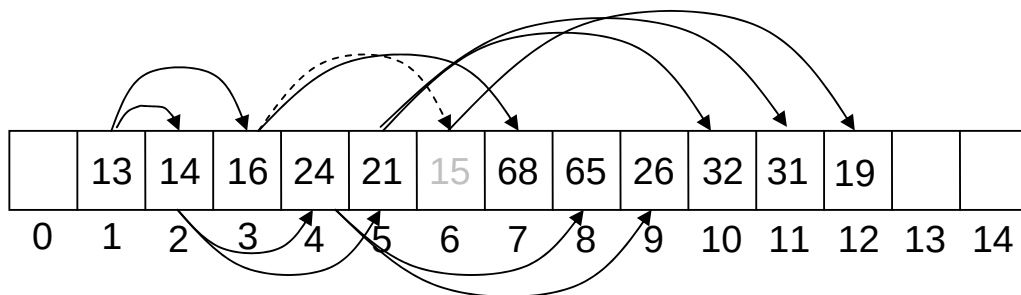
Now we compare this node 14 with its new parent i.e. 13. Here the heap property stands preserved due to the fact that the parent node i.e. 13 is less than its child node i.e. 14. So this tree is a heap now. The following figure shows this heap and array representation of it.



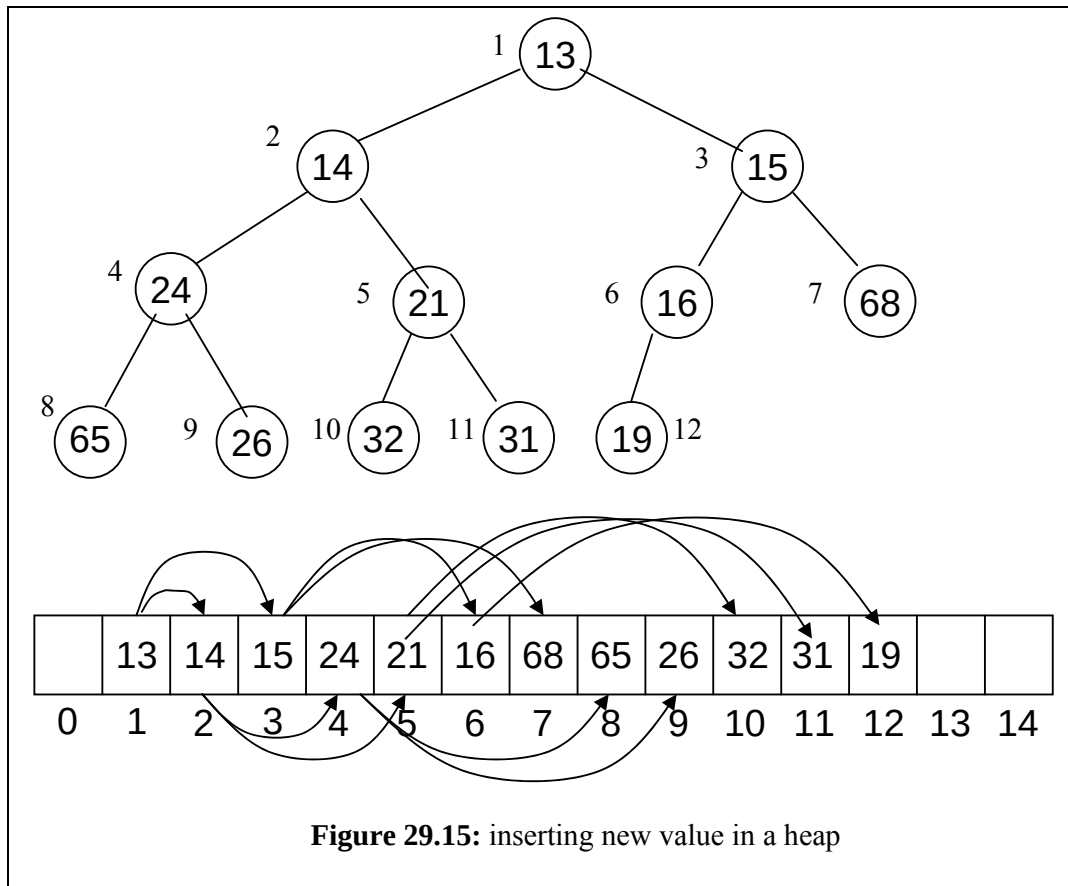
Suppose we want to add another node to this heap. This new node has value 15. As the heap is a complete binary tree, so this new node will be added as left child of node 19. In the array, we see that the position of 19 is 6 so the position of its left child will be (by formula of $2i$) 6×2 that is 12. This is shown in the following figure.



Here the new node 15 is less than its parent node (i.e. 19). To preserve the heap property, we have to exchange these values. Thus the value 15 goes to the position 6 and value 19 attains the position 12 as shown in the array in figure below.



Now we compare the value 15 with its new parent i.e. 16. Here again the parent (16) is greater than its child (15). So to preserve the heap property we need to exchange these values. After the exchange, the value 15 is now at position 3 and value 16 is seen position 6. The following figure shows this step.



The figure also shows that the node 15 is greater than its new parent i.e. 13. In other words, the parent node is less than its child. So the heap property has been preserved, warranting no exchange of values. The node 15 has gone to its proper position. All the nodes in this tree now follow the heap order. Thus it is a heap in which a new node has been inserted.

Data Structures

Lecture No. 30

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 6

6.3

Summary

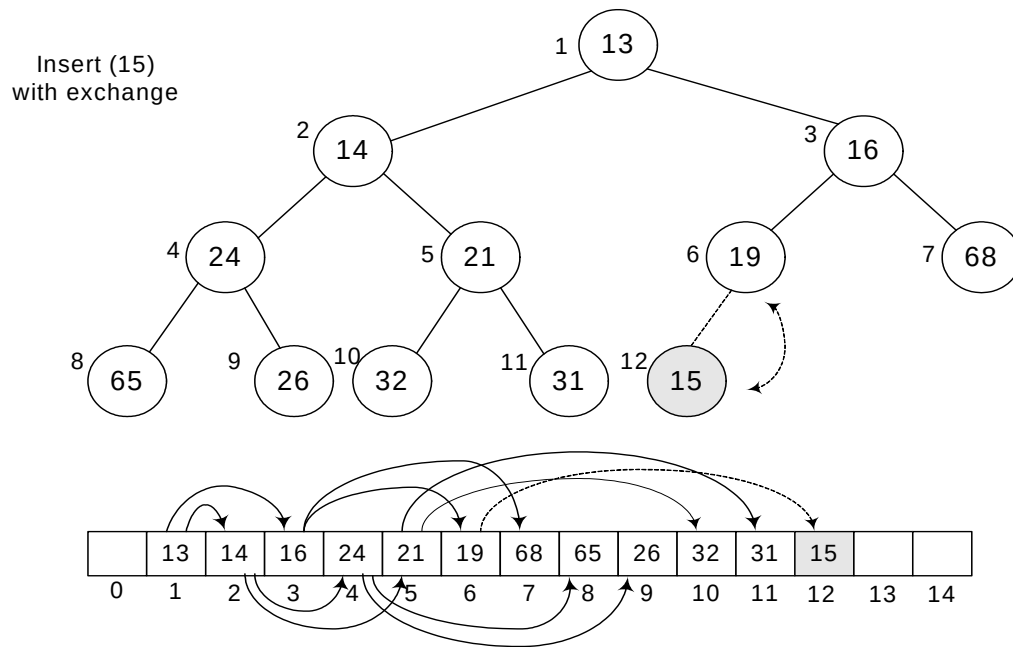
- Inserting into a Min-Heap
- Deleting from a Min-Heap
- Building a Heap

Inserting into a Min-Heap

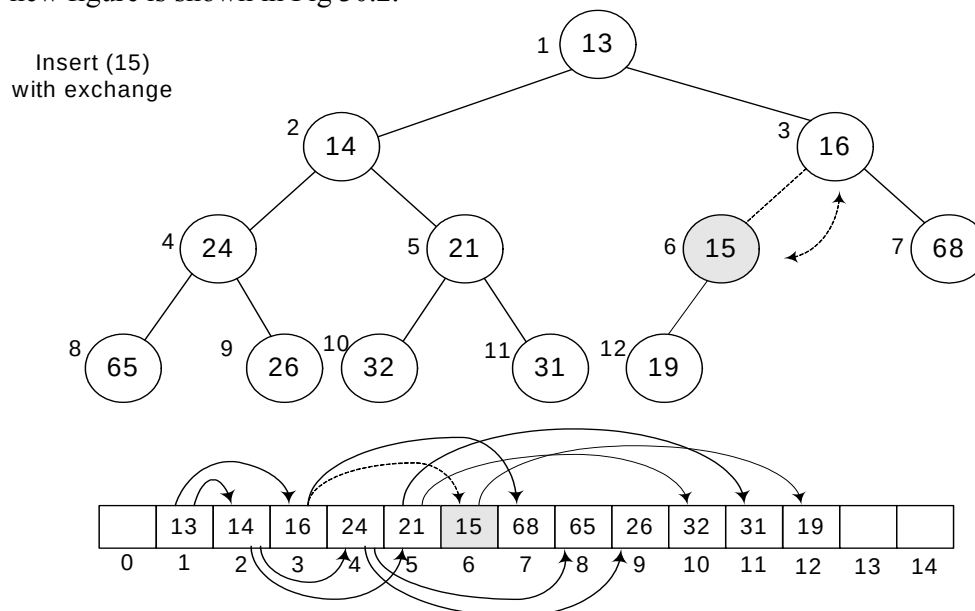
In the previous lecture, we discussed about the heap data structure besides touching upon the concept of storage of complete binary tree in an array. For the study of parent-child relationship, the ' $2i$ and $2i+1$ scheme' was used. Then we changed our focus to heap and found it best for the priority queues. In the previous lecture, we did not really discuss the uses of heap. Rather, most of the discussion remained focused on insertion method in the binary tree employed at the time of implementation with the help of an array. After inserting a new element in the array and by moving few elements, a programmer can have minimum or maximum heap. In case of minimum heap, the minimum value in the tree is always in the *root* of the tree. However, in case of maximum heap, the maximum value in the tree lies in the *root* node.

When we insert a new element in the tree implemented with the help of an array, we insert element at the last position (of the array). Due to this insertion at the end, the heap order may be violated. Therefore, we start moving this element upwards. While moving upward, this element may reach at the *root* of the tree. It is important to note that only one branch of the tree is affected because of this movement in the upward direction. This process, being of localized nature, will not disturb the entire tree.

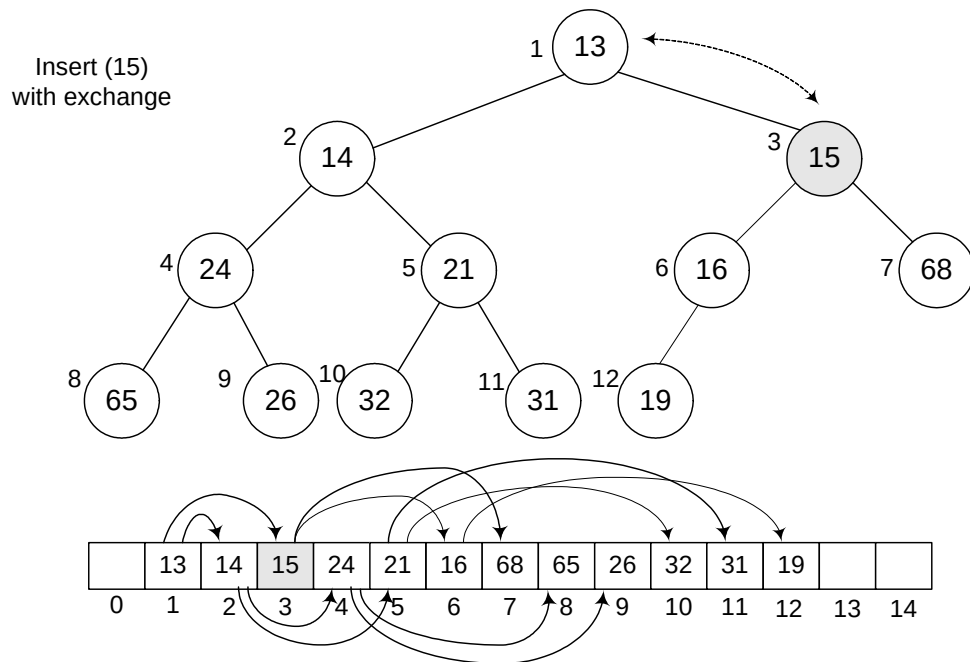
To recap, see the figure Fig 30.1, where we are inserting an element 15.

**Fig 30.1**

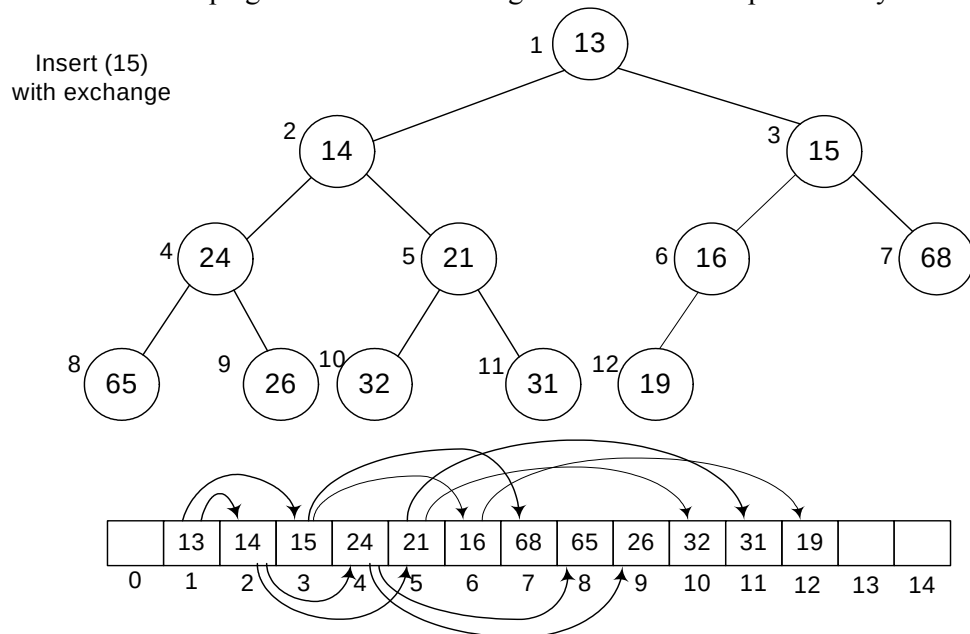
The new element 15 is inserted at the last array position 12. Being a complete binary tree, the next new node will be the left child of node 19. As node 19 is at array position 6 (or level order traversal), its left child will be $6 * 2 = 12^{\text{th}}$ position node or the value at 12^{th} position in the array. Now, we see where we will have to carry out the exchange operation. As the parent of 15, the number 19 is greater than it, the first exchange will be among 19 and 15 as shown in the above figure. After exchange, the new figure is shown in Fig 30.2.

**Fig 30.2**

You can see that both the elements have exchanged positions i.e. 19 has come down and 15 gone up. But number 15 is still less than its parent 16, so we will have another exchange operation.



Now the new parent of 15 is 13 which is less than it. Therefore, the exchange operation will stop here as 15 has found its destination in the tree. This new tree is not violating any condition of heap order as witnessed before insertion of 15. It has become min-heap again while maintaining its status as a complete binary tree.



A view of the path shows that the number 15 has gone through to find its final destination in the tree. It has only affected the right sub-tree's left branch that was containing 16, 19 and 15. It has not affected any other branch. This is not a binary search tree. Here we are only fulfilling one condition that the parent value should be

less than that of its two-children. These exchanges will not take long as we did mathematically that a tree containing N nodes; can go to $\log_2 N$ level maximum. You know, when we built binary tree for balancing, it had turned into a linked list. Here the case is different. the number of levels in complete binary tree will be around $\log_2 N$ while building a complete binary tree.

Now, we should be clear that a new element can go up to what maximum level in the tree. If the number is greater than all parent nodes, it will be already at its destination place, needing no exchange. If we insert a number smaller than all parent nodes, the exchange operation reaches to the *root* of the tree. But this exchange operation's intensity does not increase from the one we saw in our previous case of linked lists.

Deleting from a Min-Heap (deleteMin)

Now, we will see how the delete operation is executed in the heap. We do a lot of insert and delete (removal from the data structure) operations especially when heap is used in the priority queue. Normally the delete operation is a bit complex. However, in this case, it is quite easy-to-do.

We want to write a function *deleteMin()*, which will find and delete the minimum number from the tree.

– *Finding the minimum is easy; it is at the top of the heap.*

See the heap below. It is a min-heap. Now the question arises where the minimum number will be lying. It is clear from the definition of min-heap that it should be in the *root* of the tree. So finding the minimum is very easy.

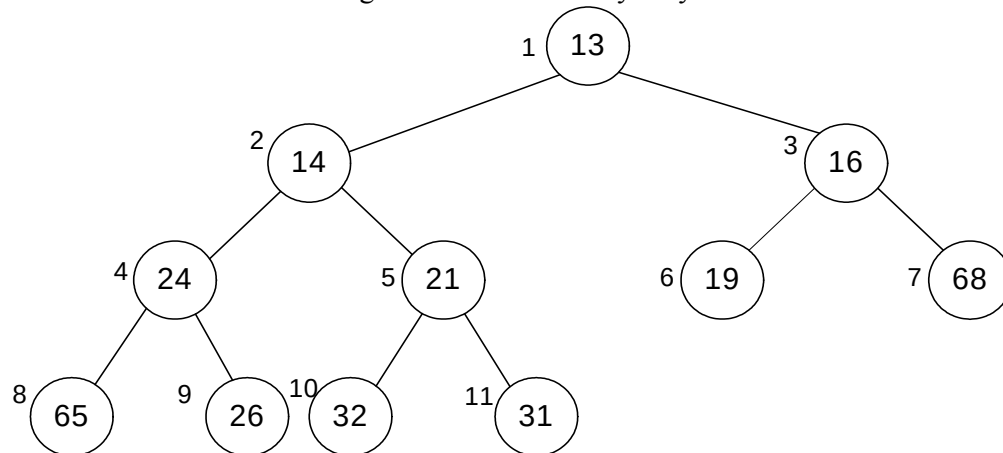
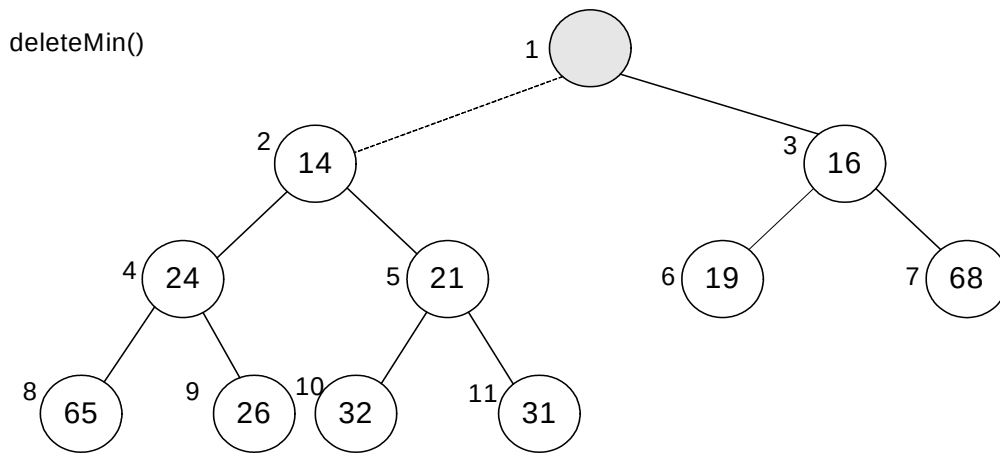


Fig 30.5

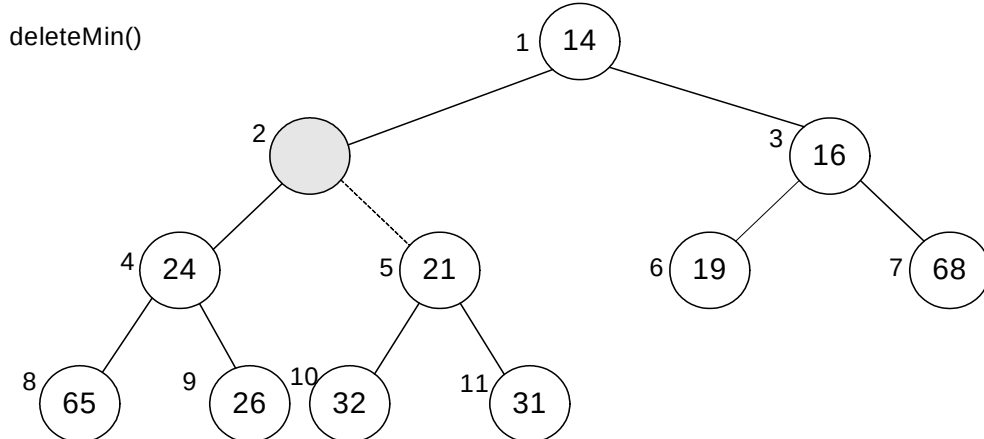
To understand it better, consider the tree form first, instead of thinking about the array. Remember, we have implemented the complete binary tree with the help of an array for the sake of efficiency. It is not necessary to implement it this way. We can also implement it with pointers. However, heap is normally implemented through array.

Coming back to the deletion, it is necessary to understand that

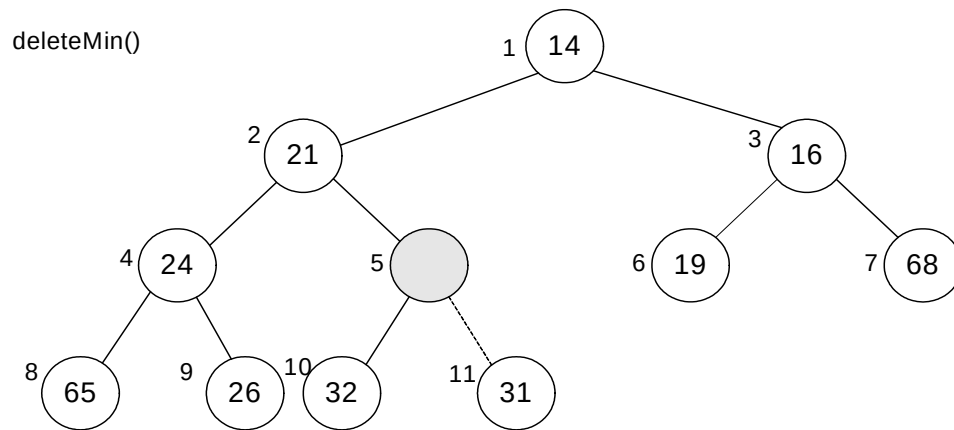
– *Deletion (or removal) causes a hole which needs to be filled.*

**Fig 30.6**

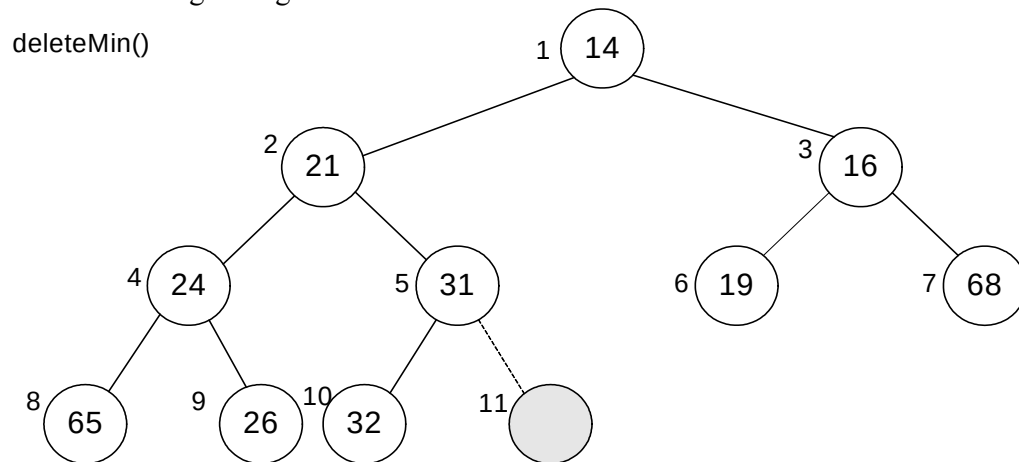
In the above figure, we have deleted the *root* node, resulting in a hole at the *root* position. To maintain it as a complete binary tree, we have to fill this hole. Now, we will identify the appropriate candidates to fill this hole. As the parent is the minimum as compared to its right and left children, the appropriate candidates are both right and left children of the *root* node. Both children will be compared and the smaller one will take the vacant place. In the above figure, the children 14 and 16 are candidates for the vacant position. Being the minimum, 14 will be put in the hole.

**Fig 30.7**

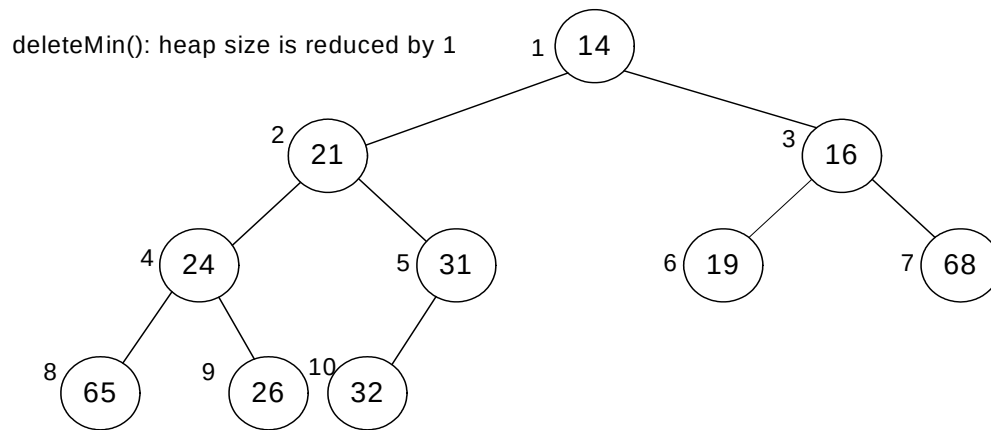
In the Fig 30.7, we can see that the 14 has been placed at the *root* position. However, the vacuum at the previous position of 14 has created a hole. To fill this hole, the same logic is applied i.e. we compare both right and left children of the hole and let the minimum take over the place. Here 24 and 21 are compared. Being the smaller one, 21 is placed at the vacant position. This way we get the latest figure of the tree as seen in the following figure. (Fig 30.8.)

**Fig 30.8**

The hole is now created at previous position of 21. This time, the children- 32 and 31 are compared and being smaller, 31 takes over the place and the tree becomes as shown in the figure Fig 30.9.

**Fig 30.9**

The hole has been transferred from top to the bottom. It has reached at such a position where it can be deleted.

**Fig 30.10**

While using array to store complete binary tree, we can free array element at the end. From the above figure Fig 30.10, the last array index 11 is no more there and the node has been deleted.

We saw that the data element was inserted at the bottom of the tree and moved upwards while comparing it with the parents (following the $i/2$ scheme) until its destination was found. Similarly, when we deleted a node, the hole was produced. While following the definition of min-heap, the hole kept on moving from top to the bottom (following the $2i$ or $2i+1$ scheme).

Building a Heap (buildHeap)

Now, we will see how a heap can be made and in which peculiar conditions, it should be built. We may have the data (for example, numbers) on hand, needed to be used to construct the tree. These data elements may be acquired one by one to construct the tree. A programmer may face either the situations. If we consider priority queue, it is normally empty initially. Events are inserted in the queue as these are received. You can also consider priority queue in another application where data can be inserted into the queue or taken out at one time.

Let's say we have a large unsorted list of names and want to sort it. This list can be sorted with the help of the heap sort algorithm. When we construct the heap or complete binary tree of the list, the smallest name (considering alphabet 'a' as the smallest) in the list will take the *root* node place in the tree. The remaining names will take their places in the nodes below the *root* node according to their order. Remember, we are not constructing binary search tree but a min-heap to sort the data. After the construction of the min-heap from all the names in the list, we start taking the elements out (deleting) from the heap in order to retrieve the sorted data. The first element that is taken out would be the smallest name in the heap. The remaining heap after taking out the deleted node will be reduced by one element in size and the next name will be at the root position. It is taken out to further reduce the tree size by one. This way, the continuation of the process of taking out the elements from the heap will ultimately lead to a situation when there is no more node left in the tree.

While keeping in view the time consumption factor, can we find a better way than this? We may change the data structure or algorithm for that. There are data structures which are more efficient than heap. But at the moment, we are focusing the heap data structure. We will see, what we can do algorithmically to improve the performance, having all the data simultaneously to construct the tree.

Following are some of the important facts about building a heap.

- Suppose we are given as input N keys (or items) to build a heap of the keys.
- Obviously, this can be done with N successive inserts.
- Each call to insert will either take unit time (leaf node) or $\log_2 N$ (if new key percolates all the way up to the root).
- The worst time for building a heap of N keys could be $N \log_2 N$.
- It turns out that we can build a heap in linear time.
- Suppose we have a method *percolateDown*(p) that moves down the key in node p downwards.
- This is what happens in *deleteMin*. We have used the word *keys* in the first point. This is the same concept as that of the database keys, used to identify information uniquely. Using the example of telephone directory (also discussed in a number of previous lectures), we see that it contains the person name (first, middle and last), address (street address, district and postal code) and telephone number. All these information together form a data record. This set of information fields (record) will always be together. The key item here can be the unique name of the person or the telephone number.

Consider that we have N number of items i.e. names. One way is to call insert for every element one by one. For N elements, it will be N number of calls obviously. Being an iterative process, this insertion technique can be executed in a loop fashion.

Let's talk about the time taken to insert a node in the tree. It depends on the value to be inserted and the values already present in the tree. The number of exchanges can be variable. Therefore, the insertion time is also variable. Each call will be a unit time (in

case of leaf node) or $\log_2 N$ (if new key percolates all the way up to the root node). For the whole tree, the worst time for N keys would be $N \log_2 N$.

Next point tells about the possibility of reducing the time from $N \log_2 N$ and it can turn out to be a linear time. This is our target and we want to achieve it. Let's talk about the ways to achieve the linear time.

The word *percolate* above means something is going upward from downward. This terminology is coming from coffee maker, where the hot water moves upwards. This phenomenon happens at the time of insertion. On the contrary, in delete operation, the hole moves towards the bottom of the tree. You must be remembering that we discussed about insertion in the heap without making exchanges and by finding the destination position first before making any change. Similarly in case of deletion, the hole moves downward while the data moves upwards. Let's write a method for the delete operation *percolateDown(p)*. This method will find which child of the hole will be moved upwards and the movement of hole to the bottom of the tree.

We will see, using this method, how can we build a heap that will take lesser time than $N \log_2 N$.

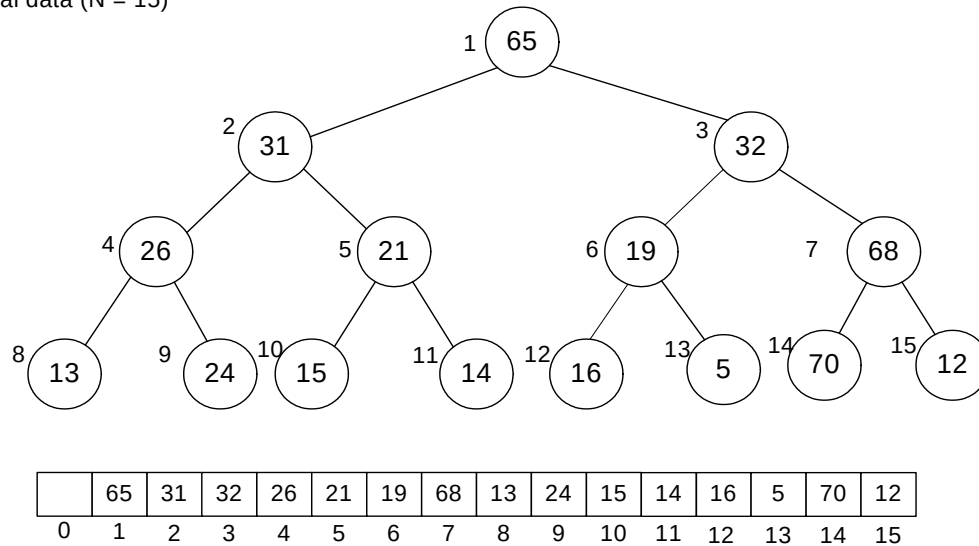
Initial data ($N = 15$)

	65	31	32	26	21	19	68	13	24	15	14	16	5	70	12
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig 30.11

As shown in the figure Fig 30.11, we are given a list of 15 elements and asked to construct a heap of them. The numbers are stored in an array, starting from index (position) 1. This start of array from position 1 is to make the tree construction easier. It will be clear in a moment. Contrary to the previous situation when array was used to store heap or complete binary tree, we should now think what would be the picture of complete binary tree out of it. It may seem complex. But actually it is quite easy. You start from the very first element in the array i.e. the *root* of the tree. The children of root are present at $2i$ and $2i+1$, which in this case, are at positions $2(1) = 2$ and $2(1)+1=3$. The children nodes for the node at position 2 are at positions $2(2)=4$ and $2(2)+1=5$. Similarly, the children nodes for the node at position 3 are at positions 6 and 7. Apply this logic to the whole of this array and try to construct a tree yourself. You should build a tree as given in the figure Fig 30.12.

Initial data (N = 15)

**Fig 30.12**

Is this tree binary one? Yes, every node has only two left and right children.

Is it a complete binary tree? It surely is as there is no node that has missing left or right child.

The next question is; is it a min-heap. By looking at the root node and its children, as the root node is containing 65 and the children nodes are containing 31 and 32, you can abruptly say that no, it is not a min-heap.

How can we make min-heap out of it? We may look for *percolateDown(p)*.method to convert it into a min-heap. As discussed above, the *percolateDown(p)*.moves the node with value *p* down to its destination in the min-heap tree orders. The destination of a node, can be its next level or maximum the bottom of the tree (the leaf node). The node will come down to its true position (destination) as per min-heap order and other nodes (the other data) will be automatically moving in the upward direction.

- The general algorithm is to place the *N* keys in an array and consider it to be an unordered binary tree.
- The following algorithm will build a heap out of *N* keys.
for (*i* = *N*/2; *i* > 0; *i*--)
 percolateDown(i);

A close look on the above loop shows that it is starts from *N*/2 position and goes down to 0. The loop will be terminated, once the position 0 is reached. Inside, this loop is a single function call *percolateDown(i)*. We will try to understand this loop using figures.

$$i = 15/2 = 7$$

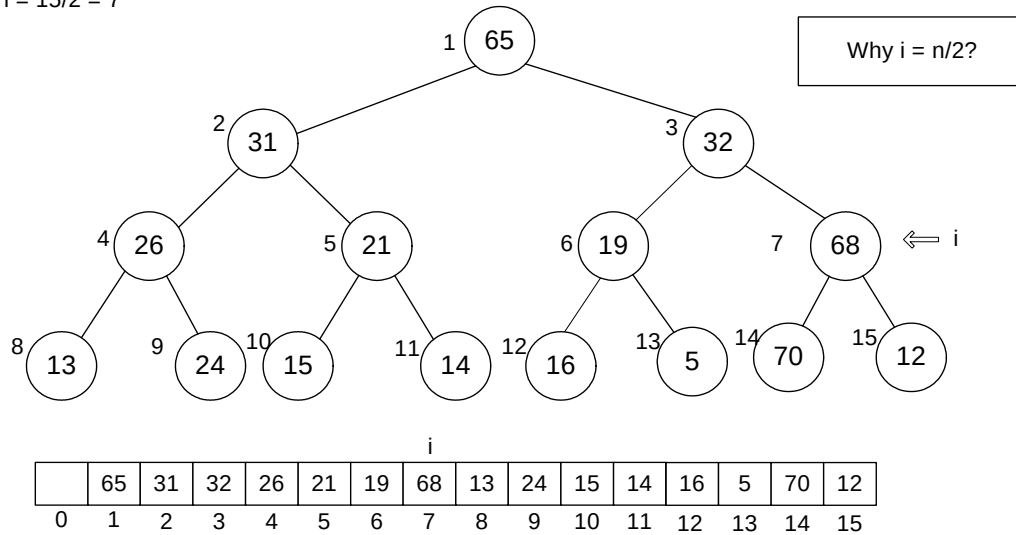


Fig 30.13

You can see the top left of the figure Fig 30.13. It is given $i = 15/2 = 7$. This is the initial value of i . The value in the array at position 7 is 68. In our discussion, we will interchangeably discuss array and binary tree. The facts arising in the discussion would be applicable to both. Considering this position 7, we will try to build min-heap below that. We know that for position 7, we have children at positions $2(7)=14$ and $2(7)+1=15$. So as children of the number 68 (which is the value at position 7), we have 70 and 12. After applying the *percolateDown(i)*, we get the tree as shown in Fig 30.14.

$$i = 15/2 = 7$$

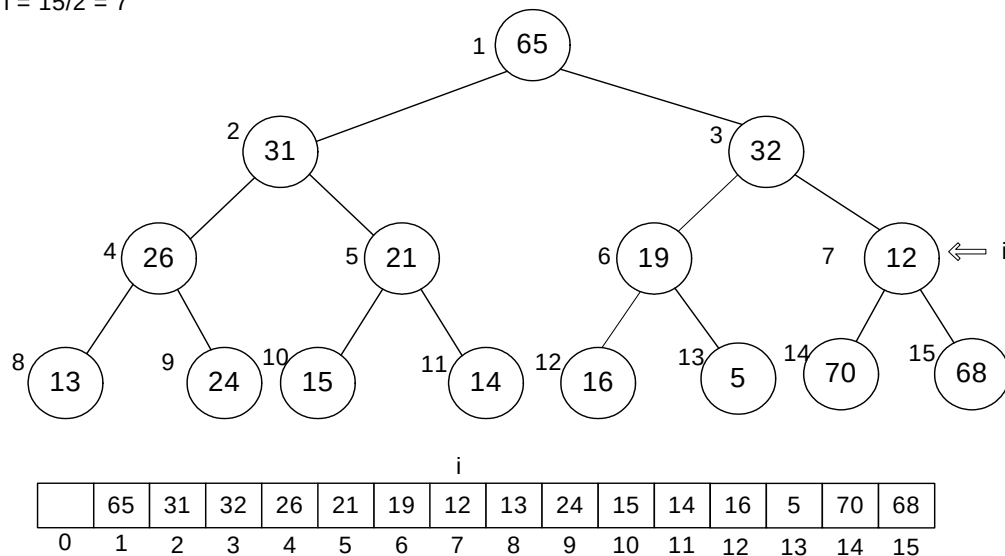
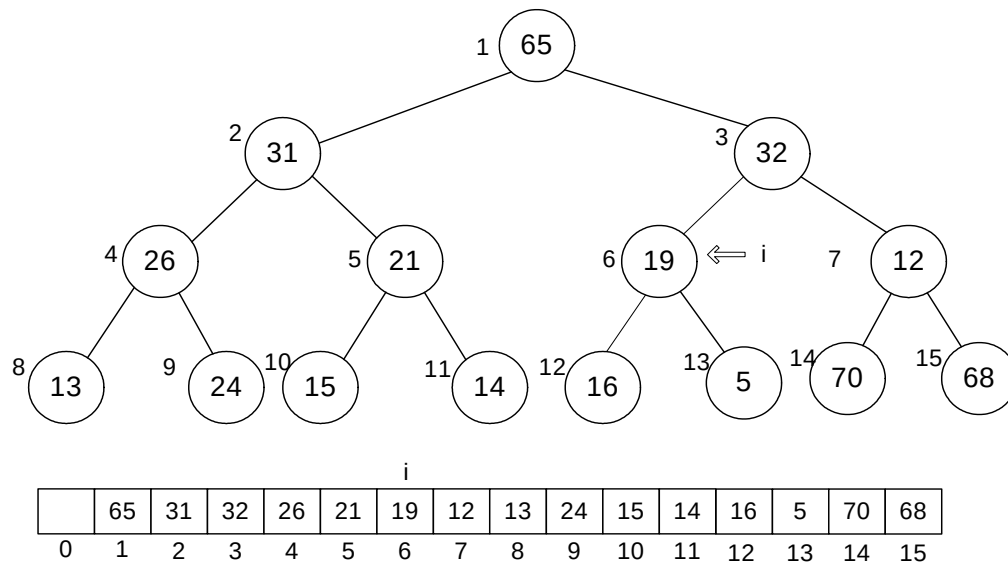
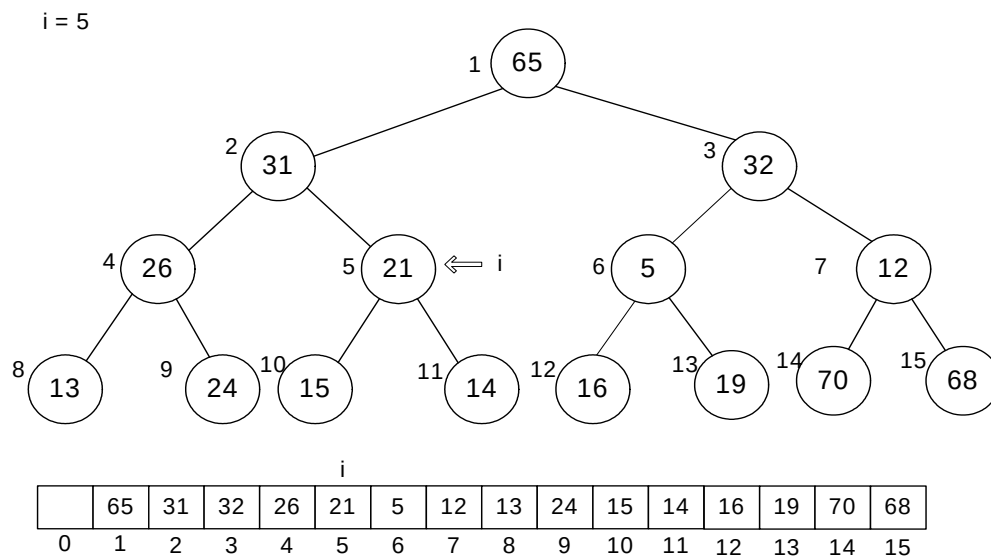


Fig 30.14

You can see in the figure that 12 has moved upward and 68 has gone down. Now, what about this little tree starting from 12 and below it is min-heap. Surely, it is. Next, we go for second iteration in the loop, the value of i is decremented and it becomes 6.

**Fig 30.15**

The node at position 6 is 19. Here the method *percolateDown(i)* is applied and we get the latest tree as shown in Fig 30.16.

**Fig 30.16**

The node 5 has come upward while the node 19 has moved downward. Its value of has decremented. It is now ready for next iteration. If we see the positions of *i* in the tree, we can see that it is traveling in one level of the tree, which is the second last level.

The question might have already arisen in your minds that why did we start *i* by $N/2$ instead of N . You think about it and try to find the answer. The answer will be given in the next lecture. We will continue with this example in the next lecture. You are strongly encouraged to complete it yourself before the next lecture.

Data Structures

Lecture No. 31

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 6
6.3.3, 6.3.4

Summary

- BuildHeap
- Other Heap methods
- C++ Code

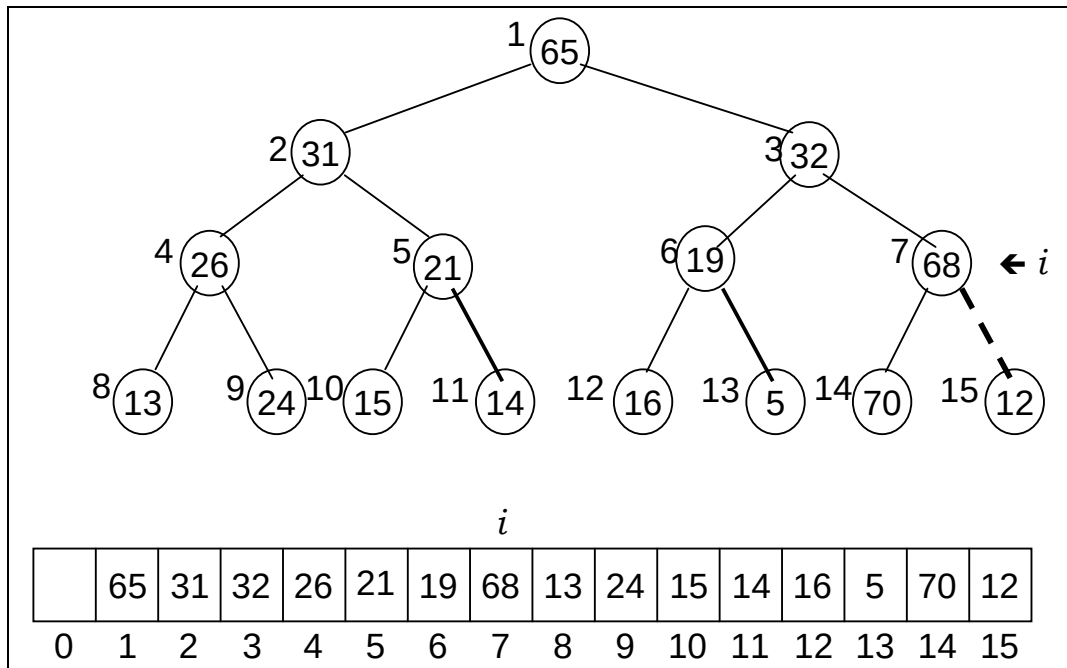
BuildHeap

In the previous lecture, we discussed about the *BuildHeap* method of heap abstract data structure. In this handout, you are going to know why a programmer adopts this method. Suppose we have some data that can be numbers or characters or in some other form and want to build a *min-Heap* or *max-Heap* out of it. One way is to use the *insert()* method to build the heap by inserting elements one by one. In this method, the heap property will be maintained. However, the analysis shows that it is $N\log N$ algorithm i.e. the time required for this will be proportional to $N\log N$. Secondly, if we have all the data ready, then it will be better to build a heap at once as it is a better option than $N\log N$.

In the delete procedure, when we delete the root, a new element takes the place of root. In case of *min-heap*, the minimum value will come up and the larger elements will move downwards. In this regard, percolate procedures may be of great help. The *percolateDown* procedure will move the smaller value up and bigger value down. This way, we will create the heap at once, without calling the *insert()* method internally. Let's revisit it again:

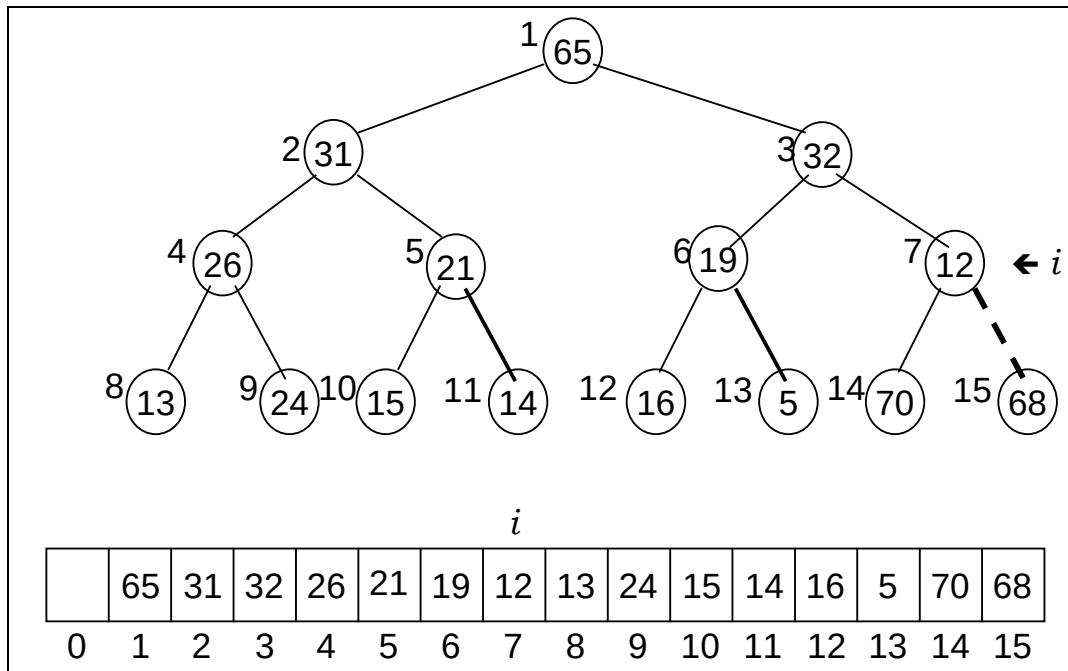
- The general algorithm is to place the N keys in an array and consider it to be an unordered binary tree.
- The following algorithm will build a heap out of N keys.
for ($i = N/2$; $i > 0$; $i--$)
 percolateDown(i);

Suppose, there are N data elements (also called as N Keys). We will put this data in an array and call it as a binary tree. As discussed earlier, a complete binary tree can be stored in an array. We have a tree in an array but it is not a heap yet. It is not necessary that the heap property is satisfied. In the next step, we apply the algorithm. Why did we start the i from $N/2$? Let's apply this algorithm on the actual data. We will use the diagram to understand the *BuildHeap*. Consider the diagram given below:

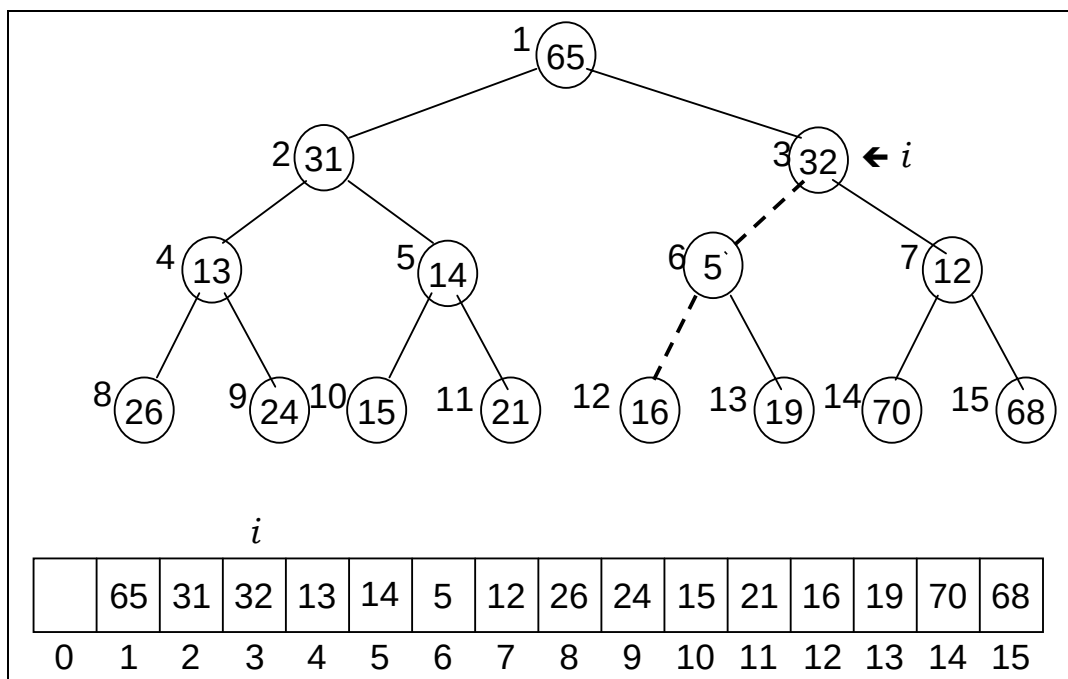


In the above diagram, there is an array containing the elements as 65, 31, 32, 26, 21, 19, 68, 13, 24, 15, 14, 16, 5, 70, 12. We have all the data in this array. The zeroth element of the array is empty. We kept it empty intentionally to apply the $2i$, $2i+1$ scheme. We will take this array as a complete binary tree. Let's take the first element that is 65, applying the $2i$ and $2i+1$ formula. This element has 31 and 32 as its children. Take the second element i.e. 32 and use the formula $2i$ and $2i+1$. Its children are 26 and 21. We can take the remaining elements one by one to build the tree. The tree is shown in the above figure. This tree is not a *min-heap* as it has 65 as root element while there are smaller elements like 13, 15, 5 as its children. So, this being a binary tree has been stored in an array.

Let's think about the above formula, written for the heap building. What will be the initial value of i . As the value of N is 15, so the value of i ($i=N/2$) will be 7 (integer division). The first call to *percolateDown* will be made with the help of the value of i as 7. The element at 7th position is 68. It will take this node as a root of subtree and make this subtree a minimum heap. In this case, the left child of node 68 is 70 while the right child of node 68 is 12. The element 68 will come down while the element 12 moves up.



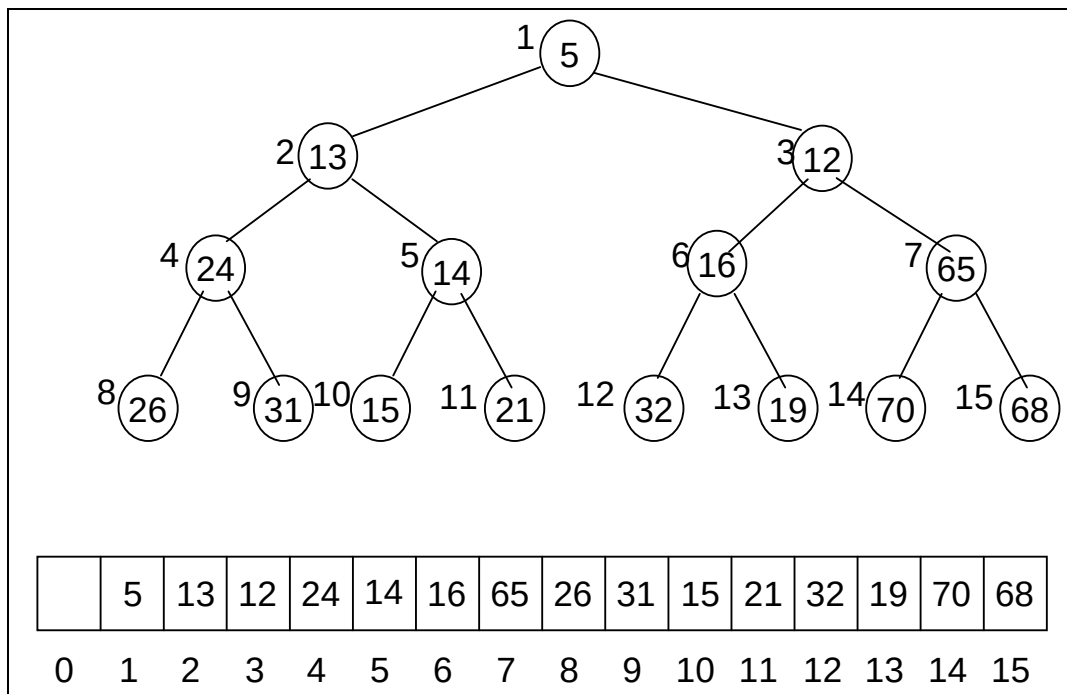
Look at this subtree with 12 as the root node. Its left and right children are smaller than root. This subtree is a minimum heap. Now in the loop, the value of i is decreased by 1, coming to the mark of 6. The element at 6th position is 19 that has been passed to *percolateDown* method. It will convert this small tree into a minimum heap. The element 19 is greater than both of its children i.e. 16 and 5. One of these will move up. Which node should move up? The node 5 will move up while node 19 will move down. In the next repetition, the value of i becomes 5, i.e. the element 21. We apply the *percolateDown()* method again and convert it into minimum heap. Now the subtree with node 26 will be converted into a minimum heap.



Here the value of i is 3. The element at 3rd position is 32. You can see that the level of the node has also been changed. We have moved one level up. Now the *percolateDown* method will take this node as the root of the tree. It needs to move the minimum value at this position. The minimum value is 5 so the values 32 and 5 are going to be exchanged with each other. Now 32 becomes the parent of node 16 and node 19 which are smaller values. Therefore, it will further move down to node 16. The dotted line in the above figure shows the path of nodes that will be replaced. This subtree has been converted into the *min-heap*.

Now, the value of i is 2 while the element is 31. It will change its position with 13 first and then with 24. So the node 31 will move down and become a leaf node. You have seen that some nodes are moving up and some moving downwards. This phenomenon is known as percolate. Therefore we have named this method as *percolateDown* as it makes big values move down and smaller values upward.

Finally the value of i is 1. The element at first position is 65. As it is not the smallest value, so it will move down and the smallest value will move up. It will move to the position 7 as it is greater than 68 and 70. The node 5 will move to the top of the tree. The final figure is shown below:



Is this a minimum heap? Does the heap property satisfy or not? Let's analyze this tree. Start from the root of the tree. Is the value of the root is smaller than that of its left and right children? The node 5 is smaller than the node 13 and node 12. Check this on the next level. You will see that the values at each node are smaller than its children. Therefore it satisfies the definition of the *min-heap*.

Now we have to understand why we have started the 'for loop' with $i = N/2$. We can start i from N . As i moves from level to level in the tree. If we start i from N , it will

start from the right most leaf node. As long as i is 1 less than $N/2$ it will remain on leaf nodes level. Is the leaf node alone is *min-heap* or not? Yes it is. As it does not have any left or right child to compare that it is smaller or not. All the leaf nodes satisfy the *min-heap* definition. Therefore we do not need to start with the leaf nodes. We can do that but there is no reason to do that. For efficiency purposes we start i from $N/2$. This is the one level up to the leaf node level. We applied the *percolateDown* method on each node at this level. Due to this some nodes move down and some nodes move up. We apply this on each level and reaches at the root. In the end we have a minimum heap.

If we have data available, the use of *BuildHeap* method may be more appropriate as compared to *insert* method. Moreover, *BuildHeap* method takes less time. How can we prove that the *BuildHeap* method takes less time? Some mathematical analysis can help prove it. As *insert* method is $N\log N$ and *BuildHeap* method should be better than that.

Other Heap Methods

Let's have a look on some more methods of heap and see the C++ codes.

decreaseKey(p, delta)

This method lowers the value of the key at position 'p' by the amount 'delta'. Since this might violate the heap order, so it (the heap) must be reorganized with percolate up (in *min-heap*) or down (in *max-heap*).

This method takes a pointer to the node that may be the array position as we are implementing it as an array internally. The user wants to decrease the value of this node by delta. Suppose we have a node with value 17 and want to decrease it by 10. Its new value will be 10. By decreasing the value of a node, the heap order can be violated. If heap order is disturbed, then we will have to restore it. We may not need to build the whole tree. The node value may become smaller than that of the parents, so it is advisable to exchange these nodes. We use *percolateUp* and *percolateDown* methods to maintain the heap order. Here the question arises why we want to decrease the value of some node? The major use of heap is in priority queues. Priority queues are not FIFO or LIFO. The elements are taken out on some key value, also known as priority value. Suppose we have some value in the priority queue and want to decrease its priority. If we are using heap for the priority queue, the priority of the some elements can be decreased so that it could be taken out later and some other element that now has higher priority will be taken out first. You can find many real life examples to understand why we need to increase or decrease the priority of elements.

One of these examples is priorities of processes. You will read about this in the operating system course. You can see the priorities of the processes using the task manager. Task manager can be activated by pressing Ctrl+Alt+Del. Under the process tab, we may see the priority of each process. Normally, there is one processor in a computer and there is lot of processes running in it. How can we decide which process should be given the CPU time and how much time should be given to this process? We will cover all of these topics in the operating system course. Suppose we have 25

processes with different priorities. How can we decide which process should be given the CPU time? We gave the CPU to the highest priority process for one second. We can schedule this by using the priority queue. For some reason, we decrease the priority of some process. Now its turn will come later. We may increase the priority of some process so that it should be given the CPU time. Therefore, we adopt the increase and decrease methods. If we are implementing the priority queue by using the heap, the increase or decrease method of priority queue will use the increase and decrease method of heap internally. Using this method, the turn of the element will be according to the new priority.

increaseKey(p, delta)

This method is the opposite of *decreaseKey*. It will increase the value of the element by delta. These methods are useful while implementing the priority queues using heap.

remove(p)

This method removes the node at position p from the heap. This is done first by *decreaseKey(p, ϵ)* and then performing *deleteMin()*. First of all, we will decrease the value of the node by ϵ and call the method *deleteMin*. The *deleteMin* method deletes the root. If we have a *min-heap*, the root node contains the smallest value of the tree. After deleting the node, we will use the *percolateDown* method to restore the order of the heap.

The user can delete any node from the tree. We can write a special procedure for this purpose. Here we will use the methods which are already available. At first, the *decreaseKey* method will be called and value of node decreased by ϵ . It will result in making the value of this node smallest of all the nodes. If the value is in integers, this node will have the smallest integer. Now this node has the minimum value, so it will become the root of the heap. Now we will call the *deleteMin()* and the root will be deleted which is the required node. The value in this node is not useful for us. The ϵ is a mathematical notation. It is not available in the C++. Actually we want to make the minimum possible value of this node supported by the computer.

C++ Code

Now we will look at the C++ code of the *Heap* class. The objects of *Heap* may be got from this factory class. This class will contain the methods including those discussed earlier and some new ones. Heap is used both in priority queues and sorting.

We have some public and private methods in the class. Let's have a look on the code.

```
/* The heap class. This is heap.h file */

template <class eType>

class Heap
{
public:
    Heap( int capacity = 100 );
```

```

    Void insert( const eType & x );
    Void deleteMin( eType & minItem );
    Const eType & getMin( );
    Bool isEmpty( );
    Bool isFull( );
    int Heap<eType>::getSize( );

private:
    int  currentSize; // Number of elements in heap
    eType* array;    // The heap array
    int  capacity;

    void percolateDown( int hole );
};

```

We may like to store different type of data in the heap like integers, strings, floating-point numbers or some other data type etc. For this purpose, template is used. With the help of template, we can store any type of object in the heap. Therefore first of all we have:

```
template <class eType>
```

Here *eType* will be used as a type parameter. We can use any meaningful name. Then we declare the *Heap* class. In the public part of the class, there is a constructor as given below.

```
Heap( int capacity = 100 );
```

We have a parameter *capacity* in the constructor. Its default value is 100. If we call it without providing the parameter, the capacity will be set to 100. If we call the constructor by providing it some value like 200, the capacity in the *Heap* object will be 200.

Next we have an insert method as:

```
void insert( const eType & x );
```

Here we have a reference element of *eType* which is of constant nature. In this method, we will have the reference of the element provided by the caller. The copy of element is not provided through this method. We will store this element in the *Heap*.

Similarly we have a delete method as:

```
void deleteMin( eType & minItem );
```

This method is used to delete the element from the heap.

If we want to know the minimum value of the heap, the *getMin* method can be useful.

The signatures are as:

```
const eType & getMin( );
```

This method will return the reference of the minimum element in the *Heap*. We have some other methods in the *Heap* class to check whether it is empty or full. Similarly, there is a method to check its size. The signatures are as:

```
bool isEmpty( );
bool isFull( );
int Heap<eType>::getSize( );
```

When will *Heap* become full? As we are implementing the *Heap* with the help of an array, the fixed data type, so the array may get full at some time. This is the responsibility of the caller not to insert more elements when the heap is full. To facilitate the caller, we can provide the methods to check whether the heap is full or not. We can call the *getSize()* method to ascertain the size of the *Heap*.

In the private part, we have some data elements and methods. At first, we have *currentSize* element. It will have the number of elements in the heap. Then we have an array of *eType*. This array will be dynamically allocated and its size depends on the *capacity* of the *Heap*. We have one private method as *percolateDown*.

This is our *.h* file, now we will see the implementation of these methods that is in our *.cpp* file. The code is as under:

```
/* heap.cpp file */

#include "Heap.h"

template <class eType>
Heap<eType>::Heap( int capacity )
{
    array = new eType[capacity + 1];
    currentSize=0;
}

/* Insert item x into the heap, maintaining heap order. Duplicates
are allowed. */

template <class eType>
bool Heap<eType>::insert( const eType & x )
{
    if( isFull( ) ) {
        cout << "insert - Heap is full." << endl;
        return 0;
    }
    // Percolate up
    int hole = ++currentSize;
    for(; hole > 1 && x < array[hole/2 ]; hole /= 2)
```

```

        array[ hole ] = array[ hole / 2 ];
        array[hole] = x;
    }
    template <class eType>
    void Heap<eType>::deleteMin( eType & minItem )
    {
        if( isEmpty( ) ) {
            cout << "heap is empty." << endl;
            return;
        }

        minItem = array[ 1 ];
        array[ 1 ] = array[ currentSize-- ];
        percolateDown( 1 );
    }

```

We include the *heap.h* file before having the constructor of the heap. If we do not provide the value of *capacity*, it will be set to 100 that is default. In the constructor, we are dynamically creating our array. We have added 1 to the *capacity* of the array as the first position of the array is not in use. We also initialize the *currentSize* to zero because initially *Heap* is empty.

Next we have *insert* method. This method helps insert item *x* into the heap, while maintaining heap order. Duplicates are allowed. Inside the *insert* method, we will call the *isFull()* method. If the heap is full, we will display a message and return 0. If the heap is not full, we will insert the element in the heap and take a variable *hole* and assign it *currentSize* plus one value. Then we will have a ‘for loop’ which will be executed as long as *hole* is greater than 1. This is due to the fact that at position one, we have root of the array. Secondly, the element which we want to insert in the array is smaller than the *array[hole/2]*. In the loop, we will assign the *array[hole/2]* to *array[hole]*. Then we divide the hole by 2 and again check the loop condition. After exiting from the loop, we assign *x* to the *array[hole]*. To understand this insert process, you can get help from the pictorial diagrams of the *insert*, discussed earlier. We have a complete binary tree stored in an array and placed this new element at the next available position in the array and with respect to tree it was left most leaf node. This new node may have smaller value so it has to change its position. If we are building the *min-heap*, this value should be moved upward. We will compare this with the parent. If the parent is bigger, it will move down and child will move up. Using the array notation, the parent of a child is at $i/2$ if child is at i position. We will first try to find the final position of this new node and exchange the values. You might want to remember the example of insert discussed in some earlier lecture. We have shown a hole going up. Now the swapping function generally contains three statements and is not an expensive operation. But if we perform swapping again and again, it may cost some time. Therefore we will first find the final position of the new node and then insert it at that position. In the ‘for loop’ we are finding the final position of the node and the hole is moving upwards. In the statement *array[hole] = array[hole/2]*; we are moving the parent down till the time final position of the new node is achieved. Then we insert the node at its final position. It is advisable to execute this code only for actual data and see how it works.

The next method is *deletMin*. First of all, it calls the *isEmpty()* method. If heap is empty, it will display a message and return. If the heap is not empty, it will delete the node. As the minimum value lies at the first position of the array, we save this value in a variable and store the *currentSize* at the first position of the array. At the same time, we reduce the *currentSize* by one as one element is being deleted. Then we call the *percolateDown* method, providing it the new root node. It will readjust the tree. We put the last element of the array which is the right most leaf node of the tree as the root element will be deleted now. With this operation, the heap order can be disturbed. Therefore we will call the *percolateDown* method which has the ability to readjust the tree. This method will make the tree as *min-heap* again. There are some more methods which we will cover in the next lecture.

Data Structures

Lecture No. 32

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 6
6.3

Summary

- percolateDown Method
- getMin Method
- buildHeap Method
- buildHeap in Linear Time
- Theorem

In the previous lecture, we defined a heap class and discussed methods for its implementation. Besides, the insert and delete procedures were talked about. Here, we will continue with discussion on other methods of the heap class. C++ language code will be written for these procedures so that we could complete the heap class and use it to make heap objects.

We are well aware of the *deleteMin* method. In this method, *array[1]* element (a root element) is put in the *minItem* variable. In the min heap case, this root element is the smallest among the heap elements. Afterwards, the last element of the array is put at the first position of the array. The following two code lines perform this task.

```
minItem = array[ 1 ];  
array[ 1 ] = array[ currentSize-- ];
```

percolateDown Method

Then we call the *percolateDown* method. This is the same method earlier employed in the build heap process. We passed it the node, say *i*, from where it starts its function to restore the heap order. Let's look at this *percolateDown* method. It takes the array index as an argument and starts its functionality from that index. In the code, we give the name *hole* to this array index. Following is the code of this method.

```
// hole is the index at which the percolate begins.  
template <class eType>  
void Heap<eType>::percolateDown( int hole )  
{  
    int child;  
    eType tmp = array[ hole ];  
    for( ; hole * 2 <= currentSize; hole = child )  
    {  
        child = hole * 2;
```

```

        if( child != currentSize && array[child+1] < array[ child ] )
            child++; // right child is smaller
        if( array[ child ] < tmp )
            array[ hole ] = array[ child ];
        else break;
    }
    array[ hole ] = tmp;
}

```

In the code of this function, it declares an integer variable named *child* before putting the value at the index *hole* of the array in *tmp* variable. It is followed by a ‘for loop’, the core of this method. The termination condition of this *for* loop is $hole * 2 \leq currentSize$; We know the $2i$ (or $i * 2$) formula that gives us index position of the left child of i . In the code given above, we are using *hole* instead of i . This variable i.e. *hole* is the loop variable. In other words, the termination condition means that the loop will continue as long as the left child ($hole * 2$) is less than or equal to the current size (*currentSize*) of the heap. Before looking at the third condition in which we change the value of loop variable, let’s have a view of the iteration of *for* loop.

In the body of the ‘*for* loop’, we assign the value $hole * 2$ to the *child* variable. The left child of an index i is located at the index $2i$. So it is obvious that the *child* has the index number of the left child of *hole*. Then there is an *if* statement. In this statement, we check two conditions combined with $\&\&$ (logical AND) operator. This *if* statement returns TRUE only if both the conditions are TRUE. The first condition in this *if* statement checks that the *child* (index) should not be equal to the *currentSize* while in the second condition, we compare the value at index $child + 1$ with the value at index *child*. We check whether the value at $child + 1$ is less than the value at *child*. The value at index *child* is the left child of its parent as we have found it by the formula of $2i$ (here we used $hole * 2$). Thus the value at $child + 1$ will be the right child of the parent (i.e. *hole*). Actually, we are comparing the left and right children of *hole*. If both the conditions are TRUE, it means that right child is less than the left child. Resultantly, we increment the *child* index by 1 to set it to the right child. Now again in an *if* statement, we compare the value at index *child* (which is left or right child depending upon our previous check in *if* statement) with the *tmp* value. If value at *child* is less than the *tmp* value, we will put the value of index *child* at the index *hole*. Otherwise, if *tmp* value is less than the *child*, we exit *for* loop by using the *break* statement. Thus the value of *child* comes to the position of the parent (i.e. *hole*). The *hole* gets downward and *child* goes upward. The ‘*for* loop’ continues, bringing the *hole* downward and setting it to its proper position. When the *hole* reaches its position, the ‘*for* loop’ exits (it may exit by meeting the *break* statement) and we put the *tmp* value in the array at the index *hole*. In this method, at first, the final position of *hole* is determined before putting value in it. Here the *percolateDown* procedure ends.

Now the question arises why we are bringing the *hole* (index) downward? Why don’t we exchange the values? We can execute the process of exchanging values. We will use swap procedure to do this exchange. The swap process is carried out in three statements. We have to do this swapping in the *for* loop. It is time-consuming process. Contrast to it, in the *percolateDown* method, we execute one statement in *for* loop instead of three swap statements. So it is better to use one statement instead of three statements. This increases the efficiency with respect to time.

getMin Method

We discussed this method in the previous lectures while talking about the interface of heap. Under this method, the minimum value in the heap is determined. We just try that element and do not remove it. It is like the top method of stack that gives us the element on the top of the stack but do not remove it from the stack. Similarly, the getMin method gives us the minimum value in the heap, which is not deleted from the heap. This method is written in the following manner.

```
template <class eType>
const eType& Heap<eType>::getMin( )
{
    if( !isEmpty( ) )
        return array[ 1 ];
}
```

Now we will discuss the buildHeap method.

buildHeap Method

This method takes an array along with its size as an argument and builds a heap out of it. Following is the code of this method.

```
template <class eType>
void Heap<eType>::buildHeap(eType* anArray, int n )
{
    for(int i = 1; i <= n; i++)
        array[i] = anArray[i-1];
    currentSize = n;
    for( int i = currentSize / 2; i > 0; i-- )
        percolateDown( i );
}
```

In the body of this method, at first, there is a ‘for loop’ that copies the argument array (i.e. anArray) into our internal array, called ‘array’. We do this copy as the array that this method gets as an argument starts from index zero. But we want that the array should start from index 1 so that the formula of $2i$ and $2i + 1$ could be used easily. In this for loop, we start putting values in the internal array from index 1. Afterwards, we set the currentSize variable equal to the number of the elements got as an argument. Next is the *for* loop that we have already seen while discussing the build of heap. In the previous lecture, we used ‘N’ for number of elements but here, it will be appropriate to use the currentSize, which is also the number of elements. We start this loop from the value $i = \text{currentSize} / 2$ and decrement the value of i by 1 for each iteration and execute this loop till the value of i is greater than 0. In this for loop, we call the method percolateDown to find the proper position of the value given as an argument to this method. Thus we find the position for each element and get a heap. Then there are three small methods. The isEmpty method is used to check whether the heap is empty. Similarly the isFull method is used to check if the heap is full. The getSize method is used to get the size of the heap. These three methods i.e. isEmpty, isFull and getSize are written below.

```

//isEmpty method
template <class eType>
bool Heap<eType>::isEmpty( )
{
    return currentSize == 0;
}

//isFull method
template <class eType>
bool Heap<eType>::isFull( )
{
    return currentSize == capacity;
}

//getSize method
template <class eType>
int Heap<eType>::getSize( )
{
    return currentSize;
}

```

buildHeap in Linear Time

We have seen that *buildHeap* takes an array to make a heap out of it. This process of making heap (*buildHeap* algorithm) works better than $N \log_2 N$. We will prove it mathematically. Although our course is Data Structures, yet we have been discussing the efficiency of the data structures from the day one of this course. We talked about how efficiently a data structure uses memory storage and how much it is efficient with respect to time. In different cases, mathematics is employed to check or compare the efficiency of different data structures.

Here, we will show that the *buildHeap* is a linear time algorithm and is better than $N \log_2 N$ algorithm which is not of linear nature. Linear algorithm is such a thing that if we draw its graph, it will be a straight line with some slope. Whereas the graph of a non-linear algorithm will be like a curve. We will discuss it in detail later. To prove the superiority of the *buildHeap* over the $N \log_2 N$, we need to show that the sum of heights is a linear function of N (number of nodes). Now we will a mathematical proof that proves that the *buildHeap* algorithm is better than $N \log_2 N$. Now consider the following theorem.

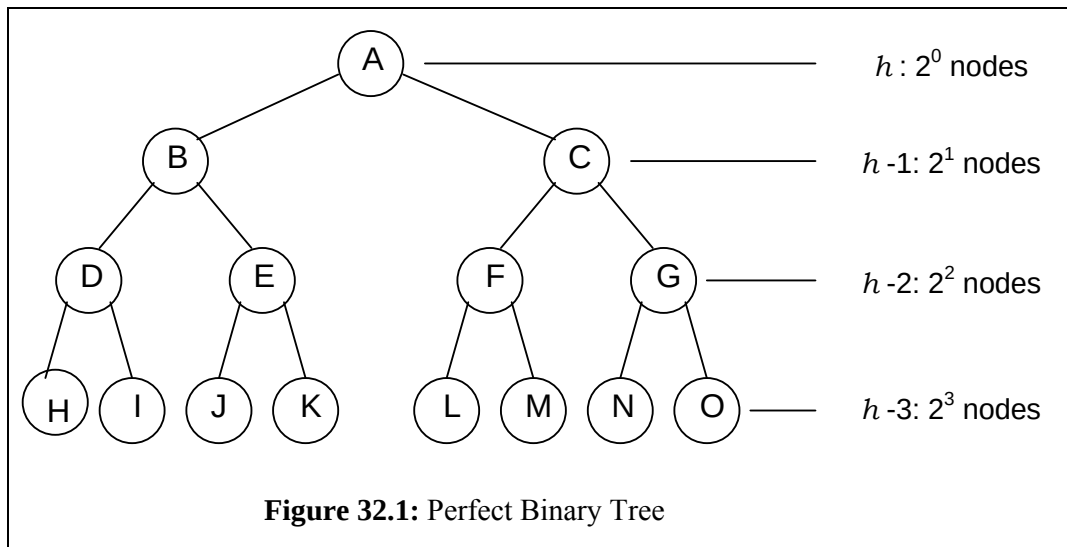
Theorem

According to this theorem, “For a perfect binary tree of height h containing $2^{h+1} - 1$ nodes, the sum of the heights of nodes is $2^{h+1} - 1 - (h + 1)$, or $N - h - 1$ ”.

This theorem is about heights of tree. Let's try to understand the concept of height. If we start from the root node and go to the deepest leaf node of the tree, the number of links we pass to go to the deepest leaf node is called the height of the tree. We have been using the term depth and level for it also. Height can also be measured with reference to the deepest leaf node at height zero and going upward to the root node. Normally we start from the root and go to the deepest level to determine the height. A perfect binary tree of height h has total number of nodes N equal to $2^{h+1} - 1$. Now according to the theorem, the sum of heights of nodes will be equal to $N - h - 1$. Let's

prove it mathematically.

In a perfect binary tree, there are 2^0 nodes at level 0, 2^1 nodes at level 1, 2^2 nodes at level 2 and so on. Generally, level n has 2^n nodes. Now in terms of height, in a perfect binary tree, if h is the height of the root node, there are 2^0 nodes at height h , 2^1 nodes at height $h-1$, 2^2 nodes at height $h-2$ and so on. In general, there are 2^i nodes at height $h-i$. Following figure shows a perfect binary tree with the number of nodes at its heights.



In the above figure, we can see that there are 2^1 nodes at height $h-1$ and 2^3 nodes at height $h-3$. The same is true about the number of nodes at other heights.

By making use of this property, we find the sum of heights of all the nodes mathematically in the following manner. Suppose S is the sum of the heights of all the nodes. Then

$$\begin{aligned}
 S &= \sum 2^i (h - i), \text{ for } i = 0 \text{ to } h - 1 \\
 S &= h + 2(h-1) + 4(h-2) + 8(h-3) + \dots + 2^{h-1} \quad (1) \\
 \text{Now multiplying by 2 the both sides of equation} \\
 2S &= 2h + 4(h-1) + 8(h-2) + 16(h-3) + \dots + 2^h \quad (2) \\
 \text{Now subtracting the equation 2 from equation 1} \\
 -S &= h - 2 - 4 - 8 - 16 - \dots - 2^{h-1} - 2^h \\
 S &= -h + 2 + 4 + 8 + 16 + \dots + 2^{h-1} + 2^h \\
 S &= (2^{h+1} - 1) - (h+1)
 \end{aligned}$$

Here S is the sum of heights. Thus it proves the theorem.

As stated earlier, in a perfect binary tree of height h , the total number of nodes N is $(2^{h+1} - 1)$. So by replacing it in the above equation, we get $S = N - (h + 1)$.

Since a binary complete tree has nodes between 2^h and 2^{h+1} , the equation

$$S = (2^{h+1} - 1) - (h+1)$$

Can also be written as

$$S \simeq N - \log_2(N+1)$$

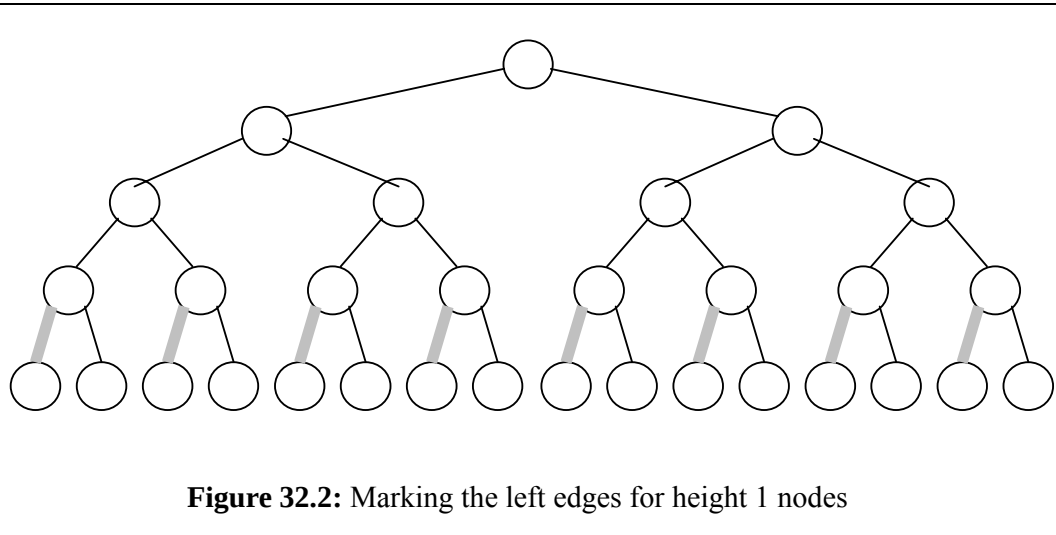
In this equation, N stands for $(2^{h+1} - 1)$ while $\log_2(N+1)$ has taken the place of $(h+1)$. This equation is in term of N (number of nodes) and there is no h i.e. height in this equation. Moreover, it also shows that with N getting larger, the $\log_2(N+1)$ term becomes insignificant and S becomes a function of N .

Suppose if N has a value 1000000, the value of $\log_2(1000000)$ will be approximately 20. We can see that it is insignificant with the value of N i.e. 1000000. Thus we can say that S is approximately equal to N . This insignificance of $\log N$ shows that when we build a heap from N elements, the values move up and down to maintain heap order. Thus in buildHeap case, the values traverse the tree up and down. Now the question arises what will be the maximum number of these traversals? This means that if every node has to go up and down, what will be the maximum number of level or height. We have proved that the sum of heights (i.e. S) is approximately equal to the total number of nodes (i.e. N) as N becomes larger. Now in the buildHeap, the maximum number of up down movement of values is actually the sum of heights i.e. S . This S is the upper limit of number of movements in buildHeap. As this S is equal to N , so this upper limit is N and there is not any log term involved in it. Thus buildHeap is a linear time application.

Now let's prove the previous theorem with a non-mathematical method. We know that the *height* of a node in the tree is equal to the number of edges on the longest downward path to a leaf. So if we want to find the height of a node, it will be advisable to go to the deepest leaf node in the tree by following the links (edges) and this path (the number of links traveled) to the leaf node is the height of that node. The height of a tree is the height of its root node. After these definitions, consider the following two points that prove the theorem.

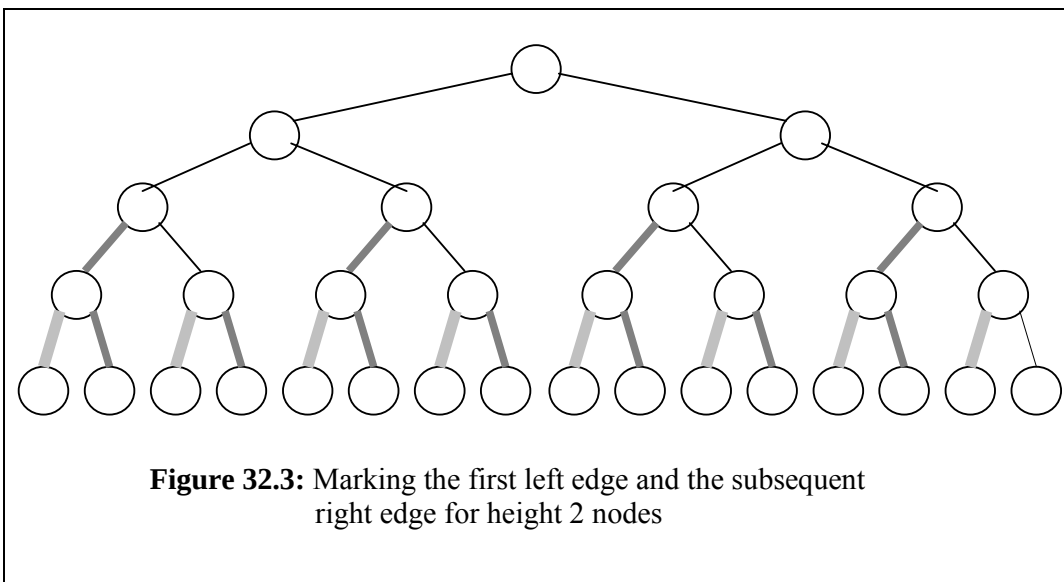
- For any node in the tree that has some height h , darken h tree edges
–Go down the tree by traversing left edge then only right edges.
- There are $N - 1$ tree edges, and h edges on right path, so number of darkened edges is $N - 1 - h$, which proves the theorem.

We will explain these two points with the help of figures.
Consider the perfect binary tree, shown in the figure below.

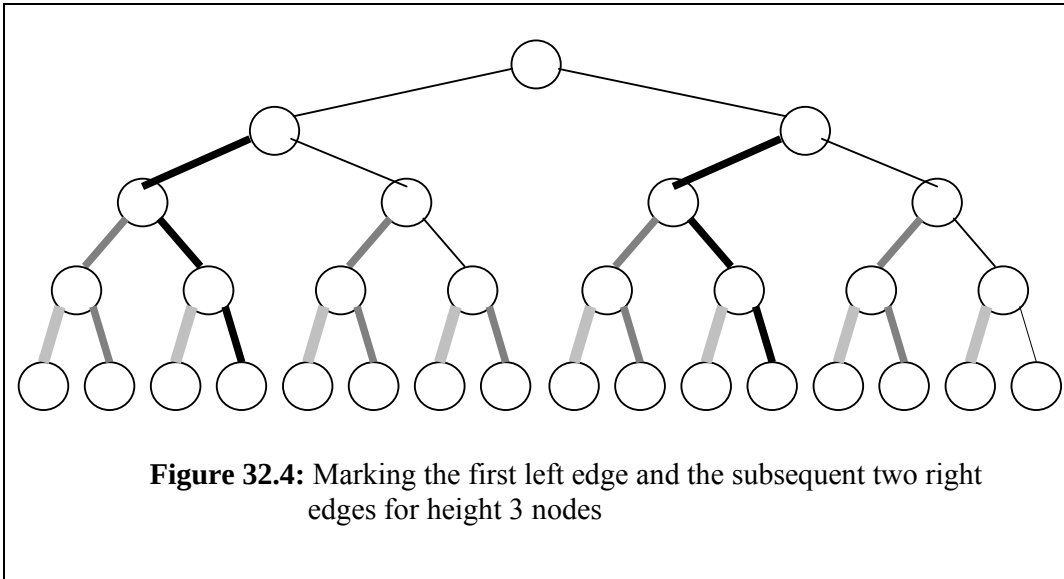


There are no values shown in the nodes. We are concerned with the heights of nodes while the values do not matter. We mark the left edges of the nodes having height 1. The leaf nodes are at height 0, so the one level above nodes has height 1, the left edges of which are marked (shown light gray in above figure).

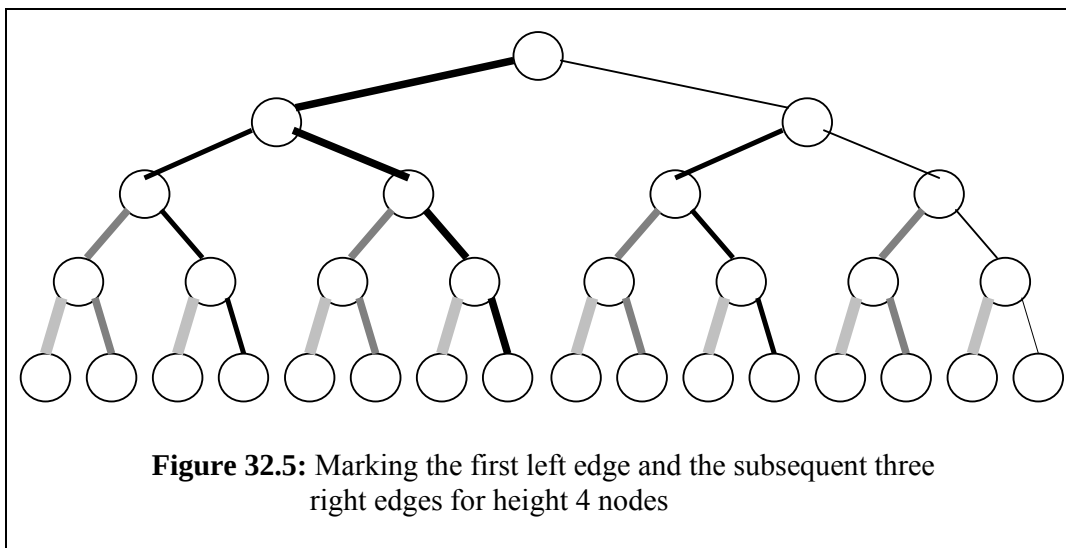
Now we mark the first left edge and the subsequent right edge up to the leaf node for each node at height 2. These marked edges are shown in the figure below with gray color.



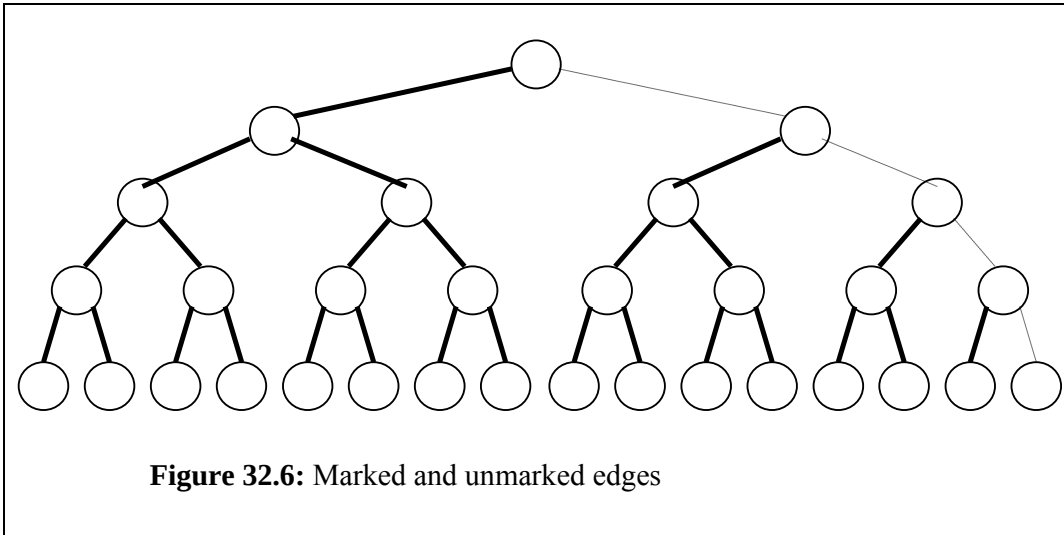
Similarly we go to the nodes at height 3, mark the first left edge and the subsequent right edge up to the leaf node. Thus we mark one left and then two right edges for the nodes at height 3. This is shown in the following figure. The edges marked in this step are seen as the dark lines.



Now we reach at the root whose height is 4. We mark the first left edge and the subsequent three right edges for this node. This is shown in the figure below.



Now consider the following figure. In this figure, we show all the marked edges (that we have marked in the previous steps) in gray and the non-marked edges are shown with dotted lines.



The marked links are the ones through which the data can move up and down in the tree. We can move the data in any node at a height to a node at other height following the marked links. However, for the movement of the data in the root node to right side of it, we can have opposite of the above figure. The opposite of the above figure can be drawn by symmetry of the above steps. That means we first mark the right edge and then the subsequent left edges. This will give us the figure of marked links in which we can move to the right subtree of root.

Now let's sort out different aspects of the tree shown in above figure. We know that in case of tree of N nodes, there will be $N - 1$. Now in the tree above there are 31 nodes that means $n = 31$, so the number of edges is $31 - 1 = 30$. The height of the tree is 4. Height is represented by the letter H . so $H = 4$. The number of dotted edges (that were not marked) in the tree is 4 that is the same as the height of the tree. Now we put these values in the formula for the sum of the heights of the nodes. We know the formula i.e.

$$S = N - H - 1$$

By putting values in this formula, we get

$$S = 31 - 4 - 1 = 26$$

If we count the darkened edges (marked links) in the above tree, it is also 26 that is equal to the sum of heights. Thus with the help of these figures, the theorem earlier proved mathematically, is established by a non-mathematical way.

Data Structures

Lecture No. 33

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 6, 8
6.3, 8.1

Summary

- Priority Queue Using Heap
- The Selection Problem
- Heap Sort
- Disjoint Set ADT
- Equivalence Relations

Priority Queue Using Heap

As discussed in the previous lecture, we generally prefer to employ the `buildHeap` to construct heap instead of using `insert` if we have all the required data. *buildHeap* is optimized as compared to *insert* method as it takes lesser time than $N\log_2 N$. In the previous discussion, we had proved a theorem that if the number of links in the tree are counted, the sum of all is $N-h-1$. We have been iterating a lot of times that the best use of heap is priority queue's implementation. Let's try to develop a class of priority queue with the help of a heap. You are very familiar with the concept of bank simulation. While explaining the bank simulation, we had used a priority queue that was implemented by using an array. Now, we will implement the same queue with the help of a heap. For this purpose, the code will be modified so that heap is used in place of array. The interface (.h file) will remain the same. However, the implementation (.cpp file) will be changed. Let's see the following cpp code, which also shows the change in the .cpp file:

```
1.  #include "Event.cpp"
2.  #include "Heap.cpp"
3.  #define PQMAX 30
4.  class PriorityQueue
5.  {
6.      public:
7.          PriorityQueue()
8.          {
9.              heap = new Heap<Event> ( PQMAX );
10.         };
11.
12.         ~PriorityQueue()
13.         {
14.             delete heap;
15.         };
16.
```

```

17.         Event * remove()
18.         {
19.             if( !heap->isEmpty() )
20.             {
21.                 Event * e;
22.                 heap->deleteMin( e );
23.                 return e;
24.             }
25.             cout << "remove - queue is empty." << endl;
26.             return (Event *) NULL;
27.         };
28.
29.         int insert(Event * e)
30.         {
31.             if( !heap->isFull() )
32.             {
33.                 heap->insert( e );
34.                 return 1;
35.             }
36.             cout << "insert queue is full." << endl;
37.             return 0;
38.         };
39.
40.         int full(void)
41.         {
42.             return heap->isFull();
43.         };
44.
45.         int length()
46.         {
47.             return heap->getSize();
48.         };
49.     };

```

The first line has a file *Event.cpp* that contains all the events used for simulation. We are including .cpp files here as done in case of templates of C++. In the second line, there is *heap.cpp* while a constant PQMAX has been defined in third line, declaring the maximum size of the priority queue to be 30. In line 4, class *PriorityQueue* is declared. *public* keyword is given at line 6 indicating that all class members below will be of public scope. Line 7 starts the class constructor's definition while in the line 9; a new *heap* object is being created. This object is a collection of *Event* type objects and the number of elements in the collection is *PQMAX*. The address (pointer) of the newly created *heap* is stored in the *heap* object pointer. The *heap* is a private pointer variable of *PriorityQueue* class. Now there is the destructor of the class, where we are deleting (deallocating the The first line is including a file *Event.cpp*, this is containing all the events used for simulation. At the second line, we have included *heap.cpp*. In the third line, we are defining a constant PQMAX, declaring the maximum size of the priority queue to be 30. In line 4, class *PriorityQueue* is

declared. *public* keyword is given at line 6 indicating that all class members below will be of public scope. Line 7 is starting the class constructor's definition. At line 9, a new heap object is being created, this object is a collection of *Event* type objects and the number of elements in the collection is *PQMAX*. The address (pointer) of the newly created heap is stored in the *heap* object pointer. The *heap* is a private pointer variable of *PriorityQueue* class. Next comes the destructor of the class, where we are deleting (deallocating the allocated resources) the pointer variable *heap*. Next is the *remove* method that was sometimes, named as *dequeue* in our previous lectures. *remove* method is returning an *Event* pointer. Inside *remove*, the first statement at line 19 is an if-condition; which is checking whether *heap* is not empty. If the condition is true (i.e. the *heap* is not empty), a local variable (local to the block) *Event** variable *e* is declared. In the next line at line 22, we are calling *deleteMin* method to delete (remove) an element from the *heap*, the removed element is assigned to the passed in parameter pointer variable *e*. You might have noticed already that in this version of *deleteMin*, the parameter is being passed by pointer, in the previous implementation we used reference. In the next line, the retrieved value is returned, the function returns by returning the pointer variable *e*. But if the heap was empty when checked at line 19, the control is transferred to the line 25. At line 25, a message is displayed to show that the heap is empty and the next line returns a NULL pointer.

It is important to understand one thing that previously when we wrote the array based *remove* method. We used to take an element out from the start of the array and shifted the remaining elements to left. As in this implementation, we are using heap, therefore, all the responsibilities of maintaining the heap elements lie with the heap. When the *deleteMin()* is called, it returns the minimum number.

We now observe the *insert* method line by line. It is accepting a pointer type parameter of type *Event*. As the heap may become full, if we keep on inserting elements into it, so the first line inside the insert function is checking whether the heap has gone full. If the heap is not full, the if-block is entered. At line 33 inside the if-block, an element that is passed as a parameter to *insert* method of heap is inserted into the queue by calling the *insert* method of heap. This insert call will internally perform percolate up and down operations to place the new element at its correct position. The returned value 1 indicates the successful insertion in the queue. If the heap has gone full, a message is displayed i.e. '*insert queue is full*'. Note that we have used the word queue, not heap in that message. It needs to be done this way. Otherwise, the users of the *PriorityQueue* class are unaware of the internal representation of the queue (whether it is implemented using a heap or an array). Next line is returning 0 while indicating that the *insert* operation was not successful. Below to *insert*, we have *full()* method. This method returns an *int* value. Internally, it is calling *isFull()* method of heap. The *full()* method returns whatever is returned by the *isFull()* method of heap. Next is *length()* method as the size of heap is also that of the queue. Therefore, *length()* is internally calling the *getSize()* method of heap.

In this new implementation, the code is better readable than the *PriorityQueue*'s implementation with array. While implementing the *PriorityQueue* with an array, we had to sort the internal array every time at each insertion in the array. This new implementation is more efficient as the heap can readjust itself in $\log_2 N$ times. Gain in performance is the major benefit of implementing *PriorityQueue* with heap as compared to implementation with array.

There are other significant benefits of the heap that will be covered in this course time to time. At the moment, we will have some common example usages of heap to make you clear about other uses of heap. The heap data structure will be covered in the Algorithms course also.

The Selection Problem

- Given a list of N elements (numbers, names etc.) which can be totally ordered and an integer k , find the k^{th} smallest (or largest) element.

Suppose, we have list of N names (names of students or names of motor vehicles or a list of numbers of motor vehicles or list of roll numbers of students or id card numbers of the students, whatever). However, we are confronting the problem of finding out the k^{th} smallest element. Suppose we have a list of 1000 numbers and want to find the 10th smallest number in it. The sorting is applied to make the elements ordered. After sorting out list of numbers, it will be very easy to find out any desired smallest number.

- One way is to put these N elements in an array and sort it. The k^{th} smallest of these is at the k^{th} position.

It will take $N \log_2 N$ time, in case we use array data structure. Now, we want to see if it is possible to reduce the time from $N \log_2 N$ by using some other data structure or by improving the algorithm? Yes, we can apply heap data structure to make this operation more efficient.

- A faster way is to put the N elements into an array and apply the *buildHeap* algorithm on this array.
- Finally, we perform k *deleteMin* operations. The last element extracted from the heap is our answer.

The *buildHeap* algorithm is used to construct a heap of given N elements. If we construct 'min-heap, the minimum of the N elements, will be positioned in the root node of the heap. If we take out (*deleteMin*) k elements from the heap, we can get the K^{th} smallest element. *BuildHeap* works in linear time to make a min or a max-heap.

- The interesting case is $k = N/2$ as it is also known as the *median*.

In Statistics, we take the average of numbers to find the minimum, maximum and median. Median is defined as a number in the sequence where the half of the numbers are greater than this number while the remaining half are smaller ones. Now, we can come up with the mechanism to find median from a given N numbers. Suppose, we want to compute the median of final marks of students of our class while the maximum aggregate marks for a student are 100. We use the *buildHeap* to construct a heap for N number of students. By calling *deleteMin* for $N/2$ times, the minimum marks of the half number students will be taken out. The $N/2^{\text{th}}$ marks would be the median of the marks of our class. The alternate methods are there to calculate median. However, we are discussing the possible uses of heap. Let's see another use of heap.

Heap Sort

To take the 100th minimum element out from the min-heap, we will call *deleteMin* to

take out 1st element, 2nd element, 3rd element. Eventually we will call *deleteMin* 100th time to take out our required 100th minimum element. Suppose, if the size of the heap is 100 elements, we have taken out all the elements out from the heap. Interestingly the elements are sorted in ascending order. If somehow, we can store these numbers, let's say in an array, all the elements sorted (in ascending order in this min-heap case) can be had.

Hence,

- If $k = N$, and we record the *deleteMin* elements as they come off the heap. We will have essentially sorted the N elements.
- Later in the course, we will fine-tune this idea to obtain a fast sorting algorithm called *heapsort*.

We conclude our discussion on the heap here. However, it will be discussed in the forthcoming courses. At the moment, let's see another Abstract Data Type.

Disjoint Set ADT

Before actually moving to an Abstract Data Type (ADT), we need to see what that ADT is, how it works and in which situations it can be helpful. We are going to cover Disjoint Set ADT. Firstly, we will have its introduction, examples and later the ways of its implementation.

- Suppose we have a database of people.
- We want to figure out who is related to whom. Initially, we only have a list of people, and information about relations is obtained by updating the form “Haaris is related to Saad”. Let's say we have a list of names of all people in a locality but are not aware of their relationships to each other. After having the list of all people, we start getting some information about their relationships gradually. For example, “Ali Abbas is the son of Abbas”.

The situation becomes interesting when we have relationships like “Ali Abbas is first cousin of Ayesha Ali (i.e. fathers of both are brothers) but Ayesha Ali has other immediate cousins also from her mother's side. Therefore, other immediate cousins of Ayesha Ali also get related to Ali Abbas despite the fact that they are not immediate to him”.

So as we keep on getting relationship details of the people, the direct and indirect relationships can be established.

- Key property: If Haaris is related to Saad and Saad is related to Ahmad, then Haaris is related to Ahmad.

See the key property line's first part above “Harris is related to Saad and Saad is related to Ahmad”. Suppose we have a program to handle this list of people and their relationships. After providing all the names “Harris, Saad and Ahmad” to that program and their relationships, the program might be able to determine the remaining part “Harris related to Ahmad”.

The same problem (the intelligence required in the program) is described in the sentence below:

- Once we have relationships information, it will be easy for us to answer queries like “Is Haaris related to Ahmad?”

To answer this kind of queries and have that intelligence in our programs, *Disjoint Set ADT* is used. Before going for more information about *Disjoint Set ADT*, we see another application of this ADT in image analysis. This problem is known as *Blob Coloring*.

Blob Coloring

A well-known low-level computer vision problem for black and white images is the following:

Put together all the picture elements (pixels) that belong to the same "blobs", and give each pixel in each different blob an identical label.

You must have heard of robots that perform certain tasks automatically. How do they know from the images provided to them that in which direction should they move? They can catch things and carry them. They do different things the way human beings do. Obviously, there is a software working internally in robots' body, which is doing all this controlling part and vision to the robot. This is very complex problem and broad area of *Computer Science* and *Electrical Engineering* called *Robotics* (*Computer Vision* in particular).

Consider the image below:



Fig 33.1

This image is black and white, consisting of five non-overlapping black colored regions of different shapes, called blobs. These blobs are two ellipses- n and u shaped (two blobs) and one arc at the bottom. We can see these five blobs. How can robot identify them? So the problem is:

- We want to *partition* the pixels into *disjoint sets*, one set per blob.

If we make one set per blob, there will be five sets for the above image. To understand the concept of disjoint sets, we can take an analogy with two sets- A and B (as in Mathematics) where none of the elements inside set A is present in set B. The sets A and B are called disjoint sets.

Another problem related to the Computer Vision is the *image segmentation problem*. See the image below on the left of an old ship in gray scales.

**Fig 33.2**

We want to find regions of different colors in this picture e.g. all regions in the picture of color black and all regions in the image of color gray. The image on the right represents the resultant image.

Different scanning processes carried out in hospitals through MRI (Magnetic Resonance Imaging), CAT *Scan* or CT *Scan* views the inner parts of the whole body of human beings. These scanned images in gray scales represent organs of the human body. All these are applications of *Disjoint Set ADT*.

Equivalence Relations

Let's discuss some Mathematics about sets. You might have realized that Mathematics is handy whenever we perform analysis of some data structure.

- A binary relation R over a set S is called an *equivalence relation* if it has following properties:
 1. Reflexivity: for all element $x \in S$, $x R x$
 2. Symmetry: for all elements x and y , $x R y$ if and only if $y R x$
 3. Transitivity: for all elements x , y and z , if $x R y$ and $y R z$ then $x R z$
 - The relation “is related to” is an equivalence relation over the set of people.
- You are advised to read about equivalence relations yourself from your text books or from the internet.

Data Structures

Lecture No. 34

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 8
8.1, 8.2, 8.3

Summary

- Equivalence Relations
- Disjoint Sets
- Dynamic Equivalence Problem

Equivalence Relations

We will continue discussion on the abstract data structure, ‘disjointSets’ in this lecture with special emphasis on the mathematical concept of Equivalence Relations. You are aware of the rules and examples in this regard. Let’s discuss it further and define the concept of Equivalence Relations.

‘A binary relation R over a set S is called an *equivalence relation* if it has following properties’:

1. Reflexivity: for all element $x \in S$, $x R x$
2. Symmetry: for all elements x and y , $x R y$ if and only if $y R x$
3. Transitivity: for all elements x , y and z , if $x R y$ and $y R z$ then $x R z$

The relation “is related to” is an equivalence relation over the set of people. This is an example of Equivalence Relations. Now let’s see how the relations among people satisfy the conditions of Equivalence Relation. Consider the example of Haris, Saad and Ahmed. Haris and Saad are related to each other as brother. Saad and Ahmed are related to each other as cousin. Here Haris “is related to” Saad and Saad “is related to” Ahmed. Let’s see whether this binary relation is Equivalence Relation or not. This can be ascertained by applying the above mentioned three rules.

First rule is reflexive i.e. for all element $x \in S$, $x R x$. Suppose that x is Haris so $Haris R Haris$. This is true because everyone is related to each other. Second is Symmetry: for all elements x and y , $x R y$ if and only if $y R x$. Suppose that y is Saad. According to the rule, $Haris R Saad$ if and only if $Saad R Haris$. If two persons are related, the relationship is symmetric i.e. if I am cousin of someone so is he. Therefore if Haris is brother of Saad, then Saad is certainly the brother of Haris. The family relationship is symmetric. This is not the symmetric in terms of respect but in terms of relationship. The transitivity is: ‘for all elements x , y and z . If $x R y$ and $y R z$, then $x R z$ ’. Suppose x is Haris, y is Saad and z is Ahmed. If Haris “is related to” Saad, Saad “is related to” Ahmed. We can deduce that Haris “is related to” Ahmed. This is also true in relationships. If you are cousin of someone, the cousin of that person is also related to you. He may not be your first cousin but is related to you.

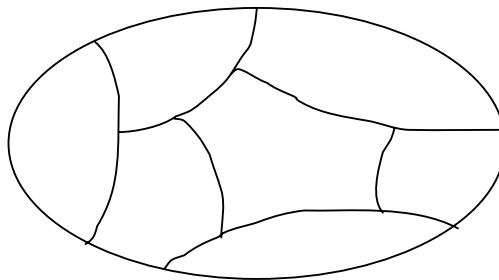
Now we will see an example of binary relationship that is not based on equivalence relationship. The $\leq$ relationship is *not* an equivalence relation. We will prove this by applying the three rules. The first rule is reflexive. It is reflexive, since $x \leq x$, as x is not less than x but surely is equal to x . Let's check the transitive condition. Since $x \leq y$ and $y \leq z$ implies $x \leq z$, it is also true. However it is *not symmetric* as $x \leq y$ does not imply $y \leq x$. Two rules are satisfied but symmetric rule does not. Therefore $\leq$ is not an equivalence relation.

Let's see another example of binary relationship that is also equivalence relation. This is from the electric circuit domain. Electrical connectivity, where all connections are by metal wires, is an equivalence relation. You can make the circuit diagram on the paper including transistor, resistors and capacitors etc. These parts are connected to each other with the metal wire. Now let's apply the rules of equivalence relations on it. It is clearly reflexive, since the component is connected to itself. In a circuit, a transistor is connected to itself. It is symmetric due to the fact that if component a is connected to component b , then b must be electrically connected to a . Suppose we have two capacitors a and b . If capacitor a is connected to b , capacitor b is also connected to a . It is also transitive. If component a is connected to component b and b is connected to c , then a is connected to c . This is also true in electrical connectivity. All the three rules of equivalence relations satisfy in this case. Therefore this is equivalence relation.

Disjoint Sets

In the beginning of the lecture, it was told that equivalence relationship partitioned the set. Suppose we have a set of elements and a relation which is also equivalence relation. Let's see what it does mathematically to the set. An equivalence relation R over a set S can be viewed as a partitioning of S into disjoint sets. Here we have an equivalence relationship which satisfies the three conditions. Keep in mind the examples of family relationships or electrical circuits. Here the second point is that each set of the partition is called an *equivalence class* of R (all elements that are related).

Consider the diagram below. We have a set in the shape of an ellipse.



This set has been partitioned into multiple sets. All these parts are disjoint sets and belong to an equivalence class.

Let's discuss an example to understand it. Suppose there are many people around you. How can we separate those who are related to each other in this gathering? We make

groups of people who are related to each other. The people in one group will say that they are related to each other. Similarly we can have many groups. So every group will say that they are related to each other with family relationship and there is no group that is related to any other group. The people in the group will say that there is not a single person in the other group who is related to them. There is a possibility that a boy from one group marries to a girl from the other group. Now a relation has been established between these two groups. The people in both groups are related to each other due to this marriage and become a bigger family. With the marriages people living in different families are grouped together. We will do operations like this in case of disjoint sets by combining two sets. You must be aware of the union and intersection operations in sets.

Every member of S appears in exactly one equivalence class. We have divided a set into many disjoint sets. One member of the set can appear in only one equivalence class. Keeping in mind the example of family relations, if a person belongs to a group he cannot go to some other group. The second point is to decide if $a R b$. For this purpose, we need only to check whether a and b are in the same equivalence class. Consider the example of pixels. If a pixel $p1$ is in a region and want to know the relation of pixel $p2$ with it, there is only the need to confirm that these two pixels are in the same region. Lastly, this provides a strategy to solve the equivalence problem. With the help of second point we can get help to solve the equivalence problem. So far, we have not seen the disjoint sets and its data structure.

Dynamic Equivalence Problem

Let's talk about the data structure. Suppose we have a set and converted into disjoint sets. Now we have to decide if these disjoint sets hold equivalence relation or not. Keep in mind the example of family relations in which we had divided the people into groups depending upon their relations. Suppose we have to decide that two persons belong to the same family or not. You have to make this decision at once. We have given a set of people and know the binary relation between them i.e. a person is a cousin of other, a person is brother of other etc. So the relation between two persons is known to us. How we can take that decision? What is the data available to us? We have people (with their names) and the binary relation among them. So we have names and the pair of relations. Think that Haris and Ahmed are related to each other or not i.e. they belong to the same family or not. How can we solve this? One way to find out the relationship is given below.

If the relation is stored as a two-dimensional array of booleans, this can be done in constant time. The problem is that the relation is usually not explicitly defined, but shown in implicit terms. Suppose we are given 1000 names of persons and relations among them. Here we are not talking about the friendship as it is not a family relation. We are only talking about the family relations like son-father, brother-brother, cousin-cousin, brother-sister, etc. Can we deduce some more relations from these given relations? Make a two dimensional array, write the names of persons as the columns and rows headings. Suppose Haris is the name of first column and first row. Saad is the name of 2nd col and 2nd row. Similarly Ahmed, Omar, Asim and Qasim are in the 3rd, 4th, 5th, and 6th columns and rows respectively. The names are arranged in the same way in columns and rows.

	H a a r i s	S a a d	A h m e d	O m a r	A s i m	Q a s i m		
Haaris	<i>T</i>	<i>T</i>	<i>T</i>					
Saad		<i>T</i>	<i>T</i>					
Ahmed			<i>T</i>					
Omar				<i>T</i>				
Asim					<i>T</i>	<i>T</i>		
Qasim						<i>T</i>		

Now visit each cell of matrix. If two persons are related, store T(true) in the corresponding column and row. As Haris and Saad are related to each other, so we take the row of Haris and column of Saad and store T in that cell. Also take the row of Saad and column of Ahmed and mark it with T. By now, we do not know that Haris and Ahmed are related so the entry in this regard will be false. We cannot write T(True) here as this relationship has not been stated explicitly. We can deduce that but cannot write it in the matrix. Take all the pairs of relations and fill the matrix. As Haris is related to Haris so we will write T in the Haris row and Haris column. Similarly, the same can be done in the Saad row and Saad column and so on. This will be square matrix as its rows and columns are equal. All of the diagonal entries are T because a person is related to himself. This is a self relation. Can we find out that Haris and Ahmed are related? We can find the answer of this question with the help of two relations between Haris & Saad and Saad & Ahmed. Now we can write T in the Haris row and Ahmed column. The problem is that this information was not given initially but we deduced this from the given information. We have 1000 people so the size of the matrix will be 1000 rows * 1000 columns. Suppose if we have 100,000 people, then the size of the array will be 100,000*100,000. If each entry requires one byte then how much storage will be required to store the whole array? Suppose if we have one million people, the size will be 10 to the power 14. Do we have as much memory available in the computer? We have one solution but it is not efficient in terms of memory. There is always need of efficient solution in terms of space and time. For having a fast algorithm, we can compromise on the space. If we want to conserve memory, a slow algorithm can be used. The algorithm we discussed is fast but utilized too much space.

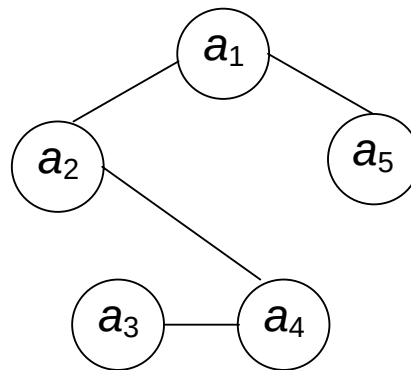
Let's try to find out some more efficient solution of this problem. Consider another example. Suppose the equivalence relation is defined over the five-element set {a1,

$a2, a3, a4, a5\}$. These elements can be the name of people, pixels, electrical components etc. How many pairs we can have in this set? We have a pair $a1-a1, a1-a2, a1-a3, a1-a4, a1-a5, a2-a2, a2-a3$ etc. The pair $a1-a2$ and $a2-a1$ are equal due to symmetric. We also find some self-pairs (i.e. $a1-a1, a2-a2$ etc). There are 25 pairs of elements, each of which is related or not ($30 \text{ pairs} - 5 \text{ self-pairs} = 25$). These are the total pairs and we may not have as much relations. What will be size of array with five elements? The size will be $5*5 = 25$. We are also given relations i.e.

- $a1 R a2$,
- $a3 R a4$,
- $a5 R a1$,
- $a4 R a2$,

We have five people in the example and their relation is given in four pairs. If two persons are brothers, then cousin of one brother is also the cousin of other. We can find out this information with the help of a matrix. We would like to be able to infer this information quickly.

We made nodes of each element. These five nodes are as $a1, a2, a3, a4, a5$.



As $a1 R a2$, so we link the nodes $a1$ and $a2$. Similarly $a3 R a4$, these two nodes are also linked. Following this pattern, it is established that $a5 R a1$ and $a4 R a2$. So we connect these nodes too. It is clear from the figure that all of these nodes are related to each other. If they are related to each other as cousin, then all of these five persons belong to the same family. They need to be in the same set. They are not in disjoint sets. Is $a3 R a5$? You can easily say yes. How you get this information? This relation is not provided in the given four relations. With the above tree or graph we can tell that $a3$ is related to $a5$. We can get the information of relationship between different persons using these nodes and may not need the matrix.

We want to get the information soon after the availability of the input data. So the data is processed immediately. The input is initially a collection of n sets, each with one element. Suppose if we have 1000 people, then there will be need of 1000 sets having only one person. Are these people related to each other? No, because every person is in different set. This initial representation is that all relations (except reflexive relations) are false. We have made 1000 sets for 1000 people, so only the reflexive relation (every person is related to himself) is true. Now mathematically speaking, each set has a different element so that $S_i \cap S_j = \emptyset$ which makes the sets

disjoint. A person in one set has no relation with a person in another set, therefore their intersection is null. Now here we have 1000 sets each containing only one person. Only the reflexive relation is true and all the 1000 sets are disjoint. If we take intersection of any two sets that will be null set i.e. there is no common member in them.

Sometimes, we are given some binary relations and are asked to find out the relationship between two persons. In other words, we are not given the details of every pair of 1000 persons. The names of 1000 persons and around 50 relations are provided. With the help of these 50 relations, we can find out more relations between other persons. Such examples were seen above while finding out the relationship with the help of graph.

There are two permissible operations in these sets i.e. *find* and *union*. In the *find* method, we are given one element (name of the person) and asked to find which set it belongs to. Initially, we have 1000 sets and asked in which set person 99 is? We can say that every person is in a separate set and person 99 is in set 99. When we get the information of relationships between different persons, the process of joining the sets together can be started. This is the union operation. When we apply union operation on two sets, the members of both sets combined together and form a new set. In this case, there will be no duplicate entry in the new sets as these were disjoint. The definitions of *find* and *union* are:

- *Find* returns the name of the set (equivalence class) that contains a given element, i.e., $S_i = \text{find}(a)$
- *Union* merges two sets to create a new set $S_k = S_i \cup S_j$.

We give an element to the *find* method and it returns the name of the set. The method *union* groups the member of two sets into a new set. We will have these two operations in the disjoint abstract data type. If we want to add the relation $a R b$, there is need to see whether a and b are already related. Here a and b may be two persons and a relation is given between them. First of all we will see that they are already related or not. This is done by performing *find* on both a and b to check whether they are in the same set or not. At first, we will send a to the *find* method and get the name of its set before sending b to the *find* method. If the name of both sets is same, it means that these two belong to the same set. If they are in the same set, there is a relation between them. We did not get any useful information with this relation. Let's again take the example of the Haris and Saad. We know that Haris is the brother of Saad, so they can be placed into a new set. Afterwards, we get the information that Saad is the brother of Haris.

Now the question arises, if they are not in the same set, what should we do? We will not waste this information and apply *union* which merges the two sets a and b into a new set. This information will be helpful later on. We keep on building the database with this information. Let's take the example of pixels. Suppose we have two pixels and are told that these two belong to the same region, have same color or these two pixels belong to the liver in CT scan. We will keep these two pixels and put them in one set. We can name that set as *liver*. We will use union operation to merge two sets. The algorithm to do this is frequently known as *Union/Find* for this reason.

There are certain points to note here. We do not perform any operations comparing the relative values of set elements. That means that we are not considering that one person is older than the other, younger or one electrical component is bigger than other etc. We merely require knowledge of their location, i.e., which set an element, belongs to.

Let's talk about the algorithm building process. We can assume that all elements are numbered sequentially from 1 to n . Initially we have $S_i = \{i\}$ for $i = 1$ through n . We can give numbers to persons like person1, person2 etc. Consider the example of jail. In a jail, every prisoner is given a number. He is identified by a number, not by the name. So you can say that persons in a cell of jail are related. We will use these numbers to make our algorithm run faster. We can give numbers to every person, every pixel, each electrical component etc. Under this scheme, the member of S_{10} is number 10. That number 10 can be of a person, a pixel or an electrical component. We are concerned only with number.

Secondly, the name of the set returned by *find* is fairly arbitrary. All that really matters is that $find(x) = find(y)$ if and only if x and y are in the same set. We will now discuss a solution to the *union/find* problem that for any sequence of at most m finds and up to $n-1$ unions will require time proportional to $(m + n)$. We are only storing numbers in the sets and not keeping any other relevant information about the elements. Similarly we are using numbers for the names of the sets. Therefore we may want that the method *find* just returns the number of the set i.e. the person 10 belongs to which set number. The answer may be that the person 10 belongs to set number 10. Similarly the set number of person 99 may be set 99. So if the set numbers of two persons is not equal, we can decide that these two persons do not belong to the same set. With the help of this scheme, we will develop our algorithm that will run very fast. In the next lecture, we will discuss this algorithm and the data structure in detail.

Data Structures

Lecture No. 35

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 8

8.2, 8.3

Summary

- Dynamic Equivalence Problem
- Example 1
- Parent Array
 - o Initialization
 - o Find (i)
 - o Union (i, j)
- Example 2
 - o Initialization
 - o union operation
 - o find Operation
- Running Time Analysis

Before talking about the data structure implementation in detail, we will recap the things discussed in the previous lecture. The concepts that came under discussion in the last lecture included the implementation of the disjoint set. It was observed that in case of data elements, we assign a unique number to each element to make sets from these numbers. Moreover, techniques of merger and searching in these sets by using the operations union and find were, talked about. It was seen that if we send a set item or a number to *find* method, it tells us the set of which this item is a member. It will return the name of the set. This name of set may also be a number. We talked about the *union* method in which members of two sets join together to form a new set. The *union* method returns the name or number of this new set.

Now we will discuss the data structure used to implement the disjoint sets, besides ascertaining whether this data structure implements the find and union operations efficiently.

Dynamic Equivalence Problem

We are using sets to store the elements. For this purpose, it is essential to remember which element belongs to which set. We know that the elements in a set are unique and a tree is used to represent a set. Each element in a tree has the same root, so the root can be used to name the set. The item (number) in the root will be unique as the set has unique values of items. We use this root as the name of set for our convenience. Otherwise, we can use any name of our choice. So the element in the root (the number) is used as the name of the set. The *find* operation will return this name. Due to the presence of many sets, there will be a collection of trees. In this collection, each tree will be representing one set. If there are ten elements, we will

have ten sets initially. Thus there will be ten trees at the beginning. In general, we have N elements and initially there will be N trees. Each tree will have one element. Thus there will be N trees of one node. Now here comes a definition for this collection of trees, which states that a collection of trees is called a *forest*.

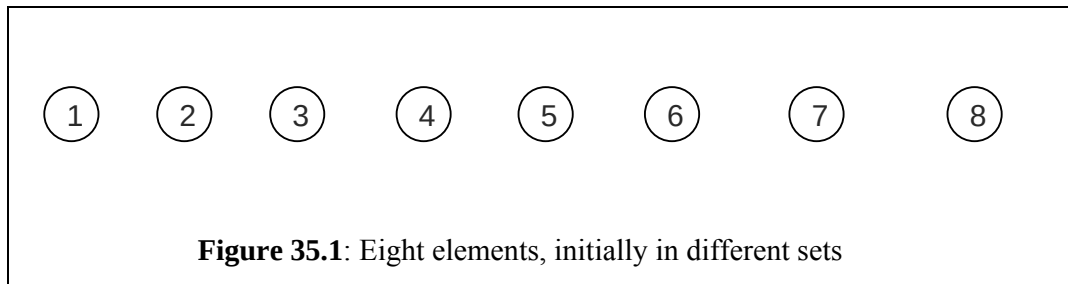
The trees used for the sets are not necessarily binary trees. That means it is not necessary that each node should have a maximum of two children nodes. Here a node may have more than two children nodes.

To execute the union operation in two sets, we merge the two trees of these sets in such a manner that the root of one tree points to the root of other. So there will be one root, resulting in the merger of the trees. We will consider an example to explain the union operation later.

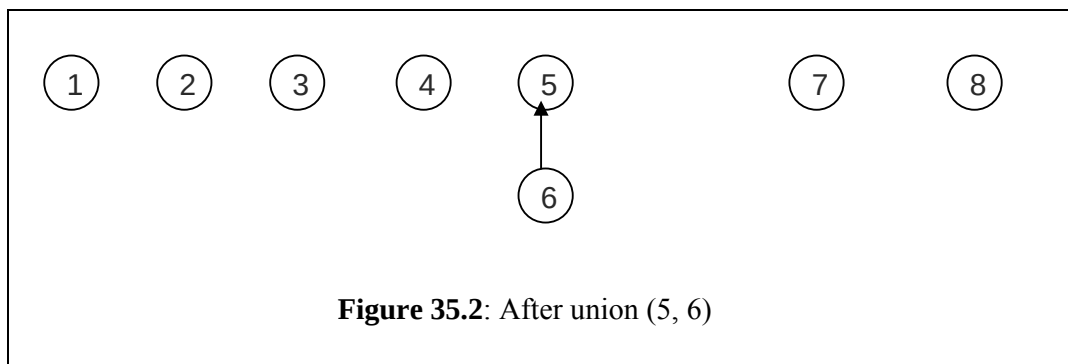
In the find operation, when we call *find* (x), it helps us to know which set this x belongs to. Internally, we find this x in a tree in the *forest*. When this x is found in a tree the *find* returns the number at root node (the name of the set) of that tree.

Example 1

Now let's consider an example to understand this phenomenon. Suppose, we have developed a data structure and apply the union and find operations on it. There are eight elements in this data structure i.e. 1 to 8. These may be the names of people to which we have assigned these numbers. It may be other distinct eight elements. We will proceed with these numbers and make a set of each number. The following figure shows these numbers in different sets.

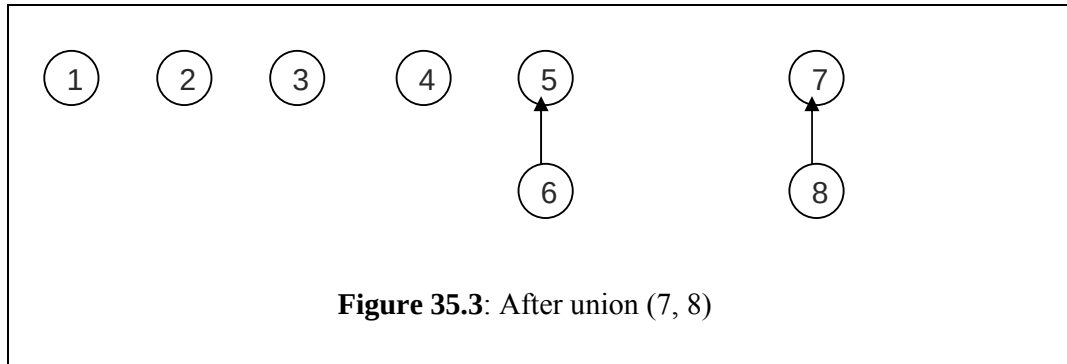


Now we carry out union operation on the sets 5 and 6. The call *union*(5,6) means, merge the sets 5 and 6 and return the new set developed due to the merger of the sets-5 and 6. In the following figure, we see this union. We put the set 6 below set 5, which join together.

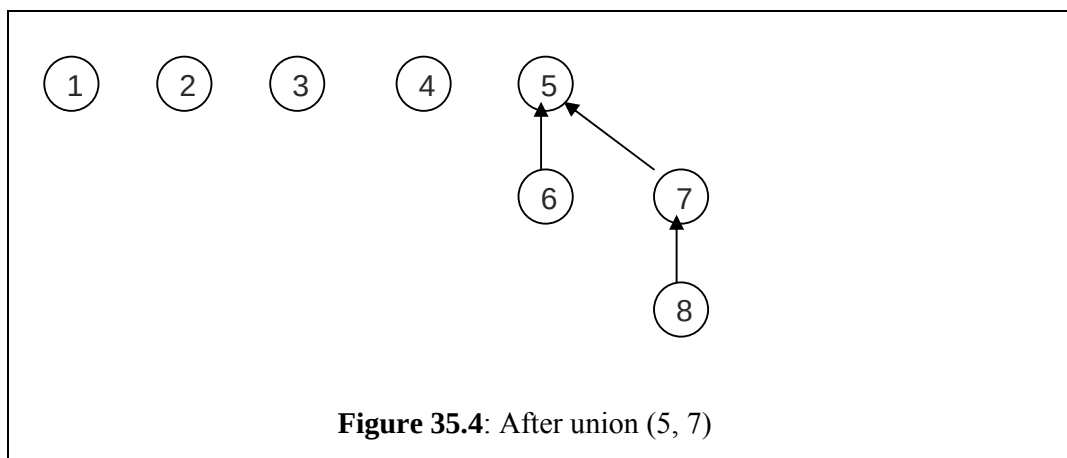


After the merger, the new set contains two members 5 and 6. Now the question arises what will be the name of this new set. In the above union operation, the set 6 becomes

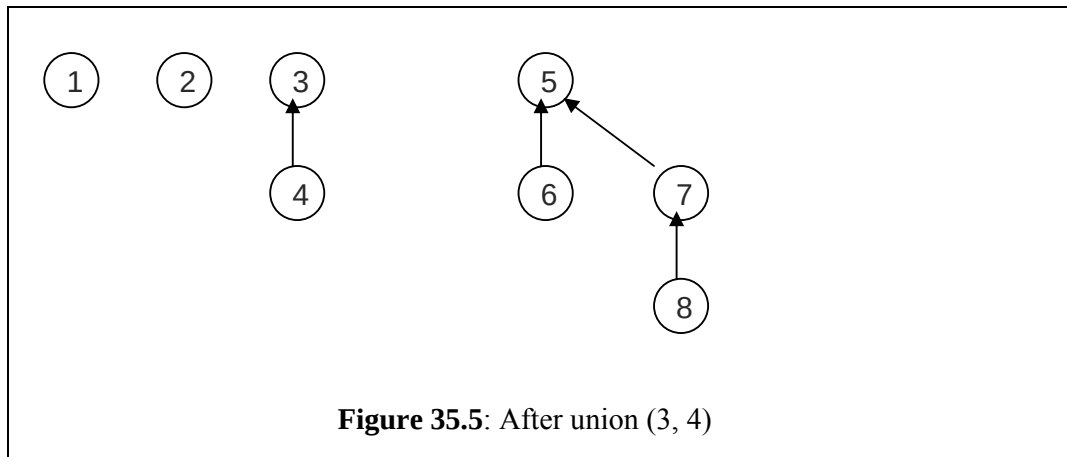
the node of 5. It may be in reverse i.e. 5 could be a node of 6. In the above figure, we put 6 as node of 5. Moreover the arrow joining these is from 6 to 5. In the new set formed due to the merger of 5 and 6, it seems that 5 has some superiority. So the name of the new set is 5. We passed two arguments 5 and 6 to the *union* function. And the union made 6 a member of 5. Thus, if we pass S1 and S2 to the *union* function, the *union* will make S2 a member of S1. And the name of the new set will be S1. That means the name of first argument will be the name of the new set. Now we call *union* (7,8). After this call, 7 and 8 form a new set in which 8 is a member of 7 and the name of this new set is 7 (that is the first argument in the call). In other words, 7 is root and 8 its child. This is shown in the following figure.



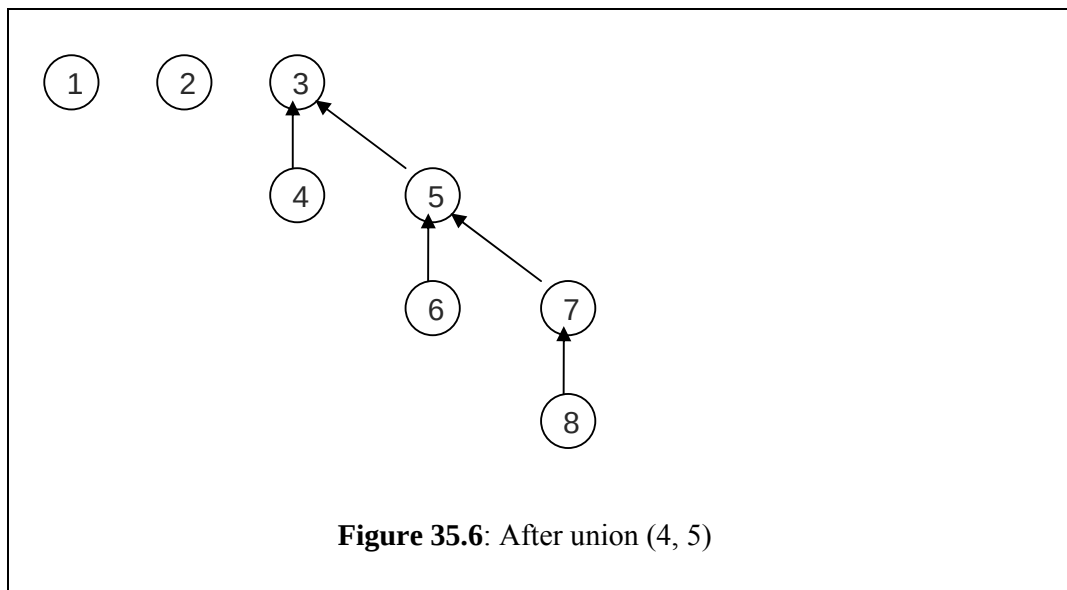
Now we call the union function by passing it set 5 and set 7 as arguments i.e. we call *union* (5,7). Here the sets 5 and 7 have two members each. 5 and 6 are the members of 5 and similarly the two members of 7 are 7 and 8. After merging these sets, the name of the new set will be 5 as stated earlier that the new set will be named after the first argument. The following figure (figure 35.4) shows that now in the set 5, there are four members i.e. 5, 6, 7 and 8.



We can see that there are four unique set members in this set (i.e. 5). We will demonstrate the union operation with the help of another example. Suppose, we have made another union that is *union* (3,4). This call merges the set 4 with the set 3. The following figure shows this.



In this figure, we see that there are four sets that are 1, 2, 3 and 5. Now we unite the sets 4 and 5 and make a call *union* (4,5). The following figure shows the outcome of this call. In this figure, the set 5 points to 3 whereas we made a call *union* (4,5).



So we conclude that it is not necessary that the caller should send the roots to union. It is necessary that the union function should be such that if a caller sends elements of two sets, it should find the sets of those elements, merge them and return the name of new set. Thus our previous call i.e. *union* (4,5) was actually carried out in the way that first the *union* finds the root of 4 that is 3. Then it looks for 5 that itself is a root of its set. After this, it merges both the trees (sets). This merger is shown in the above figure i.e. Figure 35.6.

Up to now, we have come to know that the formation of this tree is not like a binary tree in which we go down ward. Moreover, a binary tree has left and right children that are not seen yet in the tree we developed. This is a tree like structure with some properties.

Let's talk about these properties.

Here we see that typical tree traversal (like inorder, preorder or postorder) is not required. So there is no need for pointers to point to the children. Instead, we need a pointer to parent, as it's an up-tree. We call it up-tree due to the fact that it is such a tree structure in which the pointers are upward. These parent pointers can be stored in an array. Here we have to keep (and find) pointer to the parent of a node unlike a binary tree in which we keep pointer to child nodes. In the array, we will set the parent of root to -1 . We can write it as

Parent[i] = -1 // if i is the root

Now we will keep these tree structures (forest) in an array in the same manner. With the merger of the trees, the parents of the nodes will be changed. There may be nodes that have no parent (we have seen this in the previous example). For such nodes, we will keep -1 in the array. This shows that this node has no parent. Moreover, this node will be a root that may be a parent of some other node.

Now we will develop the algorithm for *find* and *union*. Let's consider an example to see the implementation of this disjoint set data structure with an array.

Parent Array

Initialization

We know that at the start we have n elements. These n elements may be the original names of the elements or unique numbers assigned to them. Now we take an array and with the help of a *for* loop, keep these n elements as root of each set in the array. These numbers are used as the index of the array before storing -1 at each location from index zero to n . We keep -1 to indicate a number as root. In code, this *for* loop is written as under.

```
for ( i = 0; i < n ; i ++)  
    Parent [i] = -1 ;
```

Find (i)

Now look at the following code of a loop. This loop is used to find the parent of an element or the name of the set that contains that element.

```
// traverse to the root (-1)  
for(j=i; parent[j] >= 0; j=parent[j])  
    ;  
return j;
```

In this loop, i is an argument passed to the find function. The execution of the loop starts from the value, passed to the find function. We assign this value to a variable j and check whether its parent is greater than zero. It means that it is not -1 . If it is greater than zero, its parent exists, necessitating the re-initialization of the j with this parent of j for the next iteration. Thus this loop continues till we find parent of j (parent [j]) less than zero (i.e. -1). This means that we come to the root before

returning this number j .

Union (i, j)

Now let's see the code for the function of union. Here we pass two elements to the function union. The union finds the roots of i and j . If i and j are disjoint sets, it will merge them. Following is the code of this function.

```

root_i = find(i);
root_j = find(j);
if (root_i != root_j)
    parent[root_j] = root_i;

```

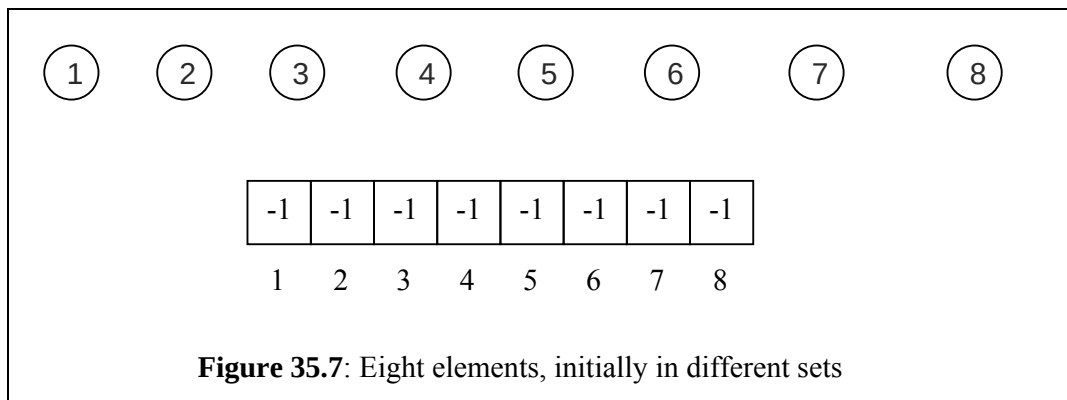
In the code, we see that at first it finds the root of tree in which i exists by the $\text{find}(i)$ method and similarly finds the root of the set containing j by $\text{find}(j)$. Then there is a check in if statement to see whether these sets (roots) are same or not. If these are not the same, it merges them in such a fashion that the root i is set as the parent of root j . Thus, in the array where the value of root j exists, the value of root i becomes there.

Example 2

To understand these concepts, let's consider an example. We re-consider the same previous example of eight numbers and see how the initialization, union and find work.

Initialization

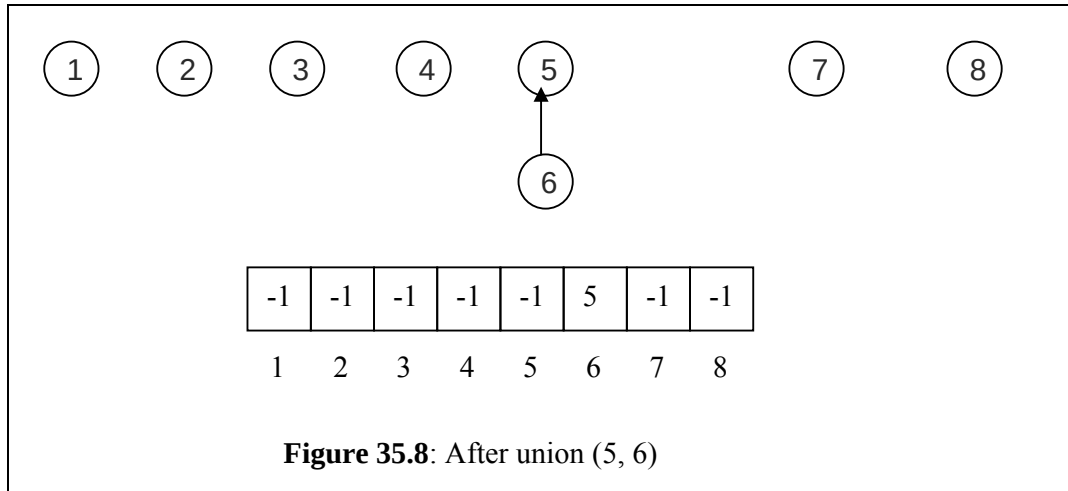
In the following figure (figure 35.7), we have shown the initialization step. Here we make an array of eight locations and have initialized them with -1 as these are the roots of the eight sets. This -1 indicates that there is no parent of this number. We start the index of array from 1 for our convenience. Otherwise, we know that the index of an array starts from zero.



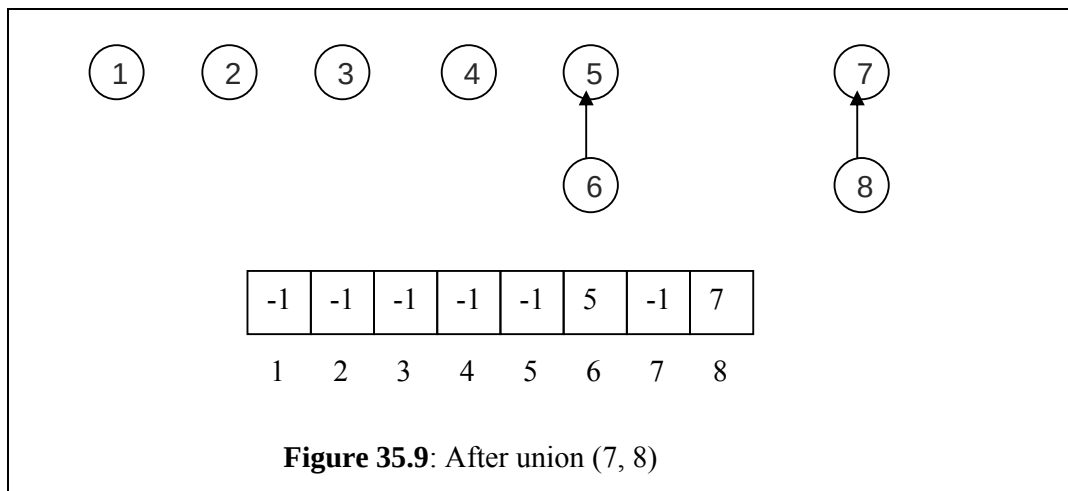
Union Operation

Now we come to the union operation. First of all, we call union (5,6). This union operation forms an up-tree in which 5 is up and 6 down to it. Moreover 6 is pointing

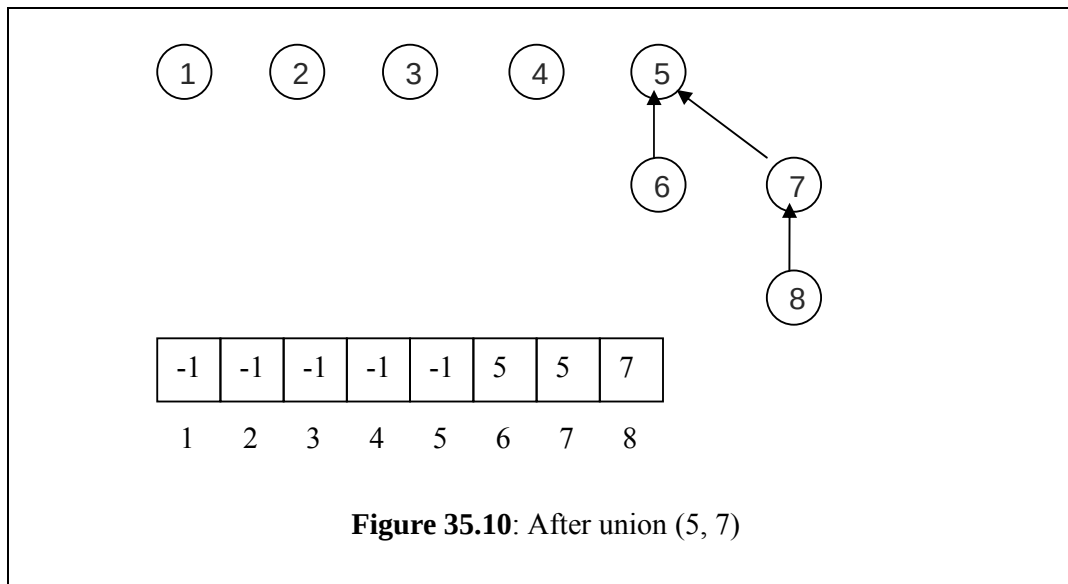
to 5. In the array, we put 5 instead of -1 at the position 6. This shows that now the parent of 6 is 5. The other positions have -1 that indicates that these numbers are the roots of some tree. The only number, not a root now is 6. It is now the child of 5 or in other words, its parent is 5 as shown in the array in the following figure.



Now we carry out the same process (i.e. union) with the numbers 7 and 8 by calling `union (7,8)`. The following figure represents this operation. Here we can see that the parent of 8 is 7 and 7 itself is still a root.

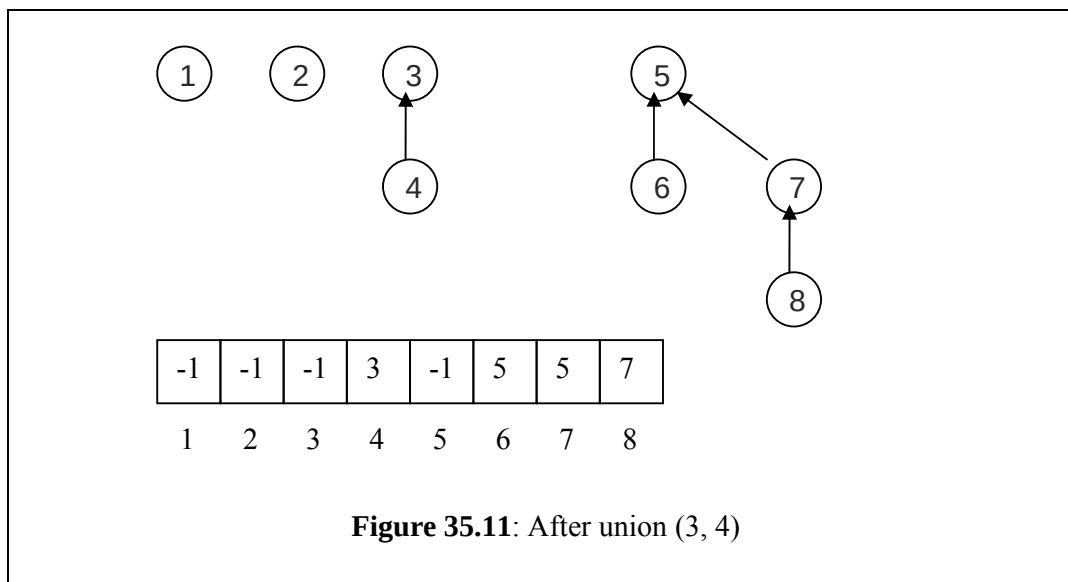


Now we execute the `union (5,7)`. The following figure represents the tree and array status after the performance of this union operation.



The tree in the figure shows that 7 points to 5 now. In the array, we see that the value at position 7 is 5. This means that the parent of 7 is 5 now. Whereas the parent of 5 is still -1 that means it is still a root. Moreover, we can see that the parent of 8 is 7 as before.

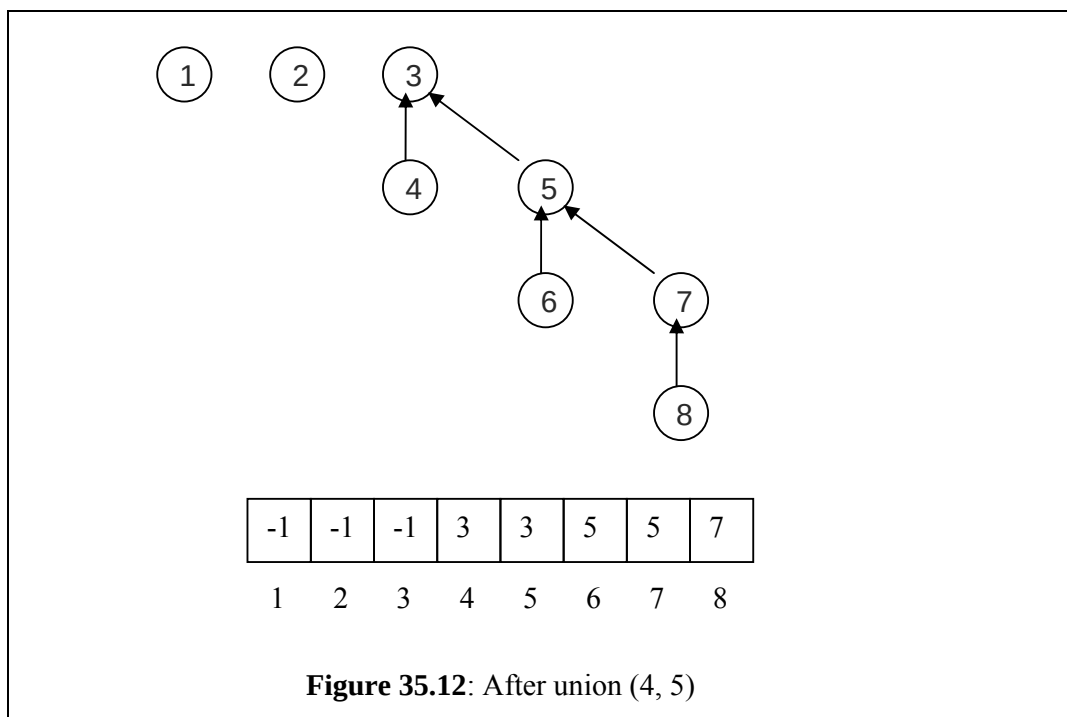
Afterwards, we call *union* (3,4). The effect of this call is shown in the following figure. Here in the array at position 4, there is 3 instead of -1. This shows that the parent of 4 is 3.



By looking at the array only, we can know that how many trees are there in the collection (forest). The number of trees will be equal to the number of -1 in the array. In the above figure, we see that there are four -1 in the array so the number of trees is

four and the trees in the figure confirms this. These four trees are with the roots 1, 2 3 and 5 respectively.

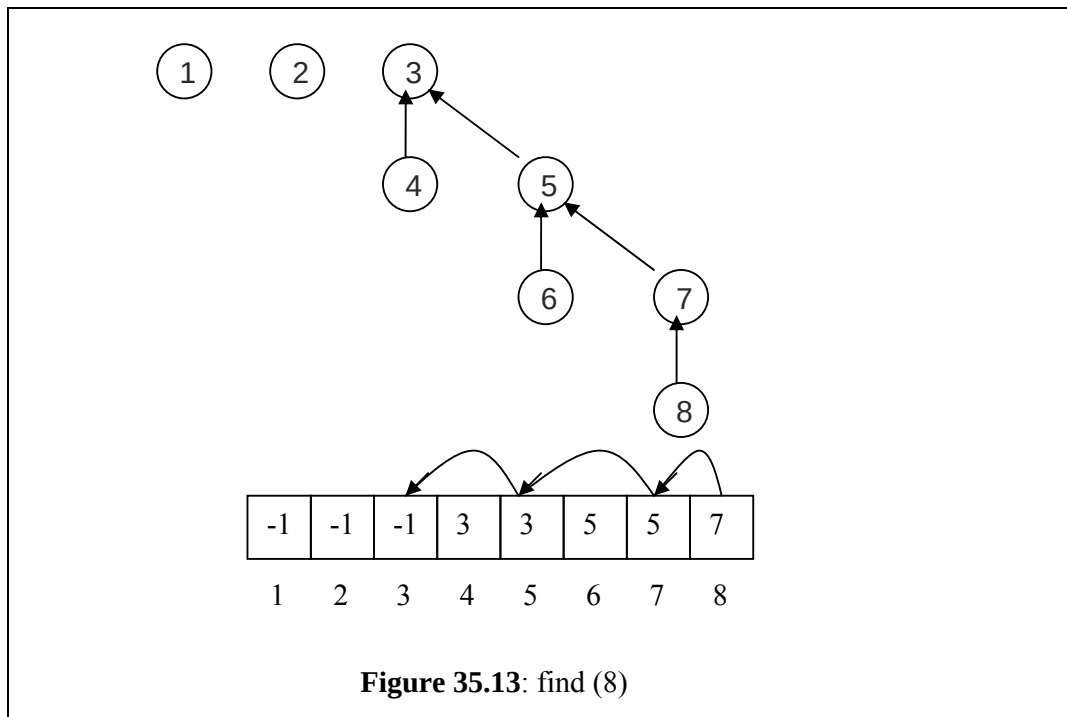
Now we carry out another union operation. In this operation, we do the union of 4 and 5 i.e. union (4, 5). Here the number 4 is not a root. Rather, it is an element of a set. We have already discussed that the union operation finds the set (root) of the element passed to it before putting together those sets. So here the union finds the set containing 4 that is 3. The element 5 itself is a name (root) of a set. Now the union operation merges the two sets by making the second set a member of the first set. In this call (union (4, 5)), the first set is 3 (that is found for the element 4) and second is 5. So the union joins 5 with 3 while 5 is pointing to 3. Due to this union operation, now, in the array, there is 3 at position 5 instead of -1. The 3 at this position 5 indicates that the parent of 5 is 3. The following figure shows the tree and array after the operation union (4,5).



These were the union operations while using parent array. Now let's look at the find operation.

Find Operation

To understand the find operation, we make a call *find* (8) in the above forest. This call means that the caller wants to know the set in which number 8 is lying. We know that the root of the tree is also the name of the set. By looking at the trees in the figure below, we come to know that 8 is in the tree with 3 as root. This means 8 is in set 3.



Thus from the figure, we find the set containing 8. This is being implemented with the parent array to ultimately find it in the array. For this find operation in the array, we have discussed the algorithm in which we used a *for* loop. This 'for loop' starts from the position that given to the *find* function. Here it is 8. The condition in the for loop was that as long as parent [*j*] is greater than zero, set this parent [*j*] to *j*. Now we execute the loop with the value of *j* equal to 8. First of all, the loop goes to position 8 and looks for the value of parent of 8. This value is 7, which sets the value of *j* to 7 and goes to position 7. At that position, the value of parent of 7 is 5. It goes to position 5. At position 5, the value of parent is 3. So the loop sets the value of *j* equal to 3 and goes to the position 3. Here it finds that the parent of 3 is -1 i.e. less than zero so the loop ends. This position 3 is the root and name of the set as parent of it is -1. Thus the name of the set that contains 8 is 3. The find will return 3 which mean that 8 is a member of set 3.

Similarly, in the above array, we can execute the operation *find* (6). This will also return 3 as the loop execution will be at the positions 6 before going to 5 and finally to 3. It will end here, as the parent of 3 is -1. Thus *find* (6) returns 3 as the set that contains 6.

Running Time analysis

Now we will discuss how the implementation of disjoint set is better. We must remember that while discussing the implementation of disjoint set, we talked about Boolean matrix. This is a two dimensional structure in which the equivalence relations is kept as a set of Boolean values. Here in the parent array, we also are keeping the same information. The union will be used when the two items are related. In the two-dimensional matrix, we can easily find an item by its index. Now we are

using an array that is a single dimensional structure and seems better with respect to space. We keep all the information in this array which is at first kept in a matrix. Now think about a single dimension array versus a two dimensional array. Suppose we have 1000 set members i.e. names of people. If we make a Boolean matrix for 1000 items, its size will be 1000 x 1000. Thus we need 1000000 locations for Boolean values. In case of an array, the number of locations will be 1000. Thus this use of array i.e. tree like structure is better than two-dimensional array in terms of space to keep disjoint sets and doing union and find operations. Moreover, we do not use pointers (addresses) that are used in C++. We use array indices as pointers. In case of find, we follow some indices (pointers) to find the set containing a particular member. From these points of discussion, we conclude that

- *union* is clearly a constant time operation.
- Running time of *find(i)* is proportional to the height of the tree containing node *i*.
- This can be proportional to *n* in the worst case (but not always)
- Goal: Modify *union* to ensure that heights stay small

We will discuss these points in detail in the next lecture.

Data Structures

Lecture No. 36

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 8
8.4, 8.5, 8.6

Summary

- Running Time Analysis
- Union by Size
- Analysis of Union by Size
- Union by Height
- Sprucing up find
- Timing with Optimization

Running Time Analysis

In the previous lecture, we constructed up trees through arrays to implement disjoint sets. *union* and *find* methods along with their codes were discussed with the help of figures. We also talked about the ways to improve these methods. As far as the *union* operation is concerned, considering the parent array, it was related to changing the index of the array. The root value of the merging tree was put into that. In case of *find*, we have to go to the root node following the up pointers.

While studying the balanced trees, we had constructed a binary search tree from the already sorted data. The resultant tree becomes a linked list with height N . Similar kind of risks involved here. Suppose, we have eight members i.e. 1, 2, 3, 4, 5, 6, 7 and 8 as discussed in an example in the previous lecture. Suppose their unions are performed as *union*(1,2), *union*(2,3), *union*(3,4) and so on up to *union*(7,8). When the trees of these unions are merged together, there will be a longer tree of height 8. If we perform a *find* operation for an element that is present in the lowest node in the tree then $N-1$ links are traversed. In this particular case, $8-1=7$ links.

We know that with reduced tree (with lesser height), the time required for *find* operation will also be reduced. We have to see whether it is possible to reduce the size (in terms of height) of the resultant tree (formed after unions of trees). Therefore, our goal is:

- **Goal:** Modify *union* to ensure that heights stay small

We had already seen this goal in the last slide of our previous lecture, which is given below in full:

- *union* is clearly a constant-time operation.
- Running time of *find*(i) is proportional to the height of the tree containing node i .
- This can be proportional to n in the worst case (but not always)

- Goal: Modify *union* to ensure that heights stay small

You might be thinking of employing the balancing technique here. But it will not be sagacious to apply it here. Due to the use of array, the ‘balancing’ may prove an error-prone operation and is likely to decrease the performance too. We have an easier and appropriate method, called *Union by Size* or *Union by Weight*.

Union by Size

Following are the salient characteristics of this method:

- Maintain sizes (number of nodes) of all trees, and during *union*.
- Make smaller tree, the subtree of the larger one.
- Implementation: for each root node *i*, instead of setting `parent[i]` to `-1`, set it to `-k` if tree rooted at *i* has *k* nodes.
- This is also called *union-by-weight*.

We want to maintain the sizes of the trees as given in the first point. Until now, we are not maintaining the size but only the *parent* node. If the value for *parent* in a node is `-1`, then this indicates that the node is the *root* node. Consider the code of the *find* operation again:

```
//find(i):
// traverse to the root (-1)
for(j=i; parent[j] >= 0; j=parent[j])
    ;
return j;
```

The terminating condition for loop is checking for non-negative values. At any point when the value of `parent[j]` gets negative, the loop terminates. Note that the condition does not specify exactly the negative number. It may be the number `-1`, `-2`, `-3` or some other negative number that can cause the loop to be terminated. That means we can also put the number of nodes (size) in the tree in this place. We can put the number of nodes in a tree in the *root* node in the negative form. It means that if a tree consists of six nodes, its *root* node will be containing `-6` in its *parent*. Therefore, the node can be identified as the *root* being the negative number and the magnitude (the absolute value) of the number will be the size (the number of nodes) of the tree.

When the two trees would be combined for union operation, the tree with smaller size will become part of the larger one. This will cause the reduction in tree size. That is why it is called union-by-size or union-by-weight.

Let’s see the pseudo-code of this union operation (quite close to C++ language). This contains the logic for reducing tree size as discussed above:

```
//union(i,j):
1. root1 = find(i);
2. root2 = find(j);
```

```

3.   if (root1 != root2)
4.   if (parent[root1] <= parent[root2])
5.       {
6.           //first tree has more nodes
7.           parent[root1] += parent[root2];
8.           parent[root2] = root1;
9.       }
10.  else
11.  {
12.      // second tree has more nodes
13.      parent[root2] += parent[root1];
14.      parent[root1] = root2;
15.  }

```

This operation performs the *union* of two sets; element *i*'s set and element *j*'s set. The first two lines are finding the *roots* of the sets of both the elements i.e. *i* and *j*. Line 3 is performing a check that the *root1* is not equal to *root2*. In case of being unequal, these are merged together. In next statement (line 4), the numbers in *parent* variables of *root* nodes are compared. Note that, the comparison is not on absolute values. So the tree with greater number of nodes would contain the lesser number mathematically. The condition at line 4 will be true, if the tree of *root1* is greater in size than that of *root2*. In line 7, the size of the smaller tree is being added to the size of the greater one. Line 8 is containing the statement that causes the trees to merge. The *parent* of *root2* is being pointed to *root1*. If the *root2* tree is greater in size than the *root1* tree, the if-condition at line 4 returns false and the control is transferred to *else* part of the if-condition. The *parent* of *root1* (the size of the *root1* tree) is added in the *parent* of *root2* in line 13 and the *root1* tree is merged into the *root2* tree using the statement at line 14.

Let's practice this approach using our previous example of array of eight elements.



-1	-1	-1	-1	-1	-1	-1	-1
1	2	3	4	5	6	7	8

Eight elements, initially in different sets.

Fig 36.1

These are eight nodes containing initially -1 in the *parent* indicating that each tree contains only one node. Let's unite two trees of 4 and 6. The new tree will be like the one shown in Fig 36.2.

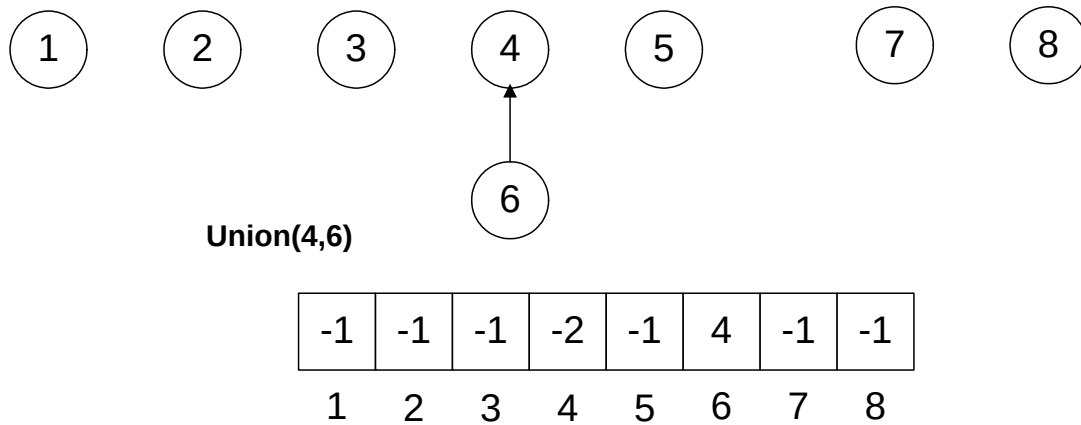


Fig 36.2

From the figure, you can see the node 6 has come down to node 4. The array also depicts that the *parent* at position 4 is containing -2 . The number of nodes has become 2. The position 6 is set to 4 indicating that *parent* of 6 is 4. Next, we similarly perform the *union* operation on nodes 2 and 3.

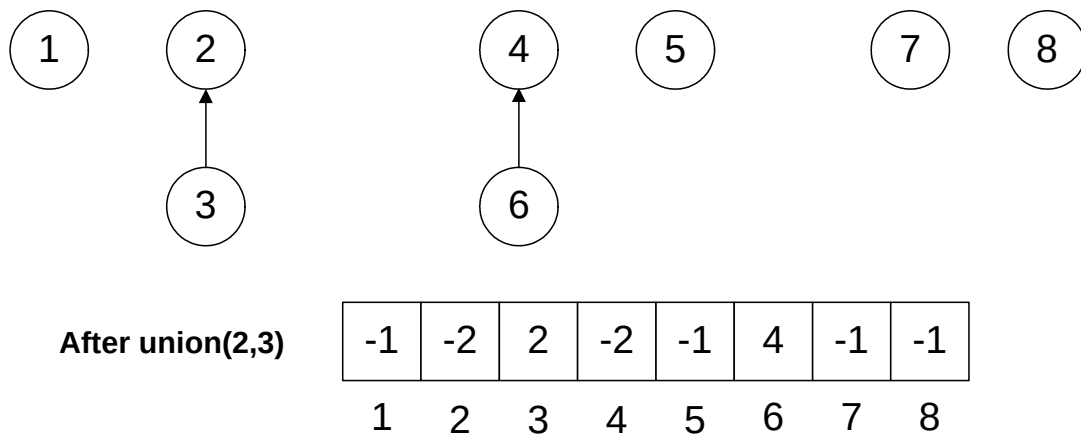


Fig 36.3

Let's perform the *union* operation further and merge the trees 1 and 4.

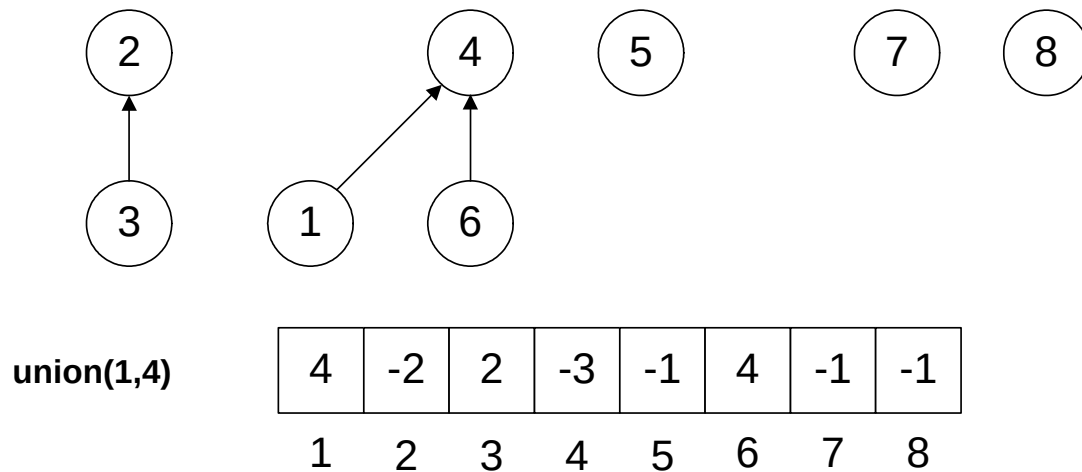


Fig 36.4

1 was a single node tree (-1 in the array at position 1) and 4 was the root node of the tree with two elements (-2 in the array at position 4) 4 and 6. As the tree with root 4 is greater, therefore, node 1 will become part of it. Previously (when the union was not based on weight), it happened contrary to it with second argument tree becoming the part of the first tree.

In this case, the number of levels in the tree still remains the same (of two levels) as that in the greater tree (with root 4). But we apply our previous logic i.e. the number of levels of the trees would have been increased from two to three after the union. Reducing the tree size was our goal of this approach.

In the next step, we merge the trees of 2 and 4. The size of the tree with root 2 is 2 (actually -2 in the parent array) while the size of the tree with root 4 is 3 (actually -3). So the tree of node 2 will be joined in the tree with root 4.

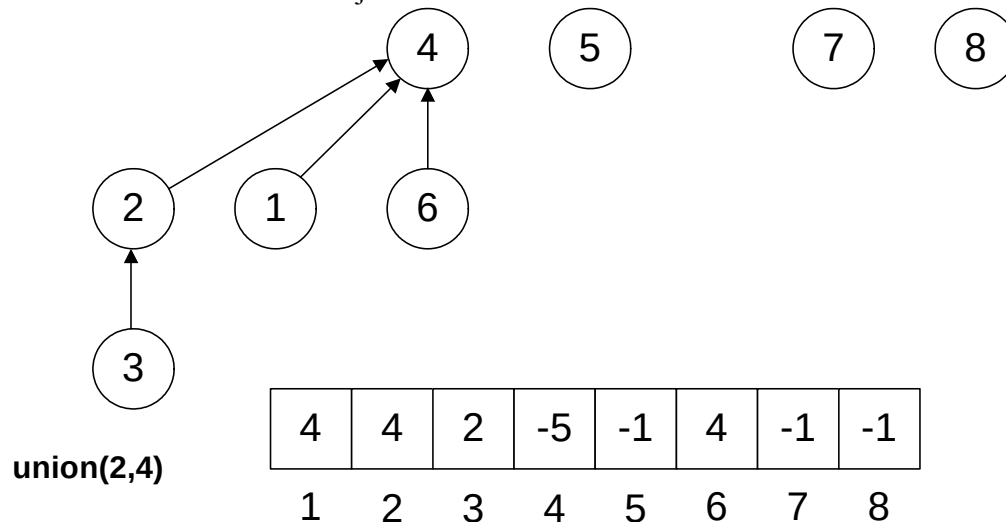


Fig 36.5

The latest values in the array i.e. at position 4, have been updated to -5 (indicating the size of the tree has reached 5). At position 2, the new value is 4, which indicates that the up node of 2 is 4.

Next, we perform the $\text{union}(5,4)$ operation. We know that the size of the tree with node 4 is 5 (-5 in the array) and node 5 is single node tree. As per our rules of union by size, the node 5 will become part of the tree 4.

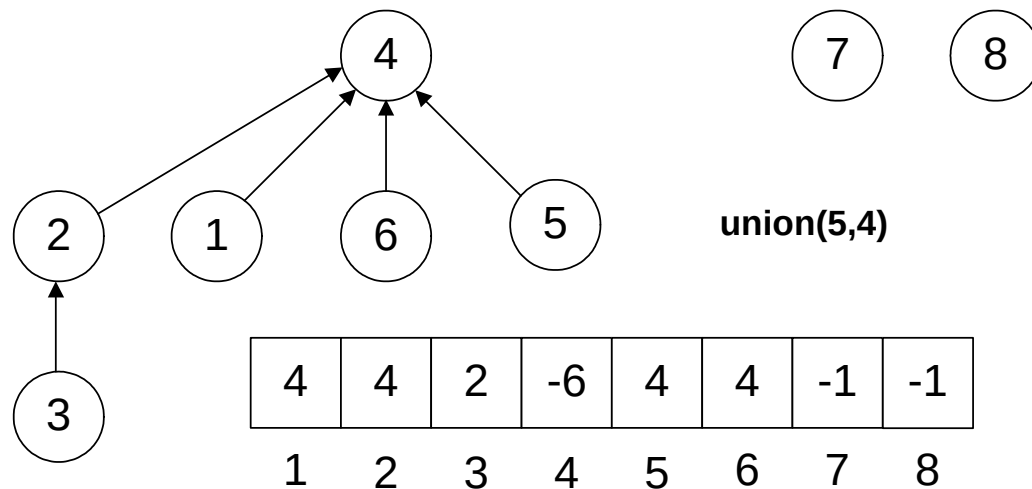


Fig 36.6

The updates inside the array can be seen as the value at position 4 has been changed to -6 and new value at position 5 is 4. This is the tree obtained after applying the *union-by-size approach*. Consider, if we had applied the previous method of union, tree's depth would have been increased more.

It seems that we have achieved our goal of reducing the tree size up to certain extent. Remember, the benefit of reducing the tree height is to increase performance while finding the elements inside the tree.

Analysis of Union by Size

- If unions are done by weight (size), the depth of any element is never greater than $\log_2 n$.

By following the previous method of union where second tree becomes the part of the first tree, the depth of the tree may extend to N . Here N is the number of nodes in the tree. But if we take size into account and perform union by size, the depth of tree is $\log_2 n$ maximum. Suppose the N is 100,000 i.e. for 100,000 nodes. The previous methods may give us a tree with depth level as 100,000. But on the other hand, $\log_2 100000$ is approximately 20. *Union-by-size* gives us a tree of 20 levels of depth maximum. So this is a significant improvement.

Mathematical proof of this improvement is very complex. We are not covering it here but only providing the logic or reasoning in intuitive proof.

Intuitive Proof:

- Initially, every element is at depth zero.
- When its depth increases as a result of a union operation (it's in the smaller tree), it is placed in a tree that becomes at least twice as large as before (union of two equal size trees).
- How often can each union be carried out? -- $\log_2 n$ times, because after at most $\log_2 n$ unions, the tree will contain all n elements.

Union by Height

- Alternative to *union-by-size* strategy: maintain heights,
- During *union*, make a tree with smaller height a subtree of the other.
- Details are left as an exercise.

This is an alternate way of *union-by-size* that we maintain heights of the trees and join them based on their heights. The tree with smaller height will become part of the one with greater height. This is quite similar with the *union-by-size*. In order to implement Disjoint Set ADT, any of these solutions can work. Both the techniques i.e. *union-by-size* and *union-by-height* are equivalent, although, there are minor differences when analyzed thoroughly.

Now, let's see what can we do for *find* operation to make it faster.

Sprucing up Find

- So far we have tried to optimize *union*.
- Can we optimize *find*?
- Yes, it can be achieved by using *path compression* (or compaction).

Considering performance, we can optimize the *find* method. We have already optimized the trees *union* operation and now we will see how can optimize the *find* operation using the *path compression*.

- During *find(i)*, as we traverse the path from i to *root*, update *parent* entries for all these nodes to the *root*.
- This reduces the heights of all these nodes.
- Pay now, and reap the benefits later!
- Subsequent *find* may do less work.

To understand the statements above, let's see the updated code for *find* below:

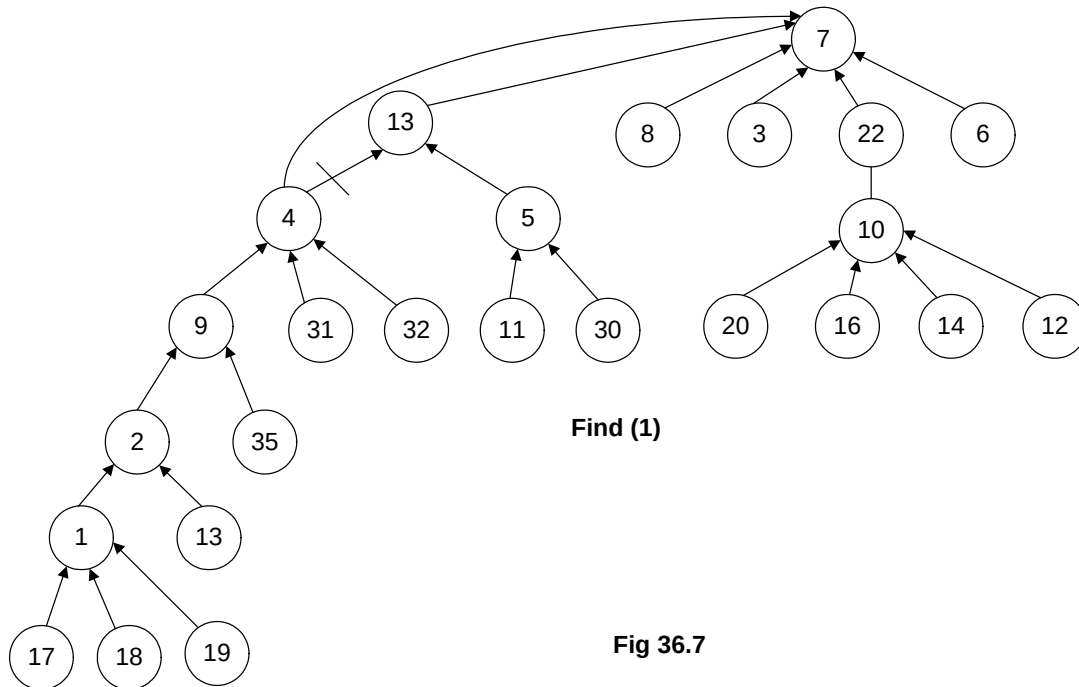
```

find (i)
{
    if (parent[i] < 0)
        return i;
    else
        return parent[i] = find(parent[i]);
}

```

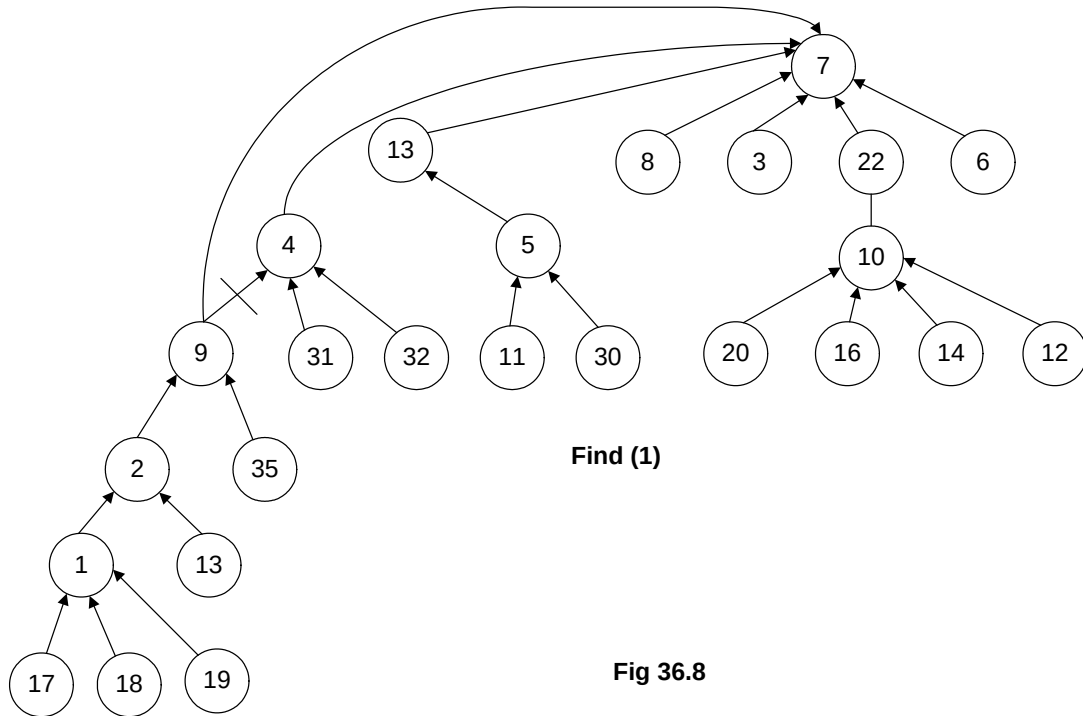
}

We have modified the code in the body of the *find*. This implementation is of recursive nature. *parent[i]* negative means that we have reached the *root* node. Therefore, we are returning *i*. If this is not the *root* node, then *find* method is being called recursively. We know that *parent[i]* may be positive or negative. But in this case it will not be negative due to being an index. To understand this recursive call, we elaborate it further with the help of figure. We call the *find* method with argument 1.

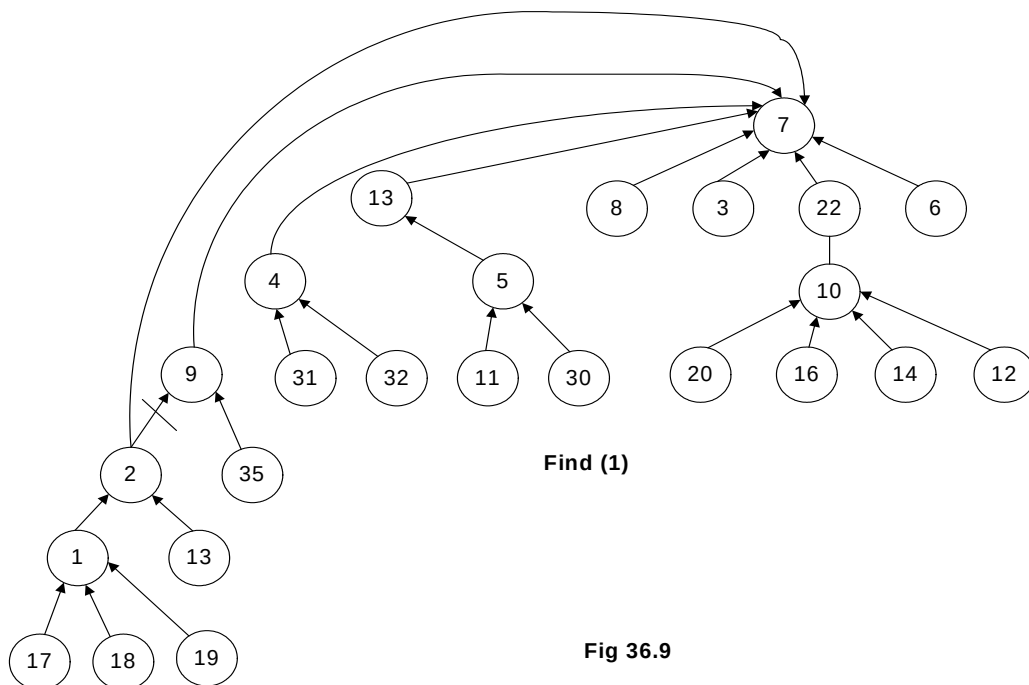


This tree is bigger as compared to the ones in our previous examples, it has been constructed after performing some *union* operations on trees. Now we have called *find* method with argument 1. We want to find the set with node 1. By following the previous method of *find*, we will traverse the tree from 1 to 2, from 2 to 9, 9 to 4, 4 to 13 and finally from 13 to 7, which is the *root* node, containing the negative number. So the *find* will return 7.

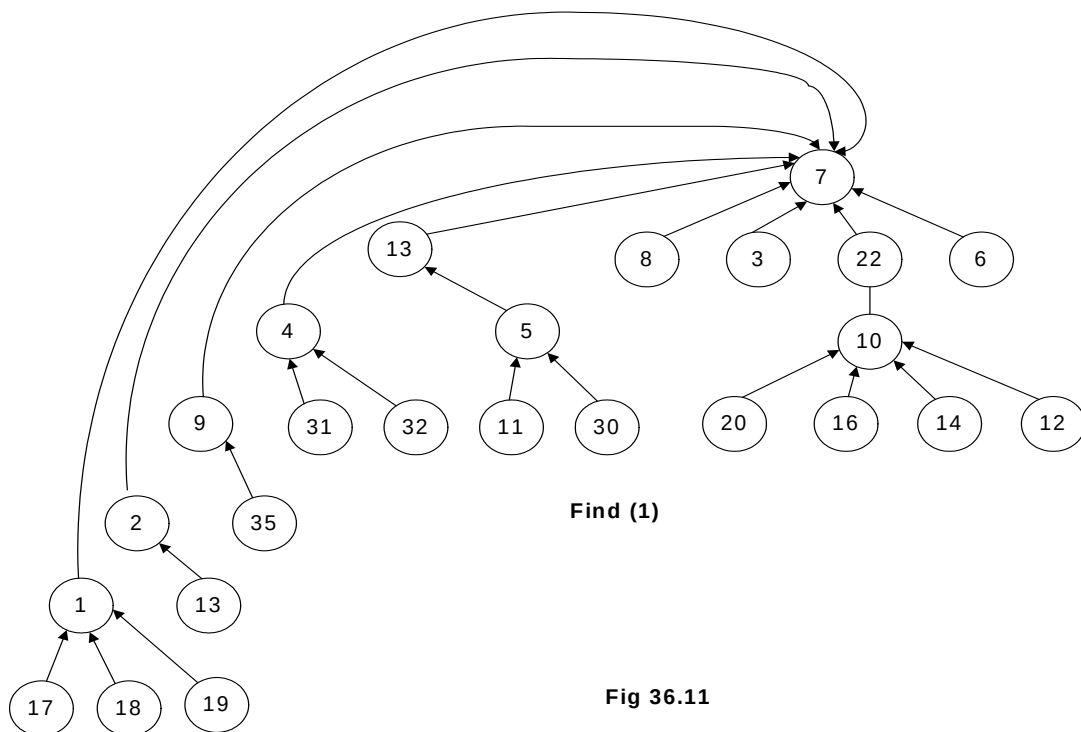
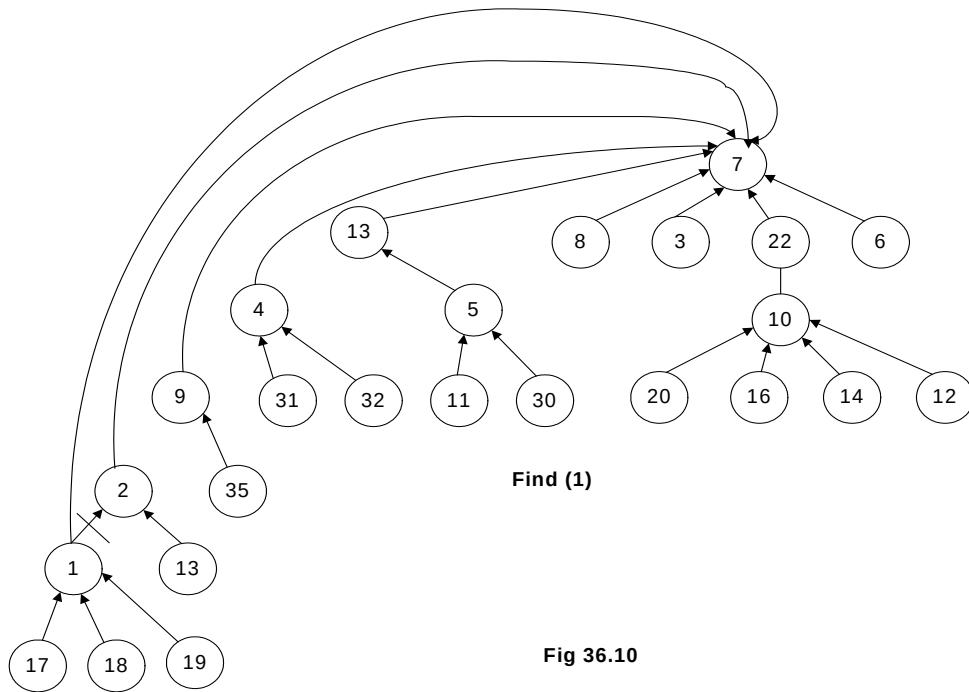
If we apply recursive mechanism given in the above code, the recursive call will start from *find(1)*. The *parent (find(1))* will return 2. The recursive call for 2 will take the control to node 9, then to 4, 13 and then eventually to 7. When *find(7)* is called, we get the negative number and reach the recursion stopping condition. Afterwards, all the recursive calls, on the stack are executed one by one. The call for 13, *find(13)* returns 7. *find(4)* returns 7 because *find(parent(4))* returns 7. That is why the link between 4 and 13 has been dropped and a new link is established between 4 and 7 as shown in the Fig 36.7. So the idea is to make the traversal path shorter from a node to the *root*. The remaining recursive calls will also return 7 as the result of *find* operation. Therefore, for subsequent calls we establish the links directly from the node to the *root*.



Similarly the node 2 is directly connected to the root node and the interlink between the node 2 and 9 is dropped.



The same will happen with 1.



The *union* operation is based on size or weight but the reducing the in-between links or path compression from nodes to the *root* is done by the *find* method. The *find* method will connect all the nodes on the path to the *root* node directly. The *path compression* is depicted by the following tree:

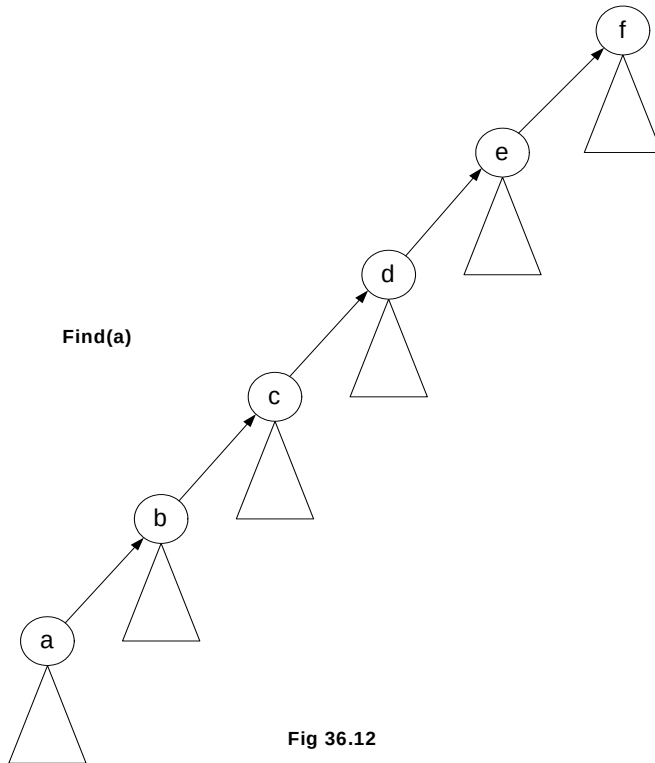


Fig 36.12

If we have called `find(a)` then see the path from node `a` to the *root* node `f`. `a` is connected to root node `f` through `b`, `c`, `d` and `e` nodes. Notice that there may be further subtrees below these nodes. After we apply this logic of *path compression* for *find* operation, we will have the tree as follows:

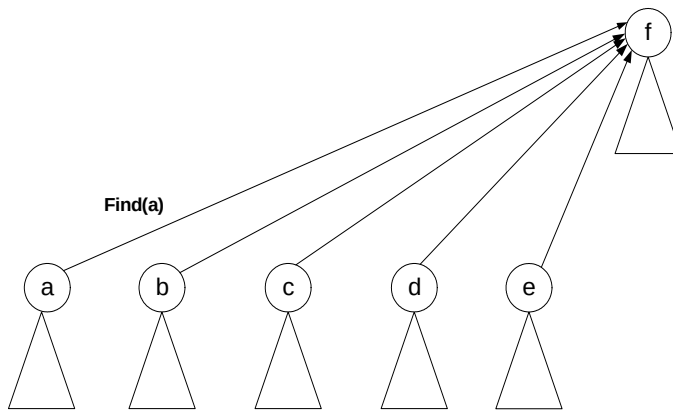


Fig 36.13

We see, how much is the impact on performance in *find* by a theorem.

Timing with Optimization

- Theorem: A sequence of m *union* and *find* operations, n of which are *find* operations, can be performed on a disjoint-set forest with union by rank (weight or height) and path compression in worst case time proportional to $(m\tilde{N}(n))$.
- $\tilde{N}(n)$ is the inverse Ackermann's function which grows extremely slowly. For all

practical purposes, $\tilde{N}(n)$ R 4.

- Union-find is essentially proportional to m for a sequence of m operations, linear in m .

There are number of things present in this theorem regarding analysis, which are difficult to cover in this course. We will study these in course of Algorithm Analysis. At the moment, consider if there are m *union* operations out of which n are *finds*. The average time for union-find will be linear in m and it will grow not quadratically or exponentially. *union* and *find* operations using this data structure are very efficient regarding both space and time.

Data Structures

Lecture No. 37

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 8

Summary

- Review
- Image Segmentation
- Maze Example
- Pseudo Code of the Maze Generation

Review

In the last lecture, we talked about *union* and *find* methods with special reference to the process of optimization in *union*. These methods were demonstrated by reducing size of tree with the help of the techniques-union by size or union by weight. Similarly the tree size was reduced through path optimization in the *find* method. This was due to the fact that we want to reduce the tree traversal for the *find* method. The time required by the *find/union* algorithm is proportional to m . If we have m union and n find, the time required by find is proportional to $m+n$. Union is a constant time operation. It just links two trees whereas in the *find* method tree traversal is involved. The disjoint sets are increased and forest keeps on decreasing. The *find* operation takes more time as unions are increased. But on the average, the time required by *find* is proportional to $m+n$.

Now we will see some more examples of disjoint sets and *union/find* methods to understand their benefits. In the start, it was told that disjoint sets are used in image segmentation. We have seen the picture of a boat in this regard. We have also talked about the medical scanning to find the human organs with the help of image segmentation. Let's see the usage of *union/find* in image segmentation.

Image segmentation is a major part of image processing. An image is a collection of pixels. A value associated with a pixel can be its intensity. We can take images by cameras or digital cameras and transfer it into the computer. In computer, images are represented as numbers. These numbers may represent the gray level of the image. The zero gray level may represent the black and 255 gray level as white. The numbers between 0 and 255 will represent the gray level between black and white. In the color images, we store three colors i.e. RGB (Red, Green, Blue). By combining these three colors, new ones can be obtained.

Image Segmentation

In image segmentation, we will divide the image into different parts. An image may

be segmented with regard to the intensity of the pixels. We may have groups of pixels having high intensity and those with pixels of low intensity. These pixels are divided on the basis of their threshold value. The pixels of gray level less than 50 can be combined in one group, followed by the pixels of gray level less between 50 and 100 in another group and so on. The pixels can be grouped on the basis of threshold for difference in intensity of neighbors. There are eight neighbors of the pixel i.e. top, bottom, left, right, top-left, top-right, bottom-left and bottom-right. Now we will see the difference of the threshold of a pixel and its neighbors. Depending on the difference value, we will group the pixels. The pixels can be grouped on the basis of texture (i.e. a pattern of pixel intensities). You will study all these in detail in the image processing subject.

Let's see an example. Consider the diagram below:



It seems a sketch or a square of black, gray and white portions. These small squares represent a pixel or picture element. The color of these picture elements may be white, gray or black. It has five rows and columns each. This is a 5 * 5 image. Now we will have a matrix of five rows and five columns. We will assign the values to the elements in the matrix depending on their color. For white color, we use 0, 4 for black and 2 for gray color respectively.

	0	1	2	3	4
0	0	0	0	4	4
1	2	0	4	4	0
2	4	2	2	4	4
3	4	4	0	4	4
4	0	2	2	4	0

When we get the image from a digital device, it is stored in numbers. In the above matrix, we have numbers from 0 to 4. These can be between 0 and 255 or may be more depending upon the digital capturing device. Now there is a raw digital image. We will apply some scheme for segmentation. Suppose we need the pixels having value 4 or more and the pixels with values less than 4 separately. After finding such pixels, put 1 for those that have values more than 4 and put 0 for pixels having less than 4 values. We want to make a binary array by applying the threshold. Let's apply this scheme on it.

	0	1	2	3	4

0	0	0	0	1	1
1	0	0	1	1	0
2	1	0	0	1	1
3	1	1	0	1	1
4	0	0	0	1	0

We have applied threshold equal to 4. We have replaced all 4's with 1 and all the values less than 4, have been substituted by zero. Now the image has been converted into a binary form. This may represent that at these points, blood is present or not. The value 4 may represent that there is blood at this point and the value less than 4 represents that there is no blood. It is very easy to a program for this algorithm.

Our objective was to do image segmentation. Suppose we have a robot which captures an image. The image obtained contains 0, 2 and 4 values. To know the values above and below the threshold, it may use our program to get a binary matrix. Similarly, for having the knowledge regarding the region of 1's and region of 0's, the robot can use the algorithm of disjoint sets. In the figure above, you can see that most of the 1's are on the right side of the matrix.

We will visit each row of the matrix. So far, there are 25 sets containing a single element each. Keep in mind the example discussed in *union/find* algorithm. In the zeroth row, we have three 0's. We can apply union on 0's. Yet we are interested in 1's. In the 3rd column, there is an entry of 1. Now the left neighbor of this element is 0 so we do not apply union here and move to the next column. Here, the entry is 1. Its left neighbor is also 1. We apply union here and get a set of two elements. We have traversed the first row completely and moved to the next row. There is 0 in the 0th and 1st column while in the 2nd column; we have an entry of 1. Its left neighbor is 0 so we move to the next column where entry is also 1. We apply union here. There is a set of 1 at the top of this set also. So we take union of these two sets and combine them. There is a set of four 1's, two from row 0 and two from row 1. In the next row, we have 1, 0, 0, 1, 1. In the row 2 column 3, there is 1. At the top of this element, we have 1. We apply union to this 1 and the previous set of four 1's. Now we have a set of five 1's. In the next column again, there is 1, also included in the set as the sixth element. In the row 3, we have 1 at the 0 column. There is 1 at the top of this so we have set of these two 1's. The next element at the row 3 is again 1, which is included in the left side set of 1's. Now we have another set of 1's having 3 elements. The 1's at column 3 and column 4 are included in the set of six 1's making it a set of eight elements. In the last row, we have single 1 at the column 3 which will be included with the eight element set. So there are two sets of 1's in this matrix.

	0	1	2	3	4
0	0	0	0	1	1
1	0	0	1	1	0
2	1	0	0	1	1
3	1	1	0	1	1
4	0	0	0	1	0

We have two disjoint sets and it is clearly visible that where are the entries of 1's and 0's. This is the situation when we have set the threshold of 4 to obtain a binary image. We can change this threshold value.

Let's take the value 2 as threshold value. It means that if the value of pixel is 2 or more than 2, replace it by 1 otherwise 0. The new image will be as:

	0	1	2	3	4
0	0	0	0	1	1
1	1	0	1	1	0
2	1	1	1	1	1
3	1	1	0	1	1
4	0	1	1	1	0

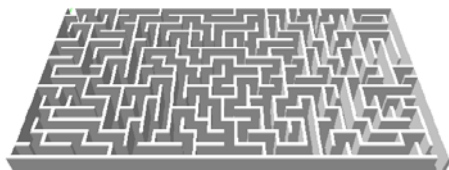
We can apply the union scheme on it to find the region of 1's. Here we have a blob of 1.

	0	1	2	3	4
0	0	0	0	1	1
1	1	0	1	1	0
2	1	1	1	1	1
3	1	1	0	1	1
4	0	1	1	1	0

The region is shaded. The image has been segmented. With the help of *union/find* algorithm, we can very quickly segment the image. The *union/find* algorithm does not require much storage. Initially, we have 25 sets that are stored in an array i.e. the up-tree. We do all the processing in this array without requiring any extra memory. For the image segmentation, disjoint sets and *union-find* algorithm are very useful. In the image analysis course, you will actually apply these on the images.

Maze Example

You have seen the maze game in the newspapers. This is a puzzle game. The user enters from one side and has to find the path to exit. Most of the paths lead to blind alley, forcing the user to go back to square one.



This can be useful in robotics. If the robot is in some room, it finds its path between the different things. If you are told to build mazes to be published in the newspaper, a new maze has to be developed daily. How will you do that? Have you done like this before? Did anyone told you to do like this? Now you have the knowledge of disjoint sets and *union-find* algorithm. We will see that the maze generation is very simple with the help of this algorithm.

Let's take a 5 * 5 grid and generate a 5 * 5 maze. A random maze generator can use *union-find* algorithm. Random means that a new maze should be generated every time.

Consider a 5x5 maze:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Here we have 25 cells. Each cell is isolated by walls from the others. These cells are numbered from 0 to 24. The line between 0 and 1 means that there is a wall between them, inhibiting movement from 0 to 1. Similarly there is a wall between 0 and 5. Take the cell 6, it has walls on 4 sides restricting to move from it to anywhere. The internal cells have walls on all sides. We will remove these walls randomly to establish a path from the first cell to the last cell.

This corresponds to an equivalence relation i.e. two cells are equivalent if they can be reached from each other (walls been removed so there is a path from one to the other). If we remove the wall between two cells, then there can be movement from one cell to the other. In other way, we have established an equivalence relationship between them. This can be understood from the daily life example that when the separation wall of two persons is removed, they become related to each other. In maze generation, we will remove the walls, attaching the cells to each other in some sequence. In the end, we will have a path from the start cell to the end cell.

First of all, we have to decide an entrance and an exit. From the entrance, we will move into the maze and have to reach at the exit point. In our example, the entrance is from the cell 0 and the exit will be at cell 24.

→	0	1	2	3	4
	5	6	7	8	9
	10	11	12	13	14
	15	16	17	18	19
	20	21	22	23	24 →

How can we generate maze? The algorithm is as follows:

- Randomly remove walls until the entrance and exit cells are in the same set.
- Removal of the wall is the same as doing a union operation.
- Do not remove a randomly chosen wall if the cells it separates are already in the same set.

We will take cells randomly. It means that the probability of each cell is equal. After selecting a cell, we will choose one of its surrounding walls. The internal cells have four walls around them. Now we will randomly choose one wall. In this way, we will choose the neighboring cell. Initially we have 25 sets. Now after removing a wall between 2 cells, we have combined these two cells into one set. Here the union method has been applied to combine these two cells. If these cells are already in the same set, we will do nothing.

We will keep on randomly choosing the cells and removing the walls. The cells will merge together. The elements of the set will keep on growing and at some point, we may have the entrance cell (cell 0) and the exit cell (cell 24) in the same set. When the entrance cell and the exit cell are in the same set, it means that we have a set in which the elements are related to each other and there is no wall between them. In such a situation, we can move from start to the end going through some cells of this set. Now we have at least one path from entrance to exit. There may be other paths in which we have the entrance cell but not the exit. By following this path, you will not reach at the exit as these paths take you to the dead end. As these are disjoint sets, so unless the entrance and exit cell are in the same set, you will not be able to find the solution of the maze.

Pseudo Code of the Maze Generation

Let's take a look at the pseudo code of the maze generation. We will pass it to the size argument to make a maze of this size. We will use the entrance as 0 and exit as *size-1* by default.

```

MakeMaze(int size) {
    entrance = 0; exit = size-1;
    while (find(entrance) != find(exit)) {
        cell1 = randomly chosen cell
        cell2 = randomly chosen adjacent cell
        if (find(cell1) != find(cell2)) {
            knock down wall between cells
            union(cell1, cell2)
        }
    }
}

```

After initializing the entrance and exit, we have a while loop. The loop will keep on executing till the time the *find(entrance)* is not equal to *find(exit)*. It means that the loop will continue till the set returned by the *find(entrance)* and *find(exit)* are not same. When the entrance and exit cells are in the same set, the loop will stop. If these cells are not in the same set, we will enter into the loop. At first, we will randomly choose a cell from the available cells e.g. from 0 to 24 from our example. We are not discussing here how to randomly choose a cell in C++. There may be some function available to do this. We store this value in the variable *cell1* and choose randomly its neighboring cell and store it in *cell2*. This cell may be its top, bottom, left or right cell, if it is internal cell. If the cell is at the top row or top bottom, it will not have four neighbors. In case of being a corner cell, it will have only two neighboring cells. Then we try to combine these two cells into one set. We have an if statement which checks that the set returned by *find(cell1)* is different than the set returned by *find(cell2)*. In this case, we remove the wall between them. Then we apply union on these two cells to combine them into a set.

We randomly choose cells. By applying union on them, the sets are joined together and form new sets. This loop will continue till we have a set which contains both entrance and exit cell. This is a very small and easy algorithm. If we have the *union-find* methods, this small piece of code can generate the maze. Let's take a pictorial

look at different stages of maze generation. You can better understand this algorithm with the help of these figures. We are using the disjoint sets and *union-find* algorithm to solve this problem.

Initially, there are 25 cells. We want to generate a maze from these cells.

→

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

→

Apply the algorithm on it. We randomly choose a cell. Suppose we get *cell 11*. After this, we randomly choose one of its walls. Suppose we get the right wall. Now we will have *cell 11* in the variable *cell1* and *cell 12* in the variable *cell2*.

→

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

→

By now, each cell is in its own set. Therefore *find(cell 11)* will return *set_11* and *find(cell 12)* will return *set_12*. The wall between them is removed and union is applied on them. Now we can move from *cell 11* to *cell 12* and vice versa due to the symmetry condition of disjoint sets. We have created a new set (*set_11* = {11,12}) that contains *cell 11* and *cell 12* and all other cells are in their own cells.

→

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

→

Now we randomly choose the *cell 6* and its bottom wall. Now the *find(cell 6)* will return *set_6*. The *find(cell 11)* will return *set_11* which contains *cell 11* and *cell 12*. The sets returned by the two *find* calls are different so the wall between them is removed and union method applied on them. The set *set_11* now contains three elements *cell 11*, *cell 12* and *cell 6*. These three cells are joined together and we can move into these cells.

Now we randomly select the *cell 8*. This cell is not neighbor of *set_11* elements. We can randomly choose the *cell 6* again and its bottom wall. However, they are already in the same set so we will do nothing in that case. We randomly choose the *cell 8* and its top wall (*cell 3*). As these two cells are in different sets, the wall between them is removed so that union method could be applied to combine them into a set (*set_8* = {8, 3}).

→

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24



Now we randomly choose the *cell 14* and its top i.e. *cell 9*. Now we keep on combining cells together but so far entrance and exit cells are not in the same set.

→

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24



We randomly choose *cell 0* and its bottom wall so the *cell 0* and *cell 5* are combined together.

→

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24



If you keep on using this algorithm, the walls between cells will keep on vanishing, resulting in the growth of set elements. At some point, the entrance and exit cell will come into one set and there will be a path taking us from the start to exit. There may be some other sets which do not contain the exit cell. Take it as a programming exercise and do it. This is very interesting exercise. Find some information on the internet regarding maze generation. In the next lecture, we will talk about the new data type i.e. table.

Data Structures

Lecture No. 38

Summary

- Table and Dictionaries
- Operations on Table ADT
- Implementation of Table
 - Unsorted Sequential Array
 - Sorted Sequential Array
- Binary Search

We will discuss the concepts of Tables and Dictionaries in this lecture with special reference to the operations on Table ADT and the implementation.

Tables and Dictionaries

The table, an abstract data type, is a collection of rows and columns of information. From rows and columns, we can understand that this is like a two dimensional array. But it is not always a two dimensional array of same data type. Here in a table, the type of information in columns may be different. The data type of first column may be integer while the type of second column is likely to be string. Similarly there may be other different data types in different columns. So the two-dimensional array used for table will have different data types.

A table consists of several columns, known as fields. These fields are some type of information. For example a telephone directory may have three fields i.e. *name*, *address* and *phone number*. On a computer system, the user account may have fields- *user ID*, *password* and *home folder*. Similarly a bank account may have fields like *account number*, *account title*, *account type* and *balance of account*.

Following is a table that contains three fields (We are calling columns as fields). These three fields are *name*, *address* and *phone*. Moreover, there are three rows shown in the table.

Name	Address	Phone
Sohail Aslam	50 Zahoor Elahi Rd, Gulberg-4, Lahore	567-4567
Imran Ahmad	30-T Phase-IV, LCCHS, Lahore	517-2349
Salman Akhtar	131-D Model Town, Lahore	756-5678

Figure 38.1: A table having three fields.

Each row of the table has a set of information in the three fields. The fields of a row are linked to each other. The row of a table is called a record. Thus the above table contains three records. It resembles to the telephone directory. There may be other fields in the phone directory like *father's name* or the *occupation* of the user. Thus the data structure that we form by keeping information in rows and columns is called table.

During the discussion on the concept of table, we have come across the word,

dictionary. We are familiar with the language dictionary, used to look for different words and their meanings. But for a programmer, the dictionary is a data type. This data is in the form of sets of fields. These fields comprise data items.

A major use of table is in Databases where we build and use tables for keeping information. Suppose we are developing payroll software for a company. In this case, a table for employees will be prepared. This table of employees has the name, id number, designation and salary etc of the employees. Similarly there may be other tables in this database e.g. to keep the leave record of the employees. There may be a table to keep the personal information of an employee. Then we use SQL (Structured Query Language) to append, select, search and delete information (records) from a table in the database.

Another application of table is in compilers. In this case, symbol tables are used. We will see in compilers course that symbol table is an important structure in a compiler. In a program, each variable declared has a, type and scope. When the compiler compiles a program, it makes a table of all the variables in the program. In this table, the compiler keeps the name, type and scope of variables. There may be other fields in this table like function names and addresses. In the compiler construction course, we will see that symbol table is a core of compiler.

When we put the information in a table, its order will be such that each row is a record of information. The word *tuple* is used for it in databases. Thus a row is called a record or tuple. As we see there is a number of rows (records) in a table. And a record has some fields.

Now here are some things about tables.

To find an *entry* in the table, we only need to know the contents of one of the fields (not all of them). This field is called *key*. Suppose we want to see information (of a record) in dictionary. To find out a record in the dictionary, we usually know the name of the person about whom information is required. Thus by finding the name of the person, we reach a particular record and get all the fields (information) about that person. These fields may be the address and phone number of the person. Thus in telephone directory, the *key* is the *name* field. Similarly in a user account table, the key is usually *user id*. By getting a user id, we can find its information that consists of other fields from the table. The *key* should be unique so that there is only one record for a particular value of *key*. In the databases, it is known as *primary key*. Thus, in a table, a key uniquely identifies an entry. Suppose if the name is the *key* in telephone book then no two entries in the telephone book have the same name. The *key* uniquely identifies the entries. For example if we use the *name* “imran ahmed” in the following table that contains three entries, the name “imran ahmed” is there only at one place in the following table. In this record, we can also see the address and phone number of “imran ahmed”.

Name	Address	Phone
Sohail Aslam	50 Zahoor Elahi Rd, Gulberg-4, Lahore	567-4567
Imran Ahmad	30-T Phase-IV, LCCHS, Lahore	517-2349
Salman Akhtar	131-D Model Town, Lahore	756-5678

Figure 38.2: A table having three fields.

Similarly if we use “salman akhtar” as key we find out one record for this key in the table.

The purpose of this key is to find data. Before finding data, we have to add data. On the addition of the data in the table, the finding of data will be carried through this *key*. We may need to delete some data from the table. For example, in the employees' table, we want to delete the data of an employee who has left the company. To delete the data from the table, the *key* i.e. *name* will be used. However in the employees' table, the *key* may be the *employee id* and not the *name*.

Operations on Table ADT

Now we see the operations (methods) that can be performed with the table abstract data type.

insert

As the name shows this method is used to insert (add) a record in a table. For its execution, a field is designated as *key*. To insert a record (entry) in the table, there is need to know the *key* and the entry. The insert method puts the *key* and the other related fields in a table. Thus we add records in a table.

find

Suppose we have data in the table and want to find some particular information. The find method is given a *key* and it finds the entry associated with the key. In other words, it finds the whole record that has the same *key* value as provided to it. For example, in employees table if *employee id* is the key, we can find the record of an employee whose *employee id* is, say, 15466.

remove

Then there is the remove method that is given a value of the *key* to find and remove the entry associated with that key from the table.

Implementation of Table

Let's talk about why and how we should implement a table. Our choice for implementation of the Table ADT depends on the answers to the following.

- How often entries are inserted, found and removed?
- How many of the possible key values are likely to be used?
- What is the likely pattern of searching for keys? Will most of the accesses be to just one or two key values?
- Is the table small enough to fit into the memory?
- How long will the table exist?

In a table for searching purposes, it is best to store the key and the entry separately (even though the key's value may be inside the entry). Now, suppose we have a record of a person 'Saleem' whose address is '124 Hawkers Lane' while the phone number is '9675846'. Similarly we have another record of a person 'Yunus'. The address and phone fields of this person are '1 Apple crescent' and '622455' respectively. For these records in the table, we will have two parts. One part is the complete entry while the other is the key in which we keep the unique item of the entry. This unique item is twice in a record one as part of the entry and the second in a field i.e. the key field. This key will be used for searching and deletion purposes of

records. With the help of key, we reach at the row and can get other fields of it. We also call *TableNode* to row of the table. Its pictorial representation is given below.

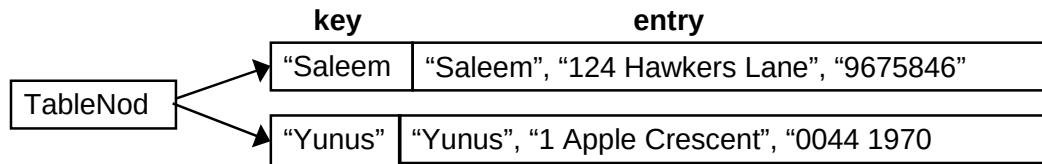


Figure 38.3: key and entry in table

Now we will discuss the implementation of table with different data structures. The first implementation is the unsorted sequential array.

Unsorted Sequential Array

In this implementation, we store the data of table in an array such that *TableNodes* are stored consecutively in any order. Each element of the row will have a key and entry of the record.

Now let's think how we can store the data of a table in an array. For a telephone directory, there may be a field name i.e. a string. Then there is 'address', which is also a string but of large size as compared to the name. The phone number may be a string or integer. Each record in the directory has these three fields with different types. How can we store this information in an array? Here comes the class. Suppose we make a class *Employee*, with an ultimate aim of developing objects. An object of *Employee* will contain the name, address, designation and salary. Similarly we can make a class of *PhoneDirectoryEntry* that will have name, address and phone number. We can add other fields to these classes. Thus we have *Employee* and *PhoneDirectoryEntry* objects. Now we make an array of objects. In this array, every object is a table entry. To insert a new object (of *Employee* or *PhoneDirectoryEntry*), we add it to the back of the array. This insert operation in the array is fast as we directly put the data at the last available position. Thus we add the data to the array as soon as it is available. In other words, we don't have an order of the data. Now if there are n entries in the array, the find operation searches through the keys one at a time and potentially all of the keys to find a particular key. It has to go through all the entries (i.e. n) if the required key is not in the table. Thus the time of searching will be proportional to the number of entries i.e. n . Similarly the remove method also requires time proportional to n . The remove method, at first, has to find the key, needed to be removed. It consumes the time of find operation that is proportional to the number of entries i.e. n . If the entry is found, the remove method removes it. Obviously it does nothing if the entry is not found.

Here we see that in unsorted sequential array, the insertion of data is fast but the find operation is slow and requires much time. Now the question arises if there is any way to keep an array in which the search operation can be fast? The use of sorted sequential array is a solution to this problem.

Sorted Sequential Array

We have studied in the tree section that binary search tree is used to search the information rapidly. Equally is true about the sorted sequential array. But in this case, we want to put the data in an array and not in a tree. Moreover, we want to search the data in this array very fast. To achieve this objective, we keep the data in the array in a sorted form with a particular order. Now suppose we are putting the information in the *telephoneDirectory* table into an array in a sorted form in a particular order. Here, for example, we put the data alphabetically with respect to name. Thus data of a person whose name starts with 'a' will be at the start of the table and the name starting with 'b' will be after all the names that start with 'a'. Then after the names starting with 'b' there will be names starting with 'c' and so on. Suppose a new name starting from 'c' needs to be inserted. We will put the data in the array and sort the array so that this data can be stored at its position with respect to the alphabetic order. Later in this course, we will read about sorting.

Let's talk about the insert, find and remove methods for sorted data.

insert

For the insertion of a new record in the array, we will have to insert it at a position in the array so that the array should be in sorted form after the insertion. We may need to shift the entries that already exist in the array to find the position of the new entry. For example, if a new entry needs to be inserted at the middle of the array, we will have to shift the entries after the middle position downward. Similarly if we have to add an entry at the start of the array, all the entries will be moved in the array to one position right (down). Thus we see that the *insert* operation is proportional to n (number of entries in the table). This means *insert* operation will take considerable time.

find

The *find* operation on a sorted array will search out a particular entry in $\log n$ time by the binary search. The binary search is a searching algorithm. Recall that in the tree, we also find an item in $\log n$ time. The same is in the case of sorted array.

remove

The *remove* operation is also proportional to n . The *remove* operation first finds the entry that takes $\log n$ time. While removing the data, it has to shuffle (move) the elements in the array to keep the sorted order. This shuffling is proportional to n . Suppose, we remove the first element from the array, then all the elements of the array have to be moved one position to left. Thus *remove* method is proportional to n .

Binary Search

The binary search is an algorithm of searching, used with the sorted data. As we have sorted elements in the array, binary search method can be employed to find data in the array. The binary search finds an element in the sorted array in $\log n$ time. If we have 100000 elements in the array, the $\log 100000$ will be 20 i.e. very small as compared to 100000. Thus binary search is very fast.

The binary search is like looking up a phone number in the directory or looking up a word in the dictionary. For looking a word in the dictionary, we start from the middle in the dictionary. If the word that we are looking for comes before words on the page, it shows that the word should be before this page. So we look in the first half.

Otherwise, we search for the word in the second half of the dictionary. Suppose the word is in the first half of the dictionary, we consider first half for looking the word. We have no need to look into the second half of the dictionary. Thus the data to be searched becomes half in a step. Now we divide this portion into two halves and look for the word. Here we again come to know that the word is in the first half or in the second half of this portion. The same step is repeated with the part that contains the required word. Finally, we come to the page where the required word exists. We see that in the binary search, the data to be searched becomes half in each step. And we find the entry very fast. The number of maximum steps needed to find an entry is $\log n$, where n is the total number of entries. Now if we have 100000 entries, the maximum number of attempts (steps) required to find out the entry will be 20 (i.e. $\log 100000$).

Data Structures

Lecture No. 39

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 10

10.4.2

Summary

- Searching an Array: Binary Search
- Binary Search - Example 1
- Binary Search - Example 2
- Binary Search - Example 3
- Binary Search – C++ Code
- Binary Search – Binary Tree
- Binary Search - Efficiency
- Implementation 3 (of Table ADT): Linked List
- Implementation 4 (of Table ADT): Skip List
- Skip List - Representation
- Skip List - Higher Level Chains
- Skip List - Formally

Searching an Array: Binary Search

In the previous lecture, we had started discussion on Binary Search Tree algorithm. The discussion revealed that if already sorted data is available, then it is better to apply an algorithm of binary search for finding some item inside instead of searching from start to the end in sequence. The application of this algorithm will help get the results very quickly. We also talked about the example of directory of employees of a company where the names of the employee were sorted. For efficient searching, we constructed the binary search tree for the directory and looked for information about an employee named 'Ahmed Faraz'.

We also covered:

- Binary search is like looking up a phone number or a word in the dictionary
- Start in middle of book
- If the name you're looking for, comes before names on the page, search in the first half
- Otherwise, look into the second half

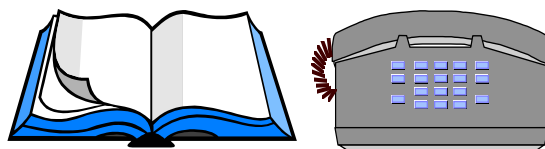


Fig. 39.1

The telephone directory is the quotable example to understand the way the binary

search method works.

In this lecture, we will focus on data structures for performing search operation. Consider the data is present in an array as we discussed in the previous lecture. For the first implementation, we supposed that the data is not sorted in the array. For second implementation, we considered that the data inside the array is put in sorted array. The advantage of the effort of putting the data in the array in sorted order pays off when the searches on data items are performed.

Now, let's first see the algorithm (in pseudo code) for this operation below. It is important to mention that this algorithm is independent of data type i.e. the data can be of any type numeric or string.

```
if ( value == middle element )
    value is found
else if ( value < middle element )
```

search left half of list with the same method

else

search right half of list with the same method

The item we are searching for in this algorithm is called *value*. The first comparison of this *value* is made with the *middle element* of the array. If both are equal, it means that we have found our desired search item, which is present in the middle of the array. If this is not the case, then the *value* and the *middle element* are not the same. The *else-if* part of the algorithm is computed, which checks if the *value* is less than the *middle element*. If so, the left half part of the array is searched further in the same fashion (of logically splitting that half part of array into two further halves and applying this algorithm again). This search operation can also be implemented using the recursive algorithm but that will be discussed later. At the moment, we are discussing only the non-recursive algorithm. The last part of the algorithm deals with the case when the *value* is greater than the *middle element*. This processes the right half of the array with the same method.

Let's see this algorithm in action by taking an example of an array of elements:

Binary Search – Example 1

Case 1: $val == a[mid]$

$val = 10$

$low = 0, high = 8$

$mid = (0 + 8) / 2 = 4$

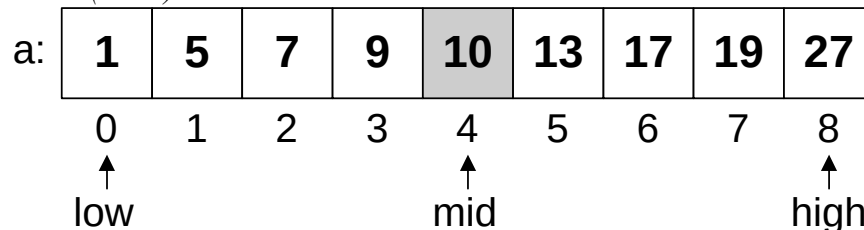


Fig 39.2

You can see an array *a* in the Fig 39.2 with indexes 0 to 8 and values 1, 5, 7, 9, 10, 13, 17, 19 and 27. These values are our data items. Notice that these are sorted in ascending (increasing) order.

You can see in the first line of case 1 that $val = 10$, which indicates that we are searching for value 10. From second line, the range of data to search is from 0 to 8. In this data range, the middle position is calculated by using a simple formula $(low + high)/2$. In this case, it is $mid = (0+8)/2=4$. This is the middle position of the data array. See the array in the above figure Fig 39.2, which shows that the item at array position 4 is 10, exactly the value we are searching for. So, in this case, we have found the value right away in the middle position of the array. The search operation can stop here and an appropriate value can be returned back.

Let's see the case 2 now:

Binary Search – Example 2

Case 2: $val > a[mid]$

$val = 19$

$low = 0, high = 8$

$mid = (0 + 8) / 2 = 4$

$new\ low = mid + 1 = 5$

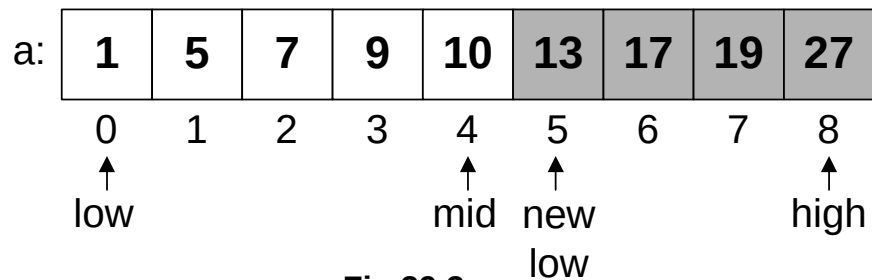


Fig 39.3

The second case is about the scenario when value (val) is greater than the middle value ($a[mid]$). The range of data items (low and $high$) is the same as that in case 1. Therefore, the middle position (mid) is also the same. But the value (val) 19 is greater than the value at the middle (mid) 10 of the array. As this array is sorted, therefore, the left half of the array must not contain value 19. At this point of time, our information about val 19 is that it is greater than the middle. So it might be present in the right half of the array. The right half part starts from position 5 to position 8. It is shown in Fig 39.3 that the $new\ low$ is at position 5. With these new low and high positions, the algorithm is applied to this right half again.

Now, we are left with one more case.

Binary Search – Example 3

Case 3: $val < a[mid]$

$val = 7$

$low = 0, high = 8$

$mid = (0 + 8) / 2 = 4$

$new\ high = mid - 1 = 5$

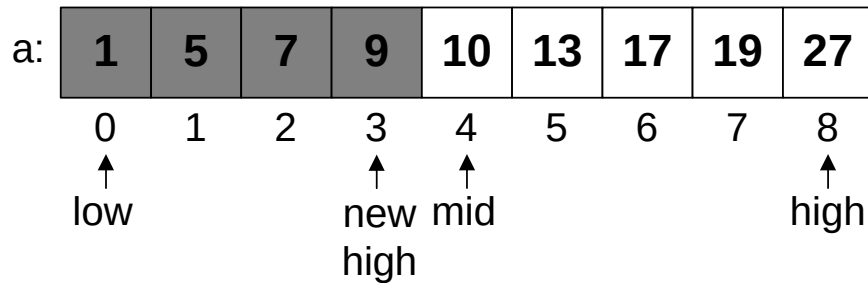


Fig 39.4

The value to be searched (*val*) in this case is 7. The data range is the same starting from *low*=0 to *high*=8. Middle is computed in the same manner and the value at the middle position (*mid*) is compared with the *val*. The *val* is less than the value at *mid* position. As the data is sorted, therefore, a value lesser than the one at *mid* position should be present in the lower half (left half) of the array (if it is there). The left half of the array will start from the same starting position *low*=0 but the *high* position is going to be changed to *mid*-1 i.e. 3. Now, let's execute this algorithm again on this left half.

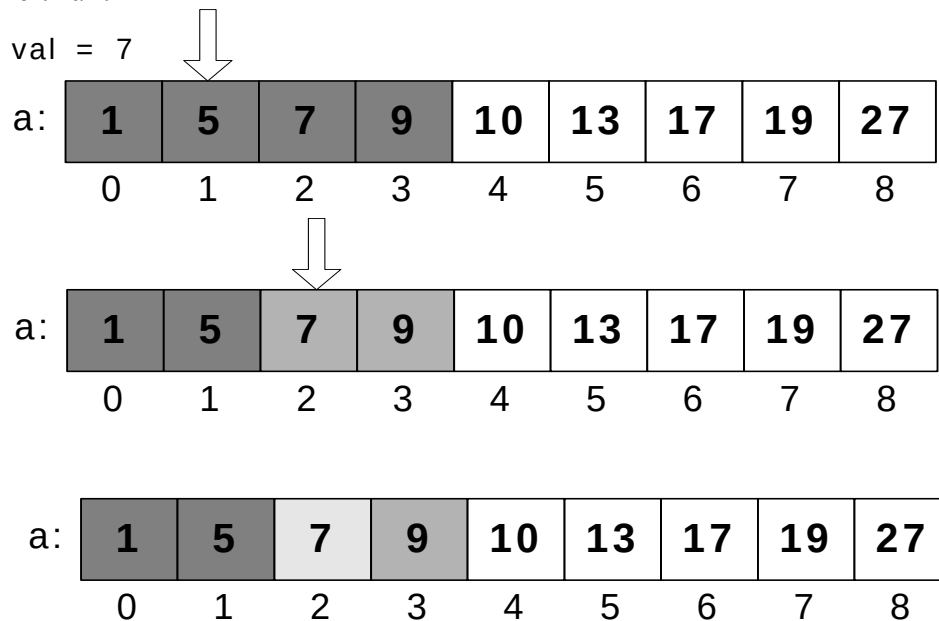


Fig 39.5

Firstly, we will compute the middle of 0 and 3 that results in 1 in this integer division. This is shown in the top array in Fig 39.5. Now, the *val* 7 is compared with the value at the middle position (*mid*) i.e.5. As 7 is greater than 5, we will process the right half of this data range (which is positioned from *low*=0 to *high*=3). The right half of this data range starts from position 2 and ends at position 3. The new data range is *low*=2 and *high*=3. The middle position (*mid*) is computed as $(2+3)/2=2$. The value at the *mid* position is 7. We compare the value at *mid* position (7) to the *val* we are looking for. These are found to be equal and finally we have the desired value.

Our desired number is found within positions- 0 to 8 at position 2. Without applying this binary search algorithm, we might have performed lot more comparisons. You might feel that finding this number 7 sequentially is easier as it is found at position 2 only. But what will happen in case we are searching for number 27. In that case, we

have to compare with each element present in the array to find out the desired number. On the other hand, if this number 27 is searched with the help of the binary search algorithm, it is found in third comparison.

Actually, we have already seen binary search while studying the binary search tree. While comparing the number with *the root* element of the tree, we had come to know that if the number was found smaller than the number in the *root*, we had to switch to left-subtree of the *root* (ensuring that it cannot be found in the right subtree).

Now, let's see the C++ code for this algorithm:

Binary Search – C++ Code

```
int isPresent(int *arr, int val, int N)
{
    int low = 0;
    int high = N - 1;
    int mid;
    while ( low <= high )
    {
        mid = ( low + high )/2;
        if (arr[mid] == val)
            return 1;    // found!
        else if (arr[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return 0; // not found
}
```

The name of the routine is *isPresent*, which expects an *int* pointer (an array in actual); an *int* value *val* is required to be searched. Another *int* value is *N* that indicates the maximum index value of the array. Inside the body of the function, *int* variables *low*, *high* and *mid* are declared. *low* is initialized to 0 and *high* is initialized to *N-1*. *while* loop is based on the condition that executes the loop until *low* <= *high*. Inside the loop, very first thing is the calculation of the middle position (*mid*). Then comes the first check inside the loop, it compares the *val* (the value being searched) with the number at middle position (*arr[mid]*). If they are equal, the function returns 1. If this condition returns false, it means that the numbers are unequal. Then comes the turn of another condition. This condition (*arr[mid] < val*) is checking if the value at the middle is less than the value being searched. If this is so, the right half of the tree is selected by changing the position of the variable *low* to *mid+1* and processed through the loop again. If both of these conditions return false, the left half of the array is selected by changing the variable *high* to *mid-1*. This left half is processed through the loop again. If the loop terminates and the required value is not found, then 0 is returned as shown in the last statement of this function.

You change this function to return the position of the value if it is found in the array otherwise return -1 to inform about the failure. It is important to note that this function requires the data to be sorted to work properly. Otherwise, it will fail.

This algorithm is depicted figurative in Fig 39.6.

Binary Search – Binary Tree

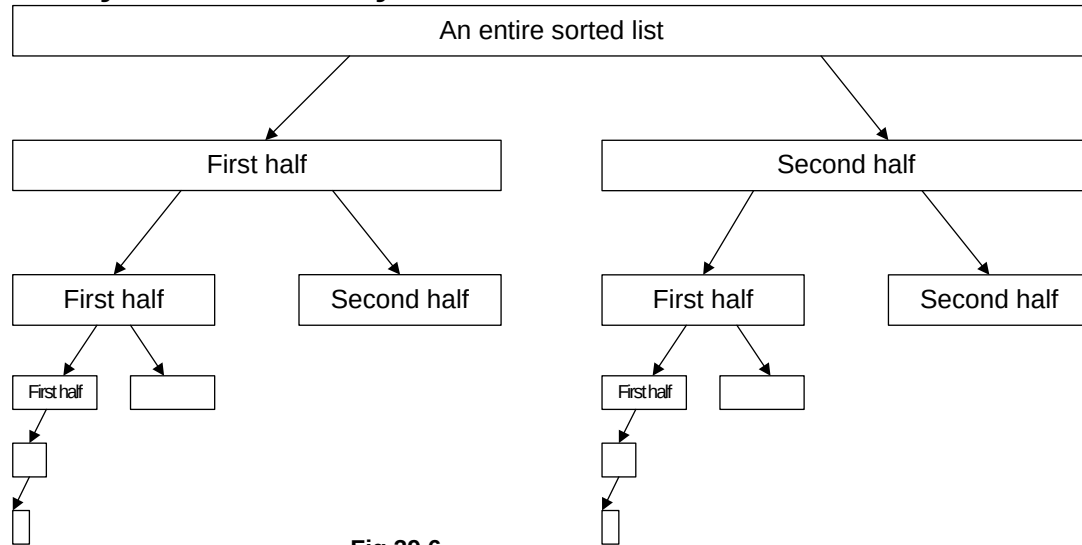


Fig 39.6

- The search divides a list into two small sub-lists till a sub-list is no more divisible. You might have realized about the good performance of binary trees by just looking at these if you remember the fully balanced trees of N items discussed earlier.

Binary Search - Efficiency

To see the efficiency of this binary search algorithm, consider when we divide the array of N items into two halves first time.

After 1 bisection	$N/2$	items
After 2 bisections	$N/4 = N/2^2$	items
...		
After i bisections	$N/2^i = 1$	item

$$i = \log_2 N$$

First half contains $N/2$ items while the second half also contains around $N/2$ items. After one of the halves is divided further, each half contains around $N/4$ elements. At this point, only one of $N/4$ items half is processed to search further. If we carry out three bisections, each half will contain $N/8$ items. Similarly for i bisections, we are left with $N/2^i$, which is at one point of time is only one element of the array. So we have the equation here:

$$N/2^i = 1$$

Computing the value of i from this gives us:

$$i = \log_2 N$$

Which shows that after maximum $\log_2 N$ bisections, either you will be successful in finding your item or fail to do so.

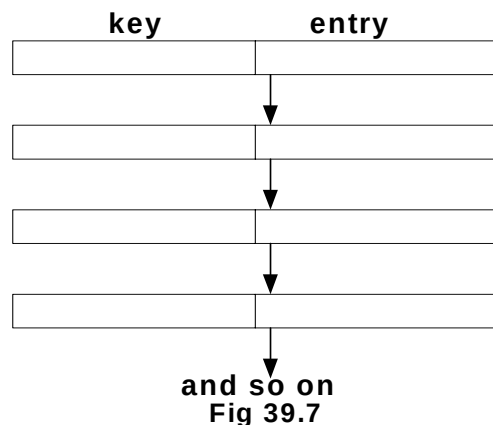
This was our second implementation of table or dictionary abstract data type using sorted sequential array. As discussed at start of it that if we implement table or

dictionary abstract data type using an array, we have to keep the array elements in sorted order. An easier way to sort them can be that whenever we want to *insert* an element in the array, firstly we find its position (in sorted order) in the array and then shift the right side (of that position) elements one position towards right to insert it. In worst case, we might have to shift all the elements one position towards right just to keep the data sorted so this will be proportional to N . Similarly the *remove* operation is also proportional to N . But after keeping the data sorted, the search operation is returned within maximum $\log_2 N$ bisections.

Implementation 3 (of Table ADT): Linked List

We might also use linked list to implement the table abstract data type. We can keep the data unsorted and keep on inserting the new coming elements to *front* in the list. It is also possible to keep the data in sorted order. For that, to insert a new element in the list, as we did for array, we will first find its position (in sorted order) in the list and then insert it there. The search operation cannot work in the same fashion here because binary search works only for arrays. Because the linked list may not be contiguous in memory, normally its nodes are scattered through, therefore, binary search cannot work with them.

- *TableNodes* are again stored consecutively (unsorted or sorted)
- **insert**: add to front; (1 or n for a sorted list)
- **find**: search through potentially all the keys, one at a time; (n for unsorted or for a sorted list)
- **remove**: find, remove using pointer alterations; (n)



Well, linked list is one choice to implement table abstract data type. For unsorted elements, the insertion at *front* operation will take constant time. (as each element is inserted in one go). But if the data inside the list is kept in sorted order then to insert a new element in the list, the entire linked list is traversed through to find the appropriate position for the element.

For *find* operation, all the keys are scanned through whether they are sorted or unsorted. That means the time of *find* is proportional to N .

For *remove* operation, we have to perform *find* first. After that the element is removed

and the links are readjusted accordingly.

We know that when we used sorted array, the *find* operation was optimized. Let's compare the usage of array and linked list for table abstract data type. The fixed size of the array becomes a constraint that it cannot contain elements more than that. Linked list has no such constraint but the *find* operation using linked list becomes slower. Is it possible to speed up this operation of *find* while using linked list? For this, a professor of University of Maryland introduced a new data structure called skip list. Let's discuss little bit about *skip list*.

Implementation 4 (of Table ADT): Skip List

- Overcome basic limitations of previous lists
 - o Search and update require linear time
- Fast Searching of Sorted Chain
- Provide alternative to BST (binary search trees) and related tree structures. Balancing can be expensive.
- Relatively recent data structure: Bill Pugh proposed it in 1990.

Important characteristics of skip list are stated above. Now, we see skip list in bit more detail.

Skip List - Representation

Can do better than n comparisons to find element in chain of length n

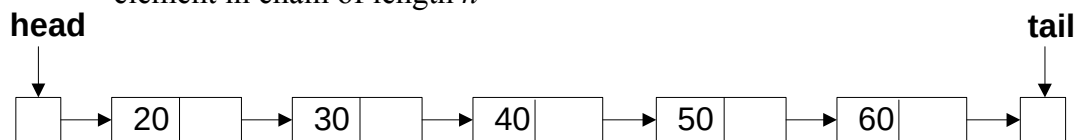


Fig 39.8

As shown in the figure. The *head* and *tail* are special nodes at start and end of the list respectively. If we have to find number 60 in the list then we have no other choice but starting from *head* traverse the subsequent nodes using the *next* pointer until the required node is found or the *tail* is reached. To find 70 in the list, we will scan through the whole list and then get to know that it is not present in it. The professor Pugh suggested something here:

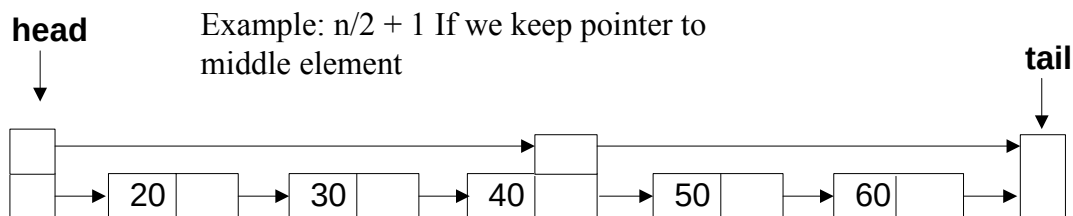


Fig 39.9

Firstly, we use two pointers *head* and *tail*. Secondly, the node in the middle has two *next* pointers; one is the old linked list pointer leading to next node 50 and the second is leading to the *tail* node. Additionally the *head* node also has two pointers, one is the old linked list pointer pointing to the next node 20 and second one is pointing to the middle element's *next* pointer, which is (as told above) further pointing to the *tail* node.

Now, if we want to find element 60 in the above list. Is it possible to search the list in relatively quick manner than the normal linked list shown in Fig 39.8? Yes, it is with the help of the additional pointers we have placed in the list. We will come to the middle of the list first and see that the middle element (40) is smaller than 60, therefore the right half part of the list is selected to process further (as the linked list is sorted). Isn't it the same we did in binary search? It definitely is.

What if we can add additional pointers (links) and boost the performance. See the figure below.

Skip List - Higher Level Chains

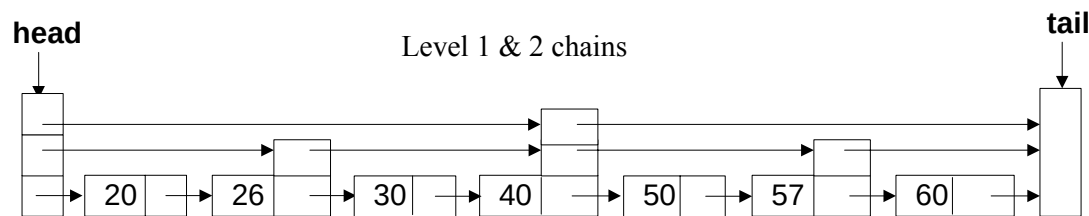


Fig 39.10

- For general n , level 0 chain includes all elements
 - level 1 every other element, level 2 chain every fourth, etc.
 - level i , every 2^i th element
- Level 0 chain is our old linked list chain as shown in Fig 39. Level 1 is new chain added to contain the link of every other node (or alternate node). Level 2 chain contains links to every 4th node. We keep on adding levels of chains so that we generalize that for level i chain includes 2^i th elements. After adding these pointers, the skip list is no more our old linked list; it has become sort of binary tree. Still, there are still few important things to consider.

- *Skip list contains a hierarchy of chains*
 - *In general level i contains a subset of elements in level $i-1$*
- Skip list becomes a hierarchy of chains and every level contains a subset of element of previous level. Using this kind of skip list data structure, we can find elements in $\log_2 n$ time. But the problem with this is that the frequency of pointers is so high as compared to the size of the data items that it becomes difficult to manage them. The *insert* and *remove* operations on this kind of skip list become very complex because single insertion or removal requires lot of pointers to readjust.

Professor Pugh suggested here that instead of doing leveling in powers of 2, it should be done randomly. Randomness in skip lists is a new topic for us. Let's see a formal definition of skip list.

Skip List - Formally

- A skip list for a set S of distinct (key, element) items is a series of lists $S_0, S_1, \dots, S_h$ such that
 - o Each list S_i contains the special keys $+\infty$ and $-\infty$
 - o List S_0 contains the keys of S in non-decreasing order. Each list is a subsequence of the previous one, i.e.,
$$S_0 \supseteq S_1 \supseteq \dots \supseteq S_h$$
 - o List S_h contains only the two special keys

You are advised to study skip list from your text books. The idea of randomness is new to us. We will study in the next lecture, how easy and useful becomes the skip list data structure after employing randomness.

Data Structures

Lecture No. 40

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 10

10.4.2

Summary

- Skip List
- Skip List Search
- Insertion in Skip List
- Deletion from Skip List

In the previous lecture, we had started the discussion on the concept of the skip lists. We came across a number of definitions and saw how the use of additional pointers was effective in the list structures. It was evident from the discussion that a programmer prefers to keep the data in the linked list sorted. If the data is not sorted, we cannot use binary search algorithm. And the insert, find and remove methods are proportional to n . But in this case, we want to use the binary search on linked list. For this purpose, skip list can be used. In the skip list, there is no condition of the upper limit of the array.

Skip List

A skip list for a set S of distinct (key, element) items is a series of lists $S_0, S_1, \dots, S_h$ such that

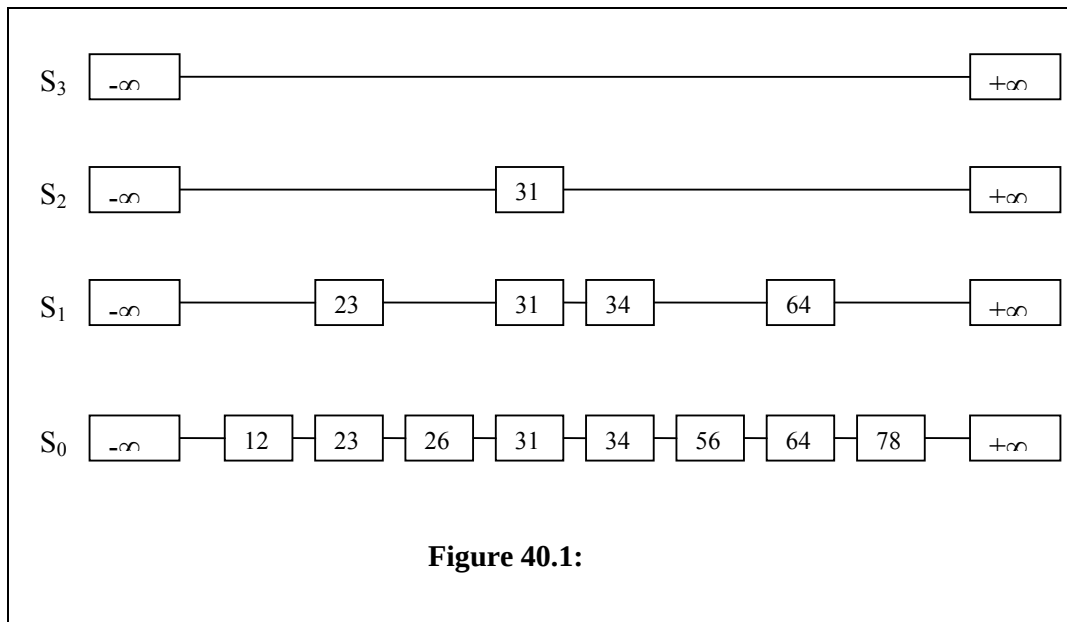
- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in non-decreasing order
- Each list is a subsequence of the previous one, i.e.,

$$S_0 \supseteq S_1 \supseteq \dots \supseteq S_h$$
- List S_h contains only the two special keys

Now let's see an example for the skip list. First of all, we have S_0 i.e. a linked list. We did not show the arrows in the list in the figure. The first and last nodes of S_0 contain $-\infty$ and $+\infty$ respectively. In the computer, we can put these values by $-\max(\text{int})$ and $\max(\text{int})$. The values can also be used about which we are sure that these will not be in the data items. However, for the sake of discussion to show these values, the $-\infty$ and $+\infty$ are the best notations. We can see that $-\infty$ is less than any value of data item while $+\infty$ is greater than any value of data item. If we insert any value much ever, it is large the $+\infty$ will be greater than it. Moreover, we see that the numbers in S_0 are in the non-decreasing order. This S_0 is the first list in which all keys are present.

Now we will take some nodes from this list and link them. That will be not every other node or every fourth node. It may happen this way. However, we will try that the node should not be every other or fourth node. It will be a random selection. Now we see S_1 i.e. a subset of S_0 . In S_1 we selected the nodes 23, 31, 34 and 64. We have

chosen these nodes randomly with out any order or preference. Now from this S1, we make S2 by selecting some elements of S1. Here we select only one node i.e. 31 for the list S2. The additional pointer, here, has to move from $-\infty$ to 31 and from 31 to $+\infty$. Now the next list i.e. S3 will be subset of S2. As there is only one node in S2, so in S3, there will only the special keys. In these lists, we use pointers to link all the nodes in S0. Then with additional pointers, we linked these nodes additionally in the other lists. Unlike 2i, there is not every other node in S1 or every fourth node in S2. The following figure represents these lists.



Now we have the list i.e. from S0 to S3. Actually, these list are made during insert operation. We will see the insert method later. Let's first talk about the search method.

Skip List Search

Suppose we have a skip list structure available. Let's see what is its benefit. We started our discussion from the point that we want to keep linked list structure but do not want to search n elements for finding an item. We want to implement the algorithm like binary search on the linked list.

Now a skip list with additional pointers is available. Let's see how search will work with these additional pointers. Suppose we want to search an item x . Then search operation can be described as under.

We search for a key x in the following fashion:

- We start at the first position of the top list
- At the current position p , we compare x with $y \leftarrow \text{key}(\text{after}(p))$
 - $x = y$: we return $\text{element}(\text{after}(p))$
 - $x > y$: we "scan forward"
 - $x < y$: we "drop down"
- If we try to drop down past the bottom list, we return *NO_SUCH_KEY*

To search a key x , we start at the first position of the top list. For example, we are discussing the top list is S_3 . We note the current position with p . We get the key in the list after the current list (i.e. in which we are currently looking for key) by the *key(after(p))* function. We get this key as y . Then we compare the key to be searched for i.e. x with this y . If x is equal to y then it means y is the element that we are searching for so we return *element(after(p))*. If x is greater than y , we scan forward and look at the next node. If x is less than y , we drop down and look in the down lists. No if we drop down and past the bottom list, it means that the element (item) is not there and we return *NO_SUCH_KEY*.

To see the working of this search strategy let's apply it on the skip list, already discussed and shown in the figure 40.1. This figure shows four lists. Remember that these four lists are actually in the one skip list and are made by the additional pointers in the same skip list. There is not such situation that S_1 is developed by extracting the data from S_0 and S_1 duplicates this data. Actually every node exists once and is pointed by additional pointers. For example, the node 23 exists once but has two next pointers. One is pointing to 26 while the other pointing to 31. In the same way, there are three pointers in node 31, two are to the node 34 and the third is toward the $+\infty$.

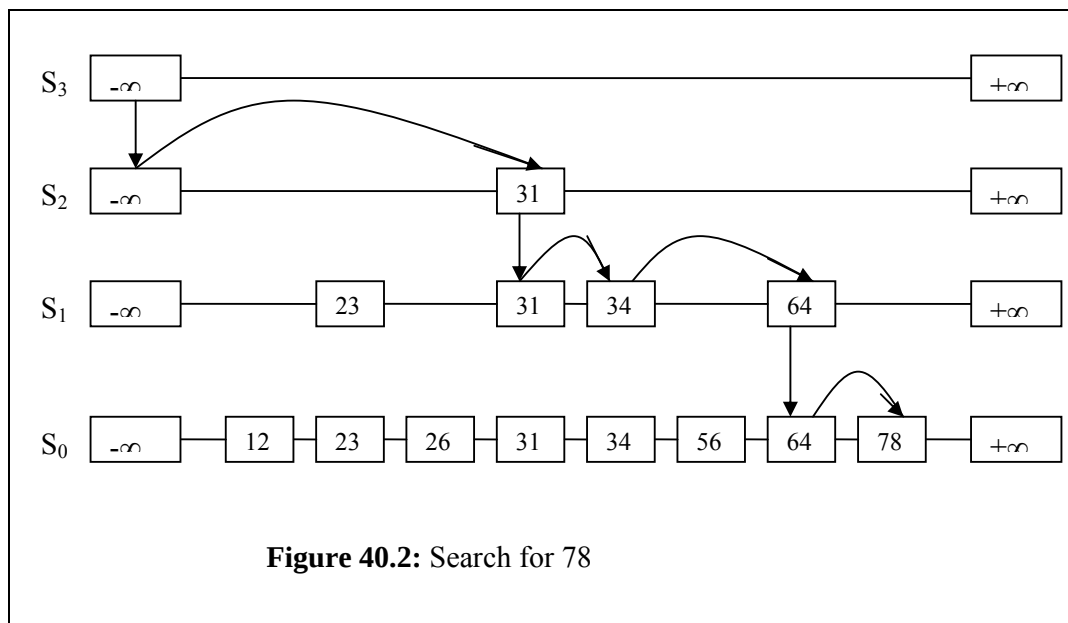


Figure 40.2: Search for 78

Suppose we want to search 78. We start from the first node of the top list i.e. S_3 . The 78 will be the current node and we denote it with p . Now we look at the value in the node after p . In the figure, it is $+\infty$. Now as the $+\infty$ is greater than 78, we drop down to S_2 . Note in the figure that we drop down vertically. We don't go to the first element p in the down list. This process of going down will be discussed later. Now we drop from S_3 to S_2 . This is our current pointer. Now we look for the value in the next node to it. We see that this value is 31. Now 31 is less than 78, so we will do scan forward. The next node is $+\infty$ that is obviously greater than 78. So we drop from here and go to the list S_1 . In this list, the current position is 34. We compare 78 with this node. As 34 is less than 78, we scan forward in the list. The next node in the list is 64 that is also less than 78. So we look at the next node and note that the next node is $+\infty$ that is greater than 78. Due to this we drop down to list S_0 . Here we look at the next node of 64 and find that this is 78. Thus at last we reach at the node that we are

searching for. Now if we look at the arrows that are actually the steps to find out the value 78 in the skip list. Then we come to know that these are much less than the links that we have to follow while starting from the $-\infty$ in the list S_0 . In S_0 we have to traverse the nodes 12, 23, 26, 31, 34, 44, 56, and 64 to reach at 78. This traversal could be larger if there were more nodes before 31. But in our algorithm, we reach at node 31 in one step. Thus we see that the search in the skip list is faster than the search in a linear list. By the steps that we do in searching an element and see that this search is like the binary search scheme. And we will see that this skip list search is also $\log_2 N$ as in binary search tree. Though the skip list is not a tree, yet its find method works like the binary search in trees. As we know that find method is also involved in the remove method, so remove method also becomes fast as the find method is fast.

Insertion in Skip List

When we are going to insert (add) an item (x, o) into a skip list, we use a randomized algorithm. Note that here we are sending the item in a pair. This is due to the fact that we keep the data and the key separately to apply the find and remove methods on table easily. In this pair, x is the key, also present in the record. The o denotes the data (i.e. whole record). Now the randomized algorithm for insertion of this value is described below.

The first step of the algorithm is that

- We repeatedly toss a coin until we get tails, and we denote with i the number of times the coin came up heads.

This first step describes that we toss a coin while knowing a 50 percent probability for both head and tail. We keep a counter denoted with i to count the heads that come up. Now if the head comes up, i becomes 1. We again toss the coin if again the head comes up we add 1 to the counter i . We continue this counting until the tail comes up. When tail comes up, we stop tossing and note the value of i .

After this, the second step of algorithm comes, stating that

- If $i \geq h$, we add to the skip list new lists $S_{h+1}, \dots, S_{i+1}$, each containing only the two special keys

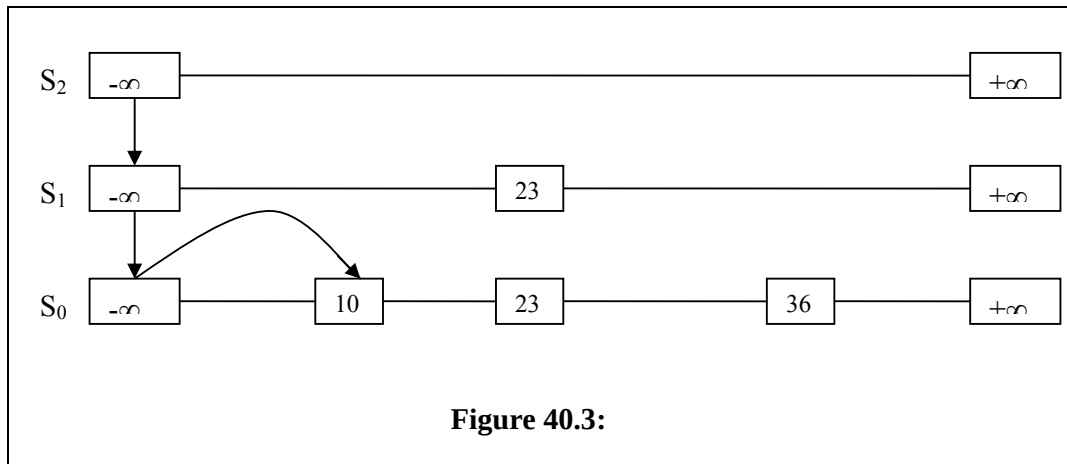
Here we compare i (that is the count of heads came up) with h (that is the number of list) if i is greater than or equal to h then we add new lists to the skip list. These new lists are $S_{h+1}, \dots, S_{i+1}$. Suppose if i is 8 and h is 4, we add additional lists S_5, S_6, S_7, S_8 and S_9 . These lists initially will contain the only two special keys that means $-\infty$ and $+\infty$. The next steps are:

- We search for x in the skip list and find the positions $p_0, p_1, \dots, p_i$ of the items with largest key less than x in each list $S_0, S_1, \dots, S_i$
- For $j \leftarrow 0, \dots, i$, we insert item (x, o) into list S_j after position p_j

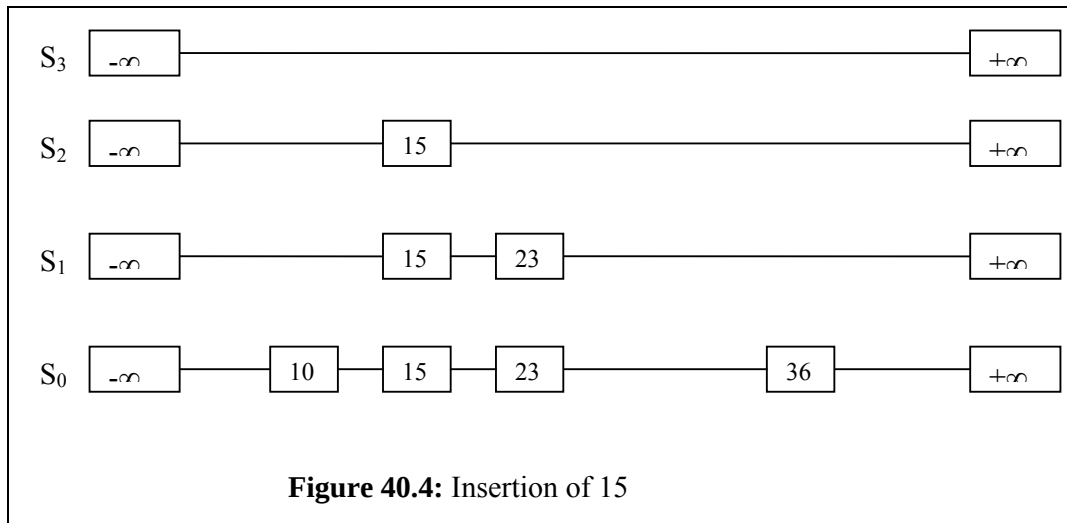
Now we will start from the left most node of the top list i.e. S_i and will find the position for the new data item. We find this position as we want to keep the data in the list sorted.

Let's see this insert position with the help of figures. Suppose we have a skip list,

shown in the figure below. Here are only three items 10, 23 and 36 in it. There are also additional layers S1 and S2. These were made when 10, 23 and 36 were inserted.



Now we proceed with this list and insert a value in it. The value that we are going to insert is 15. We have tossed the coin and figured out that the value of i is 2. As it is randomized algorithm so in this case, the value of i has become 2. The value of i is the count number of heads before the tail comes up by tossing the coin. From the above figure, we see that the value of h is 2. As h and i are equal, we are not adding S_3 , S_4 and S_5 to the list. Rather, we will apply the search algorithm. We start from the left most node of the top list. We call this position p_2 . We label these positions for their identification. We see that the item being added i.e. 15 is less than the $+\infty$, so we drop down to list S_1 . We denote this step with p_1 . In this list, the next value is 23 that is also greater than 15. So we again drop down and come in the list S_0 . Our current position is still the first node in the list as we did not have any scan forward. Now the next node is 10 that is less than 15. So we skip forward. We note this skip forward with p_0 . Now after p_0 the value in next node is 23 that is greater than 15. As we are in the bottom list, there is no more drop down. So here is the position of the new node. This position of new node is after the node 10 and before the node 23. We have labeled the positions p_0 , p_1 and p_2 to reach there. Now we add the value 15 additionally to the list that we have traversed to find the positions and labeled them to remember. After this we add a list S_3 that contains only two keys that are $-\infty$ and $+\infty$, as we have to keep such a list at the top of the skip list. The following figure shows this insertion process.



Here we see that the new inserted node is in three lists. The value that we insert must will be in S0 as it is the data list in which all the data will reside. However, it is also present in the lists that we have traversed to find its position. The value of i in the insertion method is randomly calculated by tossing the coin. How can we toss a coin in computers? There is routine library available in C and C++ that generates a random number. We can give it a range of numbers to generate a number in between them. We can ask it to give us only 0 and 1 and assign 1 to head and 0 to tail respectively. Thus the count number of 1's will give us the number of heads that come up. We can also use some other range like we can say that if the random number is less than some fixed value (whatever we fixed) that it means head otherwise it will mean tail. The algorithms that use random numbers are generally known as randomized algorithms. So

- A randomized algorithm performs coin tosses (i.e., uses random bits) to control its execution
- It contains statements of the type


```

b ← random()
if b <= 0.5 // head
    do A ...
else // tail
    do B ...
      
```
- Its running time depends on the outcome of the coin tosses, i.e, head or tail

In the randomized algorithm, we take a number from the random() function between 0 and 1. This number is up to one decimal place. However, we can keep it up to nay decimal places. As stated above ,after getting a number we check its value. If this is less than or equal to 0.5, we consider it as head and execute the process A. In our algorithm, we increase the value of i by 1. However, if value is greater than 0.5, we consider it as tail and do the process B. In our algorithm, the process B takes place when we note the value of i and stop the tossing. We do this process of tossing in a while loop. The while condition comes false when the tail (i.e. number greater than 0.5) comes. We cannot predict how many times this loop will execute as it depends upon the outcome of the toss. It is also a random number. There may be only one or a large number of heads before the tail comes up.

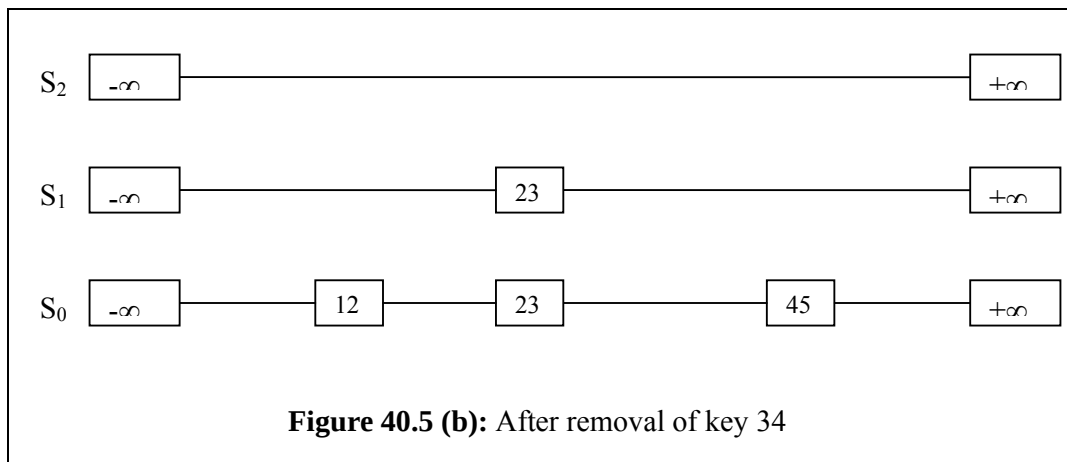
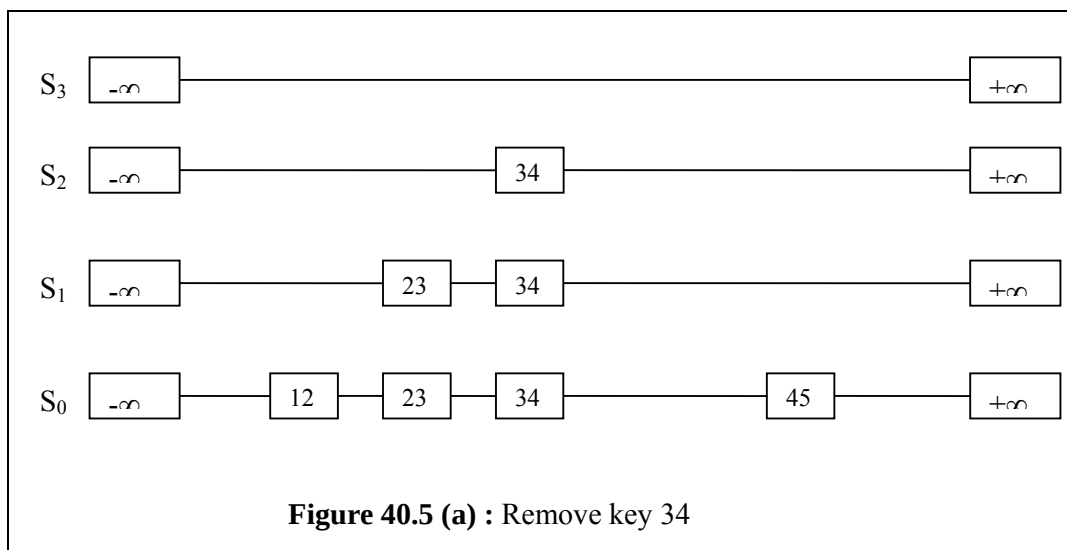
Deletion from Skip List

In the remove method, we find the item to be removed with the find item and remove it from the list. In the lists where ever this item has links, we bypass them. Thus, the procedure is quite easy. Now let's talk about this method.

To remove an item with key x from a skip list, we proceed as follows:

We search for x in the skip list and find the positions $p_0, p_1, \dots, p_i$ of the items with key x , where position p_j is in list S_j . This means that we look for the links of the item to be removed. We know that an item in the list has necessarily link in the list S_0 . Moreover it may have links in other lists up to S_i or say S_j . After this, we remove positions $p_0, p_1, \dots, p_i$ from the lists $S_0, S_1, \dots, S_i$. We remove all the lists except the list containing only the two special keys.

Let's consider the skip list shown in the figure below.



Suppose we want to remove the node 34 from the list. When this node 34 was inserted, it came not only in list S_0 but there were its additional links in S_1 and S_2 lists respectively. The list S_3 has only the two special keys. Now for finding 34, we

start from the top list. There is $+\infty$ as the next node, so we drop down to the list S2 as $+\infty$ is greater than 34. In S2, we are at node 34. Now we are at the point to remove 34. From here, we go to the remaining lists and reach list S0. We also label the links being traversed. It is evident that we have labeled the links as p2, p1 and p0. Now we remove the node 34 and change the pointers in the lists. We see that in S3, this was the single node. After removing this node, there is only one link that is from $-\infty$ to $+\infty$. The list S3 already has link from $-\infty$ to $+\infty$. Instead of keeping these two i.e. S2 and S3, we keep only S2. Thus we have removed the node 34 from the skip list. We see that the remove method is simple. We don't have randomness and need not tossing. The tossing and random number was only in the case of insertion. In the remove method, we have to change some pointers.

Thus in this data structure, we need not to go for rotations and balancing like AVL tree. In this data structure- Skip-list, we get rid of the limitation of arrays. This way, the search gets very fast. Moreover, we have sorted data and can get it in a sorted form.

Data Structures

Lecture No. 41

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 10
10.4.2

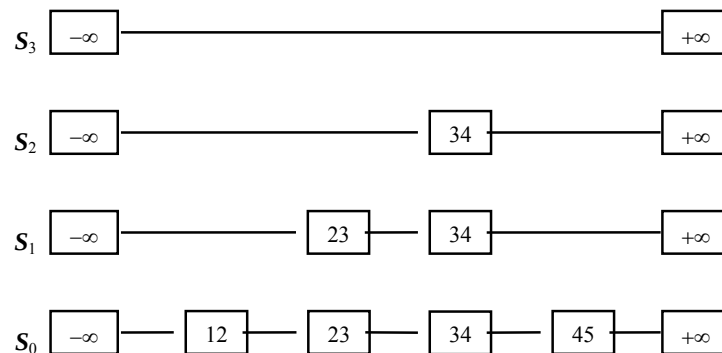
Summary

- Review
- Quad Node
- Performance of Skip Lists
- AVL Tree
- Hashing
- Examples of Hashing

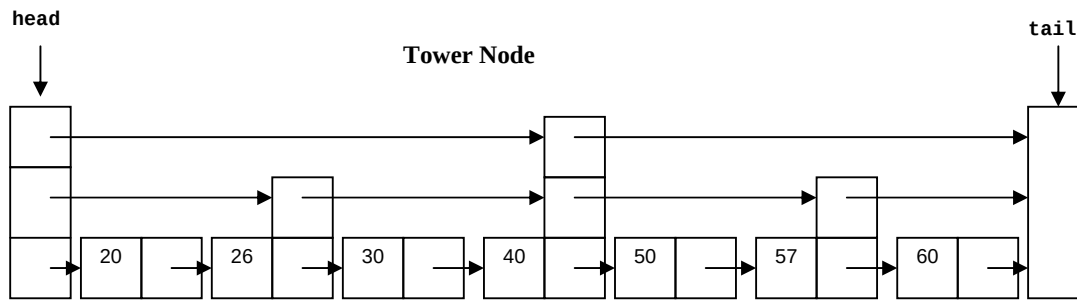
Review

In the previous lecture, we studied three methods of skip list i.e. *insert*, *find* and *remove* and had their pictorial view. Working of these methods was also discussed. With the help of sketches, you must have some idea about the implementation of the extra pointer in the skip list.

Let's discuss its implementation. The skip list is as under:



We have some nodes in this skip list. The data is present at 0, 1st and 2nd levels. The actual values are 12, 23, 34 and 45. The node 34 is present in three nodes. It is not necessary that we want to do the same in implementation. We need a structure with next pointers. Should we copy the data in the same way or not? Let's have a look at the previous example:



Here, the data is 20, 26, 30, 40, 50, 57, 60. At the lowest level, we have a link list. A view of the node 26, node 40 and node 57 reveals that there is an extra next 'pointer'. The head pointer is pointing to a node from where three pointers are pointing at different nodes.

We have seen the implementation of link list. At the time of implementation, there is a data field and a *next* pointer in it. In case of doubly link list, we have a *previous* pointer too. If we add an extra pointer in the node, the above structure can be obtained. It is not necessary that every node contains maximum pointers. For example, in the case of node 26 and node 57, there are two *next* pointers and the node 40 has three next pointers. We will name this node as 'TowerNode'.

TowerNode will have an array of *next* pointers. With the help of this array of pointers, a node can have multiple pointers. Actual number of *next* pointers will be decided by the random procedure. We also need to define *MAXLEVEL* as an upper limit on number of levels in a node. Now we will see when this node is created. A node is created at a time of calling the *insert* method to insert some data in the list. At that occasion, a programmer flips the coin till the time he gets a tail. The number of heads represents the levels. Suppose we want to insert some data and there are heads for six times. Now you know how much next pointers are needed to insert which data. Now we will create a *listNode* from the *TowerNode* factory. We will ask the factory to allocate the place for six *next* pointers dynamically. Keep in mind that the next is an array for which we will allocate the memory dynamically. This is done just due to the fact that we may require different number of *next* pointers at different times. So at the time of creation, the factory will take care of this thing. When we get this node from the factory, it has six next pointers. We will insert the new data item in it. Then in a loop, we will point all these next pointers to next nodes. We have already studied it in the separate lecture on *insert* method.

If your random number generation is not truly so and it gives only the heads. In this case, we may have a very big number of heads and the Tower will be too big, leading to memory allocation problem. Therefore, there is need to impose some upper limit on it. For this purpose, we use *MAXLEVEL*. It is the upper limit for the number of *next* pointers. We may generate some error in our program if this upper limit is crossed.

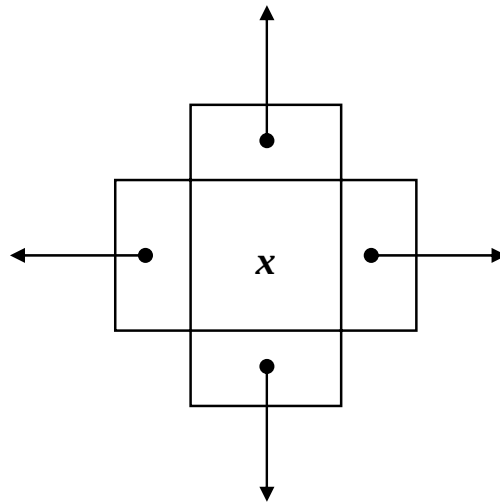
The *next* pointers of a node will point at their own level. Consider the above figure. Suppose we want to insert node 40 in it. Its 0 level pointer is pointing to node 50. The 2nd pointer is pointing to the node 57 while the third pointer pointing to tail. This is the case when we use *TowerNode*. This is one of the solutions of this problem.

Quad Node

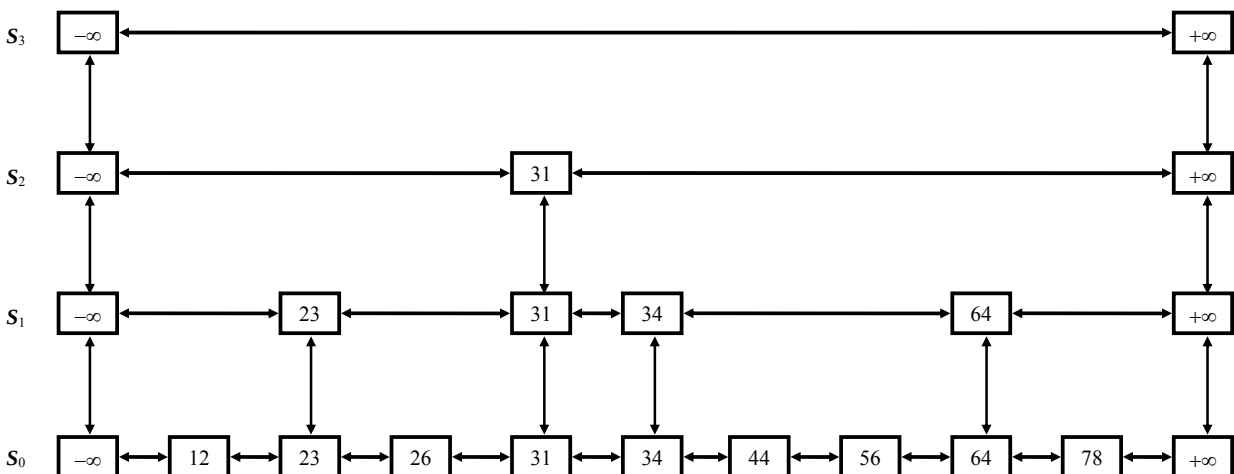
Let's review another method for this solution, called *Quad node*. In this method, we do not have the array of pointers. Rather, there are four *next* pointers. The following details can help us understand it properly.

A quad-node stores:

- item
- link to the node before
- link to the node after
- link to the node below
- link to the node above



This will require copying of the key (item) at different levels. We do not have an array of next pointers in it. So different ways are adopted to create a multilevel node of skip list. While requiring six levels, we will have to create six such nodes and copy the data item x in all of these nodes and insert these in link list structure. The following figure depicts it well.



You can see *next* and *previous* and *down* and *up* pointers here. In the bottom layer, the *down* pointer is nil. Similarly the right pointers of right column are nil. In the top

layer, the top pointers are nil. You can see that the values 23, 34, and 64 are copied two times and the value 31 is copied three times. What are the advantages of *quad node*? In *quad node*, we need not to allocate the array for next pointers. Every list node contains four pointers. The *quad node* factory will return a node having four pointers in it. It is our responsibility to link the nodes up, bottom, left and right pointers with other nodes. With the help of previous pointer, we can also move backward in the list. This may be called as doubly skip list.

Performance of Skip Lists

Let's analyze this data structure and see how much time is required for search and deletion process. The analysis is probability-based and needs lot of time. We will do this in some other course. Let's discuss about the performance of the skip list regarding insert, find and remove methods.

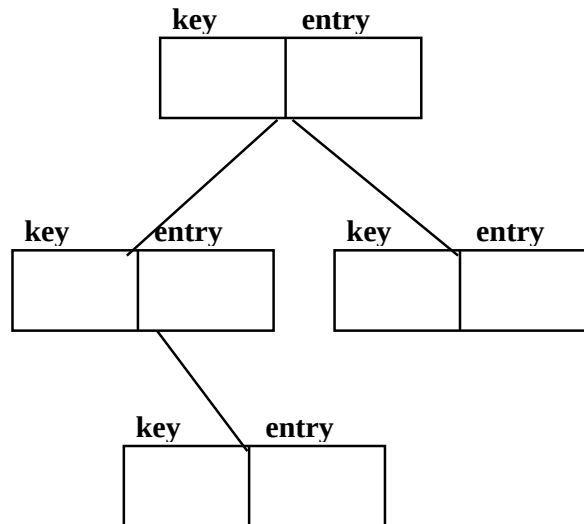
In a skip list, with n items the expected space used is proportional to n . When we create a skip list, some memory is needed for its items. We also need memory to create a link list at lowest level as well as in other levels. This memory space is proportional to n i.e. number of items. For n items, we need n memory locations. The items may be of integer data type. If we have 100 items, there will be need of 100 memory locations. Then we need space for *next* pointers that necessitates the availability of $2n$ memory locations. We have different layers in our structure. We do not make every node as *towerNode* neither we have a formula for this. We randomly select the *towerNode* and their height. The *next* pointers can be up to *maxLevel* but normally these will be few. We do not require pointers at each level. We do not need $20n$ or $30n$ memory locations. If we combine all these the value will be $15n$ to $20n$. Therefore the proportionality constant will be around 15 or 20 but it can't be n^2 or n^3 . If this is the case then to store 100 items we do need 100^2 or 100^3 memory locations. There are some algorithms in which we require space in square or cubic times. This is the space requirement and it is sufficient in terms of space requirements. It does not demand too much memory.

Let's see the performance of its methods. The expected search, insertion and deletion time is proportional to $\log n$. It looks like binary tree or binary search tree (BST). This structure is proposed while keeping in mind, the binary search tree. You have witnessed that if we have extra nodes, search process can be carried out very fast. We can prove it with the probabilistic analyses of mathematics. We will not do it in this course. This information is available in books and on the internet. All the searches, insertions and deletions are proportional to $\log n$. If we have 100,000 nodes, its $\log n$ will be around 20. We need 20 steps to insert or search or remove an element. In case of insert, we need to search that this element already exists or not. If we allow duplicate entries then a new entry would be inserted after the previous one. In case of delete too, we need to search for the entry before making any deletion. In case of binary search tree, the insertion or deletion is proportional to $\log n$ when the tree is a balanced tree. This data structure is very efficient. Its implementation is also very simple. As you have already worked with link list, so it will be very easy for you to implement it.

AVL Tree

The insertion, deletion and searches will be performed on the basis of key. In the nodes, we have key and data together. Keep in mind the example of telephone

directory or employee directory. In the key, we have the name of the person and the entry contains the address, telephone number and the remaining information. In our AVL tree, we will store this data in the nodes. Though, the search will be on the key, yet as we already noticed that the insert is proportional to $\log n$. Being a balanced tree, it will not become degenerated balance tree. The objective of AVL tree is to make the binary trees balanced. Therefore the find will be $\log n$. Similarly the time required for the removal of the node is proportional to $\log n$.



We have discussed all the five implementations. In some implementations, time required is proportional to some constant time. In case of a sorted list, we have to search before the insertion. However for an unsorted list, a programmer will insert the item in the start. Similarly we have seen the data structure where insertions, deletions and searches are proportional to n . In link list, insertions and deletions are proportional to n whereas search is $\log n$. It seems that $\log n$ is the lower limit and we cannot reduce this number more.

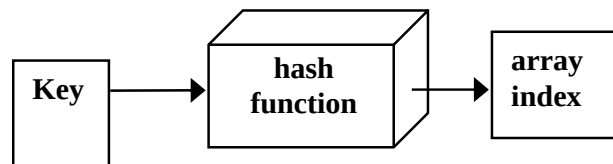
Is it true that we cannot do better than $\log n$ in case of table? Think about it and send your proposals. So far we have find, remove and insert where time varies between constant and $\log n$. It would be nice to have all the three as constant time operations. We do not want to traverse the tree or go into long loops. So it is advisable to find the element in first step. If we want to insert or delete, it should be done in one step. How can we do that? The answer is Hashing.

Hashing

The hashing is an algorithmic procedure and a methodology. It is not a new data structure. It is a way to use the existing data structure. The methods- *find*, *insert* and *remove* of table will get of constant time. You will see that we will be able to do this in a single step. What is its advantage? If we need table data structure in some program, it can be used easily due to being very efficient. Moreover, its operations are of constant time. In the recent lectures, we were talking about the algorithms and procedures rather than data structure. Now we will discuss about the strategies and methodologies. Hashing is also a part of this.

We will store the data in the array but *TableNodes* are not stored consecutively. We are storing the element's data in the *TableNodes*. You have seen the array implementation of the *Table* data structure. We have also seen how to make the data sorted. There will be no gap in the array positions whether we use the sorted or unsorted data. This means that there is some data at the 1st and 2nd position of array and then the third element is stored at the 6th position and 4th and 5th positions are empty. We have not done like this before. In case of link list, it is non-consecutive data structure with respect to memory.

In Hashing, we will internally use array. It may be static or dynamic. But we will not store data in consecutive locations. Their place of storage is calculated using the key and a *hash function*. Hash function is a new thing for you. See the diagram below:



We have a key that may be a name, or roll number or login name etc. We will pass this key to a hash function. This is a mathematical function that will return an array index. In other words, an integer number will be returned. This number will be in some range but not in a sequence.

Keys and entries are scattered throughout the array. Suppose we want to insert the data of our employee. The key is the name of the employee. We will pass this key to the hash function which will return an integer. We will use this number as array index. We will insert the data of the employee at that index.

	key	entry
4		
10		
123		

The insert will calculate place of storage and insert in *TableNode*. When we get a new data item, its key will be generated with the help of hash function to get the array

index. Using this array index, we insert the data in the array. This is certainly a constant time operation. If our hash function is fast, the insert operation will also rapid. It will take only one step to perform this.

Next we have *find* method. It will calculate the place of storage and retrieve the entry. We will get the key and pass it to the hash function and obtain the array index. We get the data element from that array position. If data is not present at that array position, it means data is not found. We do not need to find the data at some other place. In case of binary search tree, we traverse the tree to find the element. Similarly in list structure we continue our search. Therefore find is also a constant time operation with Hashing.

Finally, we have *remove* method. It will calculate the place of storage and set it to null. That means it will pass the key to the hash function and get the array index. Using this array index, it will remove the element.

Examples of Hashing

Let's see some examples of hashing and hash functions. With the help of these examples you will easily understand the working of *find*, *insert* and *remove* methods.

Suppose we want to store some data. We have a list of some fruits. The names of fruits are in string. The key is the name of the fruit. We will pass it to the hash function to get the hash key.

Suppose our hash function gave us the following values:

HashCode ("apple")	=	5
hashCode ("watermelon")	=	3
hashCode ("grapes")	=	8
hashCode ("cantaloupe")	=	7
hashCode ("kiwi")	=	0
hashCode ("strawberry")	=	9
hashCode ("mango")	=	6
hashCode ("banana")	=	2

Our hash function name is *hashCode*. We pass it to the string "apple". Resultantly, it returns a number 5. In case of "watermelon" we get the number 3. In case of "grapes" there is number 8 and so on. Neither we are sending the names of the fruits in some order to the function, nor is function returning the numbers in some order. It seems that some random numbers are returned. We have an array to store these strings. Our array will look like as:

0	kiwi
1	
2	banana
3	watermelon
4	
5	apple
6	mango
7	cantaloupe
8	grapes
9	strawberry

We store the data depending on the indices got from the *hashCode*. The array size is 10. In case of apple, we get the index 5 from *hashCode* so “apple” is stored at array index 5. As we are dealing with strings, so the array will be an array of strings. The “watermelon” is at position 3 and so on every element is at its position. This array will be in the private part of our data structure and the user will not know about it. If our array is *table* then it will look like as under:

```
table[5] = "apple"
table[3] = "watermelon"
table[8] = "grapes"
table[7] = "cantaloupe"
table[0] = "kiwi"
table[9] = "strawberry"
table[6] = "mango"
table[2] = "banana"
```

We will store our data in the Table array using the string copy. The user is storing the data using the names of the fruits and wants to retrieve or delete the data using the names of fruits. We have used the array for storage purposes but did not store the data consecutively. We store the data using the hash function which provides us the array index. You can note that there are gaps in the array positions.

Similarly we will retrieve the data using the names of fruit and pass it to the *hashCode* to get the index. Then we will retrieve the data at that position. Consider the table array, it seems that we are using the names of fruits as indices.

```
table["apple"]
table["watermelon"]
table["grapes"]
table["cantaloupe"]
table["kiwi"]
```

```
table["strawberry"]
table["mango"]
table["banana"]
```

We are using the array as table ["apple"], table ["watermelon"] and so on. We are not using the numbers as indices here. Internally we are using the integer indices using the *hashCode*. Here we have used the fruit names as indices of the array. This is known as associative array. Now this is the internal details that we are thinking it as associative array or number array.

Let's discuss about the *hashCode*. How does it work internally? We pass it to strings that may be persons name or name of fruits. How does it generate numbers from these? If the keys are strings, the hash function is some function of the characters in the strings. One possibility is to simply add the ASCII values of the characters. Suppose the mathematical notation of hash function is h . It adds all the ASCII values of the string characters. The characters in a string are from 0 to $length - 1$. Then it will take mod of this result with the size of the table. The size of the table is actually the size of our internal array. This formula can be written mathematically as:

$$h(str) = \left(\sum_{i=0}^{length-1} str[i] \right) \% TableSize$$

Example : $h(ABC) = (65 + 66 + 67) \% TableSize$

Suppose we use the string "ABC" and try to find its hash value. The ASCII values of A, B and C are 65, 66 and 67 respectively. Suppose the tableSize is 55. We will add these three numbers and take mod with 55. The result (3.6) will be the hash value. To represent character data in the computer ASCII codes are used. For each character we have a different bit pattern. To memorize this, we use its base 10 values. All the characters on the keyboard like \$, %, ' have ASCII values. You can find the ASCII table in your book or on the internet.

Let's see the C++ code of *hashCode* function.

```
int hashCode( char* s )
{
    int i, sum;
    sum = 0;
    for(i=0; i < strlen(s); i++ )
        sum = sum + s[i]; // ascii value
    return sum % TABLESIZE;
}
```

The return type of *hashCode* function is an integer and takes a pointer to character. It declares local variable i and sum , then initializes the sum with zero. We use the *strlen* function to calculate the length of the string. We run a loop from 0 to $length - 1$. In the loop, we start adding the ASCII values of the characters. In C++, characters are

stored as ASCII values. So we directly add $s[i]$. Then in the end, we take mod of sum with $TABLESIZE$. The variable $TABLESIZE$ is a constant representing the size of the table.

This is the one of the ways to implement the hash function. This is not the only way of implementing hash function. The hash function is a very important topic. Experts have researched a lot on hash functions. There may be other implementations of hash functions.

Another possibility is to convert the string into some number in some arbitrary base b (b also might be a prime number). The formula is as:

$$h(str) = \left(\sum_{i=0}^{length-1} str[i] \times b^i \right) \% T$$

$$Example: h(ABC) = (65b^0 + 66b^1 + 67b^2) \% T$$

We are taking the ASCII value and multiply it by b to the power of i . Then we accumulate these numbers. In the end, we take the mod of this summation with $tableSize$ to get the result. The b may be some number. For example, we can take b as a prime number and take 7 or 11 etc. Let's take the value of b as 7. If we want to get the hash value of ABC using this formula:

$$H(ABC) = (65 * 7^0 + 66 * 7^1 + 67 * 7^2) \bmod 55 = 45$$

We are free to implement the hash function. The only condition is that it accepts a string and returns an integer.

If the keys are integers, $key \% T$ is generally a good hash function, unless the data has some undesirable features. For example, if $T = 10$ and all keys end in zeros, then $key \% T = 0$ for all keys. Suppose we have employee ID i.e. an integer. The employee ID may be in hundreds of thousand. Here the table size is 10. In this case, we will take mod of the employee ID with the table size to get the hash value. Here, the entire employee IDs end in zero. What will be the remainder when we divide this number with 10? It will be 0 for all employees. So this hash function cannot work with this data. In general, to avoid situations like this, T should be a prime number.

Data Structures

Lecture No. 42

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 5
5.1, 5.2, 5.4.1, 5.4.2

Summary

- Collision
- Linear Probing

In the previous lecture, we were discussing about the hash functions. The hash algorithm depends on the hash function. The hash function generates the array index to enable us to insert data in the table array. We have seen two examples of hash functions in the previous lecture. Both the functions use the ASCII values of characters to generate the index. Here the question arises how can we implement the hash function in case of having integer data? We may have employee ID, user ID or student ID as integers. Here we may take mod with some number or table size and the result is used as an array index.

If the keys are integers then $key \% T$ is generally a good hash function unless the data has some undesirable features. If we want to store the employee record, user record or student record in the table, this can be done through hash function. We take the mod of the value with the T . The value of T may be 10, 15 or 100 depending on the requirements. There may be some problem. For example, if $T = 10$ and all keys end in zeros, then $key \% T = 0$ for all keys. The hash function gives 0 for all the keys, used as array index. Now it is a problem. We cannot store our values as all the records have same index i.e. 0. In general, to avoid such situations, T should be a prime number. Internally, we have to store the data in the array and there is complete freedom to use this array by taking the size of our own choice.

We have to store some data in the array. As the array is private, we will decide about its size on our own. We will take the size of the array in prime numbers. To store 100 records, we will take prime number near 100. We will select this prime number as *MAXTABLESIZE*. Then we will use this number in our hash function. This will help resolve the problem arising due to situation where all keys end with 0. Using the prime number, the values from the hash function will not be 0 for all the keys.

With the help of prime number, we cannot solve this problem completely. Similarly, it cannot be made sure that the values from the hash function are unique for all the keys. Sometimes, we may have same index for two different keys. This phenomenon is known as collision i.e. the hash values are same of two different keys. How can we solve this collision problem?

Collision

Collision takes place when two or more keys (data items) produce the same index.

Let's see the previous example of storing the names of fruits. Suppose our hash function gives us the following values:

hash("apple")	=	5	0	kiwi
hash("watermelon")	=	3		
hash("grapes")	=	8	1	
hash("cantaloupe")	=	7		
hash("kiwi")	=	0	2	banana
hash("strawberry")	=	9		
hash("mango")	=	6	3	watermelon
hash("banana")	=	2		
			4	
			5	apple
			6	mango
			7	cantaloupe
			8	grapes
			9	strawberry

We store these data items in the respective index in the array. In the above example, the index given by the hash function does not collide with any other entry. Suppose we want to add another fruit “honeydew” in it. When “honeydew” is passed to the hash function, we get the value 6 i.e.

$$\text{hash}(\text{"honeydew"}) = 6$$

This is not the responsibility of the hash function to see the data in the array before generating the index. Hash function is generally a mathematical formula that takes the keys and returns a number. It is responsibility of the caller to find its solution. We have already “mango” at position 6. The user of our ADT calls the insert function giving the value “honeydew”. We call the hash function to find out its index that comes out to be 6. Now the problem is this that position 6 is already occupied. What should we do to avoid it?

There are a lot of solutions of this problem. These solutions can be divided into two main categories. One type of solutions is the changing of the hash function. Even with the introduction of a new function, it is not guaranteed that there will be no collision with future data. Second option is that we live with the collision and do something to resolve it.

The definition of collision is:

“When two values hash to the same array location, this is called a *collision*”

We cannot say that the usage of this hash function will not result in collision

especially when the data is changing. Collisions are normally treated as a phenomenon of “first come, first served”, the first value that hashes to the location gets it. We have to find something to do with the second and subsequent values that hash to the same location.

As we have seen above in the example that at position 6, there is the data item *mango* while another item *honeydew* is trying to acquire the same position. First come first served means that the *mango* will remain at its position and we will do something for *honeydew*. We will see three solutions for this problem:

Solution #1: Search for an empty location

- Can stop searching when we find the value or an empty location.
- Search must be wrap-around at the end.

Let's apply this solution on the above example. We have *mango* at position 6. Now the position for *honeydew* is also 6. We find another empty location in the array. Keep in mind that in hashing, we do not store data at consecutive positions. Rather, data is scattered. We know that there are some empty locations in the array. We do not want to refuse the user that we cannot store the data due to the occupation of the position by some other item. Therefore, we will store the data item at the empty location. We will see an example in a short while.

Solution #2: Use a second hash function

- ...and a third, and a fourth, and a fifth, ...

We have a primary hash function, we pass it to the string *honeydew*. It returns 6. As the position 6 is occupied, so we call another hash function that is implemented in a different way. This hash function will return an integer. If this location is empty, we will store our data here and the problem is solved. But if the array location at the index number returned by this function is also occupied. Then we will call the third hash function. There is a possibility of collision so we will have three, four, five or more hash functions. When there is a collision, we call these hash functions one by one till the time, there is an index with empty location. We will store the data at this empty location.

Solution #3: Use the array location as the header of a linked list of values that hash to this location

In this solution, we will not store the data directly in the array. Our array will be an array of pointers to *TableNode*. We will create a list node and store the data in it. The array will have a pointer to the node. If we want to store some data at location 6, we will store a pointer at location 6 that points to the node containing the data. When we have another data at the position 6, we create a list node and attach it with the previous node. There is a possibility of having a link list from each location of the table. At least, it will have one node.

There are some more strategies for the resolution of this problem. But we will study only the above-mentioned three solutions. Now we will see how we can implement the methods of *Table* and *Dictionary ADT* using the hashing function. Firstly, we will see how the *insert*, *find* and *remove* methods will work with the first solution.

Linear Probing

The first solution is known as open addressing. When there is a collision, we try to find some other place in our array. This approach of handling collisions is called *open addressing*; it is also known as *closed hashing*. The word open is used with addressing and the word closed is used with hashing. Be careful when naming these. More formally, cells at $h_0(x)$, $h_1(x)$, $h_2(x)$, ... are tried in succession where

$$h_i(x) = (\text{hash}(x) + f(i)) \bmod \text{TableSize}, \quad \text{with } f(0) = 0.$$

Here *hash* is our hash function. If there is some collision, we add $f(i)$ value to it before taking its mod with *TableSize*. The function, f , is the collision resolution strategy.

We use $f(i) = i$, i.e., f is a linear function of i . Thus

$$\text{location}(x) = (\text{hash}(x) + i) \bmod \text{TableSize}$$

The function f can be any function. So our first implementation of f is that whatever integer (i) we gave to it, it returns it back. That is $f(i)$ is a linear function. We can implement function f as it returns the square of i . That will be quadratic function. Similarly, we can have cubic functions. We are free to implement it.

In the above example, we are given x for insertion. This can be string such as *honeydew*. We call the $\text{hash}(x)$ that returns a number. We will see that this location is empty or not. In case of collision, we add i to this number. The value of i can be 0,1,2,3 etc. We will take the mod of this value with *TableSize* and try to store the data there. The collision resolution strategy is called *linear probing* as it scans the array sequentially (with wrap around) in search of an empty cell. Let's consider an example to understand how it works. Keep in mind that when there is a collision, some other location in the array is found. This is known as linear probing. Here we are trying to probe the array linearly that where is the empty location.

Let's see an example. Here we have names of birds being stored in the array. We have a larger array. We already have some data in the array that is as under:

...	
141	
142	robin
143	sparrow
144	hawk
145	seagull
146	
147	bluejay
148	owl
...	

This is a part of array. There is some data before and after these locations. As shown above, the locations 142, 143, 144, 145, 147 and 148 contains data. Suppose we want to add *seagull* to this hash table. With respect to ADT, someone is using this *Table* or *Dictionary* and calls the *insert* method of this *Table*. Now we have to use the *seagull* as key and have to store its associated data. We will call the *hashCode* function to get the array index and pass it the string “seagull” as:

$\text{hashCode}(\text{“seagull”}) = 143$

Now the location 143 is not empty and we have a bird, *sparrow* at this location. That is $\text{table}[143]$ is not empty. The *seagull* is different than *sparrow* i.e. $\text{table}[143] \neq \text{seagull}$. If it already exists and we are not allowing duplicates, we can say that the data already exists. This is not the case here. We will check the next position, $143+1$ (here i is 1) i.e. 144. At location 144, we have a bird *hawk*. So this position is also not empty and not equal to *seagull*. Here also we have different data. Now we check the next position $143+2$ (here i is 2) i.e. 145. This location is empty so we store our data at this location. We have two collisions here at 143 and 144 positions. We have found the empty position in the array with single jumps. First we add 1, then 2 and so on.

Suppose we want to add some existing data. As collisions are happening here, so when we get the index value from hash function, the data may not be at that position. We have to follow this linear chain. Suppose you want to add *hawk* to this hash table. The table is as under:

...	
141	
142	robin
143	sparrow
144	hawk
145	seagull
146	
147	bluejay
148	owl
...	

We know that *hawk* is already in the table. Also *seagull* is added in the table. We will see that that data that we want to insert already exists or not. First of all, we call the hash function that is *hashCode* as:

$\text{hashCode}(\text{“hawk”}) = 143$

We use this value as the table index to insert this data in the table. The *table[143]* is not empty. Then we check the data at position 143. We have stored *sparrow* here so *table[143] != hawk*. At this point, we cannot decide whether the data item *hawk* exists in the table or not. As we are doing linear probing and there are chances that this data may exist at some other location. Therefore we will check it further taking any final decision. Now we check the next position i.e. 144. It is also not empty but the data here is *hawk* that is *table[144] == hawk*. We have two options here. One is that *hawk* is already in the table, so do nothing. The other option is that user might want to modify the data associated with *hawk*.

The size of our internal array is fixed and we store it in the constant *TableSize*. In linear probing, at the time of collisions, we add one to the index and check that location. If it is also not empty, we add 2 and check that position. Suppose we keep on incrementing the array index and reach at the end of the table. We were unable to find the space and reached the last location of the array. Now what does it mean? Is the table completely filled? There are chances that we started from the position 143. Suppose that the table size is 500. We were unable to find any empty location. Can we say that the array locations before 143 are also not available? There is a possibility that we find some empty spaces there. We will not stop at the end of the table and go to the start of this table array. We will start linear probing from the start and if we find some empty spaces, the data will be stored there. This process is called wrap around. There is a chance that in wrap around, we could not find space and come back to the 143 position. The solution of this problem is to get a larger array. We may want to create an array of size 1000 or some prime number larger than 500. Then copy all the data in this new array. This new data may be inserted with the help of linear probing. The other solution is that we may want to keep multiple arrays. In this case, when one array is filled we go to the second array.

Let's see the example of wrap around process. Suppose we want to add *cardinal* to this hash table. We will use the above table to insert this data. Also suppose that 148 is the last position of this table. We call the *hashCode* function as:

`hashCode("cardinal") = 147`

We get the array index as 147. This position is already filled in the table. The data at this position is *bluejay*. So we will add 1 to the index and check the 148 position. This position is also filled. We do not stop here and go back to the start of the array. We cannot increment the array index further. We will treat the table as circular; after 148 comes 0. Now we will check the position 0. If it is occupied, we will see the position 1 then 2 and so on. We have seen the *insert* method using the linear probing. Now we will see the *find* method.

Suppose we want to find *hawk* in this hash table. We call the *hashCode* as:

`hashCode("hawk") = 143`

The *table[143]* is not empty. However, the data at this position is not *hawk*. So with the linear probing procedure, we add 1 to the index and check the 144 position. This position is also not empty. We compare it with *hawk* that is true. We use the same procedure for looking things up in the table as generally done for inserting them.

Let's have a look on the delete procedure. If an item is placed in $array[hash(key)+4]$, the item just before it is deleted. How will probe determine that the "hole" does not indicate the item is not in the array? We may have three states for each location as:

- Occupied
- Empty (never used)
- Deleted (previously used)

Using the linear probe, we insert data where we get some empty space in the array. Firstly, we try to insert it at the index position given by the hash function. If it is occupied, we move to the next position and so on. In this way, we have a chain to follow. Now if an element of the chain is deleted, how can we know that it was filled previously. As we have seen in the above example that *seagull* collided twice before getting a space in the array. Despite having a hash value of 143, it was stored at 145. The three locations 143, 144 and 145 have different data values but are a part of a chain. These three names collide with each other with respect to hash function. Suppose we delete the 2nd name i.e. *hawk* due to some reason. Now there are two names, which collide. A space has been created due to the deletion of *hawk*. If we try to search *seagull* using the linear probing, there will be a hash value of 143. It is filled but has different data. We move to the next position that is empty. How can linear probing know that do not stop here because there was some data previously at this position, which has not been deleted. And there is some data after this position, which is part of this chain due to the collisions and that data is *seagull*.

To deal with such situations, we keep three states in our array. 1) It is empty (never used) and has no data in it. 2) It is filled with some legal data and is occupied and 3) it had some data which is now deleted and currently it is empty. We will use this state if we have to go to next position using the linear probing.

One problem with linear probing technique is the tendency to form "clusters". A cluster is a group of items not containing any open slots. The bigger a cluster gets, the more likely the new values will hash into the cluster, and make it even bigger. Clusters cause efficiency to degrade. Suppose we want to add another bird in our table. The hash function returns the index as 143. Now we have already three items that collided at 143. The array locations at 143, 144, 145 are already occupied. So we will insert this new data at 146 using the linear probing. The data is getting gathered instead of scattering because linear probing inserts the data in the next position. It seems as the normal use of the array in which we insert data in the array from first position then next position and so on. It may depend on our data or our hash function. This gathering of data is called clustering.

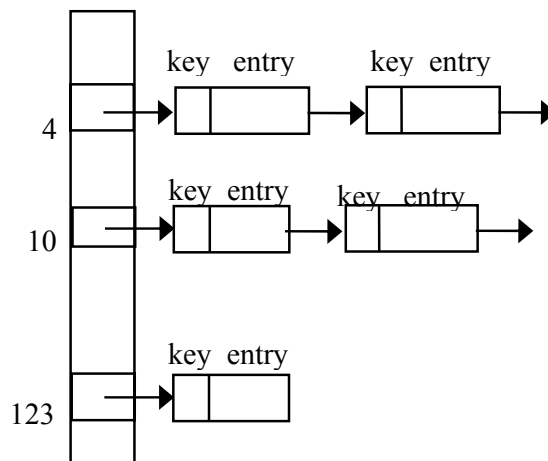
We were trying to store the data in the array in a constant time or in a single step. Similarly the *find* and *remove* methods should be of constant time. That attribute has now been lost. What should we do now? One of the solutions is quadratic probing.

Quadratic probing uses different formula:

- Use $F(i) = i^2$ (square of i) to resolve collisions
- If hash function resolves to H and a search in cell H is inconclusive, try $H + 1^2, H + 2^2, H + 3^2, \dots$

In the quadratic probing when a collision happens we try to find the empty location at $\text{index} + 1^2$. If it is filled then we add 2^2 and so on. Let's take the above example. We want to insert seagull. The hash function generates the value 143 and this location is already occupied. Now we will start probing. First of all, we add 1^2 in the 143 and have 144. It is also not empty. Now we will add 2^2 in the 143 and have 147. If it is also occupied, we will add 3^2 in 143 and we have 152. The data is now getting scattered. Unfortunately, there are some problems with quadratic probing also.

Let's discuss the third solution that we will use the link list for the collision resolution. Each table position is a linked list. When we are going to insert new data, we will insert the keys and entries anywhere in the list (front easiest). It is shown in the below diagram.



On the left side, we have vertical array that contains the pointers in it. When we insert the first item, we attach a list node. When we have collision, the new item is inserted in the start of the link list. Suppose an item is stored at position 4 and we have another data, requiring the position 4. So there is a collision and we will insert the data at the start of the list.

Let's compare the link list and open addressing.

- Advantages over open addressing:
 - Simpler insertion and removal
 - Array size is not a limitation
- Disadvantage
 - Memory overhead is large if entries are small.

The problem in linear probing is that when our array is full what we should do. This problem can be solved using the link list.

In the next lecture, we will continue our discussion on hashing. We will see an animation. Hashing is very important methodology and can be used in data structures besides *Table* and *Dictionary*.

Data Structures

Lecture No. 43

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 5, 7
5.4, 5.5, 5.6, 7.1

Summary

- Hashing Animation
- Applications of Hashing
- When Hashing is Suitable?
- Sorting
- Elementary Selection Algorithms
- Selection Sort

Hashing Animation

In the previous lecture, we discussed about collision strategies in hashing. We studied three solutions, Linear Probing, Quadratic Probing and Linked List chaining. Hashing is vast research field, which covers hash functions, storage and collision issues etc. At the moment, we will see hashing in implementation of table ADT. Operations of *insert*, *delete* and *find* are performed in constant time using this hashing strategy. Constant time means the time does not increase with the increase in data volume. However, if collisions start happening then the time does not remain constant. Especially if we see *linear probing*, we had to insert the data by sorting the array sequentially. Similar was the case with *quadratic*. In case of *linked list*, we start constructing linked list that takes time and memory. But later we will see some situations where hashing is very useful.

Today, we will study these three strategies of hash implementation using animations. These animations will be provided to you in a *Java* program. It is important to mention here that all the data structures and algorithms we have studied already can be implemented using any of the languages of *C/C++* or *Java*. However, it is an important decision to choose the programming language because every language has its strong area, where it has better application. *Java* has become very popular because of its facilities for *Internet*. As you already know *C++*, therefore, *Java* is easy to learn for you. The syntax is quite similar. If we show you the java code you will say it is *C++*.

Let's see the hashing animation. This animation will be shown in the browser. This is an applet written in java language. We will see linear probing, quadratic probing and link list chaining in it. This is an example of how do we solve collision.

We have an array shown in four different columns. The size of the array is 100 and the index is from 0 to 99. Each element of the array has two locations so we can store 200 elements in it. When we have first collision the program will use the 2nd part of the array location. When there is a 2nd collision the data is stored using the linear probing. At the top right corner we have hash function x and its definition is “mod

100". That is when a number is passed to it, it will take mod with 100 and return the result which is used as index of the array.

In this example we are using numbers only and not dealing with the characters. We also have a statistical table on the right side. This program will generate 100 random numbers and using the hash function it will store these in the array. The numbers will be stored in the array at different locations. In the bottom left we have hashing algorithms. We have chosen the linear probing. Now the program will try to solve this problem using the linear probing. Press the run button to start it. It is selecting the numbers randomly, dividing it by 100 and the remainder is the hash value which is used as array index.

Here we have number 506. It is divided by 100 and the remainder is 6. It means that this number will be stored at the sixth array location. Similarly we have a number 206, now its remainder is also 6. As location 6 is already occupied so we will store 206 at the 2nd part of the location 6. Now we have the number 806. Its remainder is also 6. As both the parts of location 6 are occupied. Using the linear probing we will store it in the next array location i.e. 7. If we have another number having the remainder as 6, we will store it at the 2nd part of location 7. If we have number 807, its remainder is 7. The location 7 is already occupied due to the linear probing. Therefore the number 807 will be stored using the linear probing in the location 8. Now you can understand how the numbers are stored in the array. You can also see some clustering effect in it. See the location 63. All the numbers having remainder as 63 are clustered around location 63.

Let's change the collision resolution algorithm to quadratic probing. Run the animation again. Now we have array size as 75 and the array is shown in three columns. Each location of the array can store two numbers. In quadratic probing we add square of one first i.e. 1 and then the square of two and so on in case of collisions. Here we have used a different hash function. We will take the mod with 75. When the both parts of the array location is filled we will use the quadratic probing to store the next numbers. Analyze the numbers and see where the collisions have happened.

Lets see the animation using the linked list chaining. Now the hash function uses 50 to take mod with the numbers. So far pointers are not shown. When both parts of the location are filled, we will see the link list appearing. We have four numbers having remainder 0. The two numbers will be stored in the array and the next two will be stored using the link list which is attached at the 0 location.

We are not covering the hashing topic in much depth here as it is done in algorithms and analysis of algorithms domain. This domain is not part of this course. For the time being, we will see the usage of hashing. For certain situations, table ADT can be used, which internally would be using hashing.

Applications of Hashing

Let's see few examples of those applications where hashing is highly useful. The hashing can be applied in table ADT or you can apply hashing using your array to store and retrieve data.

- *Compilers use hash tables to keep track of declared variables (symbol table).*

Compilers use hash tables in order to implement *symbol tables*. A *symbol table* is an important part of compilation process. Compiler puts variables inside symbol table during this process. Compiler has to keep track of different attributes of variables. The name of the variable, its type, scope and function name where it is declared etc is put into the symbol table. If you consider the operations on symbol table, those can be insertion of variable information, search of a variable information or deletion of a variable. Two of these *insert* and *find* are mostly used operations. You might have understood already that a variable name will be parameter (or the key) to these operations. But there is one slight problem that if you named a variable as *x* outside of a code block and inside that code block, you declared another variable of the similar type and name then only name cannot be the key and scope is the only differentiating factor. Supposing that all the variables inside the program have unique names, variable name can be used as the key. Compiler will insert the variable information by calling the *insert* function and by passing in the variable information. It retrieves the variable value by passing in the variable name. Well, this exercise is related to your Compiler Construction course where you will construct your own language compiler.

Another usage of hashing is given below:

- *A hash table can be used for on-line spelling checkers — if misspelling detection (rather than correction) is important, an entire dictionary can be hashed and words checked in constant time.*

You must have used spell checkers in a word processing program like MS Word. That spell checker finds out mistakes, provides you correct options and prompts you to choose any of the synonyms. You can also set the correct the words automatically.

Hashing can be used to find the spelling mistakes. For that you first take all the words from the dictionary of spoken English and construct a hash table of those. To find the spelling mistakes, you will take first word from the text that is being checked and compare it with all the words present inside the hash table. If the word is not found in the hash table then there is a high probability that the word is incorrect, although there is a low probability that the word is correct but it is not present in the dictionary. Based on the high probability a message can be displayed to the user of the application that the word is wrong. MS Word does the same. As far the automatic correct feature is concerned, it is another algorithm, which we are not going to discuss here.

Let's see few more examples in this connection.

- *Game playing programs use hash tables to store seen positions, thereby saving computation time if the position is encountered again.*

Normal computer games are graphical, there are positions that are chosen by the computer or by the player. Consider the game of chess where one player has chosen one position again. Here we can use the positions of the pieces (64 pieces) at that time as the key to store it in the hash table. If our program wants to analyze that if a player has encountered the similar situation before, can pass in the positions of the pieces to the function *find*. Inside the function, when the positions are hashed again then the previously present index is returned, which shows that the similar situation has been encountered before.

See another example below:

- *Hash functions can be used to quickly check for inequality — if two elements hash to different values they must be different.*

Sometimes in your applications, you don't want to know which value is smaller or bigger but you are only interested in knowing if they are equal or not. For this, we can use hashing. If the two data items don't collide, then their hash values will be different. Based on this two values are said to be unequal.

Above was the situation when hashing can be useful. You may like to know in what circumstances hashing is not a good solution to apply.

When Hashing is Suitable?

- *Hash tables are very good if there is a need for many searches in a reasonably stable table.*

We have just seen the excellent example of reasonably stable hash table when we discussed hash table for English dictionary. We had constructed a hash table of all the words inside dictionary and were looking for different words in it. So majorly, there were frequent look up operations and insertions were in very minor frequency.

- *Hash tables are not so good if there are many insertions and deletions, or if table traversals are needed — in this case, AVL trees are better.*

In some applications, it is required to frequently read and write data. In these kinds of applications hash table might not be a good solution, AVL tree might be a good option. But bear in mind that there are no hard and fast statistics to go for hash table and then to AVL tree. You have to be a good software engineer to choose relevant data structure.

- *Also, hashing is very slow for any operations which require the entries to be sorted*
 - o *e.g. Find the minimum key*

At times, you do other operations of *insert*, *delete* and *find* but additionally, you require the data in sorted order or the minimum or maximum value. We have discussed it many times that we insert data in the array of hash table without any sort order and it is scattered through the array in such a fashion that there are holes in the array. In these circumstances, the hash table is not useful. You might be remembering from the animation we saw earlier on in this lecture that there was no real sequence of filling of array. Some clusters were formed because of collision but there was no order as such. So hashing is not really useful in these circumstances.

We are finishing with our discussion on hashing. The important thing is how we thought about one data structure and internally we implemented in six different ways. You must be remembering that as long as the interface of the data structures remains the same, different internal implementations does not really matter from the client perspective. Occasionally, somebody might be interested in knowing the internal implementation of your data structure because that might be important for him in order to use your data structure.

Let's move on to the next topic of Sorting. It is very vast topic and cannot be covered in this course thoroughly.

Sorting

Sorting means to put the data in a certain order or sequence. We have discussed sorting before in different scattered through topics in this course but it has not been discussed so far as a separate topic. You must be remembering that when we traverse the binary search tree in in-order way, the obtained data happens to be sorted. Similarly, we saw other data structures, where we used to keep data in sorted order. In case of min-heap if we keep on removing elements one by one, we get data in sorted order.

Sorting is so useful that in 80-90% of computer applications, sorting is there in one form or the other. Normally, sorting and searching go together. Lot of research has been done on sorting; you can get lot of stuff on it from different sources. Very efficient algorithms have already been developed for it. Moreover, a vast Mathematical analysis has been performed of these algorithms. If you want to expose yourself, how these analyses are performed and what Mathematical tools and procedures are employed for performing analysis then sorting is very useful topic for you.

Sorting Integers

– *How to sort integers in this array?*

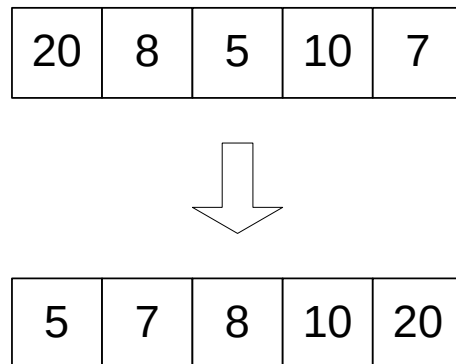


Fig 43.1

We want to sort the numbers given in the above array. Apparently, this operation may seem very simple. But think about it, if you are given a very large volume of data (may be million of numbers) then you may realize that there has to be an efficient mechanism to perform this operation. Firstly, let's put the problem in words:

We have a very large array of numbers. We want to sort the numbers inside the array in ascending order such that the minimum number of the array will be the first element of it and the largest element will be the last element at the end of the array.

Let's go to the algorithms of sorting. Point to be noted here that we are going to study algorithms of sorting; we are not talking about data structures. Until now, you might

have realized that algorithms go along data structures. We use a data structure to contain data and we use algorithms to perform certain operations or actions on that data.

Elementary Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort

These algorithms have been put as elementary because these are very simple. They will act as our baseline and we will compare them with other algorithms in order to find a better algorithm.

Selection Sort

- Main idea:
 - o find the smallest element
 - o put it in the first position
 - o find the next smallest element
 - o put it in the second position

...

- And so on, until you get to the end of the list

This technique is so simple that you might have found it yourself already. You search the whole array and find the smallest number. The smallest number is put on the first position of the array while the previous element in this position is moved somewhere else. Find the second smallest number and then put that number in the second position in the array, again the previous number in that position is shifted somewhere else. We keep on performing this activity again and again and eventually we get the array sorted. This technique is called selection sort because we select elements for their sorted positions.

In the next lecture, we will see how we can optimize this sorting operation. You read about sorting in your textbooks and from the Internet.

Data Structures

Lecture No. 44

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 7

7.1, 7.2

Summary

- Selection Sort
 - o Selection Sort analysis
- Insertion Sort
 - o Insertion Sort Analysis
- Bubble Sort
 - o Bubble Sort analysis
- Summary
- $N \log_2(N)$ Algorithms

This is the sequel of the previous lecture in which we talked about the sort algorithms. There are three elementary sorting methods being discussed in this hand out. These are- selection sort, insertion sort and bubble sort. To begin with, we will talk about the selection sort algorithm.

Selection Sort

Suppose we have an array with different numbers. For sorting it in an ascending order, selection sorting will be applied on it in the following manner. We find the smallest number in the array and bring it to the position 1. We do the same process with the remaining part of the array and bring the smallest number among the remaining array to the next position. This process continues till the time all the positions of the array are filled with the elements. Thus the main idea of selection sort is that

- find the smallest element
- put it in the first position
- find the next smallest element in the remaining elements
- put it in the second position
- ...
- And so on, until we get to the end of the array

Let's understand this algorithm by considering an example with the help of figures. Consider an array that has four numbers i.e. 19, 5, 7 and 12 respectively. Now we want to sort this array in ascending order. To sort the array, selection algorithm will be applied on it.

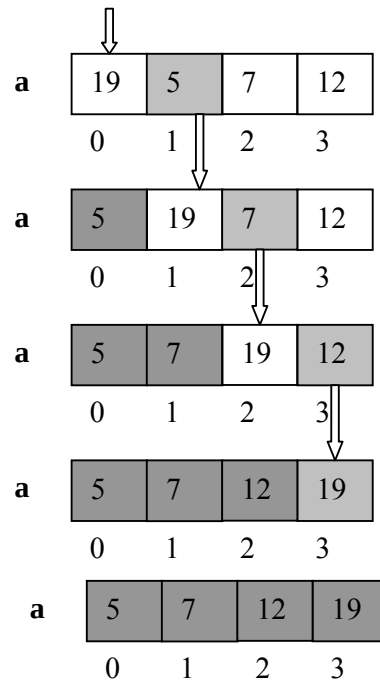


Figure 44.1: Selection Sort

The above pictorial representation explains the selection sort. It describes that at the start, we begin the search for the smallest number from the first position of the array i.e. from the index zero. We find that 5 is the smallest number and bring it to the first position i.e. index 0. Later, number 19 is put at the position that was occupied by number 5. Thus in a sense, we swap these numbers. Now 5, the smallest number in the array, has come at its final position i.e. index 0.

As index 0 has the proper number, so we start our search from position 2 i.e. index 1. Now we look at the remaining elements 19, 7, 12 and find the smallest number among them. Here 7 is the smallest so we change its position with 19 to bring 7 at its position. Thus 5 and 7 have come at their final positions. Now there are two elements are left behind i.e. 12 and 19. We search the smallest among these and find that 12 is the smallest. So we swap it with 19 to bring it at index 2 i.e. its final position. Now there is last element remaining and obviously it is at its position as there is no element to compare with it. The array is now in the sorted form with ascending numbers.

The point to remember in the selection search is that at the beginning, we start search for the smallest number from index 0 (i.e. first position). After it we start search from the index 1 (i.e. position 2). After each search one number gets its final position so we start the next search from the next position to it. Thus we do the multiple passes of the array to sort it. First, we pass through the n elements of the array and search the $n-1$ elements and then $n-2$. Thus at last, we come to the single and last element of the array.

Now let's see the code of this algorithm in C++.

```
void selectionSort(int *arr, int N)
```

```

    {
        int posmin, count, tmp ;
        for (count=0;count<N;count++)
        {
            posmin = findIndexMin(arr, count,
N) ;
            tmp=arr[posmin] ;
            arr[posmin]=arr[count] ;
            arr[count]=tmp ;
        }
    }

int findIndexMin (int *arr, int start,      int N)
{
    int posmin=start ;
    int index ;
    for(index=start; index < N; index++)
        if (arr[index]<arr[posmin])
            posmin=index ;
    return posmin ;
}

```

In the above code, we write the function *selectionSort*. This function takes an array of integers as **arr* and the size of array as *N*. There are local variables declared in function as *posmin*, *count* and *tmp*. Then there is a 'for loop' that starts from zero and goes to *N-1*. This is due to the fact that the index of array of *N* elements is from zero to *N-1*. The condition *count < N* indicates that loop will execute as long as *count* is less than *N* and will exit when *count* gets *N*. Now in the loop, we calculate the *posmin* with a function i.e. *findIndexMin*. This *findIndexMin* method is written below in the code. This routine or method works in the way that we pass to it the whole array i.e. *arr*, the value of *count* (what it is in that execution of loop) and size of the array i.e. *N*. This routine starts the search from the index equal to value of *count* and goes to *N*th position, returning the position of the minimum (smallest) element. Now in the first routine, we get this position value in the variable *posmin*. Thus the code line:

```
posmin = findIndexMin(arr, count, N) ;
```

gets the position of the smallest number returned by the method *findIndexMin* in the variable *posmin*. After finding this position, we do the swapping of this *posmin* with the *count* position. Thus we put the value of position *posmin* in the *count* position.

The *findIndexMin* routine is such that it takes *arr*, *start* and *N* as arguments. It starts searching from the *start* position in the *for* loop, finds the position of the minimum value and returns that position.

This sorting algorithm is also known as the in-place sorting algorithm as there is no need of additional storage to carry out this sorting. The pictorial representation of the swap action of this algorithm will be as under:

We want to sort the following array that has five elements. We start the search from

the first element and find that the smallest element is 5. As we have started our search from index 0 (that is the *count* in our above code) so we swap 5 with the value at index 0 i.e. 20. After this swap, 5 comes at the position of 20 while 20 goes to the position of 5. Afterwards, we start the search from index 1. We find that 7 is the smallest number among the remaining elements. It swaps its position with 8. After this, in the remaining three elements, 8 is the smallest. This number 8 swaps its position with 20. At the end, 10 is the smallest number among 10 and 20. So there is no need of swapping as 10 has already got its position.

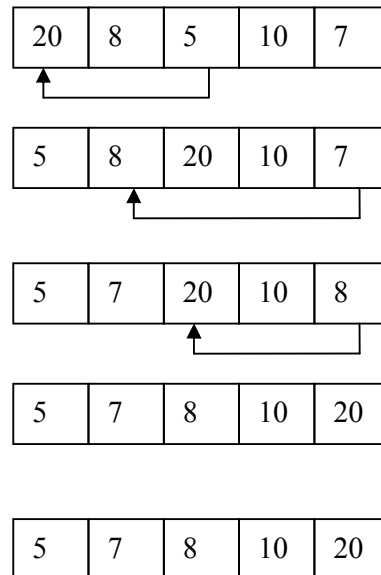


Figure 44.2: Swap Action (selection sorting)

Selection Sort Analysis

We have seen in the code for the algorithm that there are two loops. The loop in the *selectionSort* method passes the array to search the smallest element and the second loop that is in the *findIndexMin* method finds the position (index) of the smallest value. To find the first smallest element, we have to go through the N elements of the array. For the purpose of finding the second smallest element, we have to search the $N-1$ elements. During this search process, we have to find two elements for the second last smallest element. And obviously in the end, there is one element that is at its proper position, necessitating no search. We have to do all these searches. These are $N, N-1, N-2, \dots, 2, 1$ for the first, second, third, ..., second last and last element respectively. Now if we want to find the total searches, the addition of all these searches together will help us get a sum total as given in the following equation.

$$\begin{aligned}
 \text{Total searches} &= 1 + 2 + 3 + \dots + (N-2) + (N-1) + N \\
 &= \frac{N(N+1)}{2} \\
 &= \frac{(N^2 + N)}{2}
 \end{aligned}$$

Suppose if N is 10, then according to this formula, the total searches will be $(100 + 10) / 2$ i.e. 55. If N is 100, the total searches will be $(10000 + 100) / 2$ i.e. 5050. Similarly if N is 1 million, N^2 is going to be very large as compared to N . Thus there we can ignore N and can say that the time (total searches) is proportional to N^2 .

This means that larger the N , greater will be the performance of selection with respect to N^2 .

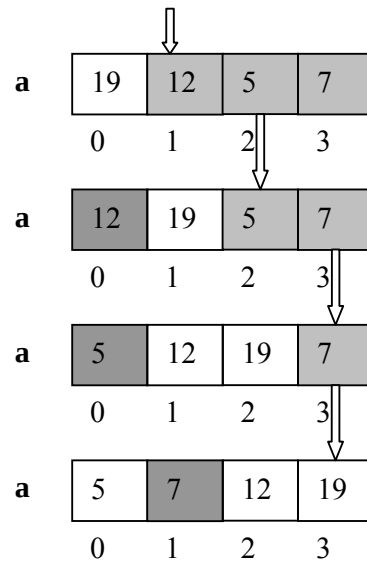
Insertion Sort

The main idea of insertion sort is

- Start by considering the first two elements of the array data. If found out of order, swap them
- Consider the third element; insert it into the proper position among the first three elements.
- Consider the fourth element; insert it into the proper position among the first four elements.
-

This algorithm is not something uncommon to the persons who know card playing. In the game of cards, a player gets 13 cards. He keeps them in the sorted order in his hand for his ease. A player looks at the first two cards, sorts them and keeps the smaller card first and then the second. Suppose that two cards were 9 and 8, the player swap them and keep 8 before 9. Now he takes the third card. Suppose, it is 10, then it is in its position. If this card is of number 2, the player will pick it up and put it on the start of the cards. Then he looks at the fourth card and inserts it in the first three cards (that he has sorted) at a proper place. He repeats the same process with all the cards and finally gets the cards in a sorted order. Thus in this algorithm, we keep the left part of the array sorted and take element from the right and insert it in the left part at its proper place. Due to this process of insertion, it is called insertion sorting.

Let's consider the array that we have used in the selection sort and sort it now with the insertion sorting. The following figure shows the insertion sort of the array pictorially.

**Figure 44.3:** Insertion Sort

The array consists of the elements 19, 12, 5 and 7. We take the first two numbers i.e. 19 and 12. As we see 12 is less than 19, so we swap their positions. Thus 12 comes at index 0 and 19 goes to index 1. Now we pick the third number i.e. 5. We have to find the position of this number by comparing it with the two already sorted numbers. These numbers are 12 and 19. We see that 5 is smaller than these two. So it should come before these two numbers. Thus the proper position of 5 is index 0. To insert it at index 0, we shift the numbers 12 and 19 before inserting 5 at index 0. Thus 5 has come at its position. Now we pick the number 7 and find its position between 5 and 12. To insert 7 after 5 and before 12, we have to shift the numbers 12 and 19 to the right. After this shifting, we put number 7 at its position. Now the whole array has been sorted so the process stops here.

Following is the code of the insertion sort in C++.

```
void insertionSort(int *arr, int N)
{
    int pos, count, val;
    for(count=1; count < N; count++)
    {
        val = arr[count];
        for(pos=count-1; pos >= 0;
pos--)
        {
            if (arr[pos] > val)
                arr[pos+1]=arr[pos];
            else break;
        }
        arr[pos+1] = val;
    }
}
```

In this sorting function, we start the process from index 0 and 1 and swap them. Afterwards, we go to the third position and put it into its proper position. While inserting a number in the sorted portion of the array, we have to shift the elements. This shifting is an additional overhead of this sorting algorithm. Due to this shifting, the sorting requires more time. This algorithm is also an in place algorithm as we don't need any additional storage. At the most, we need some variables of local nature, normally used in programs.

From the above code, we see that the name of this sorting routine is *insertionSort*. It takes an array and its size as arguments. There are local variables *pos*, *count* and *val* declared in this function. After this there is a *for* loop that starts from the *count* having value equal to 1. Now we put the value of *count* index (i.e. *arr[count]*) in variable *val*. This value has to find its place in the left sorted portion of the array. To find that position we have to execute one more *for* loop. This loop starts from *count-1* and goes to down to zero. In the body of the loop we compare the value of *val* with the value at current position in the array i.e. *arr[pos]*. If *val* is less than the *arr[pos]* i.e. value at current index. It means that the *val* has to go to the left position of *arr[pos]*. So we shift the *arr[pos]* to right to create place for the new value i.e. *val*. When the loop exits, we put this value *val* at *arr[pos + 1]*. Thus we have inserted the number in the array at its proper position.

Following is the step by step explanation for the insertion sort of the above example with same previous array.

First of all we take the first two elements 19 and 12. As 12 is less than 19, we do right shift on 19. And put 12 at its position i.e. index 0. Afterwards, we go to index 2. There is 5 at this position. As we see that 5 is less than the other elements on the left side of array, it has to come at the first position. To bring 5 to first position, the number 12 and 19 has to be shifted to right. After this shifting, the position of index 0 becomes empty and we put 5 there. Finally, there is number 7. The position of 7 will be between 5 and 12. For this, we have to shift 12 and 19 towards right so that the place for 7 could be created. We put 7 at that place. Thus the array is sorted now.

Insertion Sort Analysis

Let's analyze that when the value of *N* increases. How much time for insertion sort increases? In the code of insertion sort, we have seen that there are outer and inner loops. Due to these two loops, we can understand that it is also like N^2 algorithm. In the sort process, there may be a situation that every iteration inserts an element at the start of the array by shifting all sorted elements along. Now if we have to bring the second element to its position, there will be need of shifting the first element. This means that we have to shift one element. Similarly, for placing the third element at the start position (we are discussing the worst case scenario in which at every iteration the element has to go to the first position), we have to shift two elements. Thus we sum up all the shiftings, the total becomes $2 + 3 + 4 + \dots + N - 1 + N$.

The summation can be written as follows.

$$\begin{aligned} \text{Total} &= (2 + N) (N - 1) / 2 \\ &= O(N^2) \end{aligned}$$

From this expression, we see that when the value of *N* increases, the value of N^2 will dominate. It will increase significantly with respect to *N*. Thus we see that insertion

sort is also an N^2 algorithm like selection sort.

Bubble Sort

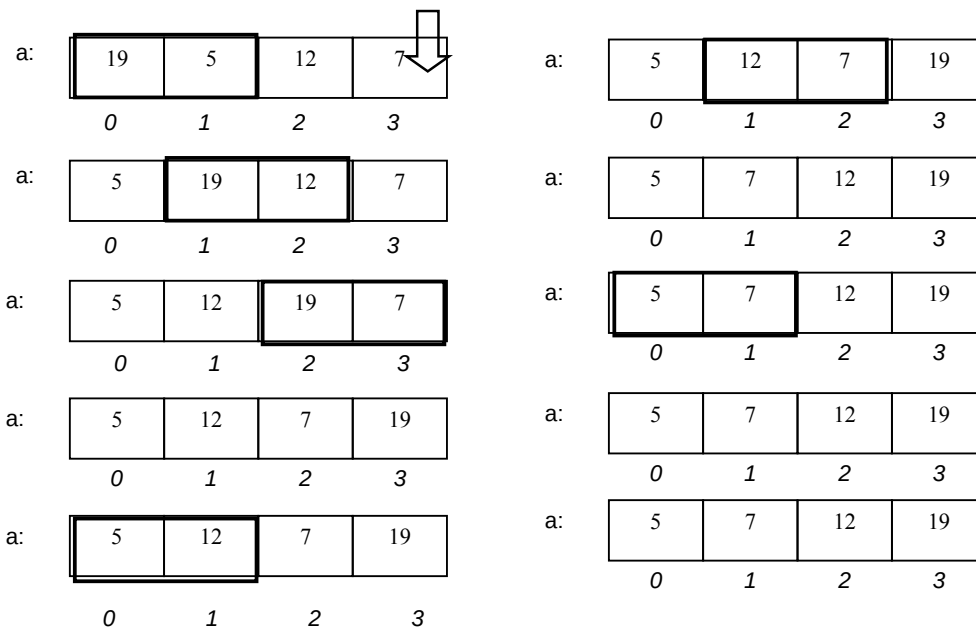
The third sorting algorithm is bubble sort. The basic idea of this algorithm is that we bring the smaller elements upward in the array step by step and as a result, the larger elements go downward. If we think about array as a vertical one, we do bubble sort. The smaller elements come upward and the larger elements go downward in the array. Thus it seems a bubbling phenomenon. Due to this bubbling nature, this is called the bubble sort. Thus the basic idea is that the lighter bubbles (smaller numbers) rise to the top. This is for the sorting in ascending order. We can do this in the reverse order for the descending order.

The steps in the bubble sort can be described as below

- Exchange neighboring items until the largest item reaches the end of the array
- Repeat the above step for the rest of the array

In this sort algorithm, we do not search the array for the smallest number like in the other two algorithms. Also we do not insert the element by shifting the other elements. In this algorithm, we do pair-wise swapping. We will take first the elements and swap the smaller with the larger number. Then we do the swap between the next pair. By repeating this process, the larger number will be going to the end of the array and smaller elements come to the start of the array.

Let's try to understand this phenomenon with the help of figures how bubble sort works. Consider the same previous array that has elements 19, 12, 5 and 7.



First of all, we compare the first pair i.e. 19 and 5. As 5 is less than 19, we swap these

elements. Now 5 is at its place and we take the next pair. This pair is 19, 12 and not 12, 7. In this pair 12 is less than 19, we swap 12 and 19. After this, the next pair is 19, 7. Here 7 is less than 19 so we swap it. Now 7 is at its place as compared to 19 but it is not at its final position. The element 19 is at its final position. Now we repeat the pair wise swapping on the array from index 0 to 2 as the value at index 3 is at its position. So we compare 5 and 12. As 5 is less than 12 so it is at its place (that is before 12) and we need not to swap them. Now we take the next pair that is 12 and 7. In this pair, 7 is less than 12 so we swap these elements. Now 7 is at its position with respect to the pair 12 and 7. Thus we have sorted the array up to index 2 as 12 is now at its final position. The element 19 is already at its final position. Note that here in the bubble sort, we are not using additional storage (array). Rather, we are replacing the elements in the same array. Thus bubble sort is also an in place algorithm. Now as index 2 and 3 have their final values, we do the swap process up to the index 1. Here, the first pair is 5 and 7 and in this pair, we need no swapping as 5 is less than 7 and is at its position (i.e. before 7). Thus 7 is also at its final position and the array is sorted.

Following is the code of bubble sort algorithm in C++.

```
void bubbleSort(int *arr, int N)
{
    int i, temp, bound = N-1;
    int swapped = 1;
    while (swapped > 0 )
    {
        swapped = 0;
        for(i=0; i < bound; i++)
            if ( arr[i] > arr[i+1] )
            {
                temp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = temp;
                swapped = i;
            }
        bound = swapped;
    }
}
```

In line with the previous two sort methods, the *bubbleSort* method also takes an array and size of the array as arguments. There is *i*, *temp*, *bound* and *swapped* variables declared in the function. We initialize the variable *bound* with N-1. This N-1 is our upper limit for the swapping process. The outer loop that is the *while* loop executes as long as swapping is being done. In the loop, we initialize the swapped variable with zero. When it is not changed in the *for* loop, it means that the array is now in sorted form and we exit the loop. The inner *for* loop executes from zero to *bound-1*. In this loop, the if statement compares the value at index *i* and *i+1*. If *i* (element on left side in the array) is greater than the element at *i+1* (element on right side in the array) then we swap these elements. We assign the value of *i* to the swapped variable that being greater than zero indicates that swapping has been done. Then after the *for* loop, we put the value of swapped variable in the bound to know that up to this index, swapping has taken place. After the *for* loop, if the value of *swapped* is not zero, the

while loop will continue execution. Thus the *while* loop will continue till the time, the swapping is taking place.

Now let's see the time complexity of bubble sort algorithm.

Bubble Sort Analysis

In this algorithm, we see that there is an outer loop and an inner loop in the code. The outer loop executes N times, as it has to pass through the whole array. Then the inner loop executes for N times at first, then for $N-1$ and for $N-2$ times. Thus its range decreases with each of the iteration of the outer loop. In the first iteration, we do the swapping up to N elements. And as a result the largest elements come at the last position. The next iteration passes through the $N-1$ elements. Thus the part of the array in which swapping is being done decreases after iteration. At the end, there remains only one element where no swapping is required. Now if we sum up these iterations i.e. $1 + 2 + 3 + \dots + N-1 + N$. Then this summation becomes $N(1 + N) / 2 = O(N^2)$. Thus in this equation, the term N^2 dominates as the value of N increases. It becomes ignorable as compared to N^2 . Thus when the value of N increases, the time complexity of this algorithm increases proportional to N^2 .

Summary

Now considering the above three algorithms, we see that these algorithms are easy to understand. Coding for these algorithms is also easy. These three algorithms are in place algorithms. There is no need of extra storage for sorting an array by these algorithms. With respect to the time complexity, these algorithms are proportional to N^2 . Here N is the number of elements. So we can see that as the value of N increases, the performance time of these algorithms increases considerably as it is proportional to N^2 . Thus these algorithms are expensive with respect to time performance. There are algorithms that have the time complexity proportional to $N \log_2(N)$. The following table shows the respective values of N^2 and $N \log_2(N)$ for some values of N .

N	N^2	$N \log_2(N)$
10	100	33.21
100	10000	664.38
1000	1000000	9965.78
10000	100000000	132877.12
100000	10000000000	1660964.04
1000000	1E+12	19931568.57

From this table we can see that for a particular value of N , the value of N^2 is very large as compared to the value of $N \log_2(N)$. Thus we see that the algorithms whose time complexity is proportional to N^2 are much time consuming as compared to the algorithms the time complexity of which is proportional to $N \log_2(N)$. Thus we see that the $N \log_2(N)$ algorithms are better than the N^2 algorithms.

$N \log_2(N)$ Algorithms

Now let's see the algorithms that are $N \log_2(N)$ algorithms. These include the following algorithms.

- Merge Sort
- Quick Sort
- Heap Sort

These three algorithms fall under ‘divide and conquer category’. The divide and conquer strategy is well known in wars. The philosophy of this strategy is ,’ divide your enemy into parts and then conquer these parts’. To conquer these parts is easy, as these parts cannot resist or react like a big united enemy. The same philosophy is applied in the above algorithms. To understand the divide and conquer strategy in sorting algorithm, let’s consider an example. Suppose we have an unsorted array of numbers is given below.

10	12	8	4	2	11	7	5

Now we split this array into two parts shown in the following figure.

10	12	8	4	2	11	7	5

Now we have two parts of the array. We sort these parts separately. Suppose we sort these parts with an elementary sort algorithm. These parts may be sorted in the following manner.

4	8	10	12	2	5	7	11

After this we merge these two parts and get the sorted array as shown below.

2	4	5	7	8	10	11	12

Data Structures

Lecture No. 45

Reading Material

Data Structures and Algorithm Analysis in C++

Chapter. 7
7.6,

7.7

Summary

- Divide and Conquer
- Mergesort
- mergeArrays
- Mergesort and Linked Lists
- Mergesort Analysis
- Quicksort
- Course Overview

Divide and Conquer

In the previous lecture, we had started discussing three new sorting algorithms; *merge sort*, *quick sort* and *heap sort*. All of these three algorithms take time proportional to $n \log_2 n$. Our elementary three sorting algorithms were taking n^2 time; therefore, these new algorithms with $n \log_2 n$ time are faster. In search operation, we were trying to reduce the time from n to $\log_2 n$.

Let's discuss these sorting algorithms; merge sort, quick sort and heap sort in detail. We had started our discussion from *divide and conquer* rule where we also saw an example. Instead of sorting a whole array, we will divide it in two parts, each part is sorted separately and then they are merged into a single array.

Split the list into two parts

10	12	8	4	2	11	7	5

Sort the two parts

4	8	10	12	2	5	7	11

Merge the two parts together

4	8	10	12	2	5	7	11

Fig 45.1

Let's see a few analyses to confirm the usefulness of the divide and conquer technique.

- To sort the halves approximate time is $(n/2)^2 + (n/2)^2$
- To merge the two halves approximate time is n
- So, for $n=100$, divide and conquer takes approximately:

$$\begin{aligned}
 &= (100/2)^2 + (100/2)^2 + 100 \\
 &= 2500 + 2500 + 100 \\
 &= 5100
 \end{aligned}$$

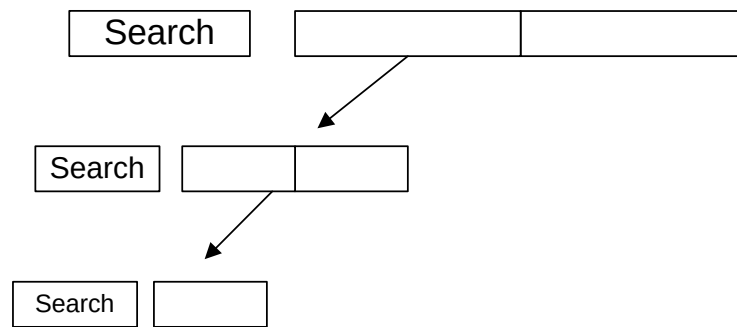
We know that elementary three sorting algorithms were taking approximately n^2 time. Suppose we are using insertion sort of those elementary algorithms. We divide the list into two halves then the time will be approximately $(n/2)^2 + (n/2)^2$. The time required for merging operation is approximately n . This operation contains a simple loop that goes to n .

Suppose that n is 100. Considering if we apply insertion sort algorithm on it then the time taken will be approximately $(100)^2 = 10000$. Now, if we apply divide and conquer technique on it. Then for first half approximate time will be $(100/2)^2$. Similarly for second half it will be $(100/2)^2$. The merging approximate time will be 100. So the whole operation of sorting using this divide and conquer technique in insertion sort will take around $(100/2)^2 + (100/2)^2 + 100 = 5100$. Clearly the time spent (5100) after applying divide and conquer mechanism is significantly lesser than the previous time (10000). It is reduced approximately to half of the previous time. This example shows the usefulness of divide and conquer technique.

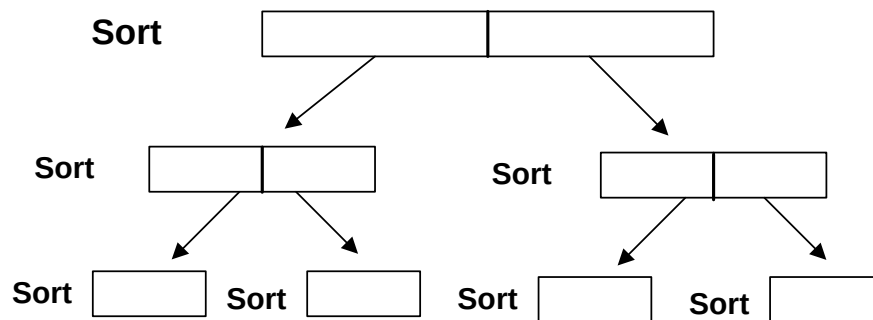
By looking at the benefit after dividing the list into two halves, some further questions arise:

- Why not divide the halves in half?
 - The quarters in half?
 - And so on . . .
 - When should we stop?
- At $n = 1$

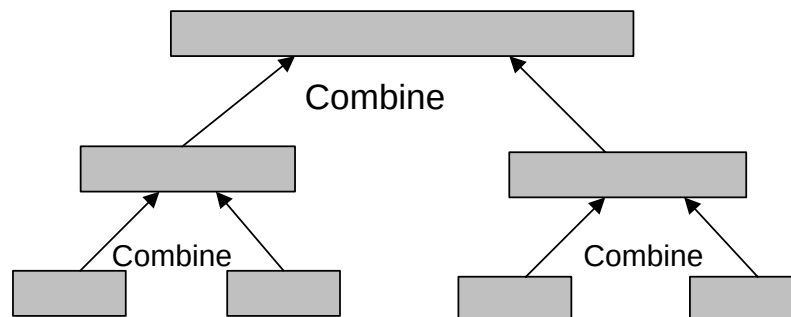
So we stop subdividing the list when we reach to the single element level. This divide and conquer strategy is not a thing, we have already prepared binary search tree on the same lines. One side of the tree contains the greater elements than the root and other part contains the smaller elements. Especially, when performing binary search in an array, we had started our search from mid of it. Subdivided the array and kept on comparing and dividing the array until we got success or failure. The subdivision process may prolong to individual element of the array.

Recall Binary Search**Fig 45.2**

Hence, we used to perform binary search on the same lines of divide and conquer strategy. Remember, we applied binary search on sorted array. From this one can realize that sorting facilitates in searching operations.

**Fig 45.3**

In figure 45.3, in order to sort the list, we have divided the list in the two parts and each part is subdivided into further subparts. At end each part is consisting of either single element or maximum two elements. If we have two numbers to sort, we can compare them (or sort them) with single if statement. After sorting individual subparts, we start merging them in the upward direction as shown in the figure Fig 45.4.

**Fig 45.4**

In Fig 45.4, we have four sorted smaller parts. We combine them to become two sorted parts and two sorted parts are further combined or merged to become one

sorted list.

Mergesort

- **Mergesort** is a divide and conquer algorithm that does exactly that.
- It splits the list in half
- Mergesorts the two halves
- Then merges the two sorted halves together
- Mergesort can be implemented recursively

Let's see the mergesort algorithm, how does that work.

- The mergesort algorithm involves three steps:
 - o If the number of items to sort is 0 or 1, return
 - o Recursively sort the first and second halves separately
 - o Merge the two sorted halves into a sorted group
- If the data is consisting of 0 or 1 element then there is nothing required to be done further. If the number of elements is greater than 1 then apply divide and conquer strategy in order to sort them. Divide the list into two halves and sort them separately using recursion. After the halves (subparts) have been sorted, merge them together to get a list of sorted elements.

We will discuss recursive sorting in a moment, before that let's see the merging operation using pictures. We have two sorted array and another empty array whose size is equal to the sum of sizes of two sorted arrays.

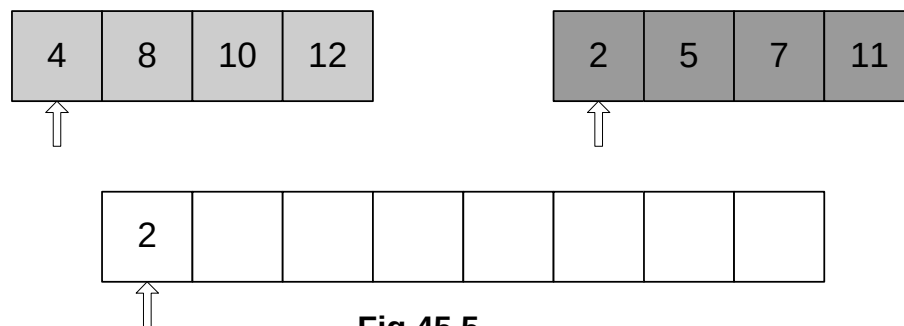
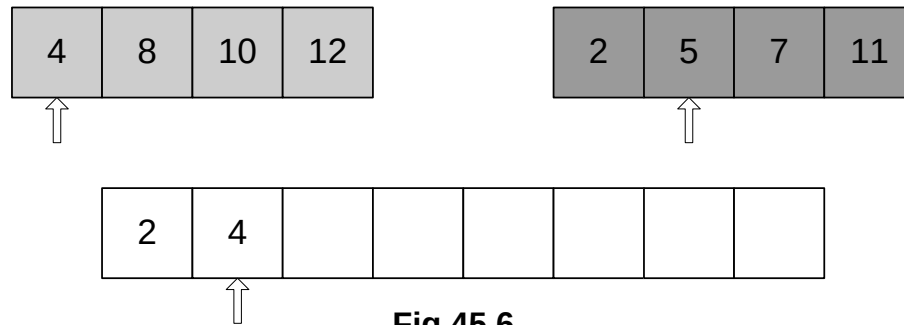


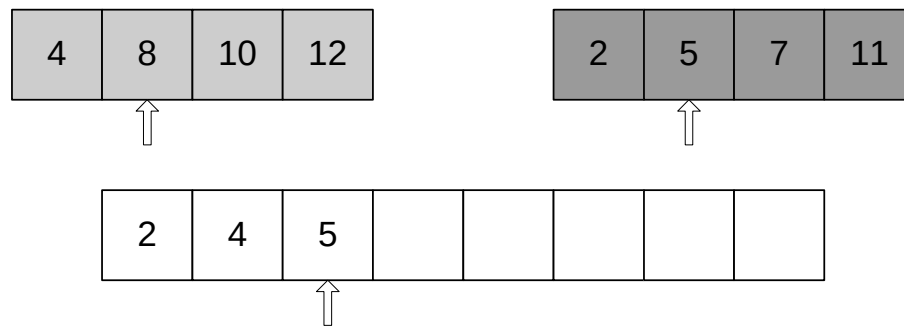
Fig 45.5

You can see from Fig 45.5, array 1 on the top left is containing 4 elements, top right is also containing 4 elements and both of them are sorted internally. Third array is containing $4+4 = 8$ elements.

Initially, very first elements (present at the starting index 0 of array) of both the arrays are compared and the smaller of them is placed in the initial position of the third array. You can see from Fig 45.5 that elements 4 and 2 are compared pointed to by the indexes (actually arrays current indexes). The smaller of them is 2, therefore, it is placed in the initial position in the third array. A pointer (the current index of array) is also shown for third array that will move forward as the array is filled in. The smaller number was from right array, therefore, its pointer is moved forward one position as shown in Fig 45.6.

**Fig 45.6**

This time the numbers at current positions in both the arrays are compared. As 4 is smaller of the two numbers, therefore, it is put in the third array and third array's pointer is moved one position forward. Also because this time, the number has been chosen from left array, therefore, its pointer is also moved forward. The updated figure is shown in Fig 45.7.

**Fig 45.7**

Next, numbers 8 and 5 are compared. As 5 is smaller of the two, it is put in the third array. The changed positions of pointers and the next comparison are shown in Fig 45.8.

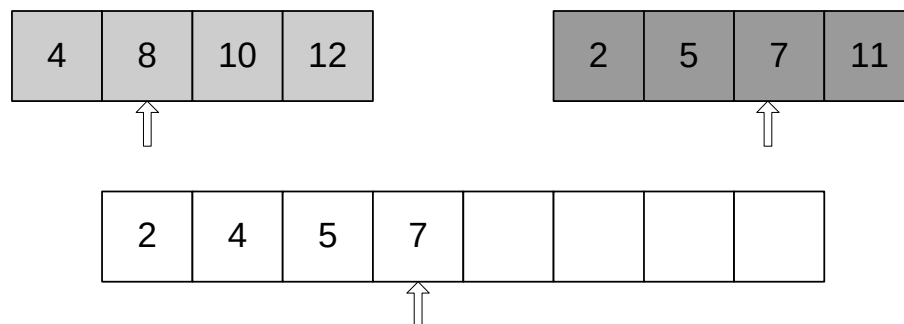
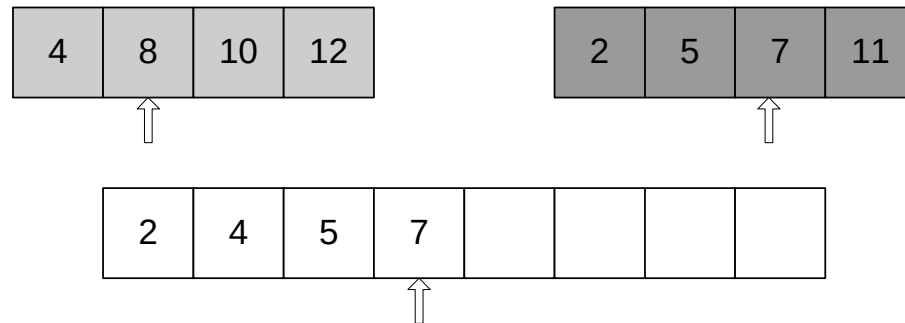
**Fig 45.8**

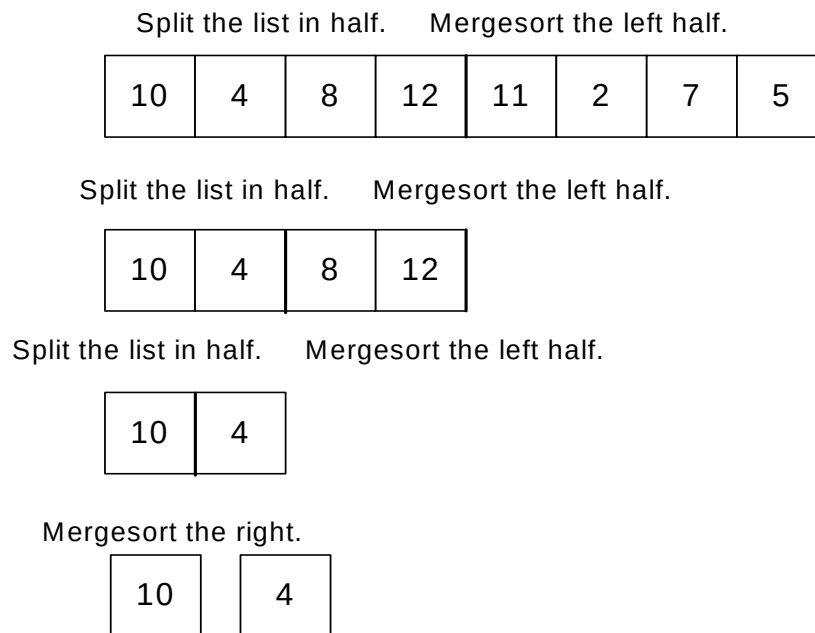
Fig 45.9 has showed the situation after next comparison done in the similar manner.

**Fig 45.9**

By now, you must have understood how the merging operation works. Remember, we do merging when we have two parts sorted already.

Consider this operation in terms of time and complexity. It is performed using a simple loop that manipulates three arrays. An index is used for first array, which starts from the initial position to the size of the array. Similar an index is used for second array, which starts from the initial position and ends at the end of the array. A third index is used for third array that sizes to the sum of the maximum values of both previous indexes. This is simple single loop operation, which is not complex.

Now, let's see sorting using recursion now pictorially.

**Fig 45.10**

At the top of Fig 45.10 is an array, which is not sorted. In order to sort it using recursive mechanism, we divide the array into two parts logically. Only logical partitions of the array are required based on size and index of the arrays. The left half of the array containing 10, 4, 8 and 12 is processed further for sorting. So actually, while calling the mergesort algorithm recursively, only that particular half of the array

is a passed as an argument. This is spitted further into two parts; 10, 4 and 8, 12. 10, 4 half is processed further by calling the *mergesort* recursively and we get both numbers 10 and 4 as separate halves. Now these halves are numbers and they cannot be subdivided therefore recursive call to *mergesort* will stop here. These numbers (new halves) 10 and 4 are sorted individually, so they are merged. When they are merged, they become as 4,10 as shown in Fig 45.11.

10	4	8	12	11	2	7	5

10	4	8	12

Mergesort the right half. Merge the two halves.

4	10	8	12

Merge the two halves.

8	12

Fig 45.11

The recursive call to *mergesort* has done its work for left half that consisted of 10 and 4. Now, we apply the same technique (the recursive calls to *mergesort*) to the right half that was consisting of 8 and 12. 8 and 12 are spitted into separate numbers as indicated in Fig 45.11. Further division of them is not possible, therefore, they are merged as shown in Fig 45.12.

10	4	8	12	11	2	7	5

Merge the two halves.

4	8	10	12

Mergesort the right half. Merge the two halves.

4	10	8	12

Fig 45.12

At this point in time, we have two parts one is containing elements as 4, 10 and other as 8, 12. These two parts are merged. The merged half is shown in Fig 45.12. Note that it is completely sorted as 4,8,10 and 12. This completes the operation on the left half of the array. We do similar process with the right half now, which is consisting of 11, 2, 7 and 5.

Mergesort the right half.

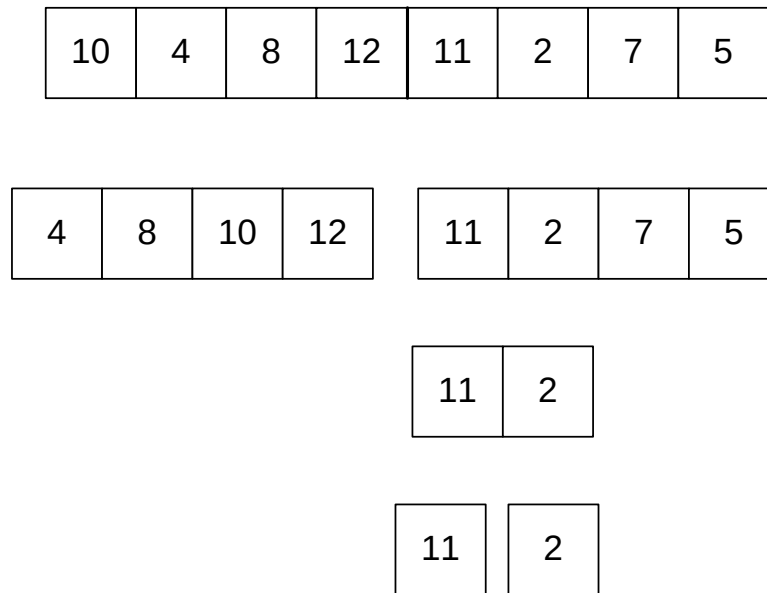


Fig 45.13

Similarly, left half of it 11,2 is processed recursively. It is divided into two halves and we have two parts as two individual numbers. The recursive call for this part stops here and the merging process starts. When the parts 11 and 2 are merged, they become one sorted part as shown in Fig 45.14.

Mergesort the right half.

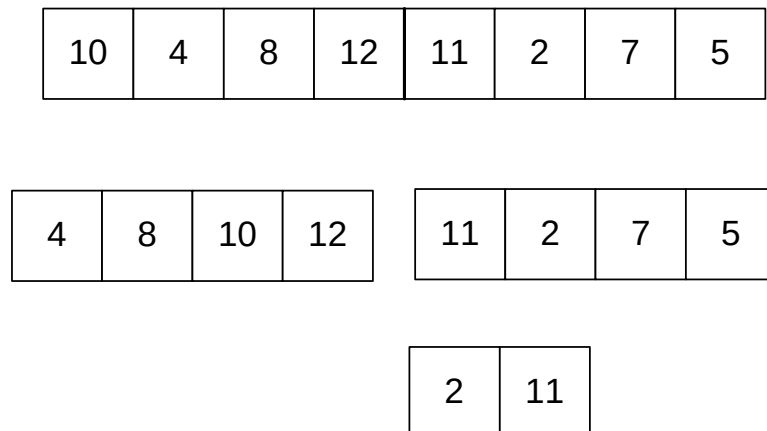


Fig 45.14

Now the same procedure is applied on right half consisting of 7 and 5. By applying recursive mechanism, it is further subdivided into two individual number parts.

Mergesort the right half.

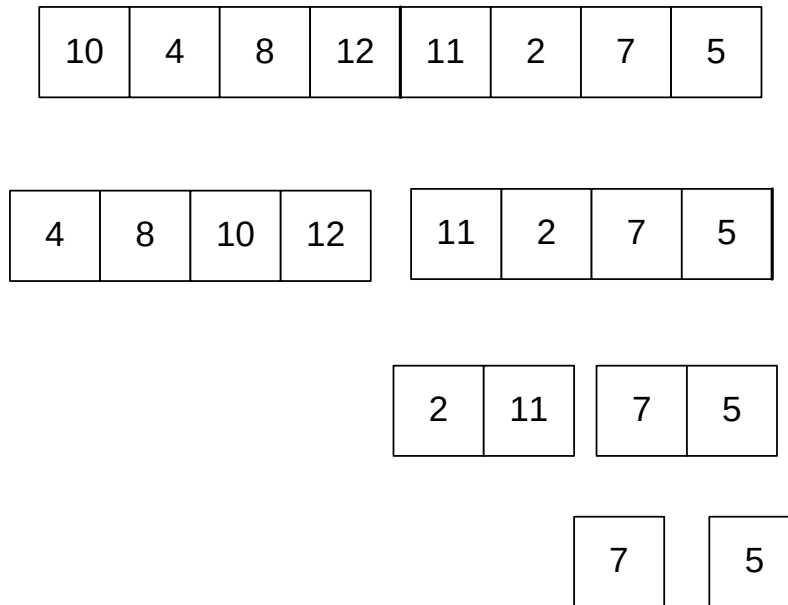


Fig 45.15

The merging operation starts, the resultant is shown in the Fig 45.16. After merging, the new part has become 5,7.

Mergesort the right half.

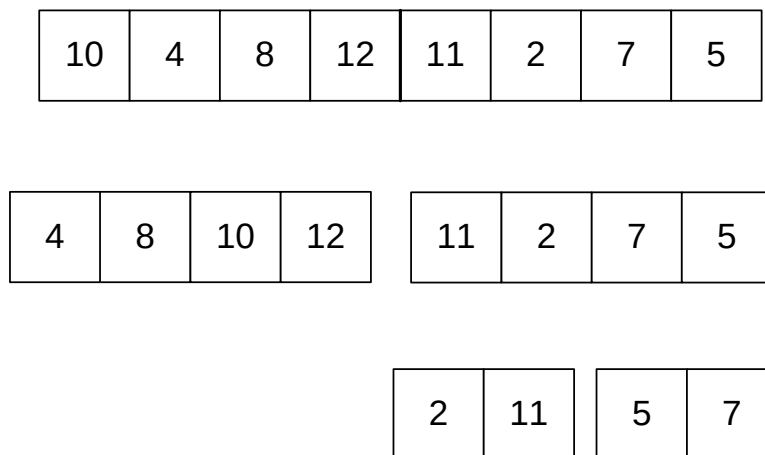


Fig 45.16

When 5,7 merged part is further merged with already merged part 2,11. The new half

becomes as 2,5,7,and 11 as shown in Fig 45.17.

Mergesort the right half.

10	4	8	12	11	2	7	5

4	8	10	12	2	5	7	11

Fig 45.17

Now, we can merge the two biggest halves to get the array in sorted order. After merging, the resulted sorted array is shown in Fig 45.18.

Merge the two halves.

2	4	5	7	8	10	11	12

Fig 45.18

Let's see the C++ code for this sorting algorithm. We are not going to write a class of it but at the moment, we are only writing it as a procedure. We can use it standalone or later on, we can also change it to make it a class member.

```
void mergeSort(float array[], int size)
{
    int * tmpArrayPtr = new int[size];
    if (tmpArrayPtr != NULL)
        mergeSortRec(array, size, tmpArrayPtr);
    else
    {
        cout << "Not enough memory to sort list.\n";
        return;
    }
    delete [] tmpArrayPtr;
}

void mergeSortRec(int array[], int size, int tmp[])
{
    int i;
    int mid = size/2;

    if (size > 1)
    {
```

```

mergeSortRec(array, mid, tmp);
mergeSortRec(array+mid, size-mid, tmp);
mergeArrays(array, mid, array+mid, size-mid, tmp);
for (i = 0; i < size; i++)
    array[i] = tmp[i];
}
}

```

The name of the procedure is *mergeSort*, it is accepting first argument as an array of *floats* and as second argument as an *int*. The second parameter is containing the size of the array. Inside the function, we have created a temporary array *tmpArrayPtr* dynamically of the same size as the size passed in the second argument. This array will be used for merging operation. In the next *if* statement *if (tmpArrayPtr != NULL)*, allocation for dynamic array is being checked, if it is allocated then another function *mergeSortRec* is called. Its first argument is *float*'s array *array*, second argument is *int* *size*, and third argument is *tmpArrayPtr*.

In *mergeSortRec*, we divided the size of the array by 2 to get the mid of the array. If the size of the array is greater than 1 then inside the if-statement, *mergeSortRec* is being called recursively two times. In first recursive call, *mergeSortRec(array, mid, tmp)*; we provided the array from index 0 to *mid*. For second recursive call, *mergeSortRec(array+mid, size-mid, tmp)*; we provided the array starting from one position ahead of *mid* position to the end of the array, however, the size of the passed array is *size-mid*. The first recursive call will keep on executing by further subdividing the left part of the half every time until the size of the array reduces to 1. Similarly, the second recursive call keeps on working on the right half of the array until the size of the array reduces to 1. After both of these recursive calls will be completed then the merging operation is performed by calling *mergeArrays*. At end, inside the for loop, the sorted data is copied from *tmp* array to the actual array *array*.

mergeArrays

Let's see the merging of two array while using their indexes as i and j.

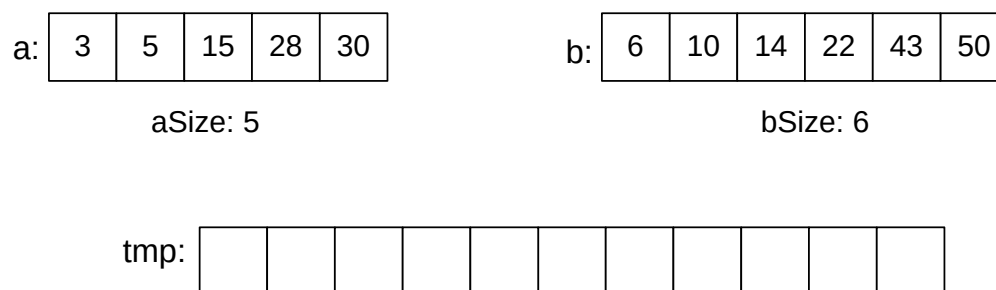
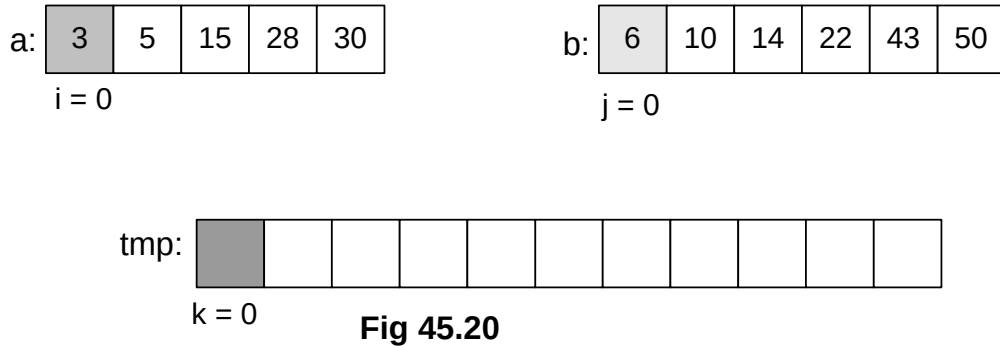
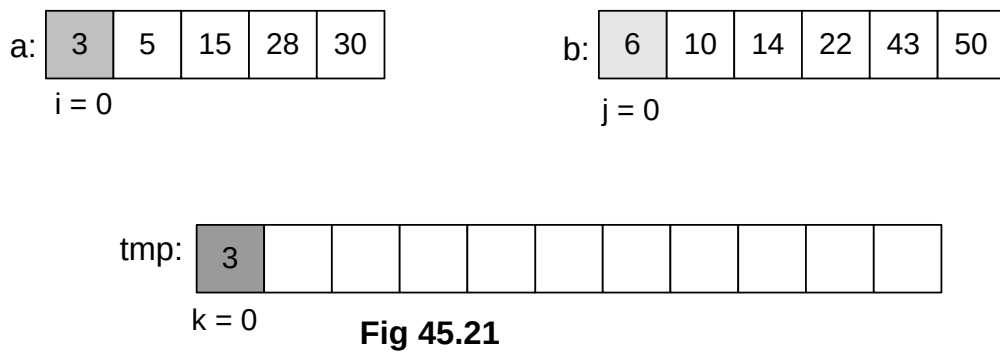


Fig 45.19

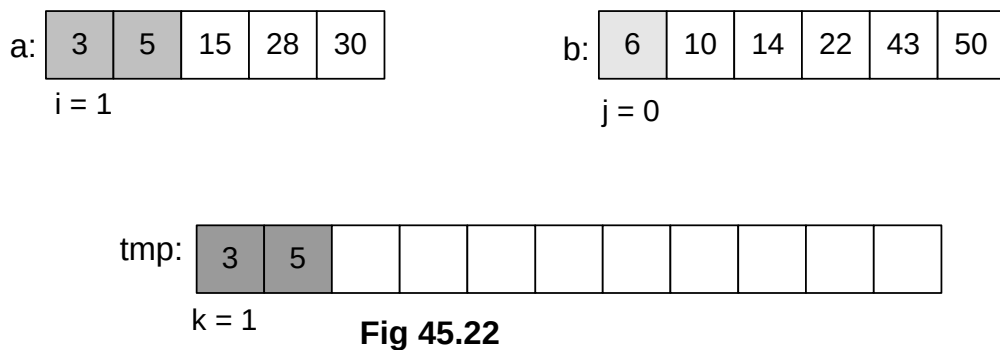
We are doing the same operation of merging of arrays but with little more detail here. On the left is an array a consisting of 2,5,15,28 and 30 and on the right is another array b consisting of 6,10,14,22,43 and 50. The size of array a is 6 while b 's size is 6. We create a temporary array named tmp , whose size is 11. We have used indexes i, j and k for arrays a, b and tmp respectively and each one of them has been initialized with 0.



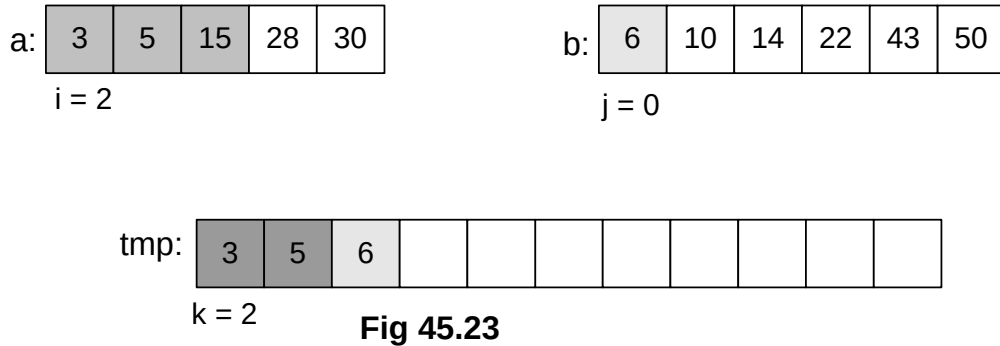
We compare the initial elements 3 and 6 of the arrays *a* and *b*. As 3 is smaller than 6, therefore, it is put in the temporary array *tmp*'s starting location where $k=0$.



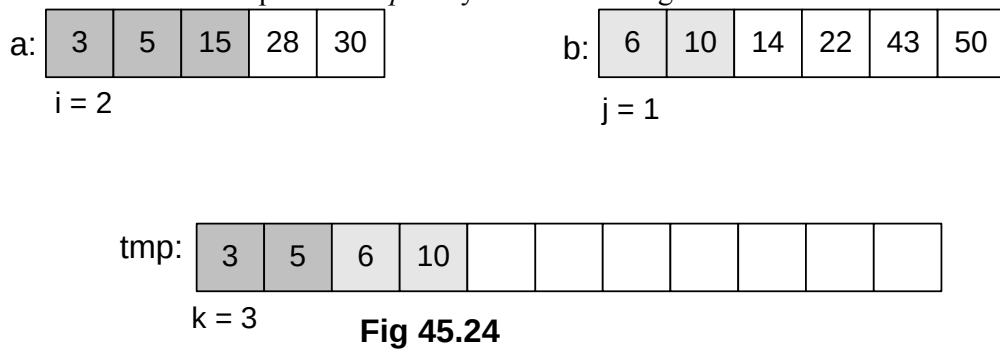
i is incremented and now elements 5 and 6 are compared. 5 being smaller of these take the place in the *tmp* array as shown in Fig 45.22.



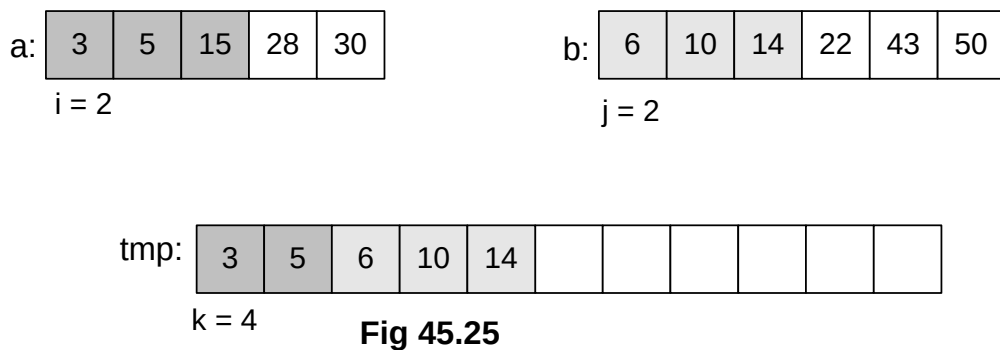
i is incremented again and it reaches the element 15. Now, elements 15 and 6 are compared. Obviously 6 is smaller than 15, therefore, 6 will take the place in the *tmp* array as shown in Fig 45.23.



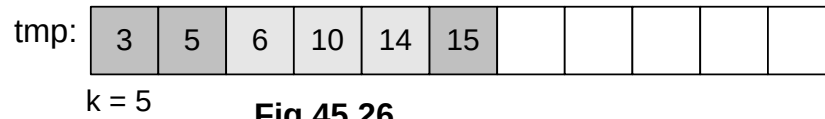
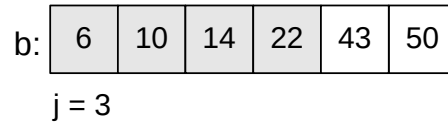
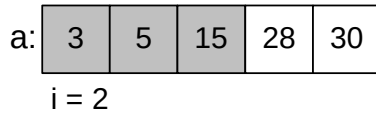
Because this time the element was taken from right array with index j , therefore, this time index j will be incremented. Keep an eye on the index k that is increasing after every iteration. So this time, the comparison is made between 15 and 10. 10 being smaller will take the place in *tmp* array as shown in Fig 45.24.



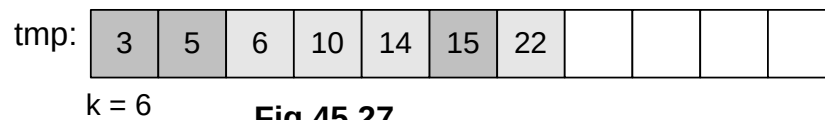
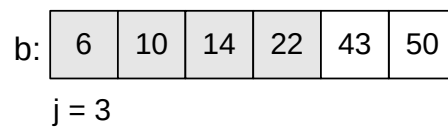
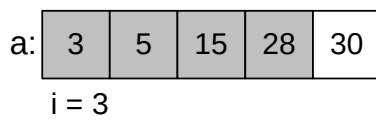
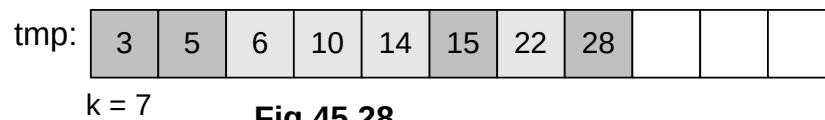
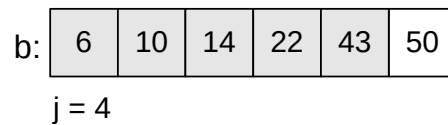
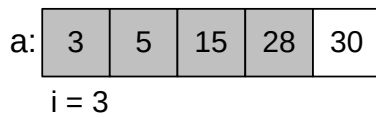
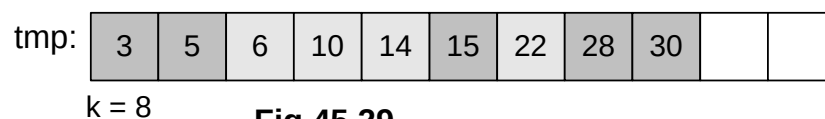
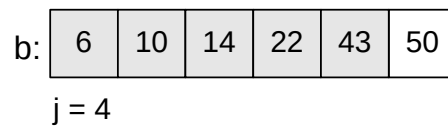
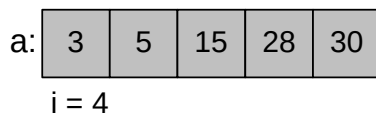
Again, j will be incremented because last element was chosen from this array. Elements 15 and 14 are compared. Because 14 smaller, so it is moved to *tmp* array in the next position.



With these movements, the value of k is increasing after each iteration. Next, 22 is compared with 15, 15 being smaller of two is put into the *tmp* array as shown in Fig 45.26.

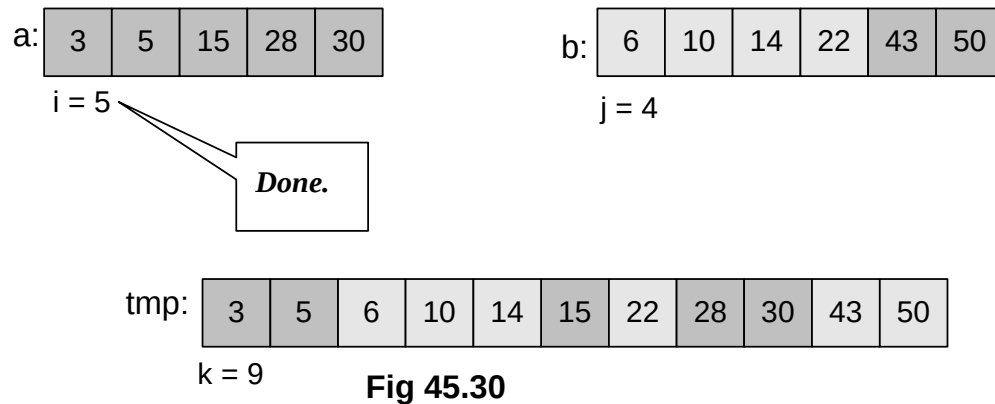
**Fig 45.26**

By now , you must have understood how this is working, let's see the next iterations pictures below:

**Fig 45.27****Fig 45.28****Fig 45.29**

Note that array *a* is all used up now, so the comparison operation will end here. All the remaining elements of array *b* are incorporated in *tmp* array straightaway as

shown in Fig 45.30. Hence, we have the resultant merged array *tmp* as shown in the figure.



Mergesort and Linked Lists

Merge sort works with arrays as well as linked lists. Now we see, how a linked list is sorted. Suppose we have a singly linked list as shown in figure Fig 45.31. We can divide the list into two halves as we are aware of the size of the linked list. Each half is processed recursively for sorting. Both of the sorted resultant halves are merged together.

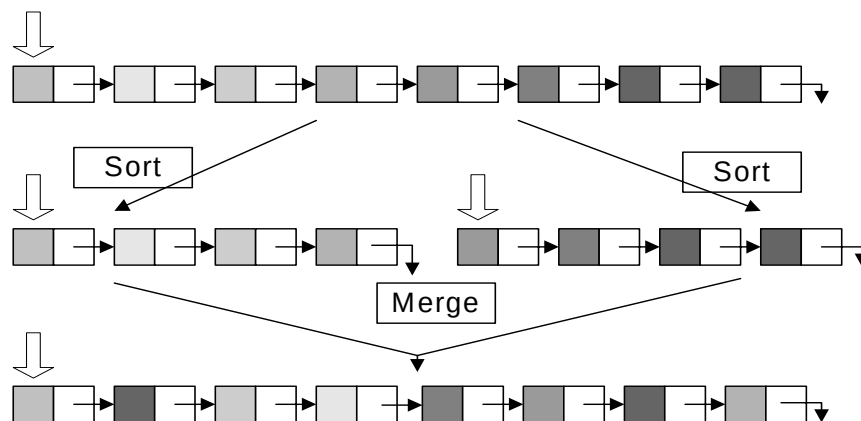


Fig 45.31

Mergesort Analysis

As you have seen, we used an additional temporary array while performing the merging operation. At the end of the merging operation, elements are copied from temporary array to the original array. The merge sort algorithm is not an inplace sorting algorithm because it requires an additional temporary array of the same size as the size of the array under process for sorting. This algorithm is still a good sorting algorithm, we see this fact from its analysis.

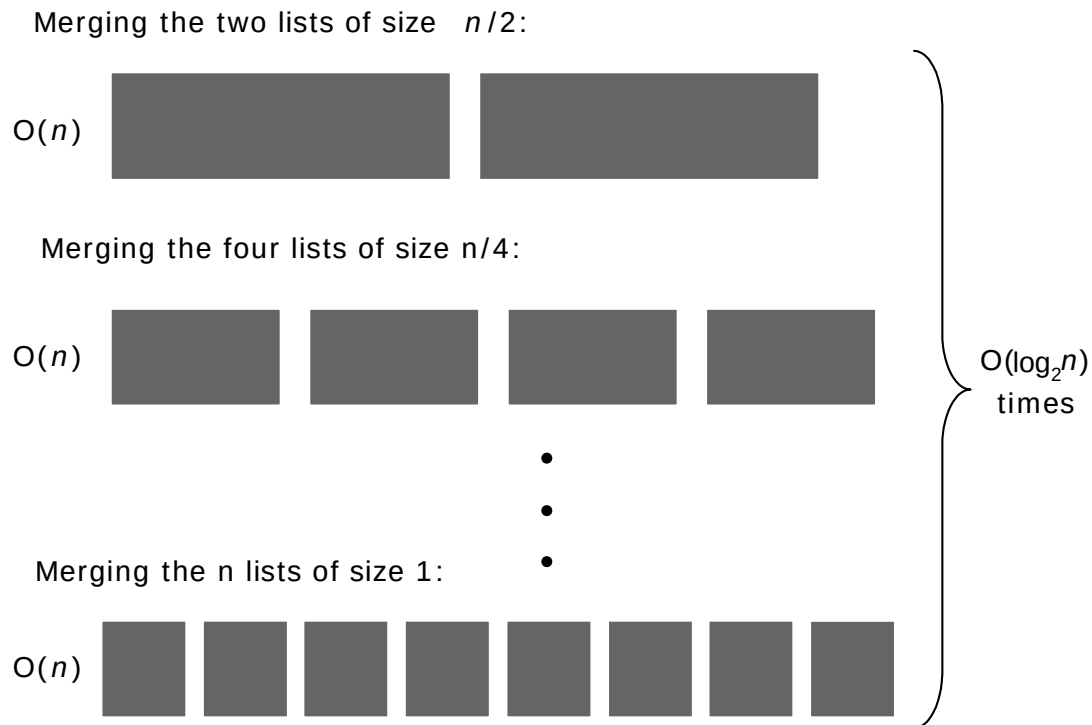


Fig 45.32

- *Mergesort is $O(n \log_2 n)$*
- *Space?*
- *The other sorts we have looked at (insertion, selection) are **in-place** (only require a constant amount of extra space)*
- *Mergesort requires $O(n)$ extra space for merging*

As shown in Fig 45.32, the array has been divided into two halves. We know that merging operation time is proportional to n , as it is done in a single loop regardless of the number of equal parts of the original array. We also know that this dividing the array into halves is similar mechanism as we do in a binary tree and a complete or perfect balance tree has $\log_2 n$ number of levels. Because of these two factors, the merge sort algorithm is called $n \log_2 n$ algorithm. Now, let's discuss about quick sort algorithm, which is not only an $n \log_2 n$ algorithm but also an inplace algorithm. As you might have guessed, we don't need an additional array while using this algorithm.

Quicksort

- **Quicksort** is another divide and conquer algorithm.
- Quicksort is based on the idea of partitioning (splitting) the list around a pivot or split value.

Quicksort is also a divide and conquer algorithm. We see pictorially, how the quick sort algorithm works. Suppose we have an array as shown in the figure Fig 45.33.

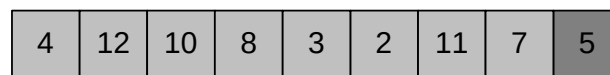


5

pivot value

Fig 45.33

We select an element from the array and call it the *pivot*. In this array, the *pivot* is the middle element 5 of the array. Now, we swap this with the last element 3 of the array. The updated figure of the array is shown in Fig 45.34.



↑
low

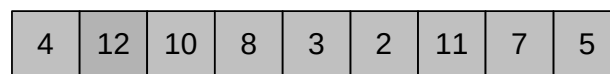
↑
high

5

pivot value

Fig 45.34

As shown in Fig 45.34, we used two indexes *low* and *high*. The index *low* is started from 0th position of the array and goes towards right until $n-1^{\text{th}}$ position. Inside this loop, an element that is bigger than the *pivot* is searched. The *low* index is incremented further as 4 is less than 5.



↑
low

↑
high

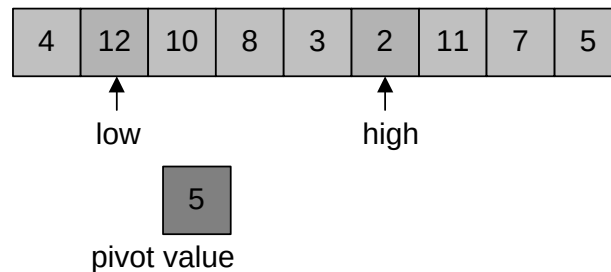
5

pivot value

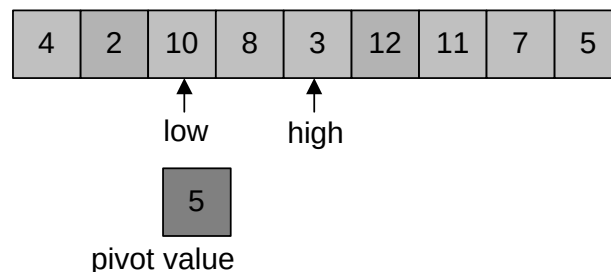
Fig 45.35

low is pointing to element 12 and it is stopped here as 12 is greater than 5. Now, we start from the other end, the *high* index is moved towards left from $n-1^{\text{th}}$ position to 0.

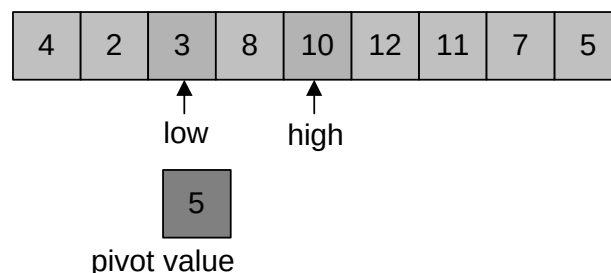
While coming from right to left, we search such an element that is smaller than 5. Elements 7 and 11 towards left are greater than 5, therefore, the *high* pointer is advanced further towards left. *high* index is stopped at the next position as next element 2 is smaller than 5. Following figure Fig 45.36 depicts the latest situation.

**Fig 45.36**

Both of the indexes have been stopped, *low* is stopped at a number 12 that is greater than the *pivot* and *high* is stopped at number 2 that is smaller than the *pivot*. In the next step, we swap both of these elements as shown in Fig 45.37.

**Fig 45.37**

Note that our *pivot* element 5 is still there at its original position. We again go to index *low* and start moving towards right, trying to find a number that is greater than the *pivot* element 5. It immediately finds the next number 10 greater than 5. Similarly, the *high* is moved towards left in search to find an element smaller than the *pivot* element 5. The very next element 3 is smaller than 5, therefore, the *high* index stops here. These elements 10 and 3 are swapped, the latest situation is shown in Fig 45.38.

**Fig 45.38**

Now, in the next iteration both *low* and *high* indexes cross each other. When the *high* pointer crosses the *low* pointer, we stop it moving further as shown in Fig 45.39 and

swap the element at the crossing position (which is 8) with the *pivot* number as shown in Fig 45.40.

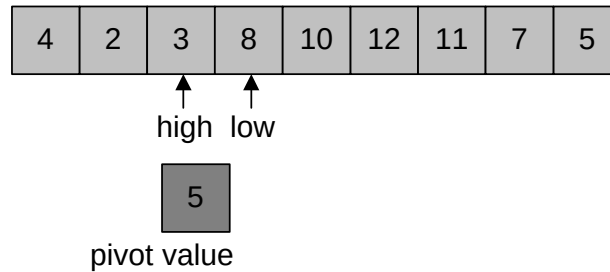


Fig 45.39

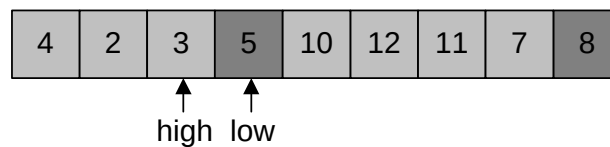


Fig 45.40

This array is not sorted yet but element 5 has found its destination. The numbers on the left of 5 should be smaller than 5 and on right should be greater than it and we can see in Fig 45.40 that this actually is the case here. Notice that smaller numbers are on left and greater numbers are on right of 5 but they are not sorted internally. Next, we recursively quick sort the left and right parts to get the whole array sorted.

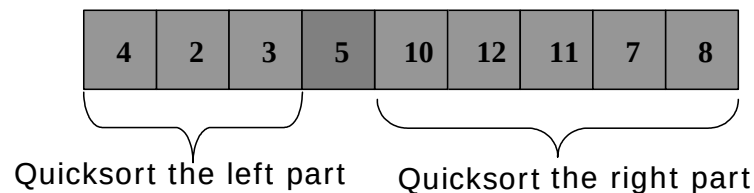


Fig 45.41

Now, we see the C++ code of quick sort.

```
void quickSort(int array[], int size)
{
    int index;
    if (size > 1)
    {
        index = partition(array, size);
        quickSort(array, index);
        quickSort(array+index+1, size - index-1);
    }
}

int partition(int array[], int size)
{
    // ...
```

```

    int k;
    int mid = size/2;
    int index = 0;
    swap(array, array+mid);
    for (k = 1; k < size; k++){
        if (array[k] < array[0]){
            index++;
            swap(array+k, array+index);
        }
    }
    swap(array, array+index);
    return index;
}

```

An array and its size are passed as arguments to the *quickSort* function. The function declared a local variable *index* and the size of the array is checked in the next statement. If the size of the array is more than 1 then the function does the recursive calling mechanism to sort the array. It divides the array into two parts by choosing the pivot element. In the subsequent calls, firstly the left side is sorted and then the right side using the recursive mechanism. Quicksort algorithm is very elegant and it can sort an array of any size efficiently. It is considered one of the best general purpose algorithms for sorting. This normally is preferred sorting method being $n\log_2 n$ and inplace algorithm. You are advised to read more about this algorithm from your text books and try to do it as an exercise.

Today's lecture being the last lecture of the course, let's have a short review of it.

Course Overview

We had started this course while keeping the objectives of data structures in mind that appropriate data structures are applied in different applications in order to make them work efficiently. Secondly, the applications use data structures that are not memory hungry. In the initial stages, we discussed *array* data structure. After we found one significant drawback of arrays; their fixed size, we switched our focus to *linked list* and different other data structures. However, in the meantime, we started realizing the significance of algorithms; without them data structures are not really useful rather I should say complete.

We also studied *stack* and *queue* data structures. We implemented them with *array* and *linked list* data structures. With their help, we wrote such applications which seemed difficult apparently. I hope, by now, you must have understood the role of stack in computer's runtime environment. Also *Queue* data structure was found very helpful in Simulations.

Later on, we also came across the situations when we started thinking that linear data structures had to be tailored in order to achieve our goals with our work, we started studying trees then. Binary tree was found specifically very useful. It also had a degenerate example, when we constructed AVL trees in order to balance the binary search tree. We also formed threaded binary trees. We studied union/find data structure, which was an up tree. At that time, we had already started putting special importance on algorithms. We found one important fact that it is not necessary that for every application we should use a new data structure. We formed new Abstract Data Types (ADT) using the existing data structures. For dictionary or table data

structure, we majorly worked with ADTs when we implemented them in six different ways. We also studied Skip List within the topic of Table ADT, which is a very recent data structure. After that we discussed about Hashing. Hashing was a purely algorithmic procedure and there was nothing much as a data structure.

In future, you will more realize the importance of algorithm. While solving your problems, you will choose an algorithm to solve your problem and that algorithm will bring along some data structure along. Actually, data structure becomes a companion to an algorithm. For example, in order to build your symbol table while constructing your own compiler, you will use hashing. For searches, trees will be employed.

One important fact here is that the data structures and algorithms covered in this course are not complete in the sense that you don't need any other data structure except them. One example is Graph, which is not discussed much in this course. Graph data structures are primarily important from algorithmic point of view.

Now, you should examine yourself, what have you learned in this course. As a software engineer, you have learned data structures and algorithmic skills to increase your domain knowledge of design choices. You can apply these design choices in order to resolve different design problems of your applications that you will come across in your student and professional life.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Definition

- ✓ Deferent B/W Mergesort and Linked Lists?
- ✓ Define mergeArrays?
- ✓ What is Bubble Sort?
- ✓ Insertion Sort?
- ✓ Hashing is Suitable?
- ✓ Applications of Hashing?
- ✓ Linear Probing and Collision?

✓ What are the constraints?

Constraint defines a limitation or restriction on resources of a system. In the computer, we have hard disk, memory and other hardware. We also have time to consider as a resource. Suppose you are writing an algorithm to solve a given problem and you have disc space of 400 Mega bytes. So, here you have space constraint that your algorithm should use only the available space in an efficient way. A solution is considered to be an efficient solution if it solves the problem within its resource constraints.

✓ What is lvalue?

In an assignment statement, lvalue refers to some memory location where some value can be stored. Lvalue variable is written on left hand side of an assignment statement. For example, in the following statement, variable 'var' is lvalue variable and value 7 will be stored in it. `int var= 7;`

What is "current" pointer ?

"Current" is a variable showing the status of currently focused element in array/list, for example if `current = 5`, it means that currently the focus is on element no. 5 in the array/list.

✓ What is algorithm?

Algorithm is a finite sequence of steps for the solution of a problem.

✓ WHAT IS ROLE OF ARRAYS IN DATA STRUCTURE?

Array is the most widely used data structure. It is also used as constituting integral part of other data structures. Array is mainly used to store similar data. For storing elements of indexed data, the particular method is array. For example, if you were storing information about each day in August, you would create an array with an index capable of addressing 31 values -- one for each day of the month. Think of a notebook with pages numbered 1 through 12. Each page may or may not contain information on it. The notebook is an array of pages. Each page is an element of the array 'notebook'. Arrays can also be multidimensional - instead of accessing an element of a one-dimensional list, elements are accessed by two or more indices, as from a matrix or tensor. Arrays guarantee constant time read and write access, $O(1)$, however many lookup operations (`find_min`, `find_max`, `find_index`) of an instance of an element are linear time, $O(n)$. Arrays are very efficient in most languages, as operations compute the address of an element via a simple formula based on the base address element of the array.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

✓ HOW MANY TYPES AN DATA CAN BE ORGANIZE?

Data can be organized in two main ways

- 1- Linear as in arrays, lists
- 2- Non Linear or Hierarchical like Trees

What is important of data structure in our today's life?

Data structures importance/applications in Daily life :

In bank service rows of men follow the Queue data structure.

In online shopping, selection and cancellation in the list of items is done/maintained via data structures.

For representing a city region telephone network.

Used to store a set of fixed key words which are referenced very frequently. DS are used to represent an image in the form of a bitmap.

To implement back functionality in the internet browser.

To implement printer spooler so that jobs can be printed in the order of their arrival. To record the sequence of all the pages browsed in one session.

To implement the undo function in a text editor.

To store information about the directories and files in a system .

✓ What is chunk?

Chunk means "a piece of reasonable size". Computer memory is divided into small parts.

Combination (allocation in sum) of these parts constitutes and can be called as chunk.

What is Basic purpose of data structure?

The main purpose of data structure is to organize and store the Data in Computer, in order to make the use of data efficient. Use of data efficiently means to use less space by the data and operations like searching, deletion, insertion etc on data took less time.

✓ What are argc and argv ?

The variables argc and argv in the definition of main function, that is, `main( int argc, char** argv)` are called command line arguments.

What is the difference between the Data structure and Algorithms?

Data Structure is the organization of Data in computer in an efficient way, while Algorithm is a finite sequence of steps for the solution of a problem.

Example,

Consider there are a number of students whose marks are needed to be stored in the computer, now we can store them in computer in an efficient way by using data structure like array, Linked List etc..

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Consider we need to find the maximum marks obtained by a student in this list, now for this we need to use an algorithm (Sequence of steps), for example,

✓ What is efficiency?

A solution is said to be efficient if it solves the problem within its resource constraints.

Space

Time

The cost of a solution is the amount of resources that the solution consumes. What is x?

X is a name of collection of items. Its individual items are numbered from zero to one less than array size. To access a cell, use the array name and an index as under:

x[0], x[1], x[2], x[3], x[4], x[5]

✓ What is Array Name?

-‘x’ is an array name but there is no variable x. ‘X’ is not an ‘l’ value.

For example, if we have the code Int a, b;

Then we can write b=2;

a=b; a=5;

But we cannot write 2=a;

List Data Structure:

✓ What are lists?

A list is collection of items that are all of the same type (grocery items, integers, names)

Operation Name

createList()

Description

Create a new list (presumably empty)

copy()

Set one list to be a copy of another

clear();

Clear a list (remove all elements)

insert(X, ?)

Insert element X at

a particular position in the list remove(?)

Remove element at

some position in the list

get(?)

Get element at a given position

update(X, ?)

Replace the

element at a given position with X find(X)

Determine if the element X is in the list

length()

Returns the length of the list.

If we use the “current” marker, the following four methods would be useful:

Functions

Description

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

start()	Moves the "current" pointer to the very first element
tail()	Moves the "current" pointer to the very last element
next()	Move the current position forward one element
back()	Move the current position backward one element

What are dynamic arrays?

In computer science, a dynamic array, grow able array, resizable array, dynamic table, or array list is a data structure that can be resized and allows elements to be added or removed. It is supplied with standard libraries in many modern mainstream programming languages.

A dynamic array is not the same thing as a dynamically-allocated array, which is a fixed-size array whose size is fixed when the array is allocated.

✓ what is the major difference between "Array" and "List"?

Array is a collection of memory cells of same type. In computer science, an array data structure or simply array is a data structure consisting of a collection of elements (values or variables), each identified by one or more integer indices, stored so that the address of each element can be computed from its index tople by a simple mathematical formula. For example, an array of 10 integer variables, with indices 0 through 9, may be stored as 10 words at memory addresses 2000, 2004, 2008, ... 2036; so that the element with index i has address $2000 + 4 \times i$

In computer science, a linked list is a data structure that consists of a sequence of data records such that in each record there is a field that contains a reference (i.e., a link) to the next record in the sequence.

✓ what is the basic concept of Data Structures?

This is very important subject as the topics covered in it will be encountered by you again and again in the future courses. Due to its great applicability, this is usually called as the foundation course. You have already studied Introduction to programming using C and C++ and used some data structures.

The focus of that course was on how to carry out programming with the use of C and C++ languages besides the resolution of different problems. In this course, we will continue problem solving and see that the organization of data in some cases is of immense importance. Therefore, the data will be stored in a special way so that the required result should be calculated as fast as possible

✓ what is list data structure ?

In computer science, a list or sequence is an abstract data structure that implements an ordered collection of values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a finite sequence, that is, a tuple. Each instance of a value in the list is usually called an item, entry, or element of the list; if the same value occurs multiple times, each occurrence is considered a distinct item.

✓ what parameters are considered to identify l value.?

If we can assign a value to a variable then we can the variable is lvalue type. Lets see with an examples

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

int x;

Here x is a integer type variable we can assign any integer value to x like x = 10 or x = 2

In the following code statement y is an array which can hold 10

values. int y[10];

In this scenario y is not lvalue, the reason is we cannot assign a single integer value to y like y = 10

Lets we elaborate it with diagram.

Suppose the box given below is int variable x. we can easily assign (save) 10 or 2 to it. Assume 10 boxes given below are representing an array y. We cannot assign single value 10 or 2 to y. The problem is, if we will assign 10 to y then which box (place) in array it will save. This is the reason array is not lvalue.

Data structure concept in easy wording?

A data structure is a group of data elements grouped together under one name. Data structure helps to store the data in computer in an organized and efficient way. Data structures are used to efficiently utilize the memory of computer. Data structures not only consider the items stored but also the relationship to each other.

✓

✓ What is Difference b/w arrays and linked list?

- i. The difference between arrays and linked lists are:
- ii. Arrays are linear data structures. Linked lists are linear and non-linear data structures.
- iii. Linked lists are linear for accessing, and non-linear for storing in memory
- iv. Array has homogenous values. And each element is independent of each other positions.
- v. Each node in the linked list is connected with its previous node which is a pointer to the node.
- vi. Array elements can be modified easily by identifying the index value. It is a complex process for modifying the node in a linked list.
- vii. Array elements cannot be added, deleted once it is declared. The nodes in the linked list can be added and deleted from the list.

✓ What are Linked lists ?

Linked lists are a way to store data with structures so that the programmer can automatically create a new place to store data whenever necessary. Specifically, the programmer writes a struct definition that contains variables holding information about something and that has a pointer to a struct of its same type (it has to be a pointer--otherwise, every time an element was created, it would create a new element, infinitely). Each of these individual structs or classes in the list is commonly known as a node or element of the list.

✓ Why we need array to implement this linked memory list?

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Using linked memory method is the other way of implementing the list. We have to implement a list of elements, so this implementation can be done in 2 ways. First is array implementation in which the elements will be stored in contiguous memory locations. There are certain limitations of this type of implementation of list i.e. the size is fixed. So to avoid these limitations, we can implement list by other method. That method is implementing via linked memory locations which do not restrict the list items to be stored contiguously.

✓ what is the difference b/w array that is dynamic allocated and that is simple ?

Simple array (Static) is created in stack and dynamic is created in heap area of the memory. In static array you have to tell the size before program runs but in dynamic you can take input from user and then make array of that size.

Example static: `int array[6];`

Example dynamic: `int *array; cout > size;`

`array = new int[size];`

What is the difference b/w array and list data structure?

Array is collection of homogeneous elements. List is collection of heterogeneous elements. For ⁶ Array memory allocated is static and continuous while for List memory allocated is dynamic and Random.

In array, user need not have to keep in track of next memory allocation while in list user has to keep in Track of next location where memory is allocated.

Array size should be specified but the list size can be determined at run time also.

✓ How can we set current position ?

“current” is a variable showing the status of currently focused element in array/list, for example if current = 5, it means that currently the focus is on element no. 5 in the array/list.

You can simply assign any array index to the variable “current” like, current = 3; where 3 is an array index.

✓ How can we shift element left or right?

This is a programming way that how can we shift element in an array. For example, consider the following small program,

```
#include <iostream.h> #include <conio.h> main()
{
    int a[5] = { 10, 20, 30, 40, 50};
    for(int i =0;i<5;i++)
    {
        cout<<a[i] <<"\t" ;
```

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

```
}  
  
coutendl;  
// after shifting  
int first = a[0]; // Save 1st element or delete for(int i =0;i<4;i++)  
{  
a[i] = a[i+1]; // Real shifting statement. couta[i] 't' ;  
}  
getche(); return 0;  
}
```

Following is the list of operations and their description:

Operation NameDescription

createList()	Create a new list (presumably empty)
copy()	Set one list to be a copy of another
clear();	Clear a list (remove all elements)
insert(X, ?)	Insert element X at a particular position in the list
remove(?)	Remove element at some position in the list
get(?)	Get element at a given position
update(X, ?)	Replace the element at a given position with X
find(X)	Determine if the element X is in the list
length()	Returns the length of the list.

✓ what is the main difference between array and list?

In computer science, an array data structure or simply array is a data structure consisting of a collection of elements (values or variables), each identified by one or more integer indices, stored so that the address of each element can be computed from its index tuple by a simple mathematical formula. For example, an array of 10 integer variables, with indices 0 through 9, may be stored as 10 words at memory addresses 2000, 2004, 2008, ... 2036; so that the element with index i has address $2000 + 4 \times i$.

In computer science, a list or sequence is an abstract data structure that implements an ordered collection of values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a finite sequence, that is, a tuple. Each instance of a value in the list is usually called an item, entry, or element of the list; if the same value occurs multiple times, each occurrence is considered a distinct item.

✓ what is dynamic memory allocation ?

In computer science, dynamic memory allocation (also known as heap-based memory allocation) is the allocation of memory storage for use in a computer program during the

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

runtime of that program. It can be seen also as a way of distributing ownership of limited memory resources among many pieces of data and code. Dynamically allocated memory exists until it is released either explicitly by the programmer, or by the garbage collector. This is in contrast to static memory allocation, which has a fixed duration. It is said that an object so allocated has a dynamic lifetime.

✓ what is the main difference between list implementation and add method?

Suppose we want to create a list of integers. For this purpose, the methods of the list can be implemented with the use of an array inside. For example, the list of integers (2, 6, 8, 7, 1) can be represented in the following manner where the current position is 3.

A	2	6	8	7	1	Current	Size
	1	2	3	4	5	3	5

In this case, we start the index of the array from 1 just for simplification against the usual practice in which the index of an array starts from zero in C++. It is not necessary to always start the indexing from zero. Sometimes, it is required to start the indexing from 1.

Add Method

Suppose there is a call to add an element in the list i.e. *add(9)*. As we said earlier that the current position is 3, so by adding the element 9 to the list, the new list will be (2, 6, 8, 9, 7, 1).

To add the new element (9) to the list at the current position, at first, we have to make space for this element. For this purpose, we shift every element on the right of 8 (the current position) to one place on the right. Thus after creating the space for new element at position 4, the array can be represented as

A	2	6	8	7	1	CurrentSize
-	1	2	3	4	5	4 6

Now in the second step, we put the element 9 at the empty space i.e. position 4. Thus the array will attain the following shape. The figure shows the elements in the array in the same order as stored in the list.

A	2	6	8	9	7	1	CurrentSize
-	1	2	3	4	5	6	4 6

We have moved the current position to 4 while increasing the size to 6. The size shows that the elements in the list. Whereas the size of the array is different that we have defined already to a fixed length, which may be 100, 200 or even greater.

✓ What is a difference between List(sequence), Array and pointer?

A list or sequence is an abstract or conceptual data type that implements an ordered collection of values or items. In order to practically implement this abstract or conceptual data type we must use some strategy like an array or linked list. So we use either arrays or linked list for the implementation or manipulation of List.

Whereas a pointer is a special type of variable in which a memory address is stored. Pointer contains a memory address, not the value of the variable.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

An array is a data structure in which identical data types are stored. Every array has a data type i.e. name and size. Data type can be any valid data type. The size of the array tells how many elements are there in the array. The arrays occupy the memory depending upon their size and have contiguous area of memory. We can access the arrays using the array index.

✓ Difference between Programming and Data Structure?

Programming is very wide topic and covers lots of computer science concepts. In simple words programming is developing software to automate or solve a problem, with the help of different data structures and algorithms.

While data structure is a part of programming. Data structures are instructions to solve a problem by using computer resources efficiently.

✓ What are LValues Variables?

Lvalue variables are those variables that can be used on left side to save a value.

Variables are labels of memory locations that we want to use in our program. Assigning a value to variable is saving value on a memory location.

In following example `int x = 10;`

Here x is Lvalue and holding a value 10.

`int y[10];`

Here y is not Lvalue and we cannot write `y = 20;`

Because y is an array and can be used to hold 10 values. In above line our program will not know on which location it should save value 20.

✓ What is Index?

Index is location of element in an array. With the help of index we can reach to any element in the array. Here is example of using index

`int grades[10];` // declaring an integer array to store 10 integer values

`grades[4] = 3;` // assigning integer 3 to 5th location of array. here 4 is index of element.

✓ What do u meant By Link Memory?

Average case: Normally we may have to move half of the elements.

✓ What is Add method and Current Pointer?

Add method is used to add an element in the list. A list can be implemented either by using an array or linked list. When we implement a list by using an array then it is called array list. When we will add an element at the start of an array list then we should have to move every element to the right to make space for the new element. When we will add the element at the end of an array list then we will not move or shift any other array element. Normally we may have to move half of the elements in average cases.

Current marker or pointer refers to a particular position in the list where we are currently standing. If we do not use current pointer then we will not be able to keep track of our list i.e. we will be unable to identify our current position so it will become difficult for us to add or remove an element from list.

What do we meant by Link list inside a computer memory ?

It means how the memory will be allocated to the linked list and how memory is linked for your list elements.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

✓ What is Node?

These are the elements that make it possible to have a linked list. Without them, there would be no listing. Each node contains two parts; a part for the data and a part for the address of the next element. As you can see, this will use pointers.

✓ What is state variables?

The products that are made by the factory have their own characteristics. For example, a car made by an automobile factory has an engine, wheels, steering and seats etc. These variables inside a class are called state variables.

getNext()

getNext() which returns a pointer to an object of type Node lying somewhere in the memory. It returns nextNode i.e. a pointer to an object of type Node. NextNode contains the address of next node in the linked list.

setNext()

setNext()

that accepts a pointer of type Node, further assigned to nextNode data member of the object. This method is used to connect the next node of the linked list with the current object. It is passed an address of the next node in the linked list.

✓ What are list interface, list implementation and analysis?

List interface means that how list works internally and how we interact with the list. If write the C++ program to implement the list then we simply need to know that what are those methods, their names and what arguments that method takes etc.

Now we will see what the implementation of the list is and how one can create a list in C++. After designing the interface for the list, it is advisable to know how to implement that interface. Suppose we want to create a list of integers. For this purpose, the methods of the list can be implemented with the use of an array inside.

For example, the list of integers (2, 6, 8, 7, 1) can be represented in the following manner where the current position is 3.

A	2	6	8	7	1	Current	Size
	1	2	3	4	5	3	5

In this case, we start the index of the array from 1 just for simplification against the usual practice in which the index of an array starts from zero in C++. It is not necessary to always start the indexing from zero.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Sometimes, it is required to start the indexing from 1. For this, we leave the zeroth position and start using the array from index 1 that is actually the second position. Suppose we have to store the numbers from 1 to 6 in the array. We take an array of 7 elements and put the numbers from the index 1. Thus there is a correspondence between index and the numbers stored in it. This is not very useful. So, it does not justify the non-use of zeroth position of the array out-rightly. However for simplification purposes, it is good to use the index from 1.

✓ What is the main purpose of node?

For the utilization of the concept of linked memory, we usually define a structure, called linked list. To form a linked list, at first, we define a node. A node comprises two fields. i.e. the object field that holds the actual list element and the next that holds the starting location of the next node.

✓ What is the importance of link list in memory?

Linked lists are among the simplest and most common data structures, and are used to implement many important abstract data structures, such as stacks, queues, hash tables, symbolic expressions, skip lists, and many more.

The principal benefit of a linked list over a conventional array is that the order of the linked items may be different from the order that the data items are stored in memory or on disk. For that reason, linked lists allow insertion and removal of nodes at any point in the list, with a constant number of operations

✓

✓ what are the operations of the linked list?

The linked list data structure provides operations to work on the nodes inside the list.

The first operation we are going to discuss here is to create a new node in the memory. The Add(9) is used to create a new node in the memory at the current position to hold '9'. You must remember while working with arrays, to add an element at the current position that all the elements after the current position were shifted to the right and then the element was added to the empty slot.

Here, we are talking about the internal representation of the list using linked list. Its interface will remain the same as in case of arrays.

We can create a new node in the following manner in the add() operation of the linked list with code in C++:

```
Node * newNode = new Node(9);
```

The first part of the statement that is on the left of the assignment is declaring a variable pointer of type Node. It may also be written as Node * newNode. On the right of this statement, the new operator is used to create a new Node object as new Node(9). This is one way in C++ to create objects of classes. The name of the class is provided with the new operator that causes the constructor of the class to be called. The constructor of a class has the same name as the class and as this a function, parameters can also be passed to it. In this case, the constructor of the Node class is called and '9' is passed to it as an int parameter.

Hence, the whole statement means:

“Call the constructor of the Node class and pass it '9' as a parameter. After constructing the object in memory, give me the starting memory address of the object. That address will be stored in the pointer variable newNode.”

Limitation of array and use of linked list

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

There are number of limitations of array data structure. Most common are as follows:-

1. Once an array is created, its size cannot be altered

2. Array provides inadequate support for inserting and deleting operations.

A linked list is a popular data structure to store data in sequential order. Meanwhile, linked list structure allow following operations which overcomes limitation of array: for example

1. Retrieve an element from list.
2. Insert a new element to the list.
3. Delete an element from the list.
4. Find how many elements are in the list.
5. Find a specific element is in the list.
6. Find if this list is empty.

✓ what is constant pointer?

Let's look at the use of const. Consider the following line of declaration: `int *const myptr = &x ;`

The right hand side of this assignment statement could be read as, myptr is a constant pointer to an integer.

Whenever we use the keyword const with a variable, the value of that variable becomes constant and no other value can be assigned to it later on. We know that when we declare a constant variable like `const int x ;` it is necessary to assign a value to x and we write `const int x = 10 ;`. After this, we cannot assign some other value to x. The value of x can not be changed as it is declared as a constant. Now consider the previous statement `int *const myptr = &x ;`

Here we declare a constant pointer to an integer. Being a constant pointer, it should immediately point to something. Therefore, we assign this pointer an address of a variable x at the time of declaration. Now this pointer cannot be changed i.e. now we cannot assign address of any other variable to pointer myptr at any stage. The pointer myptr will hold the address of variable x throughout the program.

✓

✓ What are constructors and Its use?

Constructor is a special function of a class, which is called every time a new object of the class is created.

Constructor has the same name as of the class.

It is basically to initialize the object of the class. This function does not have return type. If you do not include an explicit constructor, the compiler will call a default implicit constructor.

✓

✓ What is command line arguments?

`main()` is function from which the execution of the program is started. Your program can not be executed if main is missing in it.

`main( int argc, char** argv )`

Both of arguments in main are used to manage command line arguments passed to our program. The first argument argc (arguments count) gives the number of command line arguments passed. While second argument argv (arguments vector) contain actual command line arguments passed.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Example:

if we are executing our program Test.exe we will use this command in cmd. Test.exe

If we want to pass arguments while executing our program, we can write this command. Test.exe abc def gh i 1 5 xyz

Here Test.exe is our program executable file and others are command line arguments we are passing during execution.

C++ will consider Test.exe as command line argument too. Now argc variable will hold 7 as it is number of command line arguments and argv will hold all command line arguments we passed.

✓ **How we can add(5) in link-list but on ascending order ?**

First of all, you will search the correct position of the new node and then insert it into that position. To search the correct position, you will start from the first node. Compare the data of first node with the new node, if it is smaller than new data, move one node forward. Repeat the process until you find the node with the greater data than new data. At the end of search, current pointer will point to the desired node. Make a connection of previous node of current node with new node, and place current node reference in new node.

✓

✓ **What is the Josephus problem (or Josephus permutation) ?**

In computer science and mathematics, the Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game.

There are people standing in a circle waiting to be executed. After the first man is executed, certain number of people are skipped and one man is executed. Then again, people are skipped and a man is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last man remains, who is given freedom.

✓

✓ **What is the difference between doubly link list and single link list?**

If you look at single link list, the chain is seen formed in a way that every node has a field next that point to the next node. This continues till the last node where we set the next to NULL i.e. the end of the list. There is a headNode pointer that points to the start of the list. We have seen that moving forward is easy in single link list but going back is difficult. For moving backward, we have to go at the start of the list and begin from there. Do you need a list in which one has to move back or forward or at the start or in the end very often? If so, we have to use double link list.

In doubly-link list, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node. Now our node factory will create a node with three parts. First part is prev i.e. the pointer pointing to the previous node, second part is element, containing the data to be inserted in the list. The third part is next pointer that points to the next node of the list. The objective of prev is to store the address of the previous node.

✓ **How we come to know which one is current point?**

current is a marker or pointer to refer to a particular position in the list.

We have a variable named current.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Now in the start, when our list is empty, the current has value 0 i.e; current is pointing to the 0th location of array which is empty.

When we want to insert first item into the list, we first increment value in the current, so now current has value 1 i.e; current is pointing to the 1st location of array. So here we insert the first item.

That is how we keep track of the current item in the array list.

Consider the following example to understand the current position.

Suppose you are reading a page and using your finger to point at the word you are reading. As you read the next word in the line, you move your finger towards next. Now at any time, if you or someone wants to check which word you are reading, simply check which word your finger is pointing at.

What are Arrays advantages over Linked list? Arrays advantages over Linked list:

Arrays allow random access to its elements and thus the complexity is order of $O(1)$.

Linked lists allow only sequential access to elements. Thus the algorithmic complexity is order of $O(n)$.

Arrays do not need an extra storage to point to next data item. Each element can be accessed via indexes.

Linked lists require an extra storage for references. This makes them impractical for lists of small data items such as characters or Boolean values.

✓ What are Linked list advantages over Arrays?

Linked list advantages over Arrays:

The size of array is restricted to declaration. If you want to increase array size, declare a new array of larger size and then copy old array elements to this new array.

The size of Linked lists is dynamic by nature.

If you want to delete something from the middle of array, you have to move half of the array to the left. But in case of Linked list, you only move two or three pointers. Same is the case with insertion.

Circularly-linked lists

The next field in the last node in a singly-linked list is set to NULL. The same is the case in the doubly-linked list. Moving along a singly-linked list has to be done in a watchful manner. Doubly-linked lists have two NULL pointers i.e. prev in the first node and next in the last node. A way around this potential hazard is to link the last node with the first node in the list to create a circularly-linked list.

✓ What is Node?

We form a link list by joining nodes, a node consists of two fields first is used to store the actual object value and second is used to store a reference to the next node. In fact a linked list is a chain of these nodes and each node have a pointer or reference to the next node while the last node have a null pointer.

✓

✓ difference between Find and back in linked list.?

Find() method uses to find an element in a data structure whereas Back() function of the list will move the current pointer one item back, for example, if the current pointer point to item no 3 of the list and we call the back() function of this list, after this call the current pointer (means current item) will be item no. 2. Find() and Back are not one step operations, it is because sometimes during Find() operation we may have to search the entire list in order to find some particular element say x whereas in the back() method, we move the current pointer one position back. Moving the current pointer back, one requires traversing the list from the start until a node reach, whose next pointer points to current node.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

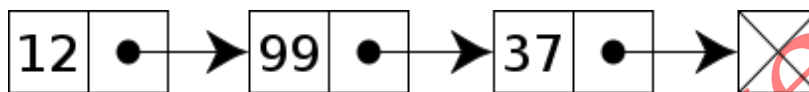
We provide all categories of final year project (CS619) and short notes

✓ how we know our current node in circularly-linked list?

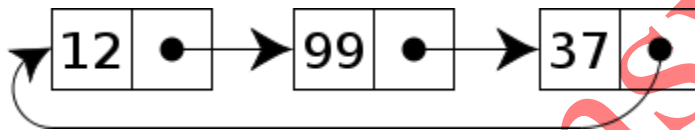
Suppose that you are given a linked list that is either circular or not circular (another word for not circular is acyclic). Take a look at the figures below if you are not sure what a circular linked list looks like.

Write a function that takes as an input a pointer to the head of a linked list and determines whether the list is circular or if the list has an ending node. If the linked list is circular then your function should return true, otherwise your function should return false if the linked list is not circular. You can not modify the linked list in any way.

This is an acyclic (non-circular) linked list:



This is a circular linked list:



You should start out this problem by taking a close look at the pictures of a circular linked list and a singly linked list so that you can understand the difference between the 2 types of lists.

The difference between the 2 types of linked lists is at the very last node of the list. In the circular linked list, you can see that the very last node (37) links right back to the first node in the list – which makes it circular. However, in the acyclic or non-circular linked list you can see that the end node does not point to another node in the linked list and just ends.

It is easy enough to know when you are in an acyclic linked list – the pointer in the end node will just be pointing to NULL. However, knowing when you are in a circularly linked list is more difficult because there is no end node, so your function would wind up in an infinite loop if it just searches for an end node. So, there has to be a solution other than just looking for a node that points to NULL, since that clearly will not work.

✓

✓ Difference between josephus problem and circularly linked?

In circularly link list, the last node is connected to the head node. Joseph problem is an application of circularly link list. We use circular link list to solve the Joseph problem.

Analysis of Linked List

\$add

we simply insert the new node after the current node. So add is a one-step operation.

\$remove

\$remove is also a one-step operation

\$find

\$worst-case: may have to search the entire list

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

\$back

\$moving the current pointer back one node requires traversing the list from the start until the node

Doubly-linked List

\$Moving forward in a singly-linked list is easy; moving backwards is not so easy.

\$To move back one node, we have to start at the head of the singly-linked list and move forward until the node before the current.

\$To avoid this we can use two pointers in a node: one to point to next node and another to point to the previous node:

✓ **Is link list and List same?if yes than explain how?**

List and linked list is same thing. List is the most generic data structure. A list is the collection of the same type of items. List can be implemented through array and linked memory. When we implement list through linked memory, it becomes linked list.

✓ **Difference between Head node and Current Node?**

headNode is a node, whose next pointer has a reference of the first node of the list. The data elements of this node do not have any values, except nextNode pointer.

Suppose, we have a list with elements 1, 2, 3, 4, 5,6. 1 is the data element of the first node of the list. headNode's next pointer will have reference of the node with the data element 1.

And current is a pointer variable in the linked list implementation which points to the node currently working upon.

✓ **What is abstract data ?**

There is no term as "abstract data" in data structure. What I perceive from your question is that you want to ask about Abstract Data Type (ADT). An abstract data type is a mathematical model of a certain type of data structure for example a Stack. It is defined in terms of operations on it; and implementation is hidden.

✓ **What are Object File and Binary File ?**

Object File:

When Compiler compiles a source code file (for example abc.cpp), the resulting compiled file is known as object file (abc.obj). Object files end in ".o" by convention, although on some operating systems (e.g. Windows, MS-DOS), they often end in ".obj". An object file contains function definitions in binary form, but they are not executable by themselves.

Binary File:

The linker links together a number of object files and as a result of that process generates binary files that can be directly executed. Binary executables have no special suffix on Unix operating systems, although they generally end in ".exe" on Windows.

✓ **What is the difference between isEmpty() and isFull()?**

Two of the methods used in the interface of stack are: isEmpty() and isFull().

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

isEmpty() is a Boolean function which will return true if stack is empty and return false if it is not empty; means there may be one or more elements in the stack.

isFull() is also a Boolean function which will return a Boolean value. If it returns true, it means that the stack is full and no more elements can be inserted and if it returns false, it means stack is not full and an element can be inserted into it.

✓ What is the difference between an array pointer and an ordinary pointer?

The difference between an array pointer and an ordinary pointer is that the array name is a constant pointer. It always contains the memory address of the first element of the array. As value of constant can never be changed or reassigned any other address so you can not perform an increment operator on an array pointer. Hence, you can not use (*Y)++ and so, in case Y is the name of an array.

✓ What are stacks and how are they affected in Data Structure?

A stack is a list with the restriction that inserts and deletes can be performed in only one position, namely the end of the list called the top.

Stack is used in infix to postfix conversion, function calls.

✓ What is Abstract data type (ADT)?

Abstract data type (ADT): A data type whose properties (domain and operations) are specified independently of any particular implementation.

List, for example, is an ADT. The functionality of an ADT can be implemented in many ways. The programs that use these ADTs will not need to know which implementation was used as long as it performs the required functionality.

The List ADT can be implemented through Arrays (elementary data structure that exists as built-in in most programming languages) or linked-structure (list implemented through linked-structure is called Linked-list).

Stacks

Definition of stacks

“Stack is a collection of elements arranged in a linear order”

Suppose we have some video cassettes. We took one cassette and put it on the table. We get another cassette and put it on the top of first cassette. Now there are two cassettes on the table- one at the top of other. Now we take the third cassette and stack it on the two. Take the fourth cassette and stack it on the three cassettes.

Now if we want to take the cassette, we can get the fourth cassette which is at the top and remove it from the stack. Now we can remove the third cassette from the stack and so on. Suppose that we have fifty cassettes stacked on each other and want to

access the first cassette that is at the bottom of the stack. What will happen? All the cassettes will fell down. It will not happen exactly the same in the computer. There may be some problem. It does not mean that our data structure is incorrect. As we see in the above example that the top most cassette will be removed first and the new cassette will be stacked at the top. The same example can be repeated with the books. In the daily life, we deal with the stacked goods very carefully.

✓ What is the difference between an array pointer and an ordinary pointer?

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

The difference between an array pointer and an ordinary pointer is that the array name is a constant pointer. It always contains the memory address of the first element of the array. As value of constant can never be changed or reassigned any other address so you can not perform an increment operator on an array pointer. Hence, you can not use $(*Y)++$ and so, in case Y is the name of an array.

✓ What are stacks and how are they affected in Data Structure?

A stack is a list with the restriction that inserts and deletes can be performed in only one position, namely the end of the list called the top.

Stack is used in infix to postfix conversion, function calls.

✓ What is Abstract data type (ADT)?

Abstract data type (ADT): A data type whose properties (domain and operations) are specified independently of any particular implementation.

List, for example, is an ADT. The functionality of an ADT can be implemented in many ways. The programs that use these ADTs will not need to know which implementation was used as long as it performs the required functionality.

The List ADT can be implemented through Arrays (elementary data structure that exists as built-in in most programming languages) or linked-structure (list implemented through linked-structure is called Linked-list).

Stacks

Definition of stacks

“Stack is a collection of elements arranged in a linear order”

Suppose we have some video cassettes. We took one cassette and put it on the table. We get another cassette and put it on the top of first cassette. Now there are two cassettes on the table- one at the top of other. Now we take the third cassette and stack it on the two. Take the fourth cassette and stack it on the three cassettes.

Now if we want to take the cassette, we can get the fourth cassette which is at the top and remove it from the stack. Now we can remove the third cassette from the stack and so on. Suppose that we have fifty cassettes stacked on each other and want to

access the first cassette that is at the bottom of the stack. What will happen? All the cassettes will fell down. It will not happen exactly the same in the computer. There may be some problem. It does not mean that our data structure is incorrect. As we see in the above example that the top most cassette will be removed first and the new cassette will be stacked at the top. The same example can be repeated with the books. In the daily life, we deal with the stacked goods very carefully.

Abstract, dynamic data type and stack.

Abstract data type:

If for a particular collection of data only the structure of data and the functions to be performed on the data is defined but the implementation is not defined, then such a collection of data is called Abstract data type. It means "Only declaration is there but implementation is not i.e it never uses further but it can be inherited".

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Dynamic data:

Dynamic data or transactional data denotes information that is asynchronously changed as further updates to the information become available. The opposite of this is persistent data, which is data that is infrequently accessed and not likely to be modified. Dynamic data is also different from streaming data, in that there is no constant flow of information. Rather, updates may come at any time, with periods of inactivity in between.

Stack:

in computer science, a stack is a last in, first out (LIFO) abstract data type and data structure. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: push and pop. The push operation adds to the top of the list, hiding any items already on the stack, or initializing the stack if it is empty. The pop operation removes an item from the top of the list, and returns this value to the caller. A pop either reveals previously concealed items, or results in an empty list. Stacks have many applications. For example, as processor executes a program, when a function call is made, the called function must know how to return back to the program, so the current address of program execution is pushed onto a stack. Once the function is finished, the address that was saved is removed from the stack, and execution of the program resumes. If a series of function calls occur, the successive return values are pushed onto the stack in LIFO order so that each function can return back to calling program. Stacks support recursive function calls in the same manner as conventional non recursive calls.

✓ **Difference between stack and Linked list?**

A stack is an abstract or conceptual data type that implements a collection of values or items in which the principal (or only) operations on the collection are the addition of an entity/element to the collection, known as push and removal of an entity/element, known as pop. The relation between the push and pop operations is such that the stack is a Last-In-First-Out (LIFO) data structure. In order to practically implement this abstract or conceptual data type we must use some strategy like an array or linked list. A linked list is a data structure consisting of a group of nodes which together represent a stack. So in simple words, stack is an abstract or conceptual data type that linked list is used to implement that abstract data type.

It depends on our requirement that out of different data structures which data structure is best suited for our requirements and project e.g. for some projects linked list is useful whereas for some projects arrays may be your preference. For example, let us consider that we want to develop a real world project in which each task should be completed as early as possible. In this case we should use linked list as compare to arrays because adding or removing an element/node from the middle of a linked list takes less time as compare to arrays. Similarly if we develop a project in which our main concern is to save the memory space as much as possible then we could use arrays instead of linked list.

✓ **Difference among ADT and class?**

In simple words, ADT (abstract data type) is a user defined conceptual data types and a class is used to implement this ADT/user defined data type. For example, a stack is an abstract data type and we use to implement this abstract data type by using a stack class

Abstract data type:

it is the collection of values and a set of operation on those values. ADT refers to the basic mathematical concepts. that defines the data type.

Abstract Data Types and Object-Orientation

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

ADTs allows the creation of instances with well-defined properties and behaviour. In object- orientation ADTs are referred to as classes. Therefore a class defines properties of objects which are the instances in an object-oriented environment.

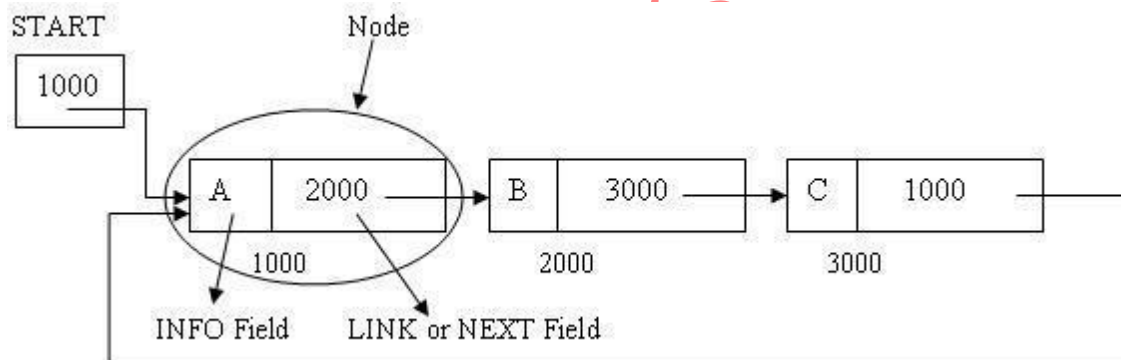
ADTs define functionality by putting main emphasis on the involved data, their structure, operations as well as axioms and preconditions. Consequently, object-oriented programming is ``programming with ADTs": combining functionality of different ADTs to solve a problem. Therefore instances (objects) of ADTs (classes) are dynamically created, destroyed and used.

✓ What is Linked list or Circular linked list ?

Linked list is a dynamic data structure that contains a "link" to the structure containing the next item. It is a collection of structures ordered not by their physical placement in memory (like array) but by logical links that are stored as part of the data in the structure itself

In it the last node does not contain NULL pointer. Instead the last node contains a pointer that has the address of first node and thus points back to the first node.

It is shown below:



✓ What are Advantages and Disadvantages of Link list ?

Advantages:

- Dynamic structure (Mem. Allocated at run-time).

We can have more than one datatype.

Re-arrange of linked list is easy (Insertion-Deletion).

It doesn't waste memory.

Disadvantages:

It is not easy to reverse the linked list.

If proper care is not taken, then the problem of infinite loop can occur.

If we at a node and go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node.

✓ Define Types of Linked lists?

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Types of Linked List Singly or Chain Linked List.

Doubly or Two Way Linked List.

Circular Linked List.

Circular Doubly Linked List.

Singly or Chain Linked List

The way to represent a linear list is to expand each node to contain a link or pointer to the next node. This representation is called a one-way chain or singly linked list.

Circular Linked List

A linked list in which the pointer of the last node points to the first node of the list is called circular linked list.

Doubly or Two-Way Linked List

A linked list which can be traversed both in backward as well as forward direction is called doubly linked list.

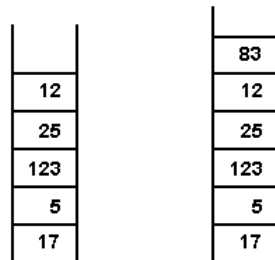
It uses double set of pointers.

Circular Doubly Linked List

It employs both the forward pointer and backward pointer in circular form.

Important thing in Stack

In a stack, all operations take place at the "top" of the stack. The "push" operation adds an item to the top of the stack. The "pop" operation removes the item on the top of the stack and returns it.



Before push(83)

After push(83)

✓ **What is the difference between $A[\text{current}++]$ and $A[++\text{current}]$ please Explain because they are totally different when implemented it in the stack?**

To understand the difference between above mentioned statements, you have to understand the difference between pre and post increment operator. Let me explain you with an example.

Suppose we have a variable x int x=10;

Now we will apply pre and post increment operator over it.

cout<<"x= "x++;

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Now in the above statement, I have used post increment operator with x. The result of the above statement will be 10. As in post increment operator, the value is used first and then the increment is applied. So firstly the value is printed and then it is incremented.

✓ What is the difference between A[current++] and A[++current] please Explain because they are totally different when implemented it in the stack?

To understand the difference between above mentioned statements, you have to understand the difference between pre and post increment operator. Let me explain you with an example.

Suppose we have a variable x int x=10;

Now we will apply pre and post increment operator over it. cout<<"x="<<"x++;

Now in the above statement, I have used post increment operator with x. The result of the above statement will be 10. As in post increment operator, the value is used first and then the increment is applied. So firstly the value is printed and then it is incremented.

```
int isEmpty(){return ( current == -1 );} // Will return true when stack is empty
int isFull(){ return ( current == size-1);}
```

In the constructor of the class stack, the member variable current is initialized to -1. When an element is inserted in the stack, current is incremented by 1. Now in isEmpty method we check whether the value of current is -1 or not. If the value is -1 the return statement will return true. If stack even has one element, the function will return false.

Stack can only hold elements equal to the size of stack. If the size is 10 then stack can only have 10 elements in it. As we are incrementing the value of current during insertion, so when the value of current reaches the value of size, it means that now there is no more capacity in the stack. isFull method compare the value of current with the value of size. If both are equal it returns true otherwise false.

```
int pop(){ return A[current--];} // The pop function
void push(int x){A[++current] = x;}
will pop and push?
```

During the push function, if the stack is not full yet than the value is inserted in the stack. Before the value is inserted the value of current is incremented first because its value is one less than the current empty space. As stack follow LIFO method to pop the value. So the pop will return the value at the index current.

when we write the method: int *y = new int[10]

Does the new method in this command returns the address of the 1st location to a pointer *y? int *y = new int[10] line is reserving the memory for 10 integer elements and giving address of first element to integer point y.

Let us consider that you declare an array size 3 to store 3 elements of stack in the following way int a[2];

In the above case, size 3 refers to the numbers of elements of stack in the array. But the array starts from index 0, so 2 will be the last index of array which is one less than size. So that's why we use

```
int isFull()
{ return ( current==size-1); }
```

✓ What is Use of This pointer?

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

this pointer is used to access data member and function of current object. It is not necessary to use this pointer. Compiler uses it implicitly. Here we are using this pointer to differentiate between local variable Object and Node class data member Object. When we want to access any data member or function using pointer then we use arrow sign. When we want to access data member or function using object of class then we use dot sign.

✓ what is the difference between stack, array and linked list?

Array:

It stores a collection of individual values that are of the same data type.

Arrays allow random access to its elements and thus the complexity is order of $O(1)$. Arrays do not need an extra storage to point to next data item. Each element can be accessed via indexes.

The size of array is restricted to declaration.

Insertion/Deletion of values in arrays is very expensive. It requires memory re-allocation.

Linked list:

In linked list, we have the objects of Node class. Each node has two parts; data and next pointer. The next pointer points to the next node in the list. A chain of these nodes forms a linked list.

Linked list allows only sequential access to elements. Thus the complexity is order of $O(n)$.

Linked list requires an extra storage for references. This makes them impractical for lists of small data items such as characters or Boolean values.

The size of Linked list is dynamic by nature.

Elements can be easily inserted or deleted from the linked list.

Stack:

Stack is known as LIFO (Last In First Out) structure.

A stack is a list with the restriction that insertion and deletion can be performed in only one position, namely the end of the list called the top. The fundamental operations on a

stack are push, which is equivalent to an insert, and pop, which deletes the most recently inserted element. The most recently inserted element can be examined prior to performing a pop by use of the top method.

✓ what is the difference between Infix , Postfix and Prefix?

here are three ways to write an expression: Infix, Postfix and Prefix. Infix: the operator is in center. For example, $A + B$

Postfix: the operator comes after the operands. For example, $A B +$ Prefix: the operator comes before the operands. For example, $+ A B$

✓ What is the back function of lists?

back() function of the list will move the current pointer one item back, for example, if the current pointer points to item no 3 of the list and we call the back() function of this list, after this call the current pointer (means current item) will be item no. 2.

✓ What is a Precedence of Operators?

In programming languages, common operator notation is just one way of notating mathematical expressions as a linear sequence of tokens, or operators, but this is not the only way. The use of operator precedence classes

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

and associativities is just one way. However, it is not the most general way: this model cannot give an operator more precedence when competing with '-' than it can when competing with '+', while still giving '+' and '-' equivalent precedences and associativities.

✓ What are push(), pop(), top()?

The push(x) method will take an element and insert it at the top of the stack. This element will become top element. The pop() method will remove the top element of the stock and return it to the calling program. The top() method returns the top-most stack element but does not remove it from the stack. The interface method names that we choose has special objective. In case of list, we have used add, remove, get, set as the suitable names. However, for stack, we are using push, pop and top. We can depict the activity from the method name like push means that we are placing an element on the top of the stack and pushing the other elements down.

Precedence of Operators

There are five binary operators, called addition, subtraction, multiplication, division and exponentiation. We are aware of some other binary operators. For example, all relational operators are binary ones. There are some unary operators as well. These require only one operand e.g. - and +. There are rules or order of execution of operators in Mathematics called precedence. Firstly, the exponentiation operation is executed, followed by multiplication/division and at the end addition/subtraction is done. The order of precedence is (highest to lowest)

✓ Binary and Unary Operator?

Unary Operators has only one operand. The operation takes place using a single operand. The Unary Operators are ++, --, &, *(indirection), -(positive), -(neg... etc.

Unary operators have precedence over binary operators

Binary operators ("bi" as in "two") have two operands. In "A*B" the * operator has two operands: A and B. In "!B" the "!" operator (meaning boolean NOT) has only one operand, and is therefore a unary operator. The "-" and "+" operators can be both binary and unary, in "-4" or "+4" it denotes a negative or a positive number, in "0-4" it acts as the subtraction operator.

✓ INFIX, POSTFIX, PREFIX?

A major application that illustrates the different types of stacks and the various operations and functions defined upon them.

A + B à Infix

+ AB à prefix AB+ à postfix

A + (B * C) A + (BC*) A(BC*)+ ABC * +

The following is the order of precedence (highest to lowest)

Exponentiation

Multiplication / division Addition / subtraction

Infix

A+B

A + B - C

(A+B) * (C - D)

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

$A \$ B * C - D + E / F / (G + H)$

postfix

AB+

AB + C - AB+ CD- *

AB\$C*D-EF / GH + / +

Prefix

+AB

- + ABC

* + AB - CD

+ - * \$ ABCD // EF + GH

✓ Define Pop() method?

pop() is one of the methods of the stack class. It is used to retrieve elements from stack one by one. As stack follow LIFO method, this method is applied in pop function. The element which is inserted last during push method is retrieved first in pop method.

For example you have inserted 5 elements in the stack i.e. 2, 13, 8, 9, 5 (order is same, 2 is inserted first and 5 at last). Now we will call pop method once, 5 will be printed on screen. We again call pop method, 9 will be printed on screen and so on.

✓ What is the benefit writing implementation separately, if we can write it in same program.?

Public member functions exposed by a class are called interface. Purpose of writing interface in a different file with .h extension and writing implementation in different file with .cpp extension is to separate interface and implementation details of your program. Separation of implementation from the interface is good software engineering.

Separating interface and implementation allows you to change the implementation independently of the interface. This helps to deal with changing requirements.

For example, let us consider that one year ago, you developed an application by separating interface and implementation. You developed that application in which data processing capability was not so fast. But with the passage of time you come to know that you should change the implementation of your application so that data processing capability may become fast. For this purpose you replaced old implementation with the some new better and faster implementation without having to change the interface of the system.

User is only concerned about ways of accessing data (interface)

User has no concern about the internal representation and implementation of the class

✓ What Is Constant time?

"Constant time" means that no matter what the input is, the program will not run longer than a known amount of time.

OR

"Constant time" means that the operation will execute in an amount of time independent of the input size

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

For example, when we say that insertion in a linked list is a constant time operation, then it means that process of inserting a node in a linked list will take a particular amount of time i.e. 5 milliseconds. And if we change the value to be inserted then the time will not exceed more than 5 milliseconds.

What is Traversing?

Traversing means visiting each and every element in a data structure, at least once. Traversing is used in searching and printing elements from a data structure.

What is the postfix expression.

Postfix expression is in the form of ABO where, A and B are numbers or also postfix expression and O is operator (+, -, *, /). e.g. AB+.

The reason to convert infix to postfix expression is that postfix expression evaluation is easier by using a stack. In postfix expression evaluation, maintain a stack and scan the postfix expression from left to right. If the element is a number, push it into the stack and if the element is an operator O, pop twice and get A and B respectively. Calculate BOA and push it back to the stack. When the expression is ended, the number in the stack is the final answer.

For example, for given $3\ 6\ 9\ * +\ 3\ -$ expression, start scanning from left. Follow the following given steps;

- i. First, push(3) into the Stack
- ii. Then, push(6) into the Stack
- iii. Push(9) into the stack
- iv. Now we see an operator *, that means we can get a new number by calculation
- v. Pop the first two numbers and perform calculations on these numbers i.e. $6*9=54$
- vi. Push the new number back into the stack
- vii. Then we see the next operator +, pop the top two values and perform the calculation $3+54=57$
- viii. Push the new number back
- ix. We see the next number 3, Push (3) into the stack
- x. Then we see the next operator -, pop the top two values and perform the calculation $57-3=54$

54 is the required result.

✓ What are Infix notation and Postfix notation ?

Infix notation:

Operators are written in-between their operands. This is the usual way we write expressions. An expression such as $A * (B + C) / D$ is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer."

Postfix notation:

Operators are written after their operands. The infix expression given above is equivalent to $A\ B\ C\ +\ *\ D\ /$

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Examples:

Infix	Postfix	Notes
$A * B + C / D$	$A B * C D / +$	multiply A and B, divide C by D, add the results
$A * (B + C) / D$	$A B C + * D /$	add B and C, multiply by A, divide by D
$A * (B + C / D)$	$A B C D / + *$	divide C by D, add B, multiply by A

✓

✓ What is the precedence rule before evaluating the expression?

The arithmetic operators in an expression are evaluated according to their precedence. The precedence means which operator will be evaluated first and which will be evaluated after that and so on. In an expression, the parentheses () are used to force the evaluation order. The operators in the parentheses () are evaluated first. If there are nested parentheses then the inner most is evaluated first.

The expressions are always evaluated from left to right. The operators *, / and % have the highest precedence after parentheses. These operators are evaluated before + and – operators. Thus + and – operators has the lowest precedence. It means that if there are * and + operators in an expression then first the * will be

evaluated and then its result will be added to other operand. If there are * and / operators in an expression (both have the same precedence) then the operator which occurs first from left will be evaluated first and then the next, except you force any operator to evaluate by putting parentheses around it.

The following table explains the precedence of the arithmetic operators:

OPERATORS	OPERATIONS	PRECEDENCE (ORDER OF EVALUATION)
()	• Parentheses	Evaluated first
*, /, or %	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated from left to right
+ or -	Addition, Subtraction	Evaluated last. If there are several, they are evaluated from left to right

in the postfix form, parentheses are not used. Consider the infix expressions as '4+3*5' and '(4+3)*5'. The parentheses are not needed in the first but are necessary in the second expression. The position of operators and operands in the expression makes it clear in which order we have to do the multiplication and addition.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

✓ How can we evaluate it?

Each operator in a postfix expression refers to the previous two operands. As the operators are binary (we are not talking

about unary operators here), so two operands are needed for each operator. The nature of these operators is not affected in the postfix form i.e. the plus operator (+) will

apply on two operands. Each time we read an operand, we will push it on the stack.

We are going to evaluate the postfix expression with the help of stack. After reaching an operator, we pop the two operands from the top of the stack, apply the operator and push the result back on the stack Using Parentheses to Specify the Order of Operators

You can enclose search terms and their operators in parentheses to specify the order that they are interpreted. Information within parentheses is read first, then information outside parentheses is read next.

If there are nested parentheses, the search engine processes the innermost parenthetical expression first.

Nested Parentheses

If there are nested parentheses, the search engine processes the innermost parenthetical expression first, then the next, and so on until the entire query is interpreted.

Infix to postfix Conversion.

We have seen how to evaluate the postfix expressions while using the stack. How can we convert the infix expression into postfix form? Consider the example of a spreadsheet. We have to evaluate expressions. The users of this spreadsheet will employ the infix form of expressions. Consider the infix expressions 'A+B*C' and '(A+B)*C'. The postfix versions are 'ABC*+' and 'AB+C*' respectively. The order of operands in postfix is the same as that in the infix. In both the infix expressions, we have the order of operands as A, B and then C. In the postfix expressions too, the order is the same i.e. A, B, followed by C. The order of operands is not changed in postfix form. However, the order of operators may be changed. In the first expression 'A+B*C', the postfix expression is 'ABC*+'. In the postfix form multiplication comes before the plus operator. In scanning from left to right, the operand 'A' can be inserted into postfix expression.

✓ What is STL of data structures?

STL stands for Standard Template Library. It is a software library partially included in the C++ Standard Library. The STL provides a ready-made set of common classes for C++, such as containers and associative arrays, that can be used with any built-in type and with any user-defined type that supports some elementary operations (such as copying and assignment). STL algorithms are independent of containers, which significantly reduces the complexity of the library.

✓ what is fatal error ?

A fatal error is an error that causes a program to abort and may return the user to the operating system. Due to this error, data that the program was processing may be lost.

A fatal error usually occurs due to many reasons; some are given below:

User may have tried to access an illegal memory location

Invalid data or code has been accessed

A Program attempts to divide an integer by zero.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

✓ What do u mean by calling template function?

Usually function perform on the type of data for which it has been defined, for example, consider the following function for calculating the square of an integer number,

```
int squar(int sq)
```

```
{  
    return sq*sq;  
}
```

This function will work only for integer data, and we can only call it in main function like, int main()

```
{  
    int ivar = 10;  
    cout<< squar(ivar) <<endl; // Can be called only with int values/variable  
    getche(); return 0;  
}
```

However if we want to make this function as general, means it work for int, float etc as well, the only method is to make the template of this function, the template version of this function will be as,

```
template <class T> T squar(T sq)
```

```
{  
    return sq*sq;  
}
```

In main function it can be called with different type of data as, int main()

```
{  
    int ivar = 10; float fvar = 2.5;  
    cout<< squar(ivar) <<endl<< squar(fvar); // Calling the same function with different type of data  
  
    getche(); return 0;  
}
```

✓ What is the purpose of using(*)?

With reference to pointers, * can be used to declare a pointer, like char * aName; (aName is a pointer of type char)

Pointers are special type of variables in which a memory address is stored. They contain a memory address, not value of the variable.

✓ What is the Thing Queue?

A queue is a linear data structure into which items can only be inserted at one end and removed from the other. In contrast to the stack, which is a LIFO (Last In First Out) structure, a queue is a FIFO (First In First Out) structure.

The usage of queue in daily life is pretty common. For example, we queue up while depositing a utility bill or purchasing a ticket. The objective of that queue is to serve persons in their arrival order; the first coming

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

person is served first. The person, who comes first, stands at the start followed by the person coming after him and so on. At the serving side, the person who has joined the queue first is served first.

✓

✓ what are Templates?

Through templates we can create generic functions and classes. In a generic function or class, the type of data upon which the function or class operates is specified as a parameter.

Thus, you can use one function or class with several different types of data without having to explicitly re-code specific versions for each data type.

The syntax of the template class is:

```
template <class T> class class-name()
```

```
{
```

```
//definition of class
```

```
};
```

In the definition of the class where the generic data type is required, we write T. For example, there is a class in which we want to write int data type. The int is the data type that may be a float or double at different times. For this, T is written wherever we are using int in the class definition.

✓ What is the difference between stack and heap?

There are two places that stores items in memory as your code executes. These are Stack and the Heap. Both the stack and heap help us run our code. They reside in the operating memory on our machine and contain the pieces of information we need to make it all happen. Stack memory stores variable types in memory, these variables in programming are called local variables and are often stored for short amounts of time while a function/method block uses them to compute a task. Once a function/method has completed its cycle the reference to the variable in the stack is removed.

Heap is similar except that its purpose is to hold information (not keep track of execution most of the time) so anything in our Heap can be accessed at any time. With the Heap, there are no constraints as to what can be accessed like in the stack Heap memory stores all instances or attributes, constructors and methods of a class/object. A Heap reference is also stored in Stack memory until the life cycle of the object has completed. Inside the Heap reference all the contents of the object are stored whereas with a local variable only the variable contents are stored in the stack.

✓ What are the local, global variables in the function and the stack in the function?

These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program. As such, they are normally implemented using a stack. Local variables cease to exist once the function that created them is completed. They are recreated each time a function is executed or called.

Global:

global variables are declared outside the boundary of the function. These variables can be accessed by any function comprising the program. They are implemented by associating memory locations with variable names. They do not get recreated if the function is recalled.

stack in function:

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Stack is a data structure that stores information about the active subroutines of a computer program. This kind of stack is also known as an execution stack, control stack, function stack, or run-time stack, and is often shortened to just "the stack". Although maintenance of the call stack is important for the proper functioning of most software. Whenever we call a function then compiler will create a stack for the function and first of all it places the return address of the function where the control will go back after executing the function. After this it places the arguments of the function on the stack. After the execution of the function called the programs start its execution from the next line after the function call by taking return address from the stack.

✓ What is difference between stack and heap?

There are two places that store items in memory as your code executes. These are Stack and the Heap. Both the stack and heap help us run our code. They reside in the operating memory on our machine and contain the pieces of information we need to make it all happen. Stack memory stores variable types in memory, these variables in programming are called local variables and are often stored for short amounts of time while a function/method block uses them to compute a task. Once a function/method has completed its cycle the reference to the variable in the stack is removed.

Heap is similar except that its purpose is to hold information (not keep track of execution most of the time) so anything in our Heap can be accessed at any time. With the Heap, there are no constraints as to what can be accessed like in the stack. Heap memory stores all instances or attributes, constructors and methods of a class/object. A Heap reference is also stored in Stack memory until the life cycle of the object has completed. Inside the Heap reference all the contents of the object are stored whereas with a local variable only the variable contents are stored in the stack.

✓

✓ What is link list and how we use it in our daily life?

List is a sequential data structure, i.e. a collection of items accessible one after another. Beginning at the head and ending at the tail. It is a widely used data structure for Applications which do not need random access. It differs from the stack and queue data structures in that additions and removals can be made at any position in the list. Link list is a type of list data structure in which we implement the concept of dynamic memory. The Linked List is stored as a sequence of linked nodes. As in the case of the stack, each node in a linked list contains data AND a reference to the next node.

Stack Abstract Data Type

The stack abstract data type is defined by structure and operations. A stack is structured, "as an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO.

✓ What are Stack operators?

The stack operations are :

Stack() creates a new stack that is empty. It needs no parameters and returns an empty stack. push(item) adds a new item to the top of the stack. It needs the item and returns nothing.

pop() removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

peek() returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.

isEmpty() tests to see whether the stack is empty. It needs no parameters and returns a boolean value. size() returns the number of items on the stack. It needs no parameters and returns an integer.

✓ C++ Templates

§We need a stack of operands and a stack of operators.

§Operands can be integers and floating point numbers, even variables.

§Operators are single characters.

§We would have to create classes FloatStack and CharStack.

§Yet the internal workings of both classes is the same.

§We can use C++ Templates to create a “template” of a stack class.

§Instantiate float stack, char stack, or stack for any type of element we want.

✓

Function Call Stack

§Stacks play a key role in implementation of function calls in programming languages.

§In C++, for example, the “call stack” is used to pass function arguments and receive return values.

§The call stack is also used for “local variables”

✓ Why the return type of push() is void ?

The return type of push method is void because this method returns nothing. This method is used to insert a new element in the stack. If you want to return something(any value) from push method then you can change its return type according to the value returned. In case of stack and the functionality explained here in this course, there is no need of changing return type.

✓ What is pseudo code?

It is an outline of a program, written in a form that can easily be converted into real programming statements. It uses a combination of informal programming structures and verbal descriptions of code. Emphasis is placed on expressing the behavior or outcome of each portion of code rather than on strictly correct syntax.

✓

✓ Use of Queues??

§Out of the numerous uses of the queues, one of the most useful is simulation.

§A simulation program attempts to model a real-world phenomenon.

§Many popular video games are simulations, e.g., SimCity, Flight Simulator

§Each object and action in the simulation has a counterpart in real world.

§If the simulation is accurate, the result of the program should mirror the results of the real-world event.

§Thus it is possible to understand what occurs in the real-world without actually observing its occurrence.

§Let us look at an example. Suppose there is a bank with four tellers.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

✓ **AVL and Binary tree node insertion?**

After inserting an element into an AVL tree, if the tree becomes unbalanced then you should have to balance the tree by performing either single or double rotation on the tree. Whereas in BST, you don't balance the tree after inserting any data element in it.

✓ **Liner and Nonlinear data structure?**

A data structure is classified into two categories: Linear and Non-Linear data structures. A data structure is said to be linear if the elements form a sequence, for example Array, Linked list, queue etc. Elements in a nonlinear data structure do not form a sequence, for example AVL Tree, Binary tree, etc.

Ways of representation:

There are two ways of representing linear data structures in memory. One way is to have the linear relationship between the elements by means of sequential memory locations. Such linear structures are called arrays. The other way is to have the linear relationship between the elements represented by means of links. Such linear data structures are called linked list.

✓ **what is wrap around?**

In simple words, wrap-around means "connect the end to the start". Circular list is an example of wrap-around in which we connect end-of-list to the start-of-list.

✓ **How queue are effective in Data Structures?**

A queue is a linear data structure into which items can only be inserted at one end and removed from the other. In contrast to the stack, which is a LIFO (Last In First Out) structure, a queue is a FIFO (First In First Out) structure.

The usage of queue in daily life is pretty common. For example, we queue up while depositing a utility bill or purchasing a ticket. The objective of that queue is to serve persons in their arrival order; the first coming person is served first. The person, who comes first, stands at the start followed by the person coming after him and so on. At the serving side, the person who has joined the queue first is served first.

Out of the numerous uses of the queues, one of the most useful is simulation. A simulation program attempts to model a real-world phenomenon. Many popular video games are simulations, e.g., SimCity, Flight Simulator etc.

✓ **What is meant by "Parameter pass by value and pass by reference" in c++?**

Parameter passed by value:

When we call a function with parameter we pass the copies of their values. For example

Int x=2, y= 3, z; z= addition (x,y);

What we did that we pass the value of x and y (i.e. 2 and 3)to our desired function.

Parameter passed by reference:

In parameter passed by reference, the first thing is that in the declaration of our function the type of each parameter was followed by an ampersand sign (&). This ampersand is what specifies that their corresponding arguments are to be passed by reference instead of by value. When a variable is passed by reference we are not passing a copy of its value, but we are somehow passing the variable itself to the function and any

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

modification that we do to the local variables will have an effect in their counterpart variables passed as arguments in the call to the function.

Example

```
Void addition (int & a, int & b, int & c)
```

```
{  
a * = 1; b* = 2;  
c* = 3;  
}
```

Now when we call the function

```
int x=5, y=3, z=2; addition (x, y, z);
```

Here x, y, z will be passed by reference to our original addition function.

✓ What is a Dangling pointer?

A dangling pointer is a pointer which has been allocated but does not point to any entity. For example uninitialized pointer.

```
int * ptr;  
*ptr = 10;
```

The correct way of doing so is:

```
int * ptr; int a;  
ptr = &a;  
*ptr = 10;
```

Another example can be a function returning reference of a local variable. The local variables are destroyed when function call ends, so if you try to access that location which has been deleted, it may lead you to unpredictable results.

```
int * function()  
{  
int a = 10; return &a;  
}  
int* ptr = function();  
*ptr = 5;
```

✓

✓ What is Queue??

A queue is a linear data structure into which items can only be inserted at one end and removed from the other. In contrast to the stack, which is a LIFO (Last In First Out) structure, a queue is a FIFO (First In First Out) structure.

The usage of queue in daily life is pretty common. For example, we queue up while depositing a utility bill or purchasing a ticket. The objective of that queue is to serve persons in their arrival order; the first coming person is served first. The person, who comes first, stands at the start followed by the person coming after him and so on. At the serving side, the person who has joined the queue first is served first. If the requirement is to serve the people in some sort of priority order, there is a separate data structure that supports priorities.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

✓ WHAT is Priority Queue?

A queue in which the items are sorted so that the highest priority item is always the next one to be extracted.

✓

✓ What is Life critical systems?

Systems on which we depend for safety and which may result in death or injury if they fail: medical monitoring, industrial plant monitoring and control and aircraft control systems are examples of life critical systems.

✓ What is Real time systems?

Systems in which time is a constraint. A system which must respond to some event (eg the change in attitude of an aircraft caused by some atmospheric event like wind-shear) within a fixed time to maintain stability or continue correct operation (eg the aircraft systems must make the necessary adjustments to the control surfaces before the aircraft falls out of the sky!).

Queues

§A stack is LIFO (Last-In First Out) structure.

§In contrast, a queue is a FIFO (First-In First-Out) structure.

§A queue is a linear structure for which items can be only inserted at one end and removed at another end.

Queue Operations

Enqueue(X) – place X at the rear of the queue. Dequeue() -- remove the front element and return it. Front() -- return front element without removing it.

IsEmpty() -- return TRUE if queue is empty, FALSE otherwise

Implementing Queue

§Using linked List: Recall

§Insert works in constant time for either end of a linked list.

§Remove works in constant time only.

§Seems best that head of the linked list be the front of the queue so that all removes will be from the front.

§Inserts will be at the end of the list.

Queue using Array

§If we use an array to hold queue elements, both insertions and removal at the front (start) of the array are expensive.

§This is because we may have to shift up to “n” elements.

§For the stack, we needed only one end; for queue we need both.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

§To get around this, we will not shift upon removal of an element.

§We have inserts and removal running in constant time but we created a new problem.

§Cannot insert new elements even though there are two places available at the start of the array.

§Solution: allow the queue to “wrap around”.

✓ **What are Simulation Models?**

Simulation model is the representation of real world problem through computer software. It is very near to original model.

AVL and Binary tree node insertion?

After inserting an element into an AVL tree, if the tree becomes unbalanced then you should have to balance the tree by performing either single or double rotation on the tree. Whereas in BST, you don't balance the tree after inserting any data element in it.

✓ **what is the real data structure?**

Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently, by efficiency here means that it executes in smaller time, take less space while storing and take less time while retrieval. Data structure is not only concerned with C++ or other specific programming language but it plays a central role in modern computer Science. As computer applications are becoming complex, so there is need for more resources. This does not mean that we should buy a new computer to make the application execute faster. Our effort should be to ensure that the solution is achieved with the help of programming, data structures and algorithm. Data structure enables an efficient storage of data for an easy access. It enables to represent the inherent relationship of the data in the real world. It enables an efficient processing of data. It helps in data protection and management.

✓ **What is FIFO and LIFO queue?**

FIFO queue:

A queue in which the first item added is always the first one out.

LIFO queue:

A queue in which the item most recently added is always the first one out.

FIFO means first in first out and LIFO means last in first out.....

In case of FIFO the items that were added firstly, they will remove firstly to complete a transaction. While in LIFO the items that were added in the last will be removed firstly from the queue.

✓ **Event based Simulation**

§Don't wait for the clock to tic until the next event.

§Compute the time of next event and maintain a list of events in increasing order of time.

§Remove a event from the list in a loop and process it.

✓ **Strict Binary Tree and Complete Binary Tree**

Strict Binary Tree: If every non-leaf node has non-empty left and right subtrees.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Complete Binary Tree: If every non-leaf node has non-empty left and right subtrees and all leaf nodes are at the same level.

So, complete binary tree is a strict binary tree with the restriction that all leaf nodes must be at the same level.

✓ What is Binary Tree?

Binary Tree The mathematical definition of a binary tree is "A binary tree is a finite set of elements that is either empty or is partitioned into three disjoint subsets. The first subset contains a single element called the root of the tree. The other two subsets are themselves binary trees called the left and right sub-trees". Each element of a binary tree is called a node of the tree.

✓ Depth of tree Explain?

There are different levels in tree. Each level has different number of nodes. Root node lies at level 0, Children of root node lie at level 1, similarly nodes which lie at level 1 may also have different children's which lie at level 2 and so on.

In formula, d stands for depth of tree. The depth of a tree is the maximum level of any node in the tree

✓

✓ Implementation of Priority Queues?

There are a variety of simple, usually inefficient, ways to implement a priority queue. They provide an analogy to help one understand what a priority queue is. For instance, one can keep all the elements in an unsorted list. Whenever the highest-priority element is requested, search through all elements for the one with the highest priority. (In big O notation: $O(1)$ insertion time, $O(n)$ pull time due to search.)

✓ What is full binary tree/Strictly?

A full binary tree (sometimes proper binary tree or 2-tree or strictly binary tree) is a tree in which every node other than the leaves has two children.

so you have no leaves with only 1 child. Appears to be the same as strict binary tree. Here is an image of a full/strict binary tree

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

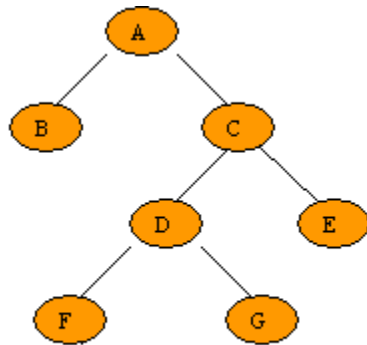
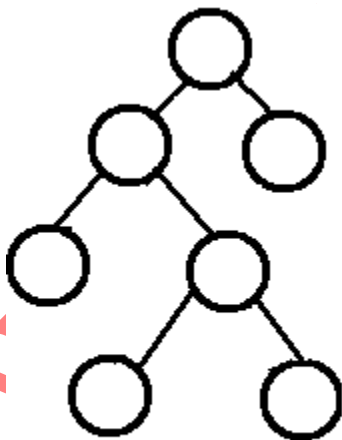


Fig .5.1.4 Strictly binary tree

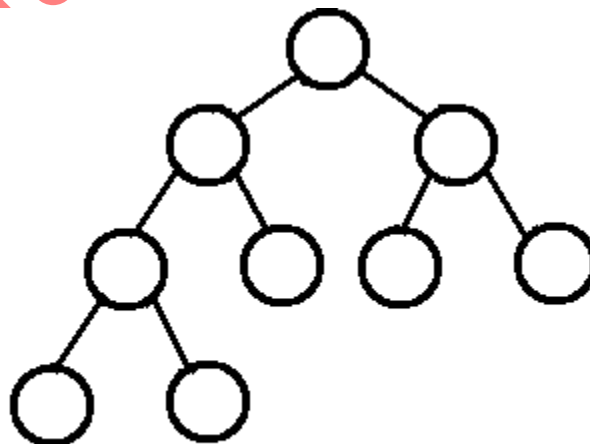
Full binary tree are a complete binary tree but reverse is not possible, and if the depth of the binary is n the no. of nodes in the full binary tree is $(2^n - 1)$. It is not necessary in the binary tree that it have two child but in the full binary it every node have no or two child.

✓ Complete Binary Tree?

A binary tree T with n levels is complete if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.



full tree



complete tree

A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

A complete binary tree can be used to represent a heap. It can be easily represented in contiguous memory with no gaps (i.e. all array elements are used save for any space that may exist at the end)

✓ What is Level?

Level

The level of a node in a binary tree is defined as follows:

Root has level 0,

Level of any other node is one more than the level its parent (father).

The depth of a binary tree is the maximum level of any leaf in the tree.

✓ What are Trace of Insert?

"Trace of Insert" means that we insert a new node into a tree and will observe step-by-step how it reaches to its proper location in the tree.

For example, in handouts, 17 is inserted into the tree.

First of all, we will compare it with the root (14) of the tree.

As 17 is greater than the root(14), so it will be moved to the right sub-tree. Then, 17 will be compared with 15, still greater, move to the right sub-tree. Then, 17 will be compared with 18, now will move to the left sub-tree. Then, 17 will be compared with 16, it will be moved to the right sub-tree. There is no node on the right of 16, so 17 will be inserted here.

This was the trace of inserting 17 into the tree.

Level of a Complete Binary Tree

We can find the depth of a complete binary tree if we know the total number of nodes. If we have a complete binary tree with n total nodes then by the equation to total number of nodes we can write..

$$\text{Total number of nodes} = 2^{d+1} - 1 = n$$

To find the value of d , we solve the above equation as under

$$2^{d+1} - 1 = n$$

$$2^{d+1} = n + 1$$

$$d + 1 = \log_2 (n + 1) \quad d = \log_2 (n + 1) - 1$$

After having n total nodes, we can find the depth d of the tree by the above equation. Suppose we have 100,000 nodes. It means that the value of n is 100,000, reflecting a depth i.e. d of the tree will be $\log_2 (100000 + 1) - 1$, which evaluates to 20. So the tree will be 20. In other words, the tree will be 20 levels deep.

✓ What is Binary Heap ?

The binary heap is a data structure that can efficiently support the basic priority-queue operations. In a binary heap, the items are stored in an array such that each key is guaranteed to be larger than (or equal to) the keys at two other specific positions.

In turn, each of those keys must be larger than two more keys, and so forth. This ordering is easy to see if we view the keys as being in a binary tree structure with edges from each key to the two keys known to be smaller.

Definition. A binary heap is a set of nodes with keys arranged in a complete heap-ordered binary tree, represented in level order in an array (not using the first entry).

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

A binary tree is heap-ordered if the key in each node is larger than (or equal to) the keys in that nodes two children (if any)

Uses and Properties of Binary tree ? Uses for Binary Trees...

Use for storing and retrieving information

Insert, delete, and search faster than with a linked list

Take advantage of logn height

Idea: Store information in an ordered way (keys)

A Property of Binary Search Trees

The key of the root is larger than any key in the left subtree

The key of the root is smaller than any key in the right subtree

Note: Duplicated keys are not allowed

✓

✓ How to insert some new numbers in the tree?

Binary tree is a tree like (means hierarchical) data structure in which each node has at most two child nodes, these child nodes are called left child (or left subtree) and right child (or right subtree), while binary search tree (BST) is a special binary tree with the following properties,

The left child (left subtree) of a node contains only those nodes which have values less than the value of this node.

The right child (right subtree) of a node contains only those nodes which have values greater than the value of this node.

Both the left and right subtrees must also be binary search trees.

So inorder to insert elements in binary search tree, you should have to keep in mind the properties of BST that has been discussed above. Now let us consider a string

14, 15, 4

Now 14 i.e. first elements will be root node of BST. As 15 is greater than 14 so 15 will become right child of 14. Similarly as 4 is less than 14 so 4 will become left child of node 14.

✓ What is Strictly Binary tree?

A strictly binary tree is a tree in which every node other than the leaves has two children. Or, perhaps more clearly, every node in a binary tree has exactly (strictly) 0 or 2 children.

What is the purpose of traversing a Tree data data structure?

Accessing and processing each data item of a data structure is called traversing.

Data is stored in computers using data structures in order to perform different operations on the data, for which each data item is needed to access and process.

✓

What is Binary Tree Traversal?

Many binary tree operations are done by performing a traversal of the binary tree .In a traversal, each element of the binary tree is visited exactly once . During the visit of an element, all action (make a clone, display, evaluate the

operator, etc.) with respect to this element is taken.

Binary Tree - Traversal

Traversal of a binary tree is defined recursively. For example,

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Visit the node (root or subroot)

Traverse the left subtree

Traverse the right subtree

Traversal Applications

§ Make a clone

§ Determine height

§ Determine number of nodes

Uses for Binary Trees...

-- Binary Expression Trees

Binary trees are a good way to express arithmetic expressions.

The leaves are operands and the other nodes are operators.

The left and right subtrees of an operator node represent sub expressions that must be evaluated before applying the operator at the root of the subtree.

✓ What are methods of traversing a tree?

Depth-first traversal: This strategy consists of searching deeper in the tree whenever possible. Certain depth-first traversals occurs frequently enough that they are given names of their own

Breadth-first traversal: This is a very simple idea which consists of visiting the nodes based on their level in the tree. It visits all nodes with depth 0, then depth 1, then depth 2, and so on

Special cases of depth-first traversal for binary trees

In binary trees there are three basic ways to traverse a tree using the a depth-first search idea

– Preorder: We visit a node, then visit the left and the right subtrees void preOrder(BinaryTreeNode t)

```
{  
if (t != null)  
{  
visit(t); preOrder(t.leftChild); preOrder(t.rightChild);  
}  
}
```

– Inorder: We visit the left subtree then we visit the node, then we visit the right subtree void

inOrder(BinaryTreeNode t)

```
{  
if (t != null)  
{  
inOrder(t.leftChild); visit(t); inOrder(t.rightChild);  
}  
}
```

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

– Postorder: We visit the left and right subtree and then we visit the node. This is what normally authors mean if they mention just depth -first traversal

```
void postOrder(BinaryTreeNode t)
{
if (t != null)
{
postOrder(t.leftChild); postOrder(t.rightChild); visit(t);
}
}
```

✓ How can we determine, whether data structure is recursive or not?

Recursive data structures are used for recursive problems, where Recursive problems are those problems in which the solution depends on the solution to smaller instances (occurrences) of the same problem.

For example, we need to find the sum of first 100 numbers (i.e. sum of 1-100). Now,

Sum of 1-100 = 1+ (Sum of 2-100)

Sum of 2-100 = 2+ (Sum of 3-100)

Sum of 3-100 = 3+ (Sum of 4-100)

Sum of 4-100 = 4+ (Sum of 5-100)

The (Sum of 2-100), (Sum of 3-100), (Sum of 4-100) etc are the smaller instances of the main problem.

When a data structure solves a problem which is recursive in nature, then the data structure may be called recursive data structure.

Preorder Traversal (cont.)

Visit the root of the tree first, then visit the nodes in the left subtree, then visit the nodes in the right subtree

Preorder(tree)

If tree is not NULL { Visit Info(tree) Preorder(Left(tree)) Pre Order(Right(tree))

}

Inorder Traversal (cont.)

Visit the nodes in the left subtree, then visit the root of the tree, then visit the nodes in the right subtree

Inorder(tree)

If tree is not NULL { Inorder(Left(tree)) Visit Info(tree) Inorder(Right(tree))

}

Postorder Traversal

Visit the nodes in the left subtree first, then visit the nodes in the right subtree, then visit the root of the tree

Postorder(tree)

If tree is not NULL { Postorder(Left(tree)) Postorder(Right(tree)) Visit Info(tree)

}

✓ What is Breadth First traversal of a tree?

Breadth-first traversal of a tree

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

A breadth-first traversal consists of visiting the nodes based on their level in the tree. It visits all nodes with level depth 0, then depth 1, then depth 2, and so on. Use a queue to implement breadth first traversal

Deleting a Node from the BST

we may also require to delete some data (nodes) from a binary tree. Consider the case where we used binary tree to implement the telephone directory, when a person leaves a city, its telephone number from the directory is deleted.

It is common with many data structures that the hardest operation is deletion. Once we have found the node to be deleted, we need to consider several possibilities.

For case 1, If the node is a leaf, it can be deleted quite easily.

See the tree figure below.

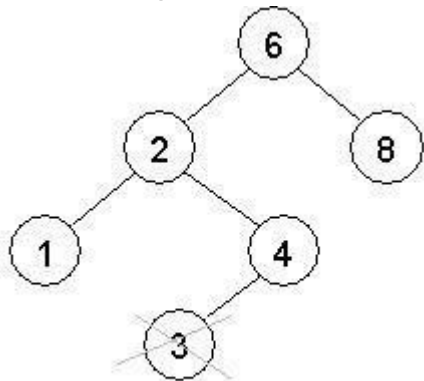


Fig 15.9: BST

✓ What is Cost of Search?

Cost of searching an element in a data structure means the time it takes to search an element or numbers of comparisons performed to search an element in a particular data structure.

✓ What is Binary Search Tree?

Binary tree is a tree like (means hierarchical) data structure in which each node has at most two child nodes, these child nodes are called left child (or left subtree) and right child (or right subtree), while binary search tree (BST) is a special binary tree with the following properties,

The left child (left subtree) of a node contains only those nodes which have values less than the value of this node.

The right child (right subtree) of a node contains only those nodes which have values greater than the value of this node.

✓ What is recursive Call?

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Both the left and right subtrees must also be binary search trees.

When a function calls itself then it is called recursion or recursive call.

✓ What is Pre order?

Pre-order traversal carried out by using recursion technique is called pre-order recursion

✓ What is the purpose of level order traversal of binary tree?

It has observed that the other traversal method that is, preorder, inorder and postorder use the recursive method while the level order traversal use a non-recursive method.

Recursive solutions may involve extensive overhead because they use function calls.

When a function call is made, it takes time to build a stack frame and push it into the stack. Conversely, when a return is executed, the stack frame must be popped from the stack and the local variables reset to their previous values.

Further, this takes time. A recursive procedure therefore generally runs slower than non recursive procedure.

In addition, each time when we make a function call we use some of our memory allocation. If the recursion is deep, that is, if there are many recursive calls, then we may run out of memory.

Void remove (const int&x);

is a user define function which take a constant integer reference as an argument.

void insert(const EType& x);

is a user define function which take a constant Template reference as an argument.

void remove(const EType& x);

is a user define function which take a constant Template reference as an argument.

Explain

✓ const EType& find(const EType & x) const;?

This method will takes an argument x of EType. It will search the tree whether x exists in the tree or not. Then we have isEmpty method that will ascertain if the tree is empty or not. There is also the printInorder method, which will print the tree in inorder traversal. If the tree has integers, we have sorted integers as a result of inorder traversal. Next thing we have is the insert (const EType& method, which inserts the x as a new node in the tree. After this, there is the remove method i.e. delete method. We have renamed delete as remove because delete is a keyword of C++. We can also name it as deleteNode or some other name which you think is suitable. The interface of the class is almost complete. Now think as a user of BinarySearchTree and decide if you need more methods. With the help of methods defined above, we can do a lot of work on BinarySearchTree. If you feel to add more methods due to the usage need, these can be added later.

✓ Q1. Why we should avoid by using the references of transient object?

Storing of references of transient objects can create problems because the transient object (object created on stack) is destroyed when the function execution finishes. The question arises, what can we do, if we do not

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

want the objects created in a function to be destroyed. The answer to this is dynamic memory allocation. All the variables or objects created in a function that we want to access later are created on memory heap (sometimes called free store) using the dynamic memory allocation functions or operators like new. Heap is an area in computer memory that is allocated dynamically. You should remember that all the objects created using new operator have to be explicitly destroyed using the delete operator.

So now, we understand that we cannot pass references to transient objects. If we want to use the objects later we create them on heap and keep the address

✓ How can we ensure the provision of a terminating condition in recursive calls ?

It is necessary to provide a terminating condition for the recursion in a recursive function, For example, consider the inorder method of binary tree,

```
void inorder(TreeNode<char> * treeNode)
{
    if( treeNode != NULL ) // Terminating condition for recursion.
    {
        inorder(treeNode->getLeft()); cout (treeNode->getInfo()) " "; inorder(treeNode->getRight());
    }
}
```

Similarly, there is always a condition for the termination of recursion depending upon the problem.

List::list()

✓ Can we only use this method in the case of template?

Scope resolution operator i.e. '::' has no relation with templates. You can use Scope resolution operator i.e. '::' either you use templates or you don't use templates. Scope resolution operator(::) is used to define a function outside a class or when we want to use a global variable but also has a local variable with same name.

C++ programming code #include <iostream> using namespace std;

```
char c = 'a'; // global variable
int main() {
    char c = 'b'; //local variable
    cout << "Local c: " << c << "\n";
    cout << "Global c: " << ::c << "\n"; //using scope resolution operator
    return 0;
}
```

Scope resolution operator in class

```
#include <iostream> using namespace std;
```

```
class programming { public:
```

```
void output(); //function declaration
```

```
};
```

```
// function definition outside the class
```

```
void programming::output() {
```

```
    cout << "Function defined outside the class.\n";
```

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

```
}  
int main() { programming x; x.output();  
return 0;  
}
```

If make a class for Node and have a method as void set(int object); what does the int object here means?

In method

```
void set(int object);
```

A variable of type int is passed by value. Here object is simply a variable name of type int. You can also use any other variable name instead of object like data e.g.

```
void set(int data);
```

✓ Use of reference Pointer?

In case of pass by value, when a variable/argument will be passed from calling function to a called function then a new copy of variable will be created of this passed argument/variable and in called function, changes will be done on new variable instead of carrying out changes in original variable of calling function. Where as in case of pass by reference, no new variable will be created and changes that will be carried out by called function in passed argument/variable will actually be the changes which will be carried out in original variable of calling function. So by using call reference we saved our self from copying of an argument/variable/object, that is why we passed by reference instead of pass by value.

✓ why we use reference variable in a program?

Passing data to functions through reference variables is used to avoid putting data on the stack during function call.

The reference variables are mostly used with user define data type (objects), as the size of an object may be enough large and may cause memory problem (stack problem) during the function call if it is pass by value, therefore objects are usually pass by reference in which the object copy does not store on stack (memory), and the calling function access the object within the caller function directly.

✓ Ampersand? and its use&?

&(ampersand) operator is used to get the address of any variable. The address of a variable can be obtained by preceding the name of a variable with an ampersand sign.

```
int a; int *b; b=a;
```

After these statement b will contain the address of a.

✓ What is Const explain?

When "const" keyword is used before a variable declaration in the parameter list of a function, it means, the function can use the variable within its body but cannot changes/update its value.

When "const" keyword is used before the return type of a function, it means, the function will return constant value.

When "const" keyword is used after a function name in a class, it means, this member function cannot change/update the object status (that is, cannot change the data members).

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

✓ What are ETYPE and findMin()?

“EType” is a class (Template class) name while findMin() is a function which finds the minimum number (node) in a binary search tree (BST).

What is a reference variable?

In fact reference variable is a synonym (second name) for a pre define variable. Consider the following program,

```
#include <iostream.h> #include <conio.h>
```

```
main()
```

```
{
```

```
int var = 10;
```

```
int &refvar = var; // refvar is a reference variable and is a synonym for variable var; coutvar;
```

```
refvar = refvar + 20; // "refvar" variable used for var
```

```
cout endl var; // value of "var" has changed by "refvar" variable.
```

```
getche(); return 0;
```

```
}
```

✓ How we can find left and right node in traversal process?

The left, right and root node can be identified if we know by which traversal these nodes are taken from the tree. Suppose tree is traversed by inorder traversal, then in the first case 4 is left child, 5 is root and 6 is right child. And if tree is traversed by postorder traversal then 4 is left child, 5 is right child and 6 is the root node.

✓ In How many Possible ways a tree can be transverse?

1: (4, 14, 15)

2: (14, 4, 15)

3: (15, 4, 14)

4: (4, 15, 14)

5: (14, 15, 4)

6: (15, 14, 4)

When we apply permutation then these 6 are the possible ways in which the above mentioned tree can be traversed. Root, left and right child will remain the same only the sequence of their traversal from the tree will be change.

✓ Difference between BST and AVL tree?

Binary tree is a tree like (means hierarchical) data structure in which each node has at most two child nodes, these child nodes are called left child (or left subtree) and right child (or right subtree), while binary search tree (BST) is a special binary tree with the following properties,

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

The left child (left subtree) of a node contains only those nodes which have values less than the value of this node. The right child (right subtree) of a node contains only those nodes which have values greater than the value of this node.

Both the left and right subtrees must also be binary search trees.

AVL tree is just like binary search tree. But difference between BST and AVL tree is that, during insertion of nodes, AVL tree re-balances itself when the balance factor of any node becomes more than -1 or 0 or 1. AVL tree re-balances itself by applying either single or double rotation on unbalanced node in such away that balance factor again becomes either -1 or 0 or 1.

What is balance AVL Tree?

The balance of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

On the basis of this definition, a Balanced AVL tree will be a an AVL tree whose each node has a balance of either 1, 0 or -1, while the balance of an empty tree is -1.

✓ .How can we evaluate $\log_2(1000000)=20$?

The $\log_2(1000000)$ is 19.93 which is about 20.

You can calculate it by yourself by using excel sheet to find the log base 2 of any value. Formula or method to find out log base 2 of any value by using excel sheet is given below.

=Log(value, base)

Let us suppose that you want to find log base 2 of value 1000000. For this purpose, you will put the value in above formula in the following way

=Log(1000000,2)

In order to calculate log base 2 of value 1000000, you will write =Log(1000000,2) in any cell of excel sheet and then you will press enter button of your keyboard to find out the log base 2 of value 1000000. Similarly you can use the formula =Log(value, base) to find out log of any value for any base by using excel sheet. E.g. Let us suppose that you want to find log base 10 of value 15. In order to calculate log base 10 of value 15, you will write =Log(15,10) in any cell of excel sheet and then you will press enter button of your keyboard to find out the log base 10 of value 15.

✓ what is Degenerate BST?

When we use sorted data to generate binary search tree then shape of binary search tree will become like linked list. This is known as degeneration of binary search tree.

As we have observed that when a BST is made from sorted data then the degeneration problem occur, means the cost of searches increases, so in order to minimize this, we balance the tree i.e. we use AVL tree In order to avoid degeneration of binary search tree in case of sorted data.

✓ What does this method do Event* remove()?

Method Event* remove() will return a pointer of type Event where Event is a user defined data type.

Event* e=node[0];

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Statement `Event* e=node[0];` is incorrect and it will give an error. It is because in order to save the first address of array in pointer of type `Event`, you should use `new` operator for dynamic memory allocation in the following way

`Event* e=new Event[10]`

Or in case of primitive data type like `int`, you should write the statement in the following way `int* e= new int[10];`

✓ Explain the difference of a binary tree and binary search tree?

Binary tree is a tree in which a node can only have two children at most. The order of insertion of nodes in the tree does not matter in case of binary tree.

Binary search tree is a type of binary tree. It is like binary tree except, the left child contains only nodes with values less than the parent node and the right child only contains nodes with values greater than the parent.

✓ AVL and Shallow Tree?

An AVL tree is said to be balanced, if each node in it has the balance 1, 0, -1. Balance of each node is determined by the difference of its left and right sub trees.

Shallow tree is a tree in which any node either do not have right or left child. The height of the left and right sub trees should be same to not be shallow tree.

Cases of Rotation

The single rotation does not seem to restore the balance. We will re-visit the tree and rotations to identify the problem area. We will call the node that is to be rotated as a (node requires to be re-balanced). Since any node has at the most two children, and a height imbalance requires that a's two sub-trees differ by two (or -2), the violation will occur in four cases:

1. An insertion into left subtree of the left child of a .
2. An insertion into right subtree of the left child of a .
3. An insertion into left subtree of the right child of a .
4. An insertion into right subtree of the right child of a .

The insertion occurs on the outside (i.e., left-left or right-right) in cases 1 and 4. Single rotation can fix the balance in cases 1 and 4.

Insertion occurs on the inside in cases 2 and 3 which a single rotation cannot fix.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

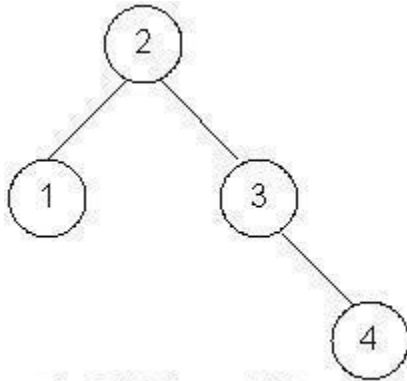


Fig 21.3: insert(4)

Once we insert a node in the tree, it is necessary to check its balance to see whether it is within AVL defined balance. If it is not so, then we have to rotate a node. The balance factor of the node containing number 4 is zero due to the absence of any left or right subtrees. Now, we see the balance factor of the node containing number 3. As it has no left child, but only right subtree, the balance factor is -1 . The balance factor of the node containing number 1 is 0. For the node containing number 2, the height of the left subtree is 1 while that of the right subtree is 2. Therefore, the balance factor of the node containing number 2 is $1 - 2 = -1$. So every node in the tree in fig. 21.3 has balance factor either 1 or less than that. You must be remembering that the condition for a tree to be an AVL tree, every node's balance needs not to be zero necessarily. Rather, the tree will be called AVL tree, if

the balance factor of each node in a tree is 0, 1 or -1 . By the way, if the balance factor of each node is 0, it will be a perfectly balanced tree.

✓ **way, if the balance factor of each node**

✓ **What is the depth of a binary tree and level of a binary tree?**

Depth of a binary tree means the number of edges along the path from the root node to the deepest leaf node.

level of a binary tree:

There are different levels in tree. Each level has different number of nodes. Root node lies at level 0, child of root nodes lie at level 1, similarly nodes which lie at level 1 may also have different children which lie at level 2 and so on.

How perform the Right-left double rotation to fix case 3??

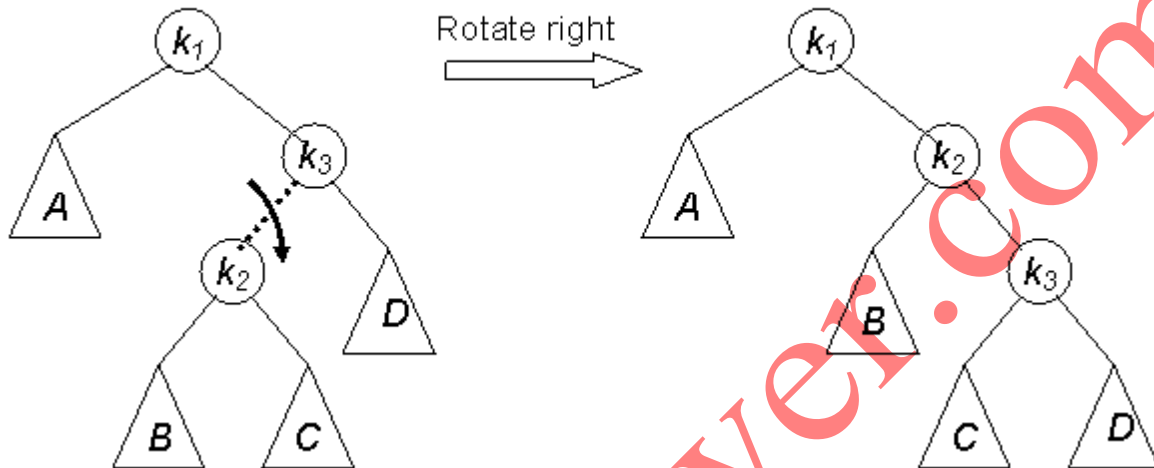
In case, the node is inserted in left sub tree of the right child, we encounter the same situation as discussed above. But here, we will perform right rotation at first before going for a left rotation.

Let's discuss this symmetric case and see how we can apply double rotation here. First we perform the right rotation.

Short Notes of CS301 By Muhammad Usman (IT OF VU)

We provide all categories of final year project (CS619) and short notes

Right-left double rotation to fix case 3.



Here k_1 is the root of the tree while k_3 is the right child of the k_1 . k_2 is the inner child. It is the Y tree expanded again here and the new node will be inserted in the k_2 's right subtree C or left subtree B. As we have to transform the k_2 into the root of the tree, so the right rotation between the link k_2 and k_3 will be carried out. As a result of this rotation, k_2 will come up and k_3 will go down. The subtree B has gone up with the k_2 while subtree C is now attached with the k_3 . To make the k_2 root of the tree, we will perform the left rotation between then k_1 and k_2 .

At the right side, we have the final shape of the tree. You can see that k_2 has become the root of the tree. k_1 and k_3 are its left and right children respectively. While performing the inorder traversal, you will see that we have preserved our inorder traversal.

•

Concepts of Single rotation and Double rotation?

Single rotation can restore the balance of the tree if the new inserted node has inserted as per the following conditions,

1. An insertion into left subtree of the left child of α .
2. An insertion into right subtree of the right child of α .

Double rotations can restore the balance of the tree if the new inserted node has inserted as per the following conditions,

1. An insertion into right subtree of the left child of α .
2. An insertion into left subtree of the right child of α .

Short Notes of CS301 By Muhammad Usman (IT OF VU)

α is a node that requires to be re-balanced. In order to know that which node should rotate left or right in balancing the AVL tree, you need to study the topic "Cases of Rotation" in lecture no. 21 of handouts. This topic has already been discussed in very deep detail by using graphical trees that which node (or sub tree) should go left or right during rotations.

✓ what r the conditions of AVL tree?

A tree is said to be an AVL tree, if it fulfills the AVL condition. An AVL tree is said to be balanced, if each node in it has the balance 1, 0, -1. Balance of each node is determined by the difference of its left and right sub trees. To check whether a given tree is AVL or not, check the balance of each and every node of the tree. If the balance of each node in the tree is 1,0 or -1, it is said AVL tree.

-



www.vuanswer.com



CS301- Data Structures

JAN 30,2011

LATEST SOLVED MCQS FROM FINALTERM PAPERS

MC100401285

Moaaz.pk@gmail.com

MC100401285@gmail.com

PSMD01

FINALTERM EXAMINATION Spring 2010

Question No: 1 (Marks: 1) - Please choose one

A solution is said to be efficient if it solves the problem within its resource constraints i.e. hardware and time.

- ▶ **True (Page 4)**
- ▶ False

Question No: 2 (Marks: 1) - Please choose one

Which one of the following is known as "Last-In, First-Out" or LIFO Data Structure?

- ▶ Linked List
- ▶ **Stack (Page 54)**
- ▶ Queue
- ▶ Tree

Question No: 3 (Marks: 1) - Please choose one

What will be postfix expression of the following infix expression?

Infix Expression : $a+b*c-d$

- ▶ $ab+c*d-$
- ▶ **$abc*+d-$**
- ▶ $abc+*d-$
- ▶ $abcd+*-$

Question No: 4 (Marks: 1) - Please choose one

For compiler a postfix expression is easier to evaluate than infix expression?

- ▶ **True [Click here for detail](#)**
- ▶ False

Question No: 5 (Marks: 1) - Please choose one

Consider the following pseudo code

declare a stack of characters

```
while ( there are more characters in the word to read )  
{
```

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**


```

    read a character
    push the character on the stack
}
while ( the stack is not empty )
{
    pop a character off the stack
    write the character to the screen
}

```

What is written to the screen for the input "apples"?

- ▶ selpa
- ▶ **selppa**
- ▶ apples
- ▶ aaappppplleess

Question No: 6 (Marks: 1) - Please choose one

Consider the following function:

```

void test_a(int n)
{
    cout << n << " ";
    if (n>0)
        test_a(n-2);
}

```

What is printed by the call test_a(4)?

- ▶ **4 2**
- ▶ 0 2 4
- ▶ 0 2
- ▶ 2 4

Question No: 7 (Marks: 1) - Please choose one

If there are N external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ **N -1 (Page 304)**
- ▶ N+1
- ▶ N+2
- ▶ N

Question No: 8 (Marks: 1) - Please choose one

If there are N internal nodes in a binary tree then what will be the no. of external nodes in this binary tree?

- ▶ N -1
- ▶ N
- ▶ **N +1 (Page 303)**
- ▶ N +2

Question No: 9 (Marks: 1) - Please choose one

If we have 1000 sets each containing a single different person. Which of the following relation will be true on each set:

- ▶ **Reflexive (page 387)**
- ▶ Symmetric
- ▶ Transitive
- ▶ Associative

Question No: 10 (Marks: 1) - Please choose one

Which one of the following is NOT the property of equivalence relation:

- ▶ Reflexive
- ▶ Symmetric
- ▶ Transitive
- ▶ **Associative (page 387)**

Question No: 11 (Marks: 1) - Please choose one

A binary tree of N nodes has _____.

- ▶ $\log_{10} N$ levels
- ▶ **$\log_2 N$ levels (Page 212)**
- ▶ $N / 2$ levels
- ▶ $N \times 2$ levels

Question No: 12 (Marks: 1) - Please choose one

The easiest case of deleting a node from BST is the case in which the node to be deleted _____.

- ▶ **Is a leaf node (Page 173)**
- ▶ Has left subtree only
- ▶ Has right subtree only
- ▶ Has both left and right subtree

Question No: 13 (Marks: 1) - Please choose one

If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is _____.

- ▶ N
- ▶ N^2
- ▶ $N \log_2 N$
- ▶ **$\log_2 N$ (page 440)**

Question No: 14 (Marks: 1) - Please choose one

Merge sort and quick sort both fall into the same category of sorting algorithms. What is this category?

- ▶ $O(n \log n)$ sorts
- ▶ Interchange sort (not sure)
- ▶ Average time is quadratic
- ▶ **None of the given options. (Page 488)**

Question No: 15 (Marks: 1) - Please choose one

If one pointer of the node in a binary tree is NULL then it will be a/an _____ .

- ▶ **External node (Page 303)**
- ▶ Root node
- ▶ Inner node
- ▶ Leaf node

Question No: 16 (Marks: 1) - Please choose one

We convert the _____ pointers of binary to threads in threaded binary tree.

- ▶ Left
- ▶ Right
- ▶ **NULL (Page 312)**
- ▶ None of the given options

Question No: 17 (Marks: 1) - Please choose one

If the bottom level of a binary tree is NOT completely filled, depicts that the tree is NOT a

- ▶ Expression tree
- ▶ Threaded binary tree
- ▶ **complete Binary tree (Page 323)**
- ▶ Perfectly complete Binary tree

Question No: 18 (Marks: 1) - Please choose one

What is the best definition of a *collision* in a hash table?

- ▶ Two entries are identical except for their keys.
- ▶ Two entries with different data have the exact same key
- ▶ **Two entries with different keys have the same exact hash value. (page 464)**
- ▶ Two entries with the exact same key have different hash values.

Question No: 19 (Marks: 1) - Please choose one

Suppose that a selection sort of 100 items has completed 42 iterations of the main loop. How many items are now guaranteed to be in their final spot (never to be moved again)

- ▶ 21
- ▶ 41
- ▶ **42 [Click here for detail](#)**
- ▶ 43

Question No: 20 (Marks: 1) - Please choose on

Suppose you implement a Min heap (with the smallest element on top) in an array. Consider the different arrays below; determine the one that *cannot* possibly be a heap:

- ▶ 16, 18, 20, 22, 24, 28, 30
- ▶ 16, 20, 18, 24, 22, 30, 28
- ▶ 16, 24, 18, 28, 30, 20, 22
- ▶ **16, 24, 20, 30, 28, 18, 22 (page 334)**

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 21 (Marks: 1) - Please choose one

Do you see any problem in the code of nextInOrder below:

```
TreeNode * nextInorder(TreeNode * p)
```

```
{
    if(p->RTH == thread)
        return( p->R );
    else {
        p = p->R;
        while(p->LTH == child)
            p = p->R;
        return p;
    }
}
```

- ▶ The function has no problem and will fulfill the purpose successfully.
- ▶ The function cannot be compile as it has syntax error.
- ▶ The function has logical problem, therefore, it will not work properly.
- ▶ The function will be compiled but will throw runtime exception immediately after the control is transferred to this function.

Question No: 22 (Marks: 1) - Please choose one

Which of the following statement is correct about find(x) operation:

- ▶ A find(x) on element x is performed by returning exactly the same node that is found.
- ▶ **A find(x) on element x is performed by returning the root of the tree containing x. [Click here for detail](#)**
- ▶ A find(x) on element x is performed by returning the whole tree itself containing x.
- ▶ A find(x) on element x is performed by returning TRUE.

Question No: 23 (Marks: 1) - Please choose on

Which of the following statement is NOT correct about find operation:

- ▶ It is not a requirement that a find operation returns any specific name, just that finds on two elements return the same answer if and only if they are in the same set.
- ▶ **One idea might be to use a tree to represent each set, since each element in a tree has the same root, thus the root can be used to name the set.**
- ▶ Initially each set contains one element.
- ▶ Initially each set contains one element and it does not make sense to make a tree of one node only.

Question No: 24 (Marks: 1) - Please choose one

In complete binary tree the bottom level is filled from _____

- ▶ **Left to right (Page 323)**
- ▶ Right to left
- ▶ Not filled at all
- ▶ None of the given options

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 25 (Marks: 1) - Please choose one

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

The array after the FIRST iteration of the large loop in a selection sort (sorting from smallest to largest).

▶ 0 3 8 9 1 7 5 2 6 4 (Page 477)

▶ 2 6 4 0 3 8 9 1 7 5

▶ 2 6 4 9 1 7 0 3 8 5

▶ 0 3 8 2 6 4 9 1 7 5

Question No: 26 (Marks: 1) - Please choose one

What requirement is placed on an array, so that *binary search* may be used to locate an entry?

- ▶ The array elements must form a heap.
- ▶ The array must have at least 2 entries.
- ▶ **The array must be sorted.** [Click here for detail](#)
- ▶ The array's size must be a power of two

FINALTERM EXAMINATION

Spring 2010

Question No: 1 (Marks: 1) - Please choose one

Which one of the following operations returns top value of the stack?

- ▶ Push
- ▶ Pop
- ▶ **Top** (page 53)
- ▶ First

Question No: 2 (Marks: 1) - Please choose one

Compiler uses which one of the following in Function calls,

- ▶ **Stack** (page 80)
- ▶ Queue
- ▶ Binary Search Tree
- ▶ AVL Tree

Question No: 3 (Marks: 1) – Please choose one

Every AVL is _____

- ▶ Binary Tree
- ▶ Complete Binary Tree
- ▶ None of these
- ▶ **Binary Search Tree** [Click here for detail](#)

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 4 (Marks: 1) - Please choose one

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ **57 (page 303)**

Question No: 5 (Marks: 1) - Please choose one

If there are 23 external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ 23
- ▶ 24
- ▶ 21
- ▶ **22 (page 303)**

Question No: 6 (Marks: 1) - Please choose one

Which one of the following is not an example of equivalence relation?

- ▶ Electrical connectivity
- ▶ Set of people
- ▶ **$\leq$ relation (page 388)**
- ▶ Set of pixels

Question No: 7 (Marks: 1) - Please choose one

Binary Search is an algorithm of searching, used with the _____ data.

- ▶ **Sorted (page 432)**
- ▶ Unsorted
- ▶ Heterogeneous
- ▶ Random

Question No: 8 (Marks: 1) - Please choose one

Which one of the following is NOT true regarding the skip list?

- ▶ Each list S_i contains the special keys + infinity and - infinity.
- ▶ List S_0 contains the keys of S in non-decreasing order.
- ▶ Each list is a subsequence of the previous one.
- ▶ **List S_h contains only the n special keys. (page 446)**

Question No: 9 (Marks: 1) - Please choose one

A simple sorting algorithm like selection sort or bubble sort has a worst-case of

- ▶ $O(1)$ time because all lists take the same amount of time to sort
- ▶ $O(n)$ time because it has to perform n swaps to order the list.
- ▶ **$O(n^2)$ time because sorting 1 element takes $O(n)$ time - After 1 pass through the list, either of these algorithms can guarantee that 1 element is sorted. (page 487)**
- ▶ $O(n^3)$ time, because the worst case has really random input which takes longer to sort.

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 10 (Marks: 1) - Please choose one

Which of the following is a property of binary tree?

- ▶ A binary tree of N external nodes has N internal node.
- ▶ **A binary tree of N internal nodes has N+ 1 external node. (page 303)**
- ▶ A binary tree of N external nodes has N+ 1 internal node.
- ▶ A binary tree of N internal nodes has N- 1 external node.

Question No: 11 (Marks: 1) - Please choose one

By using _____ we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

- ▶ Binary tree only
- ▶ **Threaded binary tree (page 306)**
- ▶ Heap data structure
- ▶ Huffman encoding

Question No: 12 (Marks: 1) - Please choose one

Which of the following statement is true about dummy node of threaded binary tree?

- ▶ This dummy node never has a value.
- ▶ This dummy node has always some dummy value.
- ▶ **This dummy node has either no value or some dummy value. (Page 321)**
- ▶ This dummy node has always some integer value.

Question No: 13 (Marks: 1) - Please choose one

For a perfect binary tree of height h, having N nodes, the sum of heights of nodes is

- ▶ $N - (h - 1)$
- ▶ **$N - (h + 1)$ (page 373)**
- ▶ $N - 1$
- ▶ $N - 1 + h$

Question No: 14 (Marks: 1) - Please choose one

What is the best definition of a *collision* in a hash table?

- ▶ Two entries are identical except for their keys.
- ▶ Two entries with different data have the exact same key
- ▶ **Two entries with different keys have the same exact hash value. (page 464)**
- ▶ Two entries with the exact same key have different hash values.

Question No: 15 (Marks: 1) - Please choose one

Which formula is the best approximation for the depth of a heap with n nodes?

- ▶ **$\log_2 n$ (page 353)**
- ▶ The number of digits in n (base 10), e.g., 145 has three digits
- ▶ The square root of n
- ▶ n

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 16 (Marks: 1) - Please choose one

Which of the following statement is NOT correct about find operation:

- ▶ It is not a requirement that a find operation returns any specific name, just that finds on two elements return the same answer if and only if they are in the same set.
- ▶ **One idea might be to use a tree to represent each set, since each element in a tree has the same root, thus the root can be used to name the set.**
- ▶ Initially each set contains one element.
- ▶ Initially each set contains one element and it does not make sense to make a tree of one node only.

Question No: 17 (Marks: 1) - Please choose one

Which of the following is not true regarding the maze generation?

- ▶ Randomly remove walls until the entrance and exit cells are in the same set.
- ▶ Removing a wall is the same as doing a union operation.
- ▶ **Remove a randomly chosen wall if the cells it separates are already in the same set. (page 424)**
- ▶ Do not remove a randomly chosen wall if the cells it separates are already in the same set.

Question No: 18 (Marks: 1) – Please choose one

In threaded binary tree the NULL pointers are replaced by ,

- ▶ preorder successor or predecessor
- ▶ **inorder successor or predecessor (page 307)**
- ▶ postorder successor or predecessor
- ▶ NULL pointers are not replaced

Question No: 19 (Marks: 1) - Please choose one

Which of the given option is NOT a factor in Union by Size:

- ▶ Maintain sizes (number of nodes) of all trees, and during union.
- ▶ Make smaller tree, the subtree of the larger one.
- ▶ **Make the larger tree, the subtree of the smaller one. (page 408)**
- ▶ Implementation: for each root node i , instead of setting $\text{parent}[i]$ to -1 , set it to $-k$ if tree rooted at i has k nodes.

Question No: 20 (Marks: 1) - Please choose one

Suppose we had a hash table whose hash function is " $n \% 12$ ", if the number 35 is already in the hash table, which of the following numbers would cause a collision?

- ▶ 144
- ▶ 145
- ▶ 143
- ▶ 148

Question No: 21 (Marks: 1) - Please choose one

What requirement is placed on an array, so that *binary search* may be used to locate an entry?

- ▶ The array elements must form a heap.
- ▶ The array must have at least 2 entries.
- ▶ **The array must be sorted. [Click here for detail](#)**
- ▶ The array's size must be a power of two

Question No: 22 (Marks: 1) - Please choose one

A binary tree with 24 internal nodes has _____ external nodes.

- ▶ 22
- ▶ 23
- ▶ 48
- ▶ **25 (page 303)**

Question No: 23 (Marks: 1) - Please choose on

In case of deleting a node from AVL tree, rotation could be prolong to the *root* node.

- ▶ **Yes (Page 267)**
- ▶ No

Question No: 24 (Marks: 1) - Please choose one

when we have declared the size of the array, it is not possible to increase or decrease it during the _____ of the program.

- ▶ Declaration
- ▶ **Execution (page 17)**
- ▶ Defining
- ▶ None of the above

Question No: 25 (Marks: 1) - Please choose one

it will be efficient to place stack elements at the start of the list because insertion and removal take _____ time.

- ▶ Variable
- ▶ **Constant (page 60)**
- ▶ Inconsistent
- ▶ None of the above

Question No: 26 (Marks: 1) - Please choose one

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

- ▶ isFull(),isEmpty()
- ▶ pop(), push()
- ▶ **isEmpty() , isFull() (page 59)**
- ▶ push(),pop()

**FINALTERM EXAMINATION
Spring 2010**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari

Question No: 1 (Marks: 1) - Please choose one

What kind of list is best to answer questions such as "What is the item at position n?"

- ▶ **Lists implemented with an array.** [Click here for detail](#)
- ▶ Doubly-linked lists.
- ▶ Singly-linked lists.
- ▶ Doubly-linked or singly-linked lists are equally best

Question No: 2 (Marks: 1) - Please choose one

Each node in doubly link list has,

- ▶ 1 pointer
- ▶ **2 pointers (page 39)**
- ▶ 3 pointers
- ▶ 4 pointers

Question No: 3 (Marks: 1) - Please choose one

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ **57 (page 303)**

Question No: 4 (Marks: 1) - Please choose one

If there are N internal nodes in a binary tree then what will be the no. of external nodes in this binary tree?

- ▶ N -1
- ▶ N
- ▶ **N +1 (page 303)**
- ▶ N +2

Question No: 5 (Marks: 1) - Please choose one

A binary tree with N internal nodes has _____ links, _____ links to internal nodes and _____ links to external nodes

- ▶ N+1, 2N, N-1
- ▶ N+1, N-1, 2N
- ▶ **2N, N-1, N+1 (page 304)**
- ▶ N-1, 2N, N+1

Question No: 6 (Marks: 1) - Please choose one

The definition of Transitivity property is

- ▶ For all element x member of S, $x R x$
- ▶ For all elements x and y, $x R y$ if and only if $y R x$
- ▶ **For all elements x, y and z, if $x R y$ and $y R z$ then $x R z$ (page 385)**
- ▶ For all elements w, x, y and z, if $x R y$ and $w R z$ then $x R z$

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 7 (Marks: 1) - Please choose one

Which one of the following is not an example of equivalence relation:

- ▶ Electrical connectivity
- ▶ Set of people
- ▶ **$\leq$ relation (page 388)**
- ▶ Set of pixels

Question No: 8 (Marks: 1) - Please choose one

Union is a _____ time operation.

- ▶ **Constant (page 405)**
- ▶ Polynomial
- ▶ Exponential
- ▶ None of the given options

Question No: 9 (Marks: 1) - Please choose one

Binary Search is an algorithm of searching, used with the _____ data.

- ▶ **Sorted (page 432)**
- ▶ Unsorted
- ▶ Heterogeneous
- ▶ Random

Question No: 10 (Marks: 1) - Please choose one

A simple sorting algorithm like selection sort or bubble sort has a worst-case of

- ▶ $O(1)$ time because all lists take the same amount of time to sort
- ▶ $O(n)$ time because it has to perform n swaps to order the list.
- ▶ **$O(n^2)$ time because sorting 1 element takes $O(n)$ time - After 1 pass through the list, either of these algorithms can guarantee that 1 element is sorted. (page 487)**
- ▶ $O(n^3)$ time, because the worst case has really random input which takes longer to sort.

Question No: 11 (Marks: 1) - Please choose one

Merge sort and quick sort both fall into the same category of sorting algorithms. What is this category?

- ▶ $O(n \log n)$ sorts
- ▶ Interchange sort
- ▶ Average time is quadratic
- ▶ **None of the given options. (Page 488)**

Question No: 12 (Marks: 1) - Please choose one

Huffman encoding uses _____ tree to develop codes of varying lengths for the letters used in the original message.

- ▶ Linked list
- ▶ Stack
- ▶ Queue
- ▶ **Binary tree (page 287)**

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 13 (Marks: 1) - Please choose one

Which of the following statement is true about dummy node of threaded binary tree?

- ▶ The left pointer of dummy node points to the itself while the right pointer points to the root of tree.
- ▶ **The left pointer of dummy node points to the root node of the tree while the right pointer points itself i.e. to dummy node (page 321)**
- ▶ The left pointer of dummy node points to the root node of the tree while the right pointer is always NULL.
- ▶ The right pointer of dummy node points to the itself while the left pointer is always NULL.

Question No: 14 (Marks: 1) - Please choose one

Consider a min heap, represented by the following array:

10,30,20,70,40,50,80,60

After inserting a node with value 31. Which of the following is the updated min heap?

- ▶ **10,30,20,31,40,50,80,60,70 (page 336)**
- ▶ 10,30,20,70,40,50,80,60,31
- ▶ 10,31,20,30,40,50,80,60,31
- ▶ 31,10,30,20,70,40,50,80,60

Question No: 15 (Marks: 1) - Please choose one

Consider a min heap, represented by the following array:

11,22,33,44,55

After inserting a node with value 66. Which of the following is the updated min heap?

- ▶ **11,22,33,44,55,66 (page 336)**
- ▶ 11,22,33,44,66,55
- ▶ 11,22,33,66,44,55
- ▶ 11,22,66,33,44,55

Question No: 16 (Marks: 1) - Please choose one

Suppose that a selection sort of 100 items has completed 42 iterations of the main loop. How many items are now guaranteed to be in their final spot (never to be moved again)?

- ▶ 21
- ▶ 41
- ▶ **42** [Click here for detail](#)
- ▶ 43

Question No: 17 (Marks: 1) - Please choose one

_____ is a data structure that can grow easily dynamically at run time without having to copy existing elements.

- ▶ Array ()
- ▶ List
- ▶ **Both of these (page 10)**
- ▶ None of these

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 18 (Marks: 1) - Please choose one

The maximum number of external nodes (leaves) for a binary tree of height H is _____

- ▶ 2^H [Click here for detail](#)
- ▶ 2^{H+1}
- ▶ $2^H - 1$
- ▶ 2^{H+2}

Question No: 19 (Marks: 1) - Please choose one

A complete binary tree of height ____ has nodes between 16 to 31 .

- ▶ 2
- ▶ 3
- ▶ 4 (page 373)
- ▶ 5

Question No: 20 (Marks: 1) - Please choose one

Which of the given option is NOT a factor in Union by Size:

- ▶ Maintain sizes (number of nodes) of all trees, and during union.
- ▶ Make smaller tree, the subtree of the larger one.
- ▶ **Make the larger tree, the subtree of the smaller one. (page 408)**
- ▶ Implementation: for each root node i, instead of setting parent[i] to -1, set it to -k if tree rooted at i has k nodes.

Question No: 21 (Marks: 1) - Please choose one

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

The array after the FIRST iteration of the large loop in a selection sort (sorting from smallest to largest).

- ▶ 0 3 8 9 1 7 5 2 6 4 (Page 477)
- ▶ 2 6 4 0 3 8 9 1 7 5
- ▶ 2 6 4 9 1 7 0 3 8 5
- ▶ 0 3 8 2 6 4 9 1 7 5

Question No: 22 (Marks: 1) - Please choose one

Suppose A is an array containing numbers in increasing order, but some numbers occur more than once when using a binary search for a value, the binary search always finds _____

- ▶ the first occurrence of a value. [Click here for detail](#)
- ▶ the second occurrence of a value.
- ▶ may find first or second occurrence of a value.
- ▶ None of the given options.

Question No: 23 (Marks: 1) - Please choose one

A binary tree with 24 internal nodes has _____ external nodes.

- ▶ 22
- ▶ 23
- ▶ 48
- ▶ 25 (page 303)

Question No: 24 (Marks: 1) - Please choose one
 it will be efficient to place stack elements at the start of the list because insertion and removal take _____ time.

- ▶ Variable
- ▶ **Constant (page 60)**
- ▶ Inconsistent
- ▶ None of the above

Question No: 25 (Marks: 1) - Please choose one
 “+” is a _____ operator.

- ▶ Unary
- ▶ **Binary (page 64)**
- ▶ Ternary
- ▶ None of the above

Question No: 26 (Marks: 1) - Please choose one
 A kind of expressions where the operator is present between two operands called _____ expressions.

- ▶ Postfix
- ▶ **Infix (page 64)**
- ▶ Prefix
- ▶ None of the above.

FINAL TERM EXAMINATION Spring 2010

Question No: 1 (Marks: 1) - Please choose one
 Here is a small function definition:

```
void f(int i, int &k)
{
  i = 1;
  k = 2;
}
```

Suppose that a main program has two integer variables x and y, which are given the value 0. Then the main program calls f(x,y); What are the values of x and y after the function f finishes?

- ▶ Both x and y are still 0.
- ▶ x is now 1, but y is still 0.
- ▶ **x is still 0, but y is now 2.**
- ▶ x is now 1, and y is now 2.

Question No: 2 (Marks: 1) - Please choose one

A binary tree with N internal nodes has _____ links, _____ links to internal nodes and _____ links to external nodes

- ▶ N+1, 2N, N-1
- ▶ N+1, N-1, 2N
- ▶ **2N, N-1, N+1 (page 304)**
- ▶ N-1, 2N, N+1

Question No: 3 (Marks: 1) - Please choose one

Each node in doubly link list has,

- ▶ 1 pointer
- ▶ **2 pointers (Page 39)**
- ▶ 3 pointers
- ▶ 4 pointers

Question No: 4 (Marks: 1) - Please choose one

If you know the size of the data structure in advance, i.e., at compile time, which one of the following is a good data structure to use.

- ▶ Array
- ▶ List
- ▶ **Both of these (page 10)**
- ▶ None of these

Question No: 5 (Marks: 1) - Please choose one

Which one of the following is not an example of equivalence relation:

- ▶ Electrical connectivity
- ▶ Set of people
- ▶ **$\leq$ relation (Page 388)**
- ▶ Set of pixels

Question No: 6 (Marks: 1) - Please choose one

If a complete binary tree has height h then its no. of nodes will be,

- ▶ Log (h)
- ▶ **$2^{h+1} - 1$ (page 125)**
- ▶ Log (h) - 1
- ▶ $2^h - 1$

Question No: 7 (Marks: 1) - Please choose one

If a max heap is implemented using a partially filled array called data, and the array contains n elements ($n > 0$), where is the entry with the greatest value? **Data[0] is correct**

- ▶ data[1]
- ▶ data[n-1]
- ▶ data[n]
- ▶ data[2*n+1]

Question No: 8 (Marks: 1) - Please choose one

Which one is a self-referential data type?

- ▶ Stack
- ▶ Queue
- ▶ [Link list](#) [Click here for detail](#)
- ▶ All of these

Question No: 9 (Marks: 1) - Please choose one

There is/are _____ case/s for rotation in an AVL tree,

- ▶ 1
- ▶ 3
- ▶ 2
- ▶ **4 (page 229)**

Question No: 10 (Marks: 1) - Please choose one

Which of the following can be the inclusion criteria for pixels in image segmentation.

- ▶ Pixel intensity
- ▶ Texture
- ▶ Threshold of intensity
- ▶ **All of the given options (page 421)**

Question No: 11 (Marks: 1) - Please choose one

Consider the following array

23 15 5 12 40 10 7

After the first pass of a particular algorithm, the array looks like

15 5 12 23 10 7 40

Name the algorithm used

- ▶ Heap sort
- ▶ Selection sort
- ▶ Insertion sort
- ▶ **Bubble sort (According to rule)**

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 12 (Marks: 1) - Please choose one

In a perfectly balanced tree the insertion of a node needs _____ .

- ▶ **One rotation (Page 225)**
- ▶ Two rotations
- ▶ Rotations equal to number of levels
- ▶ No rotation at all

Question No: 13 (Marks: 1) - Please choose one

If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is _____ .

- ▶ N
- ▶ N^2
- ▶ $N\log_2 N$
- ▶ **$\log_2 N$ (page 440)**

Question No: 14 (Marks: 1) - Please choose one

Which of the following is NOT a correct statement about Table ADT.

- ▶ In a table, the type of information in columns may be different.
- ▶ **A table consists of several columns, known as entities. (page 408)**
- ▶ The row of a table is called a record.
- ▶ A major use of table is in databases where we build and use tables for keeping information.

Question No: 15 (Marks: 1) - Please choose one

If both pointers of the node in a binary tree are NULL then it will be a/an _____ .

- ▶ Inner node
- ▶ **Leaf node (page 120)**
- ▶ Root node
- ▶ None of the given options

Question No: 16 (Marks: 1) - Please choose one

Suppose we are sorting an array of eight integers using quick sort, and we have just finished the first partitioning with the array looking like this:

2 5 1 7 9 12 11 10

Which statement is correct?

- ▶ **The pivot could be either the 7 or the 9.(page 506)**
- ▶ The pivot could be the 7, but it is not the 9.
- ▶ The pivot is not the 7, but it could be the 9.
- ▶ Neither the 7 nor the 9 is the pivot.

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 17 (Marks: 1) - Please choose one

What is the best definition of a *collision* in a hash table?

- ▶ Two entries are identical except for their keys.
- ▶ Two entries with different data have the exact same key
- ▶ **Two entries with different keys have the same exact hash value. (page 464)**
- ▶ Two entries with the exact same key have different hash values.

Question No: 18 (Marks: 1) - Please choose one

For a perfect binary tree of height h , having N nodes, the sum of heights of nodes is

- ▶ $N - (h - 1)$
- ▶ **$N - (h + 1)$ (Page 373)**
- ▶ $N - 1$
- ▶ $N - 1 + h$

Question No: 19 (Marks: 1) - Please choose one

A binary tree with 33 internal nodes has _____ links to internal nodes.

- ▶ 31
- ▶ **32 (Page 304)**
- ▶ 33
- ▶ 66

Question No: 20 (Marks: 1) - Please choose one

Suppose you implement a Min heap (with the smallest element on top) in an array. Consider the different arrays below; determine the one that *cannot* possibly be a heap:

- ▶ 16, 18, 20, 22, 24, 28, 30
- ▶ 16, 20, 18, 24, 22, 30, 28
- ▶ 16, 24, 18, 28, 30, 20, 22
- ▶ **16, 24, 20, 30, 28, 18, 22 (see min heap property at page 337)**

Question No: 21 (Marks: 1) - Please choose one

Which of the following is not true regarding the maze generation?

- ▶ Randomly remove walls until the entrance and exit cells are in the same set.
- ▶ Removing a wall is the same as doing a union operation.
- ▶ **Remove a randomly chosen wall if the cells it separates are already in the same set. (Page 424)**
- ▶ Do not remove a randomly chosen wall if the cells it separates are already in the same set.

Question No: 22 (Marks: 1) - Please choose one

Which formula is the best approximation for the depth of a heap with n nodes?

► **log (base 2) of n (Page 353)**

- The number of digits in n (base 10), e.g., 145 has three digits
- The square root of n
- n

Question No: 23 (Marks: 1) - Please choose one

In threaded binary tree the NULL pointers are replaced by ,

- preorder successor or predecessor
- **inorder successor or predecessor (Page 307)**
- postorder successor or predecessor
- NULL pointers are not replaced

Question No: 24 (Marks: 1) - Please choose one

The _____ method of list will position the *currentNode* and *lastCurrentNode* at the start of the list.

- Remove
- Next
- **Start (Page 38)**
- Back

Question No: 25 (Marks: 1) - Please choose one

Mergesort makes two recursive calls. Which statement is true after these recursive calls finish, but before the merge step?

- Elements in the first half of the array are less than or equal to elements in the second half of the array.
 - None of the given options.
 - The array elements form a heap.
 - **Elements in the second half of the array are less than or equal to elements in the first half of the array.**
- [Click here for detail](#)

Question No: 26 (Marks: 1) - Please choose one

Suppose we had a hash table whose hash function is " $n \% 12$ ", if the number 35 is already in the hash table, which of the following numbers would cause a collision?

- 144
- 145
- **143**
- 148

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

FINAL TERM EXAMINATION

Fall 2009

Question No: 1 (Marks: 1) - Please choose one

The arguments passed to a function should match in number, type and order with the parameters in the function definition.

▶ True

▶ False

Question No: 2 (Marks: 1) - Please choose one

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

▶ 48

▶ 4

▶ 222

▶ **5 (According to rule)**

Question No: 3 (Marks: 1) - Please choose one

Suppose currentNode refers to a node in a linked list (using the Node class with member variables called data and nextNode). What statement changes currentNode so that it refers to the next node?

▶ currentNode ++;

▶ currentNode = nextNode;

▶ currentNode += nextNode;

▶ **currentNode = currentNode->nextNode;**

Question No: 4 (Marks: 1) - Please choose one

A **Compound Data Structure** is the data structure which can have multiple data items of same type or of different types. Which of the following can be considered compound data structure?

▶ Arrays

▶ LinkLists

▶ Binary Search Trees

▶ **All of the given options** [Click here for detail](#)

Question No: 5 (Marks: 1) - Please choose one

Here is a small function definition:

```
void f(int i, int &k)
{
i = 1;
k = 2;
}
```

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Suppose that a main program has two integer variables x and y, which are given the value 0. Then the main program calls f(x,y); What are the values of x and y after the function f finishes?

- ▶ Both x and y are still 0.
- ▶ x is now 1, but y is still 0.
- ▶ **x is still 0, but y is now 2.**
- ▶ x is now 1, and y is now 2.

Question No: 6 (Marks: 1) - Please choose one

The difference between a binary tree and a binary search tree is that ,

- ▶ **a binary search tree has two children per node whereas a binary tree can have none, one, or two children per node** [Click here for detail](#)
- ▶ in binary search tree nodes are inserted based on the values they contain
- ▶ in binary tree nodes are inserted based on the values they contain
- ▶ none of these

Question No: 7 (Marks: 1) - Please choose one

Compiler uses which one of the following to evaluate a mathematical equation,

- ▶ Binary Tree
- ▶ Binary Search Tree
- ▶ **Parse Tree (Page 274)**
- ▶ AVL Tree

Question No: 8 (Marks: 1) - Please choose one

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ **57 (Page 303)**

Question No: 9 (Marks: 1) - Please choose one

If there are 23 external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ 23
- ▶ 24
- ▶ 21
- ▶ **22 (n-1) (Page 304)**

Question No: 10 (Marks: 1) - Please choose one

Which of the following method is helpful in creating the heap at once?

- ▶ insert
- ▶ add
- ▶ update
- ▶ **preculatDown (Page 358)**

Question No: 11 (Marks: 1) - Please choose one

The definition of Transitivity property is

- ▶ For all element x member of S , $x R x$
- ▶ For all elements x and y , $x R y$ if and only if $y R x$
- ▶ **For all elements x, y and z , if $x R y$ and $y R z$ then $x R z$ (Page 385)**
- ▶ For all elements w, x, y and z , if $x R y$ and $w R z$ then $x R z$

Question No: 12 (Marks: 1) - Please choose one

A binary tree of N nodes has _____.

- ▶ $\log_{10} N$ levels
- ▶ **$\log_2 N$ levels (Page 349)**
- ▶ $N / 2$ levels
- ▶ $N \times 2$ levels

Question No: 13 (Marks: 1) - Please choose one

If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is _____.

- ▶ N
- ▶ N^2
- ▶ $N \log_2 N$
- ▶ **$\log_2 N$ (page 440)**

Question No: 14 (Marks: 1) - Please choose one

Consider the following array

23 15 5 12 40 10 7

After the first pass of a particular algorithm, the array looks like

15 12 23 10 7 40

Name the algorithm used

- ▶ Heap sort
- ▶ Selection sort
- ▶ Insertion sort
- ▶ **Bubble sort**

Question No: 15 (Marks: 1) - Please choose one

If both pointers of the node in a binary tree are NULL then it will be a/an _____.

- ▶ Inner node
- ▶ **Leaf node (Page 313)**
- ▶ Root node
- ▶ None of the given options

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 16 (Marks: 1) - Please choose one

By using _____ we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

- ▶ Binary tree only
- ▶ **Threaded binary tree (page 306)**
- ▶ Heap data structure
- ▶ Huffman encoding

Question No: 17 (Marks: 1) - Please choose one

A complete binary tree of height 3 has between _____ nodes.

- ▶ 8 to 14
- ▶ **8 to 15 (Page 124)**
- ▶ 8 to 16
- ▶ 8 to 17

$$2^{(d+1)} - 1 = 2^{(3+1)} - 1 = 2^4 - 1 = 16 - 1 = 15$$

Question No: 18 (Marks: 1) - Please choose one

Consider a min heap, represented by the following array:

3,4,6,7,5,10

After inserting a node with value 1. Which of the following is the updated min heap?

- ▶ 3,4,6,7,5,10,1
- ▶ 3,4,6,7,5,1,10
- ▶ 3,4,1,5,7,10,6
- ▶ **1,4,3,5,7,10,6 close to correct but correct ans is 1,4,3,7,5,10,6 (page 337)**

Question No: 19 (Marks: 1) - Please choose one

Consider a min heap, represented by the following array:

10,30,20,70,40,50,80,60

After inserting a node with value 31. Which of the following is the updated min heap?

- ▶ **10,30,20,31,40,50,80,60,70 (page 337)**
- ▶ 10,30,20,70,40,50,80,60,31
- ▶ 10,31,20,30,40,50,80,60,31
- ▶ 31,10,30,20,70,40,50,80,60

Question No: 20 (Marks: 1) - Please choose one

Which one of the following algorithms is most widely used due to its good average time,

- ▶ Bubble Sort
- ▶ Insertion Sort
- ▶ **Quick Sort [Click here for detail](#)**
- ▶ Merge Sort

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 21 (Marks: 1) - Please choose one

Which of the following statement is correct about find(x) operation:

- ▶ A find(x) on element x is performed by returning exactly the same node that is found.
- ▶ **A find(x) on element x is performed by returning the root of the tree containing x.**

[Click here for detail](#)

- ▶ A find(x) on element x is performed by returning the whole tree itself containing x. (Page 10)
- ▶ A find(x) on element x is performed by returning TRUE.

Question No: 22 (Marks: 1) - Please choose one

Which of the following statement is NOT correct about find operation:

- ▶ It is not a requirement that a find operation returns any specific name, just that finds on two elements return the same answer if and only if they are in the same set.
- ▶ **One idea might be to use a tree to represent each set, since each element in a tree has the same root, thus the root can be used to name the set.**
- ▶ Initially each set contains one element.
- ▶ Initially each set contains one element and it does not make sense to make a tree of one node only.

Question No: 23 (Marks: 1) - Please choose one

The following are statements related to queues.

- The last item to be added to a queue is the first item to be removed **False statement**
- A queue is a structure in which both ends are not used **False statement**
- The last element hasn't to wait until all elements preceding it on the queue are removed **False statement**
- queue is said to be a last-in-first-out list or LIFO data structure. **False statement**

Which of the above is/are related to normal queues?

- ▶ (iii) and (ii) only
- ▶ (i), (ii) and (iv) only
- ▶ (ii) and (iv) only
- ▶ **None of the given options**

Question No: 24 (Marks: 1) - Please choose one

The maximum number of external nodes (leaves) for a binary tree of height H is _____

- ▶ **2^H** [Click here for detail](#)
- ▶ $2^H + 1$
- ▶ $2^H - 1$
- ▶ $2^H + 2$

Question No: 25 (Marks: 1) - Please choose one

In complete binary tree the bottom level is filled from _____

- ▶ **Left to right (Page 323)**
- ▶ Right to left
- ▶ Not filled at all
- ▶ None of the given options

Question No: 26 (Marks: 1) - Please choose one

We are given N items to build a heap, this can be done with _____ successive inserts.

- ▶ N-1
- ▶ **N (Page 353)**
- ▶ N+1
- ▶ N^2

Question No: 27 (Marks: 1) - Please choose one

Suppose we had a hash table whose hash function is " $n \% 12$ ", if the number 35 is already in the hash table, which of the following numbers would cause a collision?

- ▶ 144
- ▶ 145
- ▶ **143**
- ▶ 148

Question No: 28 (Marks: 1) - Please choose one

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

The array after the FIRST iteration of the large loop in a selection sort (sorting from smallest to largest).

- ▶ **0 3 8 9 1 7 5 2 6 4 (Page 477)**
- ▶ 2 6 4 0 3 8 9 1 7 5
- ▶ 2 6 4 9 1 7 0 3 8 5
- ▶ 0 3 8 2 6 4 9 1 7 5

Question No: 29 (Marks: 1) - Please choose one

What requirement is placed on an array, so that *binary search* may be used to locate an entry?

- ▶ The array elements must form a heap.
- ▶ The array must have at least 2 entries.
- ▶ **The array must be sorted. [Click here for detail](#)**
- ▶ The array's size must be a power of two.

Question No: 30 (Marks: 1) - Please choose one

In case of deleting a node from AVL tree, rotation could be prolonged to the *root* node.

- ▶ **Yes (Page 267)**
- ▶ No

FINAL TERM EXAMINATION

Fall 2009

Question No: 1 (Marks: 1) - Please choose one

_____ only removes items in reverse order as they were entered.

- ▶ **Stack (Page 81)**
- ▶ Queue
- ▶ Both of these
- ▶ None of these

Question No: 2 (Marks: 1) - Please choose one

Here is a small function definition:

```
void f(int i, int &k)
{
    i = 1;
    k = 2;
}
```

Suppose that a main program has two integer variables x and y, which are given the value 0. Then the main program calls f(x,y); What are the values of x and y after the function f finishes?

- ▶ Both x and y are still 0.
- ▶ x is now 1, but y is still 0.
- ▶ **x is still 0, but y is now 2.**
- ▶ x is now 1, and y is now 2.

Question No: 3 (Marks: 1) - Please choose one

Select the one *FALSE* statement about binary trees:

- ▶ **Every binary tree has at least one node. (Page 113)**
- ▶ Every non-empty tree has exactly one root node.
- ▶ Every node has at most two children.
- ▶ Every non-root node has exactly one parent.

Question No: 4 (Marks: 1) - Please choose one

Every AVL is _____

- ▶ Binary Tree
- ▶ Complete Binary Tree
- ▶ None of these
- ▶ **Binary Search Tree** [Click here for detail](#)

Question No: 5 (Marks: 1) - Please choose one

Searching an element in an AVL tree take maximum _____ time (where n is no. of nodes in AVL tree),

- ▶ $\log_2(n+1)$
- ▶ $\log_2(n+1) - 1$
- ▶ **$1.44 \log_2 n$ (Page 227)**
- ▶ $1.66 \log_2 n$

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 6 (Marks: 1) - Please choose one

Suppose that we have implemented a *priority queue* by storing the items in a heap. We are now executing a reheapification downward and the out-of-place node has priority of 42. The node's parent has a priority of 72, the left child has priority 52 and the node's right child has priority 62. Which statement best describes the status of the reheapification.

- ▶ The reheapification is done.
- ▶ The next step will interchange the two children of the out-of-place node.
- ▶ The next step will swap the out-of-place node with its parent.
- ▶ The next step will swap the out-of-place node with its left child.

Question No: 7 (Marks: 1) - Please choose one

Suppose you implement a heap (with the largest element on top) in an array. Consider the different arrays below, determine the one that *cannot* possibly be a heap:

- ▶ 7 6 5 4 3 2 1
- ▶ 7 3 6 2 1 4 5
- ▶ 7 6 4 3 5 2 1
- ▶ **7 3 6 4 2 5 1**

According to max heap property

Question No: 8 (Marks: 1) - Please choose one

If there are 23 external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ 23
- ▶ 24
- ▶ 21
- ▶ **22 (N-1)**

Question No: 9 (Marks: 1) - Please choose one

If there are N external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ **N -1 (Page 304)**
- ▶ N+1
- ▶ N+2
- ▶ N

Question No: 10 (Marks: 1) - Please choose one

Which one of the following is NOT the property of equivalence relation:

- ▶ Reflexive
- ▶ Symmetric
- ▶ Transitive
- ▶ **Associative (Page 385)**

Question No: 11 (Marks: 1) - Please choose one

The definition of Transitivity property is

- ▶ For all element x member of S , $x R x$
- ▶ For all elements x and y , $x R y$ if and only if $y R x$
- ▶ **For all elements x , y and z , if $x R y$ and $y R z$ then $x R z$ (Page 385)**
- ▶ For all elements w , x , y and z , if $x R y$ and $w R z$ then $x R z$

Question No: 12 (Marks: 1) - Please choose one

Union is a _____ time operation.

- ▶ **Constant (Page 120)**
- ▶ Polynomial
- ▶ Exponential
- ▶ None of the given option

Question No: 13 (Marks: 1) - Please choose one

Which of the following is NOT a correct statement about Table ADT.

- ▶ In a table, the type of information in columns may be different. yes
- ▶ **A table consists of several columns, known as entities. (Page 408)**
- ▶ The row of a table is called a record.
- ▶ A major use of table is in databases where we build and use tables for keeping information.

Question No: 14 (Marks: 1) - Please choose one

In the worst case of deletion in AVL tree requires _____.

- ▶ Only one rotation
- ▶ Rotation at each non-leaf node
- ▶ Rotation at each leaf node
- ▶ **Rotations equal to $\log_2 N$ (Page 441)**

Question No: 15 (Marks: 1) - Please choose on

Binary Search is an algorithm of searching, used with the _____ data.

- ▶ **Sorted (Page 432)**
- ▶ Unsorted
- ▶ Heterogeneous
- ▶ Random

Question No: 16 (Marks: 1) - Please choose on

Which of the following statement is correct?

- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER successor.
- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its PREORDER successor.
- ▶ **A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. (Page 307)**
- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its POSTORDER successor.

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 17 (Marks: 1) - Please choose one

By using _____ we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

- ▶ Binary tree only
- ▶ **Threaded binary tree (page 306)**
- ▶ Heap data structure
- ▶ Huffman encoding

Question No: 18 (Marks: 1) - Please choose one

Which of the following statement is NOT true about threaded binary tree?

- ▶ Right thread of the right-most node points to the *dummy* node.
- ▶ Left thread of the left-most node points to the *dummy* node.
- ▶ The left pointer of dummy node points to the root node of the tree.
- ▶ **Left thread of the right-most node points to the *dummy* node. (page 321)**

Question No: 19 (Marks: 1) - Please choose one

Consider a min heap, represented by the following array:

11,22,33,44,55

After inserting a node with value 66. Which of the following is the updated min heap?

- ▶ **11,22,33,44,55,66 (page 337)**
- ▶ 11,22,33,44,66,55
- ▶ 11,22,33,66,44,55
- ▶ 11,22,66,33,44,55

Question No: 20 (Marks: 1) - Please choose one

Consider a min heap, represented by the following array:

3,4,6,7,5

After calling the function deleteMin(). Which of the following is the updated min heap?

- ▶ 4,6,7,5
- ▶ 6,7,5,4
- ▶ **4,5,6,7 (page 349)**
- ▶ 4,6,5,7

Question No: 21 (Marks: 1) - Please choose one

We can build a heap in _____ time.

- ▶ **Linear (Page 353)**
- ▶ Exponential
- ▶ Polynomial
- ▶ None of the given options

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 22 (Marks: 1) - Please choose one

Suppose we are sorting an array of eight integers using quick sort, and we have just finished the first partitioning with the array looking like this:

2 5 1 7 9 12 11 10

Which statement is correct?

- ▶ **The pivot could be either the 7 or the 9. (page 506)**
- ▶ The pivot could be the 7, but it is not the 9.
- ▶ The pivot is not the 7, but it could be the 9
- ▶ Neither the 7 nor the 9 is the pivot.

Question No: 23 (Marks: 1) - Please choose one

Which formula is the best approximation for the depth of a heap with n nodes?

- ▶ **$\log_2 n$ (Page 353)**
- ▶ The number of digits in n (base 10), e.g., 145 has three digits
- ▶ The square root of n
- ▶ n

Question No: 24 (Marks: 1) - Please choose one

Suppose you implement a Min heap (with the smallest element on top) in an array. Consider the different arrays below; determine the one that *cannot* possibly be a heap:

- ▶ 16, 18, 20, 22, 24, 28, 30
- ▶ 16, 20, 18, 24, 22, 30, 28
- ▶ 16, 24, 18, 28, 30, 20, 22
- ▶ **16, 24, 20, 30, 28, 18, 22** **It's not satisfy the min heap property.**

Question No: 25 (Marks: 1) - Please choose one

While joining nodes in the building of Huffman encoding tree if there are more nodes with same frequency, we choose the nodes _____.

- ▶ **Randomly (Page 289)**
- ▶ That occur first in the text message
- ▶ That are lexically smaller among others.
- ▶ That are lexically greater among others

Question No: 26 (Marks: 1) - Please choose one

Consider the following paragraph with blanks.

A is a linear list where and take place at the same end . This end is called the

What would be the correct filling the above blank positions?

- ▶ (i) queue (ii) insertion (iii) removals (iv) top
- ▶ (i) stack (ii) insertion (iii) removals (iv) bottom
- ▶ **(i) stack (ii) insertion (iii) removals (iv) top (Page 52)**
- ▶ (i) tree (ii) insertion (iii) removals (iv) top

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 27 (Marks: 1) - Please choose one

A binary tree with 33 internal nodes has _____ links to internal nodes.

- ▶ 31
- ▶ **32 (n-1 links to internal nodes) (Page 304)**
- ▶ 33
- ▶ 66 .

Question No: 28 (Marks: 1) - Please choose on

Which traversal gives a decreasing order of elements in a heap where the max element is stored at the top?

- ▶ post-order
- ▶ level-order
- ▶ inorder
- ▶ **None of the given options**

Question No: 29 (Marks: 1) - Please choose one

What requirement is placed on an array, so that *binary search* may be used to locate an entry

- ▶ The array elements must form a heap.
- ▶ The array must have at least 2 entries.
- ▶ **The array must be sorted** [Click here for detail](#)
- ▶ The array's size must be a power of two.

Question No: 30 (Marks: 1) - Please choose one

Which of the following is a non linear data structure?

- ▶ Linked List
- ▶ Stack
- ▶ Queue
- ▶ **Tree (Page 112)**

**FINALTERM EXAMINATION
Fall 2009**

Question No: 1 (Marks: 1) - Please choose one

The data of the problem is of 2GB and the hard disk is of 1GB capacity, to solve this problem we should

- ▶ Use better data structures
- ▶ **Increase the hard disk space (Page 5)**
- ▶ Use the better algorithm
- ▶ Use as much data as we can store on the hard disk

Question No: 2 (Marks: 1) - Please choose one

In an array list the current element is

- ▶ **The first element** [Click here for detail](#)

- ▶ The middle element
- ▶ The last element
- ▶ The element where the current pointer points to

Question No: 3 (Marks: 1) - Please choose one

Which one of the following is a valid postfix expression?

- ▶ $ab+c*d-$
- ▶ **$abc*+d-$ (According to rule)**
- ▶ $abc+*d-$
- ▶ $(abc*)+d-$

Question No: 4 (Marks: 1) - Please choose one

In sequential access data structure, accessing any element in the data structure takes different amount of time. Tell which one of the following is sequential access data structure,

- ▶ Arrays
- ▶ **Lists** [Click here for detail](#)
- ▶ Both of these
- ▶ None of these

Question No: 5 (Marks: 1) - Please choose one

I have implemented the queue with a circular array. If data is a circular array of CAPACITY elements, and last is an index into that array, what is the formula for the index after last?

- ▶ $(last \% 1) + CAPACITY$
- ▶ $last \% (1 + CAPACITY)$
- ▶ **$(last + 1) \% CAPACITY$**
- ▶ $last + (1 \% CAPACITY)$

This expression will point to field after last that will be the first field.

Question No: 6 (Marks: 1) - Please choose one

Which one of the following is TRUE about recursion?

- ▶ **Recursion extensively uses stack memory. (page 149)**
- ▶ Threaded Binary Trees use the concept of recursion.
- ▶ Recursive function calls consume a lot of memory.
- ▶ Iteration is more efficient than iteration.

Question No: 7 (Marks: 1) - Please choose one

Compiler uses which one of the following to evaluate a mathematical equation,

- ▶ Binary Tree
- ▶ Binary Search Tree
- ▶ **Parse Tree (Page 274)**
- ▶ AVL Tree

Question No: 8 (Marks: 1) - Please choose one

Which one of the following is TRUE about iteration?

- ▶ Iteration extensively uses stack memory.
- ▶ Threaded Binary Trees use the concept of iteration.
- ▶ Iterative function calls consumes a lot of memory.
- ▶ **Recursion is more efficient than iteration.** [Click here for detail](#)

Question No: 9 (Marks: 1) - Please choose one

If a max heap is implemented using a partially filled array called data, and the array contains n elements ($n > 0$), where is the entry with the greatest value? **Data[0] is correct**

- ▶ data[1]
- ▶ data[n-1]
- ▶ data[n]
- ▶ data[2*n+1]

Question No: 10 (Marks: 1) - Please choose one

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ **57 (n+1)**

Question No: 11 (Marks: 1) - Please choose one

Which of the following heap method increase the value of key at position 'p' by the amount 'delta'?

- ▶ **increaseKey(p,delta) (Page 363)**
- ▶ decreaseKey(p,delta)
- ▶ preculcateDown(p,delta)
- ▶ remove(p,delta)

Question No: 12 (Marks: 1) - Please choose one

If we have 1000 sets each containing a single different person. Which of the following relation will be true on each set:

- ▶ **Reflexive (page 387)**
- ▶ Symmetric
- ▶ Transitive
- ▶ Associative

Question No: 13 (Marks: 1) - Please choose one

Which one of the following is not an example of equivalence relation:

- ▶ Electrical connectivity
- ▶ Set of people
- ▶ **<= relation (Page 388)**
- ▶ Set of pixels

Question No: 14 (Marks: 1) - Please choose one

A binary tree of N nodes has _____.

- ▶ $\log_{10} N$ levels
- ▶ **$\log_2 N$ levels (Page 212)**
- ▶ $N / 2$ levels
- ▶ $N \times 2$ levels

Question No: 15 (Marks: 1) <http://vustudents.ning.com> - Please choose one

Binary Search is an algorithm of searching, used with the _____ data.

- ▶ **Sorted (Page 432)**
- ▶ Unsorted
- ▶ Heterogeneous
- ▶ Random

Question No: 16 (Marks: 1) - Please choose one

Consider the following array

23 15 5 12 40 10 7

After the first pass of a particular algorithm, the array looks like

15 5 12 23 10 7 40

Name the algorithm used

- ▶ Heap sort
- ▶ Selection sort
- ▶ Insertion sort
- ▶ **Bubble sort (According to rule)**

Question No: 17 (Marks: 1) - Please choose one

Which of the following statements is correct property of binary trees?

- ▶ A binary tree with N internal nodes has N+1 internal links.
- ▶ A binary tree with N external nodes has 2N internal nodes.
- ▶ **A binary tree with N internal nodes has N+1 external nodes. (page 304)**
- ▶ None of above statement is a property of the binary tree.

Question No: 18 (Marks: 1) - Please choose one

Which of the following is a property of binary tree?

- ▶ A binary tree of N external nodes has N internal node.
- ▶ **A binary tree of N internal nodes has N+ 1 external node. (Page 304)**
- ▶ A binary tree of N external nodes has N+ 1 internal node.
- ▶ A binary tree of N internal nodes has N- 1 external node.

Question No: 19 (Marks: 1) - Please choose one

Which of the following statement is true about dummy node of threaded binary tree?

- ▶ The left pointer of dummy node points to the itself while the right pointer points to the root of tree.
- ▶ **The left pointer of dummy node points to the root node of the tree while the right pointer points itself i.e. to dummy node (Page 321)**
- ▶ The left pointer of dummy node points to the root node of the tree while the right pointer is always NULL.
- ▶ The right pointer of dummy node points to the itself while the left pointer is always NULL.

Question No: 20 (Marks: 1) - Please choose one

If the bottom level of a binary tree is NOT completely filled, depicts that the tree is NOT a

- ▶ Expression tree
- ▶ Threaded binary tree
- ▶ **complete Binary tree (Page 323)**
- ▶ Perfectly complete Binary tree

Question No: 21 (Marks: 1) - Please choose one

In a selection sort of n elements, how many times the swap function is called to complete the execution of the algorithm?

- ▶ **$n-1$ [Click here for detail](#)**
- ▶ $n \log n$
- ▶ n^2
- ▶ 1

Question No: 22 (Marks: 1) - Please choose one

Which of the following statement is correct about find(x) operation:

- ▶ A find(x) on element x is performed by returning exactly the same node that is found.
- ▶ **A find(x) on element x is performed by returning the root of the tree containing x . [Click here for detail](#)**
- ▶ A find(x) on element x is performed by returning the whole tree itself containing x .
- ▶ A find(x) on element x is performed by returning TRUE.

Question No: 23 (Marks: 1) - Please choose one

Which of the following statement is NOT correct about find operation:

- ▶ It is not a requirement that a find operation returns any specific name, just that finds on two elements return the same answer if and only if they are in the same set.
- ▶ **One idea might be to use a tree to represent each set, since each element in a tree has the same root, thus the root can be used to name the set.**
- ▶ Initially each set contains one element.
- ▶ **Initially** each set contains one element and it does not make sense to make a tree of one node only.

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 24 (Marks: 1) - Please choose one

Consider the following postfix expression S and the initial values of the variables.

$$S = A B - C + D E F - + ^$$

Assume that A=3, B=2, C=1, D=1, E=2, F=3

What would be the final output of the stack?

- ▶ 1 [Click here for detail](#)
- ▶ 2
- ▶ 0
- ▶ -1

Question No: 25 (Marks: 1) - Please choose one

The maximum number of external nodes (leaves) for a binary tree of height H is _____

- ▶ 2^H [Click here for detail](#)
- ▶ $2^H + 1$
- ▶ $2^H - 1$
- ▶ $2^H + 2$

Question No: 26 (Marks: 1) - Please choose one

In threaded binary tree the NULL pointers are replaced by ,

- ▶ preorder successor or predecessor
- ▶ **inorder successor or predecessor (Page 307)**
- ▶ postorder successor or predecessor
- ▶ NULL pointers are not replaced

Question No: 27 (Marks: 1) - Please choose one

In a min heap , precluateDown procedure will move smaller value_____ and bigger value_____.

- ▶ left,right
- ▶ right,left
- ▶ **up,down (Page 358)**
- ▶ down,up

Question No: 28 (Marks: 1) - Please choose one

Which of the following statement is correct about union:

- ▶ **To perform Union of two sets, we merge the two trees by making the root of one tree point to the root of the other. (Greedy algorithms , Page 7)**
- ▶ To perform Union of two sets, we merge the two trees by making the leaf node of one tree point to the root of the other.
- ▶ To perform Union of two sets, merging operation of trees in not required at all.
- ▶ None of the given options.

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 29 (Marks: 1) - Please choose one

Suppose A is an array containing numbers in increasing order, but some numbers occur more than once when using a binary search for a value, the binary search always finds _____

- ▶ **the first occurrence of a value.** [Click here for detail](#)
- ▶ the second occurrence of a value.
- ▶ may find first or second occurrence of a value.
- ▶ None of the given options.

Question No: 30 (Marks: 1) - Please choose one

Let heap stored in an array as H = [50, 40, 37, 32, 28, 22, 36, 13]. In other words, the root of the heap contains the maximum element. What is the result of deleting 40 from this heap

- ▶ **[50,32, 37,13, 28, 22, 36] according to max heap property.**
- ▶ [37, 28, 32, 22, 36, 13]
- ▶ [37, 36, 32, 28, 13, 22]
- ▶ [37, 32, 36, 13, 28, 22]

FINAL TERM EXAMINATION
Fall 2009

Question No: 1 (Marks: 1) - Please choose one

In an array we can store data elements of different types.

- ▶ True
- ▶ **False**

Question No: 2 (Marks: 1) - Please choose one

Which one of the following statement is NOT correct .

- ▶ In linked list the elements are necessarily to be contiguous
- ▶ **In linked list the elements may locate at far positions in the memory (page 18)**
- ▶ In linked list each element also has the address of the element next to it
- ▶ In an array the elements are contiguous

Question No: 3 (Marks: 1) - Please choose one

Doubly Linked List always has one NULL pointer.

- ▶ True
- ▶ **False(page 39)**

Question No: 4 (Marks: 1) - Please choose one

A queue is a data structure where elements are,

- ▶ inserted at the front and removed from the back. .(see example at page #89 nd 90)
- ▶ inserted and removed from the top.
- ▶ inserted at the back and removed from the front.
- ▶ inserted and removed from both ends.

Question No: 5 (Marks: 1) - Please choose one

Each node in doubly link list has,

- ▶ 1 pointer
- ▶ **2 pointers(page 39)**
- ▶ 3 pointers
- ▶ 4 pointers

Question No: 6 (Marks: 1) - Please choose one

I have implemented the queue with a linked list, keeping track of a front pointer and a rear pointer. Which of these pointers will change during an insertion into an *EMPTY* queue?

- ▶ Neither changes
- ▶ Only front pointer changes.
- ▶ Only rear pointer changes.
- ▶ **Both change.**

Since it is an empty queue the front and rear are initialize to -1, so on insertion both the pointers will change and point to 0.

Question No: 7 (Marks: 1) - Please choose one

Compiler uses which one of the following to evaluate a mathematical equation,

- ▶ Binary Tree
- ▶ Binary Search Tree
- ▶ **Parse Tree(page 274)**
- ▶ AVL Tree

Question No: 8 (Marks: 1) - Please choose one

If a complete binary tree has n number of nodes then its height will be,

- ▶ **$\log_2(n+1) - 1$ (page 139)**
- ▶ 2^n
- ▶ $\log_2(n) - 1$
- ▶ $2^n - 1$

Question No: 9 (Marks: 1) - Please choose one

If a complete binary tree has height h then its no. of nodes will be,

- ▶ Log (h)
- ▶ $2^{h+1} - 1$ (page 324)
- ▶ Log (h) - 1
- ▶ $2^h - 1$

Question No: 10 (Marks: 1) <http://vustudents.ning.com> - Please choose one

A binary relation R over S is called an equivalence relation if it has following property(s)

- ▶ Reflexivity
- ▶ Symmetry
- ▶ Transitivity
- ▶ All of the given options (page 387)

Question No: 11 (Marks: 1) - Please choose one

Binary Search is an algorithm of searching, used with the _____ data.

- ▶ Sorted (page 432)
- ▶ Unsorted
- ▶ Heterogeneous
- ▶ Random

Question No: 12 (Marks: 1) - Please choose one

If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is _____ .

- ▶ N
- ▶ N^2
- ▶ $N \log_2 N$
- ▶ $\log_2 N$ (page 440)

Question No: 13 (Marks: 1) - Please choose one

Use of binary tree in compression of data is known as _____ .

- ▶ Traversal
- ▶ Heap
- ▶ Union
- ▶ Huffman encoding (page 287)

Question No: 14 (Marks: 1) - Please choose one

While building Huffman encoding tree the new node that is the result of joining two nodes has the frequency.

- ▶ Equal to the small frequency
- ▶ Equal to the greater
- ▶ Equal to the sum of the two frequencies (page 293)
- ▶ Equal to the difference of the two frequencies

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 15 (Marks: 1) - Please choose one

Which of the following statements is correct property of binary trees?

- ▶ A binary tree with N internal nodes has N+1 internal links.
- ▶ A binary tree with N external nodes has 2N internal nodes.
- ▶ **A binary tree with N internal nodes has N+ 1 external node. (page 303)**
- ▶ None of above statement is a property of the binary tree.

Question No: 16 (Marks: 1) - Please choose one

Which of the following is a property of binary tree?

- ▶ A binary tree of N external nodes has N internal node.
- ▶ **A binary tree of N internal nodes has N+ 1 external node. (page 303)**
- ▶ A binary tree of N external nodes has N+ 1 internal node.
- ▶ A binary tree of N internal nodes has N- 1 external node.

Question No: 17 (Marks: 1) - Please choose one

Which of the following statement is correct?

- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER successor.
- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its PREOREDR successor.
- ▶ **A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. (Page 307)**
- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its POSTORDER successor.

Question No: 18 (Marks: 1) - Please choose one

Which of the following statement is correct?

- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER successor.
- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its PREOREDR successor.
- ▶ **A Threaded Binary Tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER predecessor.**
- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its POSTORDER predecessor.

Question No: 19 (Marks: 1) - Please choose one

A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its _____ successor.

- ▶ levelorder
- ▶ Preorder

- ▶ **Inorder** [Click here for detail](#)
- ▶ Postorder

Question No: 20 (Marks: 1) - Please choose one

Which of the following statement is true about dummy node of threaded binary tree?

- ▶ This dummy node never has a value.
- ▶ This dummy node has always some dummy value.
- ▶ **This dummy node has either no value or some dummy value. .(page 321)**
- ▶ This dummy node has always some integer value.

Question No: 21 (Marks: 1) - Please choose one

A complete binary tree is a tree that is _____ filled, with the possible exception of the bottom level.

- ▶ partially
- ▶ **completely (page 323)**
- ▶ incompletely
- ▶ partly

Question No: 22 (Marks: 1) - Please choose one

A complete binary tree of height 3 has between _____ nodes.

- ▶ 8 to 14
- ▶ **8 to 15 (page 124)**
- ▶ 8 to 16
- ▶ 8 to 17

Question No: 23 (Marks: 1) - Please choose one

We can build a heap in _____ time.

- ▶ **Linear (page 353)**
- ▶ Exponential
- ▶ Polynomial
- ▶ None of the given options

Question No: 24 (Marks: 1) - Please choose one

Suppose that a selection sort of 100 items has completed 42 iterations of the main loop. How many items are now guaranteed to be in their final spot (never to be moved again)?

- ▶ 21
- ▶ 41
- ▶ **42** [Click here for detail](#)
- ▶ 43

Question No: 25 (Marks: 1) - Please choose one

Suppose you implement a Min heap (with the smallest element on top) in an array. Consider the different arrays below; determine the one that *cannot* possibly be a heap:

- ▶ 16, 18, 20, 22, 24, 28, 30

- ▶ 16, 20, 18, 24, 22, 30, 28
- ▶ 16, 24, 18, 28, 30, 20, 22
- ▶ **16, 24, 20, 30, 28, 18, 22** **It's not satisfy the min heap property**

Question No: 26 (Marks: 1) - Please choose one

Which of the following statement is NOT correct about find operation:

- ▶ It is not a requirement that a find operation returns any specific name, just that finds on two elements return the same answer if and only if they are in the same set.
- ▶ **One idea might be to use a tree to represent each set, since each element in a tree has the same root, thus the root can be used to name the set.**
- ▶ Initially each set contains one element.
- ▶ Initially each set contains one element and it does not make sense to make a tree of one node only.

Question No: 27 (Marks: 1) - Please choose one

Consider the following infix expression:

$$x - y * a + b / c$$

Which of the following is a correct equivalent expression(s) for the above?

- ▶ $x \ y \ -a \ * \ b \ +c \ /$
- ▶ $x \ *y \ a \ - \ b \ c \ / \ +$
- ▶ **$x \ y \ a \ * \ - \ b \ c \ / \ +$** **Hint :- $(x - y * a) + (b / c)$**
- ▶ $x \ y \ a \ * \ - \ b / \ + \ c$

Question No: 28 (Marks: 1) - Please choose one

A complete binary tree of height ____ has nodes between 16 to 31 .

- ▶ 2
- ▶ 3
- ▶ **4 (page 124)**
- ▶ 5

Question No: 29 (Marks: 1) - Please choose one

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

The array after the FIRST iteration of the large loop in a selection sort (sorting from smallest to largest).

- ▶ **0 3 8 9 1 7 5 2 6 4 (Page 477)**
- ▶ 2 6 4 0 3 8 9 1 7 5
- ▶ 2 6 4 9 1 7 0 3 8 5
- ▶ 0 3 8 2 6 4 9 1 7 5

Question No: 30 (Marks: 1) - Please choose one

What requirement is placed on an array, so that *binary search* may be used to locate an entry?

- ▶ The array elements must form a heap.

- ▶ The array must have at least 2 entries.
- ▶ **The array must be sorted.** [Click here for detail](#)
- ▶ The array's size must be a power of two.

AL-JUNAID TECH INSTITUTE



CS301-Data Structure

(Solved Macq's)

**LECTURE FROM
(23 to 45)**



Junaidfazal08@gmail.com
Bc190202640@vu.edu.pk

FOR MORE VISIT
VULMSHELP.COME

JUNAID MALIK
0304-1659294

AL-JUNAID TECH INSTITUTE



www.vulmshelp.com



Language Courses Training Available

I'm providing paid courses in different languages within 3 Months, Certificate will be awarded after completion.

- HTML
- CSS
- JAVASCRIPT
- BOOTSTRAPS
- IQUERY
- PHP MYSQL
- NODES.JS
- REACT JS

LMS Handling Services

LMS Activities Paid Task

Assignments 95% Results

Quizes 95% Results

GDB 95% Results

For CS619 Project Feel Free To Contact With Me

Ph# 0304-1659294
Email: junaidfazal08@gmail.com

AL-JUNAID TECH INSTITUTE

- 1) If there are 100 elements in a heap and 100 delete Min operation are performed, will get _____ list
 - a. **Sorted**
 - b. Unsorted
 - c. Nonlinear
 - d. Noe
- 2) Sorting procedure normally takes _____ times
 - a) **NLogN**
 - b) 2N
 - c) N*N*N
 - d) N
- 3) The expression `if(!heap-> is empty ())` Checks
 - a) Heap is empty
 - b) Heap is full
 - c) **Heap is not empty**
- 4) If the height of a perfect binary tree is 4. What will be the total number of nodes in it?
 - a) 15
 - b) 16
 - c) **31**
 - d) 32
- 5) A binary relation R over S is called an equivalence relation if it has following property(S)?
 - a) Reflexivity
 - b) Symmetry
 - c) Transitivity
 - d) **All of the given**
- 6) If a tree has 20 edges/links, then the total number of nodes in the tree will be:
 - a) 19
 - b) 20
 - c) **21**
 - d) Cannot be determined
- 7) For a perfect binary tree of height 4, what will be the sum of highest of node
 - a) 31
 - b) 30
 - c) 27
 - d) **26**
- 8) If Ahmed is cousin of Ali and Ali is cousin of Asad then Ahmed is also cousin of Asad. This statement has the following property
 - a) Reflexivity
 - b) Symmetry
 - c) **Transitivity**
 - d) All of the above
- 9) Which property of equivalence relation is satisfied if we say:
Ahmad is cousin of Ali and Ali is also cousin of Ahmed

AL-JUNAID TECH INSTITUTE

- a) Reflexivity
 - b) Symmetry
 - c) Transitivity
 - d) All of the given
- 10) Which one of the following is NOT the property of equivalence relation?
- a) Reflexive
 - b) Symmetric
 - c) Transitive
 - d) Associative
- 11) The main reason of using heap in priority queue is
- a) Improve performance
 - b) Code readable
 - c) Less code
 - d) Heap can't be used in priority queues
- 12) The total number of nodes on 10th level of perfect binary tree are
- a) 256
 - b) 512
 - c) 1024
 - d) Can't be determined
- 13) Suppose there are 100 elements in an equivalence class, so initially there will be 100 trees, the collection of these trees is called _____.
- a) Cluster
 - b) Class
 - c) Forest
 - d) Bunch
- 14) The percolate Down procedure will move the smaller value _____ and bigger value _____.
- a) Left, right
 - b) Right, left
 - c) Down, up
 - d) Up, down
- 15) For a perfect binary tree of height h , having N nodes, the Sum of height of nodes is _____
- a) $N - h - 1$
 - b) $N - 1$
 - c) $N - 1 + h$
 - d) $N - (h - 1)$
- 16) Which of the following method is helpful in creating the heap at on

AL-JUNAID TECH INSTITUTE

- a. Insert
- b. Add
- c. Update
- d. **percolateDown**

17. If ahmad is boss of Ahsan and ehsan is boss of umer then ahmad is also boss of umer, the above mentioned relation is _____.

- a. Reflexive
- b. Symmetry
- c. **Transitive**
- d. None of given

18. If we want to find 3rd minimum element from an array of element, then after applying build heap method. How many times deleteMin method will be called?

- e. 1
- f. 2
- g. **3**
- h. 4

19

If we want to find median of 50 elements, then after applying builtHeap method, how many time deleteMin method will be called?

- i. 5
- j. **25**
- k. 35
- l. 50

20 Which of the following properties are satisfied by equivalence relationship?

- m. Reflexive, symmetric
- n. Reflexive, transitive
- o. Symmetric, transitive
- p. **Reflexive, symmetric and transitive**

21. The Expression if (! heap->isFull()) Check

- q. Heap is empty
- r. Heap is full
- s. Heap is not empty
- t. **Heap is not full**

AL-JUNAID TECH INSTITUTE

- 22 Given the values are the array representation of heap:
12 23 26 31 34 44 56 64 78 100
If we perform 4 deleteMin operation, the last element deleted is _____.
a. 31
b. 34
c. 44
d. 56
- 23 Which of the following heap method increase the value if key at position 'p' by the amount 'delta'?
e. increaseKey(p, delta)
f. decreaseKey(p, delta)
g. percolateDown(p, delta)
h. remove(p, delta)
- 24 Which property of equivalence relation is satisfied if we say: Ahmad R(is related to)Ahmad
i. Reflexivity
j. Symmetry
k. Transitivity
l. All of Above
- 25 The total number of nodes on 5th level of perfect binary tree are:
m. 16
n. 15
o. 31
p. 32
- 26 Which property of equivalence relation is satisfied if we say: Ahmad is cousin of Ali and Ali is also Cousin of Ahmad
a. Reflexivity
b. Symmetry
c. Transitivity
d. All of the Above
- 27 If a tree has 50 nodes , then the total edges/links in the tree will be
a. 55
b. 51
c. 50
d. 49
- 28 If the height of perfect binary tree is 4, what will be the total number of nodes in it?
a. 15
b. 16
c. 31
d. 32

AL-JUNAID INSTITUTE GROUP

29 Suppose there are set of fruits and the set of vegetables, both sets are _____ sets.

- a. **Disjoint**
- b. Subset
- c. Whole
- d. Equal

30 A binary relation R over S is called an equivalence relation if it has following property(s)

- a. Reflexivity
- b. Symmetry
- c. Transitivity
- d. **All of Above**

31 Heap can be used to implement

- a. Stack
- b. Linked list
- c. Queue
- d. **Priority queue**

32 If a tree has 20 edges/links, then the total number of nodes in the tree will be:

- a. 19
- b. 20
- c. **21**
- d. Can't be determined

33 If there are 100 elements in an equivalence class, then we will have _____ sets initially.

- a. 50
- b. **100**
- c. 1000
- d. 80

34 Given the values are the array representation of heap; 12 23 26 31 34
44 56 64 78 100

What is the 5th smallest element in the given heap?

- a. 31
- b. **34**
- c. 44
- d. 57

35. A solution is said to be efficient if it solves the problem within its resource constraints i.e. hardware and time.

- ▶ **True (Page 4)**
- ▶ False

36 Which one of the following is known as "Last-In, First-Out" or LIFO Data Structure?

- ▶ Linked List
- ▶ **Stack (Page 54)**
- ▶ Queue
- ▶ Tree

37 What will be postfix expression of the following infix expression?

infix Expression: $a+b*c-d$

- ▶ $ab+c*d-$
- ▶ **$abc*+d-$**

AL-JUNAID INSTITUTE GROUP

- ▶ $abc+*d-$
- ▶ $abcd+*-$

38 For compiler a postfix expression is easier to evaluate than infix expression?

- ▶ **True**
- ▶ False

39 Consider the following pseudo code

declare a stack of characters

while (there are more characters in the word to read)

```
{  
  read a character  
  push the character on the stack  
}
```

while (the stack is not empty)

```
{  
  pop a character off the stack  
  write the character to the screen  
}
```

What is written to the screen for the input "apples"?

- ▶ selpa
- ▶ **selppa**
- ▶ apples
- ▶ aaappppplleess

40 A binary tree of N nodes has_____.

- ▶ $\log_{10} N$ levels
- ▶ **$\log_2 N$ levels (Page 212)**
- ▶ $N / 2$ levels
- ▶ $N \times 2$ levels

41 The easiest case of deleting a node from BST is the case in which the node to be deleted_____.

- ▶ **Is a leaf node (Page 173)**
- ▶ Has left subtree only
- ▶ Has right subtree only
- ▶ Has both left and right subtree

42 If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is_.

- ▶ N
- ▶ N^2
- ▶ $N \log_2 N$
- ▶ **$\log_2 N$ (page 440)**

AL-JUNAID INSTITUTE GROUP

43 Merge sort and quick sort both fall into the same category of sorting algorithms. What is this category?

- ▶ $O(n \log n)$ sorts
- ▶ Interchange sort (not sure)
- ▶ Average time is quadratic
- ▶ **None of the given options. (Page 488)**

44 If one pointer of the node in a binary tree is NULL then it will be a/an_____.

- ▶ **External node (Page 303)**
- ▶ Root node
- ▶ Inner node
- ▶ Leaf node

45 We convert the_____ pointers of binary to threads in threaded binary tree.

- ▶ Left
- ▶ Right
- ▶ **NULL (Page 312)**
- ▶ None of the given options

46 If the bottom level of a binary tree is NOT completely filled, depicts that the tree is NOT

- ▶ Expression tree
- ▶ Threaded binary tree
- ▶ **complete Binary tree (Page 323)**
- ▶ Perfectly complete Binary tree

47 What is the best definition of a *collision* in a hash table?

- ▶ Two entries are identical except for their keys.
- ▶ Two entries with different data have the exact same key
- ▶ **Two entries with different keys have the same exact hash value. (page 464)**
- ▶ Two entries with the exact same key have different hash values.

48 Suppose that a selection sort of 100 items has completed 42 iterations of the main loop. How many items are now guaranteed to be in their final spot (never to be moved again)

- ▶ 21
- ▶ 41
- ▶ **42**
- ▶ 43

49 .Suppose you implement a Min heap (with the smallest element on top) in an array. Consider the different arrays below; determine the one that *cannot* possibly be a heap:

- ▶ 16, 18, 20, 22, 24, 28, 30
- ▶ 16, 20, 18, 24, 22, 30, 28
- ▶ 16, 24, 18, 28, 30, 20, 22
- ▶ **16, 24, 20, 30, 28, 18, 22 (page 334)**

AL-JUNAID INSTITUTE GROUP

50 Do you see any problem in the code of nextInOrder below: `TreeNode * nextInOrder(TreeNode * p)`

```
{
    if(p->RTH == thread)
        return( p->R );
    else {
        p = p->R;
        while(p->LTH == child)
            p = p->R;
        return p;
    }
}
```

- ▶ The function has no problem and will fulfill the purpose successfully.
- ▶ The function cannot be compile as it has syntax error.
- ▶ The function has logical problem, therefore, it will not work properly.
- ▶ The function will be compiled but will throw runtime exception immediately after the control is transferred to this function.

51. Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

The array after the FIRST iteration of the large loop in a selection sort (sorting from smallest to largest).

- ▶ 0 3 8 9 1 7 5 2 6 4 (Page 477)
- ▶ 2 6 4 0 3 8 9 1 7 5
- ▶ 2 6 4 9 1 7 0 3 8 5
- ▶ 0 3 8 2 6 4 9 1 7 5

52. Which one of the following operations returns top value of the stack?

- ▶ Push
- ▶ Pop
- ▶ **Top (page 53)**
- ▶ First

53. Which one of the following is NOT true regarding the skip list?

- ▶ Each list S_i contains the special keys + infinity and - infinity.
List S_0 contains the keys of S in non-decreasing order.
Each list is a subsequence of the previous one.
- ▶ **List S_h contains only the n special keys. (page 446)**

54 By using _____ we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

- ▶ Binary tree only
- ▶ **Threaded binary tree (page 306)**
- ▶ Heap data structure

AL-JUNAID INSTITUTE GROUP

- ▶ Huffman encoding

55 What is the best definition of a *collision* in a hash table?

- ▶ Two entries are identical except for their keys.
- ▶ Two entries with different data have the exact same key
- ▶ **Two entries with different keys have the same exact hash value. (page 464)**
- ▶ Two entries with the exact same key have different hash values.

56 Which formula is the best approximation for the depth of a heap with n nodes?

- ▶ **$\log_2 n$ (page 353)**
- ▶ The number of digits in n (base 10), e.g., 145 has three digits
- ▶ The square root of n
- ▶ n

57 which of the following is not true regarding the maze generation?

- ▶ Randomly remove walls until the entrance and exit cells are in the same set.
- ▶ Removing a wall is the same as doing a union operation.
- ▶ **Remove a randomly chosen wall if the cells it separates are already in the same set. (page 424)**
- ▶ Do not remove a randomly chosen wall if the cells it separates are already in the same set.

58 Which of the given option is NOT a factor in Union by Size:

- ▶ Maintain sizes (number of nodes) of all trees, and during union.
- ▶ Make smaller tree, the subtree of the larger one.
- ▶ **Make the larger tree, the subtree of the smaller one. (page 408)**
- ▶ Implementation: for each root node i , instead of setting $\text{parent}[i]$ to -1 , set it to $-k$ if tree rooted at i has k nodes.

59. when we have declared the size of the array, it is not possible to increase or decrease it during the of the program.

- ▶ Declaration
- ▶ **Execution (page 17)**
- ▶ Defining
- ▶ None of the above

60. it will be efficient to place stack elements at the start of the list because insertion and removal take _____ time.

- ▶ Variable
- ▶ **Constant (page 60)**
- ▶ Inconsistent
- ▶ None of the above

61. is the stack characteristic but _____ was implemented because of the size limitation of the array.

AL-JUNAID INSTITUTE GROUP

- ▶ isFull(), isEmpty()
- ▶ pop(), push()
- ▶ **isEmpty(), isFull() (page 59)**
- ▶ push(), pop().

62, What kind of list is best to answer questions such as "What is the item at position n?"

- ▶ **Lists implemented with an array.**
- ▶ Doubly-linked lists.
- ▶ Singly-linked lists.
- ▶ Doubly-linked or singly-linked lists are equally best

63 Each node in doubly link list has,

- ▶ 1 pointer
- ▶ **2 pointers (page 39)**
- ▶ 3 pointers
- ▶ 4 pointers

64 If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ **57 (page 303)**
- ▶ x R z

65 A simple sorting algorithm like selection sort or bubble sort has a worst-case of

- ▶ $O(1)$ time because all lists take the same amount of time to sort
- ▶ $O(n)$ time because it has to perform n swaps to order the list.
- ▶ **$O(n^2)$ time because sorting 1 element takes $O(n)$ time - After 1 pass through the list, either of these algorithms can guarantee that 1 element is sorted. (page 487)**
- ▶ $O(n^3)$ time, because the worst case has really random input which takes longer to sort.

66 Merge sort and quick sort both fall into the same category of sorting algorithms. What is this category?

- ▶ $O(n \log n)$ sorts
- ▶ Interchange sort
- ▶ Average time is quadratic
- ▶ **None of the given options. (Page 488)**

67 Huffman encoding uses _____ tree to develop codes of varying lengths for the letters used in the original message.

- ▶ Linked list
- ▶ Stack
- ▶ Queue
- ▶ **Binary tree (page 287)**

AL-JUNAID INSTITUTE GROUP

68 Consider a min heap, represented by the following array:

10,30,20,70,40,50,80,60

After inserting a node with value 31. Which of the following is the updated min heap?

▶ 10,30,20,31,40,50,80,60,70 (page 336)

▶ 10,30,20,70,40,50,80,60,31

▶ 10,31,20,30,40,50,80,60,31

▶ 31,10,30,20,70,40,50,80,60

69 Consider a min heap, represented by the following array:

11,22,33,44,55

After inserting a node with value 66. Which of the following is the updated min heap?

▶ 11,22,33,44,55,66 (page 336)

▶ 11,22,33,44,66,55

▶ 11,22,33,66,44,55

▶ 11,22,66,33,44,55

70 Suppose that a selection sort of 100 items has completed 42 iterations of the main loop. How many items are now guaranteed to be in their final spot (never to be moved again)?

▶ 21

▶ 41

▶ 42

▶ 43

71. Is a data structure that can grow easily dynamically at run time without having to copy existing elements.

▶ Array ()

▶ List

▶ Both of these (page 10)

▶ None of these

72. A complete binary tree of height ____ has nodes between 16 to 31 .

▶ 2

▶ 3

▶ 4 (page 373)

▶ 5

73. Which of the given option is NOT a factor in Union by Size:

▶ Maintain sizes (number of nodes) of all trees, and during union.

▶ Make smaller tree, the subtree of the larger one.

▶ Make the larger tree, the subtree of the smaller one. (page 408)

▶ Implementation: for each root node i , instead of setting $\text{parent}[i]$ to -1 , set it to $-k$ if tree rooted at i has k nodes.

74. Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

AL-JUNAID INSTITUTE GROUP

The array after the FIRST iteration of the large loop in a selection sort (sorting from smallest to largest).

- ▶ 0 3 8 9 1 7 5 2 6 4 (Page 477)
- ▶ 2 6 4 0 3 8 9 1 7 5
- ▶ 2 6 4 9 1 7 0 3 8 5
- ▶ 0 3 8 2 6 4 9 1 7 5

75. Suppose A is an array containing numbers in increasing order, but some numbers occur more than once when using a binary search for a value, the binary search always finds

- ▶ the first occurrence of a value.
- ▶ the second occurrence of a value.
- ▶ may find first or second occurrence of a value.
- ▶ None of the given options.

76. A binary tree with 24 internal nodes has ___ external nodes. 22

- ▶ 23
- ▶ 48
- ▶ 25 (page 303)

77. it will be efficient to place stack elements at the start of the list because insertion and removal _____ take time.

- ▶ Variable
- ▶ Constant (page 60)
- ▶ Inconsistent
- ▶ None of the above

78. "+" is a _____ operator.

- ▶ Unary
- ▶ Binary (page 64)
- ▶ Ternary
- ▶ None of the above

79. A kind of expressions where the operator is present between two operands called _____ expressions.

- ▶ Postfix
- ▶ Infix (page 64)
- ▶ Prefix
- ▶ None of the above.

80. Here is a small function definition:

```
void f(int i, int &k)
{
    i = 1;
    k = 2;
}
```

AL-JUNAID INSTITUTE GROUP

Suppose that a main program has two integer variables x and y , which are given the value 0. Then the main program calls $f(x,y)$; What are the values of x and y after the function f finishes?

- ▶ Both x and y are still 0.
- ▶ x is now 1, but y is still 0.
- ▶ **x is still 0, but y is now 2.**
- ▶ x is now 1, and y is now 2.

81. A binary tree with N internal nodes has ___ links, ___ links to internal nodes and ___ links to external nodes

- ▶ $N+1, 2N, N-1$
- ▶ $N+1, N-1, 2N$
- ▶ **$2N, N-1, N+1$ (page 304)**
- ▶ $N-1, 2N, N+1$

82. Each node in doubly link list has,

- ▶ 1 pointer
- ▶ **2 pointers (Page 39)**
- ▶ 3 pointers
- ▶ 4 pointers

83. If you know the size of the data structure in advance, i.e., at compile time, which one of the following is a good data structure to use.

- ▶ Array
- ▶ List
- ▶ **Both of these (page 10)**
- ▶ None of these

84. Which one is a self-referential data type?

- ▶ Stack
- ▶ Queue
- ▶ **Link list**
- ▶ All of these

85. There is/are ___ case/s for rotation in an AVL tree,

- ▶ 1
- ▶ 3
- ▶ 2
- ▶ **4 (page 229)**

86. Which of the following can be the inclusion criteria for pixels in image segmentation.

- ▶ Pixel intensity
- ▶ Texture
- ▶ Threshold of intensity

AL-JUNAID INSTITUTE GROUP

- ▶ All of the given options (page 421)

87. In a perfectly balanced tree the insertion of a node needs__.

- ▶ One rotation (Page 225)
- ▶ Two rotations
- ▶ Rotations equal to number of levels
- ▶ No rotation at all

88; If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is__.

- ▶ N
- ▶ N^2
- ▶ $N\log_2 N$
- ▶ $\log_2 N$ (page 440)

89. Which of the following is NOT a correct statement about Table ADT.

- ▶ In a table, the type of information in columns may be different.
- ▶ A table consists of several columns, known as entities. (page 408)
- ▶ The row of a table is called a record.
- ▶ A major use of table is in databases where we build and use tables for keeping information.

90. Suppose we are sorting an array of eight integers using quick sort, and we have just finished the first partitioning with the array looking like this:

2 5 1 7 9 12 11 10

Which statement is correct?

- ▶ The pivot could be either the 7 or the 9.(page 506)
- ▶ The pivot could be the 7, but it is not the 9.
- ▶ The pivot is not the 7, but it could be the 9.
- ▶ Neither the 7 nor the 9 is the pivot.

91 What is the best definition of a *collision* in a hash table?

- ▶ Two entries are identical except for their keys.
- ▶ Two entries with different data have the exact same key
- ▶ Two entries with different keys have the same exact hash value. (page 464)
- ▶ Two entries with the exact same key have different hash values.

92. For a perfect binary tree of height h, having N nodes, the sum of heights of nodes is

- ▶ $N - (h - 1)$
- ▶ $N - (h + 1)$ (Page 373)
- ▶ $N - 1$
- ▶ $N - 1 + h$

93.A binary tree with 33 internal nodes has____links to internal nodes.

AL-JUNAID INSTITUTE GROUP

- ▶ 31
- ▶ **32 (Page 304)**
- ▶ 33
- ▶ 66

94. Suppose you implement a Min heap (with the smallest element on top) in an array. Consider the different arrays below; determine the one that *cannot* possibly be a heap:

- ▶ 16, 18, 20, 22, 24, 28, 30
- ▶ 16, 20, 18, 24, 22, 30, 28
- ▶ 16, 24, 18, 28, 30, 20, 22
- ▶ **16, 24, 20, 30, 28, 18, 22 (see min heap property at page 337)**

95. Which of the following is not true regarding the maze generation?

- ▶ Randomly remove walls until the entrance and exit cells are in the same set.
- ▶ Removing a wall is the same as doing a union operation.
- ▶ **Remove a randomly chosen wall if the cells it separates are already in the same set. (Page 424)**
- ▶ Do not remove a randomly chosen wall if the cells it separates are already in the same set.

96. Which formula is the best approximation for the depth of a heap with n nodes?

- ▶ **$\log(\text{base } 2) \text{ of } n$ (Page 353)**
- ▶ The number of digits in n (base 10), e.g., 145 has three digits
- ▶ The square root of n
- ▶ n

97. The _____ method of list will position the *currentNode* and *lastCurrentNode* at the start of the list.

- ▶ Remove
- ▶ Next
- ▶ **Start (Page 38)**
- ▶ Back

98. Mergesort makes two recursive calls. Which statement is true after these recursive calls finish, but before the merge step?

- ▶ Elements in the first half of the array are less than or equal to elements in the second half of the array.
- ▶ None of the given options.
- ▶ The array elements form a heap.
- ▶ **Elements in the second half of the array are less than or equal to elements in the first half of the array.**

AL-JUNAID INSTITUTE GROUP

99 The arguments passed to a function should match in number, type and order with the parameters in the function definition.

- ▶ True
- ▶ False

Question No: 100

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

- ▶ 48
- ▶ 4
- ▶ 222
- ▶ **5**

Question No: 101

Suppose currentNode refers to a node in a linked list (using the Node class with member variables called data and nextNode). What statement changes currentNode so that it refers to the next node?

- ▶ currentNode ++;
- ▶ currentNode = nextNode;
- ▶ currentNode += nextNode;
- ▶ **currentNode = currentNode->nextNode;**

Question No: 102

A **Compound Data Structure** is the data structure which can have multiple data items of same type or of different types. Which of the following can be considered compound data structure?

- ▶ Arrays
- ▶ LinkLists
- ▶ Binary Search Trees
- ▶ **All of the given options**

Question No: 103

Here is a small function definition:

```
void f(int i, int &k)
{
    i = 1;
    k = 2;
}
```

AL-JUNAID INSTITUTE GROUP

Suppose that a main program has two integer variables x and y, which are given the value 0. Then the main program calls f(x,y); What are the values of x and y after the function f finishes?

- ▶ Both x and y are still 0.
- ▶ x is now 1, but y is still 0.
- ▶ **x is still 0, but y is now 2.**
- ▶ x is now 1, and y is now 2.

Question No: 104

The difference between a binary tree and a binary search tree is that ,

- ▶ **a binary search tree has two children per node whereas a binary tree can have none, one, or two children per node**
- ▶ in binary search tree nodes are inserted based on the values they contain
- ▶ in binary tree nodes are inserted based on the values they contain
- ▶ none of these

Question No: 105

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ **57 (Page 303)**

Question No: 106

If there are 23 external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ 23
- ▶ 24
- ▶ 21
- ▶ **22 (n-1) (Page 306)**

Question No: 107

Which of the following method is helpful in creating the heap at once?

- ▶ insert
- ▶ add
- ▶ update
- ▶ **preculatDown (Page 370)**

AL-JUNAID INSTITUTE GROUP

Question No: 108

If both pointers of the node in a binary tree are NULL then it will be a/an_____.

- ▶ Inner node
- ▶ **Leaf node (Page 311)**
- ▶ Root node
- ▶ None of the given optio

Question No: 108

By using_____we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

- ▶ Binary tree only
- ▶ **Threaded binary tree (page 306)**
- ▶ Heap data structure
- ▶ Huffman encoding

Question No: 109

A complete binary tree of height 3 has between _____nodes.

- ▶ 8 to 14
- ▶ **8 to 15 (Page 124)**
- ▶ 8 to 16
- ▶ 8 to 17

$$2^{(d+1)} - 1 = 2^{(3+1)} - 1 = 2^4 - 1 = 16 - 1 = 15$$

Question No: 110

Consider a min heap, represented by the following array:

3,4,6,7,5,10

After inserting a node with value 1. Which of the following is the updated min heap?

- ▶ 3,4,6,7,5,10,1
- ▶ 3,4,6,7,5,1,10
- ▶ 3,4,1,5,7,10,6
- ▶ **1,4,3,5,7,10,6 close to correct but correct ans is 1,4,3,7,5,10,6 (page 337)**

Question No: 111

Consider a min heap, represented by the following array:

10,30,20,70,40,50,80,60

After inserting a node with value 31. Which of the following is the updated min heap?

- ▶ **10,30,20,31,40,50,80,60,70 (page 337)**
- ▶ 10,30,20,70,40,50,80,60,31
- ▶ 10,31,20,30,40,50,80,60,31
- ▶ 31,10,30,20,70,40,50,80,60

Question No: 112

Which one of the following algorithms is most widely used due to its good average time,

- ▶ Bubble Sort

AL-JUNAID INSTITUTE GROUP

- ▶ Insertion Sort
- ▶ **Quick Sort**
- ▶ Merge Sort

Question No: 113

The following are statements related to queues.

The last item to be added to a queue is the first item to be removed

A queue is a structure in which both ends are not used

The last element hasn't to wait until all elements preceding it on the queue are removed

queue is said to be a last-in-first-out list or LIFO data structure.

Which of the above is/are related to normal queues?

- ▶ (iii) and (ii) only
- ▶ (i), (ii) and (iv) only
- ▶ (ii) and (iv) only
- ▶ **None of the given options**

Question No: 114

In complete binary tree the bottom level is filled from _____

- ▶ **Left to right (Page 323)**
- ▶ Right to left
- ▶ Not filled at all
- ▶ None of the given options

Question No: 115

We are given N items to build a heap, this can be done with _____ successive inserts.

- ▶ N-1
- ▶ **N (Page 355)**
- ▶ N+1
- ▶ N^2

Question No: 116

Suppose we had a hash table whose hash function is " $n \% 12$ ", if the number 35 is already in the hash table, which of the following numbers would cause a collision?

- ▶ 144
- ▶ 145
- ▶ **143**
- ▶ 148

Question No: 117

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

The array after the FIRST iteration of the large loop in a selection sort (sorting from smallest to largest).

- ▶ **0 3 8 9 1 7 5 2 6 4 (Page 483)**
- ▶ 2 6 4 0 3 8 9 1 7 5
- ▶ 2 6 4 9 1 7 0 3 8 5
- ▶ 0 3 8 2 6 4 9 1 7 5

AL-JUNAID INSTITUTE GROUP

Question No: 118

What requirement is placed on an array, so that *binary search* may be used to locate an entry?

- ▶ The array elements must form a heap.
- ▶ The array must have at least 2 entries.
- ▶ **The array must be sorted.**
- ▶ The array's size must be a power of two.

Question No: 119

In case of deleting a node from AVL tree, rotation could be prolong to the *root* node.

- ▶ **Yes (Page 266)**
- ▶ No

Question No 119

_____ only removes items in reverse order as they were entered.

- ▶ **Stack (Page 79)**
- ▶ Queue
- ▶ Both of these
- ▶ None of these

Question No:120

Here is a small function definition:

```
void f(int i, int &k)
{
    i = 1;
    k = 2;
}
```

Suppose that a main program has two integer variables x and y, which are given the value 0. Then the main program calls f(x,y); What are the values of x and y after the function f finishes?

- ▶ Both x and y are still 0.
- ▶ x is now 1, but y is still 0.
- ▶ **x is still 0, but y is now 2.**
- ▶ x is now 1, and y is now 2.

Question No:121

Select the one *FALSE* statement about binary trees:

- ▶ **Every binary tree has at least one node. (Page 109)**
- ▶ Every non-empty tree has exactly one root node.
- ▶ Every node has at most two children.
- ▶ Every non-root node has exactly one parent.

Question No: 122

Searching an element in an AVL tree take maximum _____ time (where n is no. of nodes in AVL tree),

- ▶ $\log_2(n+1)$
- ▶ $\log_2(n+1) - 1$
- ▶ **$1.44 \log_2 n$ (Page 235)**
- ▶ $1.66 \log_2 n$

AL-JUNAID INSTITUTE GROUP

Question No: 123

Suppose that we have implemented a *priority queue* by storing the items in a heap. We are now executing a reheapification downward and the out-of-place node has priority of 42. The node's parent has a priority of 72, the left child has priority 52 and the node's right child has priority 62. Which statement best describes the status of the reheapification.

- ▶ The reheapification is done.
- ▶ The next step will interchange the two children of the out-of-place node.
- ▶ The next step will swap the out-of-place node with its parent.
- ▶ The next step will swap the out-of-place node with its left child.

Question No: 124

Suppose you implement a heap (with the largest element on top) in an array. Consider the different arrays below, determine the one that *cannot* possibly be a heap:

- ▶ 7 6 5 4 3 2 1
- ▶ 7 3 6 2 1 4 5
- ▶ 7 6 4 3 5 2 1
- ▶ 7 3 6 4 2 5 1

According to max heap property

Question No: 125

Which one of the following is NOT the property of equivalence relation:

- ▶ Reflexive
- ▶ Symmetric
- ▶ Transitive
- ▶ Associative (Page 385)

Question No: 126

The definition of Transitivity property is

- ▶ For all element x member of S , $x R x$
- ▶ For all elements x and y , $x R y$ if and only if $y R x$
- ▶ For all elements x , y and z , if $x R y$ and $y R z$ then $x R z$ (Page 385)
- ▶ For all elements w , x , y and z , if $x R y$ and $w R z$ then $x R z$

Question No: 127

Union is a _____ time operation.

- ▶ Constant (Page 416)
- ▶ Polynomial
- ▶ Exponential
- ▶ None of the given option

Question No: 128

Which of the following is NOT a correct statement about Table ADT.

- ▶ In a table, the type of information in columns may be different. yes
- ▶ A table consists of several columns, known as entities. (Page 437)
- ▶ The row of a table is called a record.
- ▶ A major use of table is in databases where we build and use tables for keeping information.

AL-JUNAID INSTITUTE GROUP

Question No: 129

In the worst case of deletion in AVL tree requires _____.

- ▶ Only one rotation
- ▶ Rotation at each non-leaf node
- ▶ Rotation at each leaf node
- ▶ **Rotations equal to $\log_2 N$**

Question No: 130

By using _____ we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

- ▶ Binary tree only
- ▶ **Threaded binary tree**
- ▶ Heap data structure
- ▶ Huffman encoding

Question No: 131

Consider a min heap, represented by the following array:

11,22,33,44,55

After inserting a node with value 66. Which of the following is the updated min heap?

- ▶ **11,22,33,44,55,66 (page 337)**
- ▶ 11,22,33,44,66,55
- ▶ 11,22,33,66,44,55
- ▶ 11,22,66,33,44,55

Question No: 132

Consider a min heap, represented by the following array:

3,4,6,7,5

After calling the function deleteMin(). Which of the following is the updated min heap?

- ▶ 4,6,7,5
- ▶ 6,7,5,4
- ▶ **4,5,6,7 (page 349)**
- ▶ 4,6,5,7

Question No: 133

We can build a heap in _____ time.

- ▶ **Linear (Page 353)**
- ▶ Exponential
- ▶ Polynomial
- ▶ None of the given options

Question No: 134

Suppose we are sorting an array of eight integers using quick sort, and we have just finished the first partitioning with the array looking like this: □

2 5 1 7 9 12 11 10

Which statement is correct?

- ▶ **The pivot could be either the 7 or the 9. (page 506)**
- ▶ The pivot could be the 7, but it is not the 9.
- ▶ The pivot is not the 7, but it could be the 9
- ▶ Neither the 7 nor the 9 is the pivot.

Question No: 135

AL-JUNAID INSTITUTE GROUP

Which formula is the best approximation for the depth of a heap with n nodes?

- ▶ **log (base 2) of n (Page 353)**
- ▶ The number of digits in n (base 10), e.g., 145 has three digits
- ▶ The square root of n
- ▶ n

Question No 136

Suppose you implement a Min heap (with the smallest element on top) in an array. Consider the different arrays below; determine the one that *cannot* possibly be a heap:

- ▶ 16, 18, 20, 22, 24, 28, 30
- ▶ 16, 20, 18, 24, 22, 30, 28
- ▶ 16, 24, 18, 28, 30, 20, 22
- ▶ **16, 24, 20, 30, 28, 18, 22**

Question No: 137

While joining nodes in the building of Huffman encoding tree if there are more nodes with same frequency, we choose the nodes _____.

- ▶ **Randomly (Page 289)**
- ▶ That occur first in the text message
- ▶ That are lexically smaller among others.
- ▶ That are lexically greater among others

Question No: 138

Consider the following paragraph with blanks.

A is a linear list where and take place at the same end . This end is called the

What would be the correct filling the above blank positions?

- ▶ (i) queue (ii) insertion (iii) removals (iv) top
- ▶ (i) stack (ii) insertion (iii) removals (iv) bottom
- ▶ **(i) stack (ii) insertion (iii) removals (iv) top (Page 52)**
- ▶ (i) tree (ii) insertion (iii) removals (iv) top

0304-1659294
www.vulmshelp.com

AL-JUNAID INSTITUTE GROUP

Question No: 139

A binary tree with 33 internal nodes has _____ links to internal nodes.

- ▶ 31
- ▶ **32 (n-1 links to internal nodes) (Page 304)**
- ▶ 33
- ▶ 66

Question No: 140

Which traversal gives a decreasing order of elements in a heap where the max element is stored at the top?

- ▶ post-order
- ▶ level-order
- ▶ inorder
- ▶ **None of the given options**

Question No: 141

What requirement is placed on an array, so that *binary search* may be used to locate an entry

- ▶ The array elements must form a heap.
- ▶ The array must have at least 2 entries.
- ▶ **The array must be sorted**
- ▶ The array's size must be a power of two.

Question No: 142

Which of the following is a non linear data structure?

- ▶ Linked List
- ▶ Stack
- ▶ Queue
- ▶ **Tree (Page 112)**

Question No: 143

The data of the problem is of 2GB and the hard disk is of 1GB capacity, to solve this problem we should

- ▶ Use better data structures
- ▶ **Increase the hard disk space (Page 5)**
- ▶ Use the better algorithm
- ▶ Use as much data as we can store on the hard disk

Question No: 144

In an array list the current element is

- ▶ **The first element**
- ▶ **The middle element**
- ▶ The last element
- ▶ The element where the current pointer points to

Question No: 145

Which one of the following is a valid postfix expression?

AL-JUNAID INSTITUTE GROUP

- ▶ $ab+cd-$
- ▶ **abc^*+d- (According to rule)**
- ▶ $abc+^*d-$
- ▶ $(abc^*)+d-$

Question No: 146

In sequential access data structure, accessing any element in the data structure takes different amount of time. Tell which one of the following is sequential access data structure,

- ▶ Arrays
- ▶ **Lists**
- ▶ Both of these
- ▶ None of these

Question No: 147

I have implemented the queue with a circular array. If data is a circular array of CAPACITY elements, and last is an index into that array, what is the formula for the index after last?

- ▶ $(last \% 1) + CAPACITY$
- ▶ $last \% (1 + CAPACITY)$
- ▶ **$(last + 1) \% CAPACITY$**
- ▶ $last + (1 \% CAPACITY)$

This expression will point to field after last that will be the first field.

Question No: 148

Which one of the following is TRUE about recursion?

- ▶ **Recursion extensively uses stack memory. (page 149)**
- ▶ Threaded Binary Trees use the concept of recursion.
- ▶ Recursive function calls consume a lot of memory.
- ▶ Iteration is more efficient than iteration.

Question no.149

Which one of the following is TRUE about iteration?

- ▶ Iteration extensively uses stack memory.
- ▶ Threaded Binary Trees use the concept of iteration.
- ▶ Iterative function calls consumes a lot of memory.
- ▶ **Recursion is more efficient than iteration.**

Question No: 150

If a max heap is implemented using a partially filled array called data, and the array contains n elements ($n > 0$), where is the entry with the greatest value? **Data[0] is correct**

- ▶ data[1]
- ▶ data[n-1]
- ▶ data[n]
- ▶ data[2*n+1]

AL-JUNAID INSTITUTE GROUP

Question No: 151

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ **57 (n+1)**

Question No: 152

Which of the following heap method increase the value of key at position „p“ by the amount „delta“?

- ▶ **increaseKey(p,delta) (Page 363)**
- ▶ decreaseKey(p,delta)
- ▶ preculatDown(p,delta)
- ▶ remove(p,delta)

Question No: 153

If we have 1000 sets each containing a single different person. Which of the following relation will be true on each set:

- ▶ **Reflexive (page 388)**
- ▶ Symmetric
- ▶ Transitive
- ▶ Associative

Question No: 154

Which one of the following is not an example of equivalence relation?

- ▶ Electrical connectivity
- ▶ Set of people
- ▶ **$\leq$ relation (Page 385)**
- ▶ Set of pixels

Question No: 155

A binary tree of nodes has_____.

- ▶ $\log_{10} N$ levels
- ▶ **$\log_2 N$ levels (Page 279)**
- ▶ $N / 2$ levels
- ▶ $N \times 2$ levels

Question No: 156

Binary Search is an algorithm of searching, used with the _____ data.

- ▶ **Sorted (Page 428)**
- ▶ Unsorted
- ▶ Heterogeneous
- ▶ Random

Question No: 157

Consider te following array

AL-JUNAID INSTITUTE GROUP

23 15 5 12 40 10 7

After the first pass of a particular algorithm, the array looks like

15 5 12 23 10 7 40

Name the algorithm used

- ▶ Heap sort
- ▶ Selection sort
- ▶ Insertion sort

▶ **Bubble sort (According to rule)**

Question No: 158

Which of the following statements is correct property of binary trees?

- ▶ A binary tree with N internal nodes has $N+1$ internal links.
- ▶ A binary tree with N external nodes has $2N$ internal nodes.

▶ **A binary tree with N internal nodes has $N+1$ external nodes. (page 304)**

- ▶ None of above statement is a property of the binary tree

Question No: 159

If the bottom level of a binary tree is NOT completely filled, depicts that the tree is NOT a

- ▶ Expression tree
- ▶ Threaded binary tree

▶ **complete Binary tree (Page 323)**

- ▶ Perfectly complete Binary tree

Question No: 160

In a selection sort of n elements, how many times the swap function is called to complete the execution of the algorithm?

▶ **$n-1$**

- ▶ $n \log n$
- ▶ n^2
- ▶ 1

Question No: 161

Which of the following statement is correct about find(x) operation:

- ▶ A find(x) on element x is performed by returning exactly the same node that is found.
- ▶ **A find(x) on element x is performed by returning the root of the tree containing x .**
- ▶ A find(x) on element x is performed by returning the whole tree itself containing x .
- ▶ A find(x) on element x is performed by returning TRUE.

Question No: 162

Which of the following statement is NOT correct about find operation:

▶ It is not a requirement that a find operation returns any specific name, just that finds on two elements return the same answer if and only if they are in the same set.

▶ **One idea might be to use a tree to represent each set, since each element in a tree has the same root, thus the root can be used to name the set.**

- ▶ Initially each set contains one element.
- ▶ **Initially each set contains one element and it does not make sense to make a tree of one node only.**

AL-JUNAID INSTITUTE GROUP

Question No: 163

Consider the following postfix expression S and the initial values of the variables.

$S = A B - C + D E F - + ^$

Assume that $A=3, B=2, C=1, D=1, E=2, F=3$

What would be the final output of the stack?

- ▶ 1
- ▶ 2
- ▶ 0
- ▶ -1

Question No: 164

The maximum number of external nodes (leaves) for a binary tree of height H is _____

- ▶ $2h$
- ▶ 2^{H+1}
- ▶ $2^H - 1$
- ▶ $2^H + 2$

Question No: 165

In threaded binary tree the NULL pointers are replaced by ,

- ▶ preorder successor or predecessor
- ▶ **inorder successor or predecessor (Page 310)**
- ▶ postorder successor or predecessor
- ▶ NULL pointers are not replaced

Question No: 166

In a min heap , preculcateDown procedure will move smaller value _____ and bigger value _____.

- ▶ left,right
- ▶ right,left
- ▶ **up,down (Page 358)**
- ▶ down,up

Question No: 167

Which of the following statement is correct about union:

- ▶ **To perform Union of two sets, we merge the two trees by making the root of one tree point to the root of the other. (Greedy algorithms , Page 7)**
- ▶ To perform Union of two sets, we merge the two trees by making the leaf node of one tree point to the root of the other.
- ▶ To perform Union of two sets, merging operation of trees is not required at all.
- ▶ None of the given options.

AL-JUNAID INSTITUTE GROUP

Question No: 168

Suppose A is an array containing numbers in increasing order, but some numbers occur more than once when using a binary search for a value, the binary search always finds _

- ▶ **the first occurrence of a value.**
- ▶ the second occurrence of a value.
- ▶ may find first or second occurrence of a value.
- ▶ None of the given options.

Question No: 169

Let heap stored in an array as H = [50, 40, 37, 32, 28, 22, 36, 13]. In other words, the root of the heap contains the maximum element. What is the result of deleting 40 from this heap

- ▶ **[50,32, 37,13, 28, 22, 36] according to max heap property.**
- ▶ [37, 28, 32, 22, 36, 13]
- ▶ [37, 36, 32, 28, 13, 22]
- ▶ [37, 32, 36, 13, 28, 22]

Question No: 170

In an array we can store data elements of different types.

- ▶ True
- ▶ **False**

we cannot store multiple datatype in an Array, we can store similar datatype only in an **Array**.

Question no 171

Which one of the following statement is NOT correct?

- ▶ In linked list the elements are necessarily to be contiguous
- ▶ **In linked list the elements may locate at far positions in the memory (page 17)**
- ▶ In linked list each element also has the address of the element next to it
- ▶ In an array the elements are contiguous

Question no 172

Doubly Linked List always has one NULL pointer.

- ▶ True
- ▶ **False(page 450)**

Question No: 173

A queue is a data structure where elements are,

- ▶ **inserted at the front and removed from the back. . (page #89 nd 90)**
- ▶ inserted and removed from the top.
- ▶ inserted at the back and removed from the front.
- ▶ inserted and removed from both ends.

Question No: 174

Each node in doubly link list has,

- ▶ 1 pointer
- ▶ **2 pointers(page 38)**
- ▶ 3 points
- ▶ 4 poiner

AL-JUNAID INSTITUTE GROUP

Question No: 175

I have implemented the queue with a linked list, keeping track of a front pointer and a rear pointer. Which of these pointers will change during an insertion into an *EMPTY* queue?

- ▶ Neither changes
- ▶ Only front pointer changes.
- ▶ Only rear pointer changes.
- ▶ **Both change**

Since it is an empty queue the front and rear are initialize to -1, so on insertion both the pointers will change and point to 0.

Question No: 176

Compiler uses which one of the following to evaluate a mathematical equation,

- ▶ Binary Tree
- ▶ Binary Search Tree
- ▶ **Parse Tree(page 279)**
- ▶ AVL Tree

Question No: 177

If a complete binary tree has n number of nodes then its height will be,

- ▶ **$\log_2(n+1) - 1$ (page 136)**
- ▶ 2^n
- ▶ $\log_2(n) - 1$
- ▶ $2^n - 1$

Question No: 178

If a complete binary tree has height h then its no. of nodes will be,

- ▶ $\log(h)$
- ▶ **$2^{h+1} - 1$**
- ▶ $\log(h) - 1$
- ▶ $2^h - 1$

Question No: 179

A binary relation R over S is called an equivalence relation if it has following property(s)

- ▶ Reflexivity
- ▶ Symmetry
- ▶ Transitivity
- ▶ **All of the given options (page 385)**

Question No: 180

If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is____.

AL-JUNAID INSTITUTE GROUP

- ▶ N
- ▶ N^2
- ▶ $N \log_2 N$
- ▶ **$\log_2 N$ (page 429)**

Question No: 181

Use of binary tree in compression of data is known as_____.

- ▶ Traversal
- ▶ Heap
- ▶ Union
- ▶ **Huffman encoding (page 292)**

Question No: 182

While building Huffman encoding tree the new node that is the result of joining two nodes has the frequency.

- ▶ Equal to the small frequency
- ▶ Equal to the greater
- ▶ **Equal to the sum of the two frequencies (page 293)**
- ▶ Equal to the difference of the two frequencies

Question No: 183

Which of the following statement is correct?

- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER successor.
- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its PREORDER successor.

▶ **A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. (Page 307)**

- ▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its POSTORDER successor.

Question No: 184

A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its_____successor.

- ▶ levelorder
- ▶ Preorder
- ▶ **Inorder**
- ▶ Postorder

Question No: 185

Which of the following statement is true about dummy node of threaded binary tree?

- ▶ This dummy node never has a value.

AL-JUNAID INSTITUTE GROUP

- ▶ This dummy node has always some dummy value.
- ▶ **This dummy node has either no value or some dummy value. .(page 324)**
- ▶ This dummy node has always some integer value.

Question No: 186

A complete binary tree is a tree that is _____ filled, with the possible exception of the bottom level.

- ▶ partially
- ▶ **completely (page 326)**
- ▶ incompletely
- ▶ partly

Question No: 187

A complete binary tree of height 3 has between _____ nodes.

- ▶ 8 to 14
- ▶ **8 to 15 (page 124)**
- ▶ 8 to 16
- ▶ 8 to 17

Question No: 188

We can build a heap in _____ time.

- ▶ **Linear (page 355)**
- ▶ Exponential
- ▶ Polynomial
- ▶ None of the given options

Question No: 189

Suppose that a selection sort of 100 items has completed 42 iterations of the main loop. How many items are now guaranteed to be in their final spot (never to be moved again)?

- ▶ 21
- ▶ 41
- ▶ **42**
- ▶ 43

Question No: 190

Which of the following statement is NOT correct about find operation:

- ▶ It is not a requirement that a find operation returns any specific name, just that finds on two elements return the same answer if and only if they are in the same set.
- ▶ **One idea might be to use a tree to represent each set, since each element in a tree has the same root, thus the root can be used to name the set.**
- ▶ Initially each set contains one element.
- ▶ Initially each set contains one element and it does not make sense to make a tree of one node only.

Question No: 191

(Consider the following infix

AL-JUNAID INSTITUTE GROUP

expression:

$$x - y a + b / c$$

Which of the following is a correct equivalent expression(s) for the above?

- ▶ $x y - a * b + c /$
- ▶ $x * y a - b c / +$
- ▶ **$x y a * - b c / +$** Hint :- $(x - y * a) + (b / c)$
- ▶ $x y a * - b / + c$

Question No: 192

A complete binary tree of height _____ has nodes between 16 to 31 .

- ▶ 2
- ▶ 3
- ▶ **4 (page 124)**
- ▶ 5

Question No: 193

What requirement is placed on an array, so that *binary search* may be used to locate an entry?

- ▶ The array elements must form a heap.
- ▶ **The array must have at least 2 entries**
- ▶ The array must be sorted.
- ▶ The array's size must be a power of two.

Question No: 194

A solution is said to be efficient if it solves the problem within its resource constraints i.e. hardware and time.

- ▶ **False** (page 4...hardware, hard disk and memory)
- ▶ ture



0304-1659294

www.vulmshelp.com

1.

If the elements are 3, 5, 1, 6 in which order they will be removed from the queue one by one

3516

2. The.....of a binary tree is a maximum level of its leaves (also the depth).

height

3. of the an empty AVL tree is define to be

Height

4. Following is the keyword of c++

Delete

5. In the perspective of memory organization every process expecting the first part of memory is for.....of the program.

code

6. The.....of a node in a binary tree is define as the height of its left subtree minus height of its right subtree.

balance

7. If p is pointing to a node in an existing tree then info (p) means

Return the content of node p.

8. Following is a keyword of c++.....

Delete

9. Every.....tree is a binary search tree

AVL

10. If a function has recursive call as last statement it is known as.....

Tail recursive

11. During in a order traversal call if we found a node is null it means this node will satisfy following condition

it will not have left child

12. In binary search tree deleting a node is easy if it is a..... node

Root

13. Which of the following operation returns the content of binary tree node?

Info (p)

14. 1 2 6
 5 8
 3
 4

1 2 3 4 5 6 8

15. in a complete binary tree the number of node of level 5 are

30

16. One should be careful about transient.....that are stored by reference I data structure

Objects

17. In a tree we link the node in such a way that it.....a linear structure

does not remain

18. the function call are mode with the help of.....

stack

18. 15 25 45
35 75

15 35 25 75 45

19. A binary is said to be a.....binary tree if every non leaf node in binary tree has non empty left and right structure

Strictly

20. In which of the following function signature the value of variable num cannot be changed in function body

Binary search tree

21. Two common method of simulations are.....are.....

Time based simulation and event based simulation

22. Generally there are case (s)to delete a node from BST

1

23. When a function calls another function the parameters and address of the calling function are put in...

Stock

24. Binary search tree () is a.....

Destructor

25. We can calculate the.....of subtree by counting its level from the bottom

Nodes

26. If there is a strictly complete binary tree of depth 3 the total number of nodes it will be

7

27. A model attempt to model real word phenomenon

Simulation

28. The process of getting the value of a variable using a pointer is called

Dereferencing

29. When a function calling itself is called as.....

Recursion

30. A.....is a tree in which every except possibly the last, is completely filled

Strict binary tree

31. In AVL tree during insertion, a single rotation can fix the balance in cases _____ and 4.

1

32. If a node is inserted in outer side of a node in binary search tree then to make it AVL tree,

We may have to apply single rotation

33. A zigzag rotation is performed. In Left-Left case of rotation in AVL tree.

False

34. The principal benefit of a linked list over a conventional array is that the order of the linked items may be _____ from the order that the data items are stored in memory.

Same

35. Stack.push(15) will push 15 on _____.

Top of the stack

36. Suppose we have the following values to be inserted in constructing AVL tree,

10,13,15,5,7,8

Tell when first rotation will take place,

After inserting the node 15

37. All the objects created using _____ operator have to be explicitly destroyed using the delete operator.

New

38. During the execution of a process, operating system constructs four things for that process. Which of the following is not part of that process?

Linked list

39. There are _____ cases of Rotation in AVL tree.

4

40. Which of the following line of code is incorrect?

Float *I = float new (not confirmed)

41. In simple or singly linked list there is/are _____ pointer/s in each node.

One

42. Suppose there are three nodes tree with three numbers 14, 4, 15. Following is not a permutation, or combination for output them.

(4, 4, 15)

43. The type of expression in which operator succeeds its operands is _____ expression.

Postfix

44. The _____ of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

Balance

45. If both left and right nodes of a node are NULL then this type of node is called a _____ node.

Non leaf (not confirmed)

46. The function calls are made with the help of _____.

Stack

47. Which data structure is needed to convert infix expression to postfix expression?

Stack

48. In case of insertion of right inner node in BST,

Left rotation then right to make AVL

49. A BST generated from the data in ascending order is _____.

Linear

50. Which operation of queue data structure is used to get front element from the queue and then remove it from the queue?

Remove ()

51. In _____ various cells of memory are not located continuously.

Linked list

52. The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

Start

53. Which operation of queue data structure is used to insert an element into the Queue?

Enqueuer ()

54. From Operating System point of view, the recursive function calls are made with the help of _____.

Queue

55. We can make a lexicographic order of characters based on their _____.

ASCII values

56. What are the basic things associated with data structures?

All of the above

57. Consider the following push operations of a Stack:

Stack.push(4);

Stack.push(6);

Stack.push(5);

Stack.push(8);

58. If a user calls a pop() operation, then which value will be returned?

8 (stack is last in first out)

59. ~BinarySearchTree() is a _____.

Destructor

60. Array cells are _____ in computer memory.

Contiguous

61. Leaf node of binary search tree contains _____

One Null Pointer

62. dequeue() operation of queue data structure is used to _____

remove an element from the front and return it

63. The depth of a binary tree is

Maximum level of a leaf

64. Which of the following operation returns but do not removes top value of the stack?

Top

65. Which of the following data structure is linear type?

All of above

66. In doubly linked list a node consists of three parts:

2 pointers and 1 object

67. Last node in circular linked list contains

No null pointer

68. A_____ model attempts to model a real-world phenomenon

Simulation

69. Factorial is an example of _____function.

Recursive

70. The balance of a node in a binary tree is defined as the height of its _____ sub tree minus height of its right sub tree.

Left

71. In case of insertion of left outer node in BST,

We first apply left rotation and then rotation to make it an AVL Tree.

72. `int htdiff = height(root->getLeft()) height(root->getRight());`
The above line of code is taken from AVL insert method.
Complete it by selecting an appropriate symbol.

Minus (-)

73. Each operator in a postfix expression refers to the previous _____ operand(s).

Two

74. For making Binary Search Tree for Strings we need, _____ data type.

Int

75. A _____ model attempts to model a real-world phenomenon

Simulation

76. The lifetime of a transient object cannot exceed that of the application.

True

77. Which one of the following calling method does not change the original value of the argument in the calling function?

Call by passing the value of the argument

78. In level-order traversal for Binary Search Tree, we visit the nodes at each level before proceeding to the next level, in a _____ order.

Left-to-right

79. In doubly linked list a node consists of three parts:

1 pointer and 2 objects

80. Binary Search Tree violates the condition of AVL tree when any node has balance equal to

1 or -1

81. To search an element in AVL tree, it takes maximum $1.88 \log 2n$ time

True

82. HOW many cases of rotation are there in AVL tree?

4

83. _____ tree has been named after two persons Adelson-Velskii and Landis.

AVL

84. Which of the following is a nonlinear data structure?

Tree

85. AVL tree is nonlinear data structure.

True

86. _____ is when function is calling to itself.

Recursion

87. Which of the following function don't belong to the stack class?

Crash ()

88. Which one is the correct function call for the following function of calculating cube?

```
int cube(int& num)
```

cube (num)

89. Binary search algorithm cannot be applied to ____

None of the given

90. copy() method of list data structure _____

Set one list to be a copy of another

91. While implementing non-recursive traversal for Binary Search Tree, we need to implement

Stack

92. Following is true in case of using Recursive method calls

The code becomes very short

93. _____ is a self-balancing tree.

Binary Search Tree

94. While implementing stack with an array and to achieve LIFO behavior, we used push and pop elements at

The start of the array

95. Generalized code written for a class is called

Structure

96. Array cells are _____ in computer memory

Contiguous

97. In singly linked list a node comprises of _____ field/s.

Two

98. Which of the following statement is false?

Pointers store the next data element of a list

99. Leaf node of binary search tree contains _____

One null Pointer

100. What will be the result of evaluating following expression?
 $5+3*2/(6-3)$

7

101. Want and de-allocating memory for linked list nodes does take more than pre-allocated array

MORE

102. Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

Compiler will automatically generate a default constructor

103. Factorial is an example of _____ function.

Recursive

104. A binary tree is said to be a _____ binary tree if every non-leaf node in a binary tree has non-empty left and right subtrees.

Strictly

105. Memory address is stored in

Pointer

106. In level-order traversal for Binary Search Tree, _____ data structure is used.

Queue

107. Which of the following is the correct option for priority Queue?

The type of queues that is FIFO i.e. the person who comes first should leave first.

108. AVL tree is linear data structure.

False

109. AVL tree is a binary search tree

True

110. The post order traversal of a binary tree is DEBFCA. Find out the pre order traversal

A

Λ

B C

Λ /

D E F

ABDCEF

111. In a tree, we link the nodes in such a way that it _____ a linear structure.

Does not remain

112. add(12) method of linked list class will:

Add 12 as value in linked list

113. Which of the following is the correct conversion of infix to postfix expression?

$Z+B-(D-H)/K$

$ZB+DH-K/-$

114. In singly linked list “next” field of node contains:

Address of next node

115. Which of the following line of code is incorrect?

The lifetime of a transient object can exceed that of the application which is accessing it.

116. A list is the collection of items of the _____

same type

117. back() method of List class is used to:

Moves the “current” pointer to backward one element.

118. Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

119. _____ is the maximum height of the AVL tree.

$1.44\log_2 n$

120. length() method of List class is used to:

Return the length of the list

121. There are four cases of rotation in an _____ tree.

AVL

122. Allocating and de-allocating memory for linked list nodes does take _____ time than pre-allocated array.

More

123. _____ is utilized at the time of memory allocation in dynamic manner.

Heap

124. In which case of insertion we require double rotation to make the AVL tree balance.

None

125. The back() method decreases the value of variable current by _____.

One

126. In array list the worst case of removing an element is

To remove the item from start

127. Left, right, info, and parent are the operations of _____ data structure.

Tree

128. Which of the following is TRUE for search operations in a binary tree as compared to linked list or an array?

It increase the number of comparison

129. Which of the following is a correct statement?

An AVL tree is identical to BST except the height of left and right subtree can differ by at least 1

130. length() method of List class is used to:

Return the length of the list

131. Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.

Return Address

132. Whenever we call a function, the compiler makes a _____ that it uses to fulfill this function call.

Stack

133. _____ is the major factor to see the efficiency of a program.

Time

134. _____ rule applies for evaluating operators of same precedence in an expression

None

135. The postfix form of the expression $A + B * C$ and $(A + B) * C$ will be _____.

Same

136. If we use array to implement list, then there is an issue that it gives difficulty when:

We will access value randomly

137. During in-order traversal using recursive calls, if we found a node is NULL. It means this node will satisfy following condition.

It will not have left child

138. Which one the following is more closer to AVL tree, (chose the best option)

Binary Search Tree

139. start() method of List class is used to:

Move the current pointer to first element

140. Consider the following infix expression. $7/8 + 9$

If one converts the above expression into postfix, what would be the resultant expression?

78/9+

141. In internal memory organization of a process, there is some area of memory for static data that holds _____ variables.

Both Static and Global

142. In Left-Right case of rotation in _____ tree. A double rotation is performed.

AVL

143. If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

5

144. Which one is not the property of binary tree?

Sibling node should be same parent (Not Sure)

145. Each node in singly linked list contains_____

One Pointers

146. What's wrong with following loop? while((i < 10) && (i > 24)) { }

The Condition is always false

147. Which one of the following is TRUE about recursion?

1. Recursive function calls consume a lot of memory.
2. Recursion extensively uses stack memory.
3. Recursion extensively uses stack memory.
4. Iteration is more efficient than iteration.

148. Which of the following is not a data structure ?

1. Linked list
2. Queue
3. stack
4. Memory cell

149. “+” is a _____operator.

1. Unary
2. Binary

- 3. Ternary
- 4. None of the above

150. When an executable program run, it is loaded in the memory and becomes a_____.

- 1. Thread
- 2. .h file
- 3. Process
- 4. None of the above

www.vustudents.ning.com

151.

In array list the worst case of removing an element is

- 1. To remove an element from the end of the list
- 2. To remove an element from the middle of the list
- 3. To remove an element from the start of the list
- 4. We cannot remove element from an array list

152. The tree data structure is a

- 1. Linear data structure
- 2. Graphical data structure
- 3. Non-linear data structure (Not Confirm)
- 4. Data structure like queue

153. In_____, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node.

1. Linked list
2. doubly-link list
3. array
4. Structure

154. Each operator in a postfix expression refers to the previous _____ operand(s)

1. one
2. two
3. three
4. four

155. Which of the following operations returns top value of the stack?

1. push
2. pop
3. top
4. First

155. In the calling function, after the execution of the function called, the program continues its execution from the _____ after the function call.

1. Previous line
2. Next line
3. Beginning
4. None of the above

156. _____ only removes items in reverse order as they were entered.

1. Queue
2. Stack
3. Both of these
4. None of these

157. In the linked list implementation of the stack class, where does the push member function places the new entry on the linked list?

1. After all other entries that are greater than the new entry.
2. At the head
3. After all other entries that are smaller than the new entry.
4. At the tail

158. Which of the following is known as "Last-In, First-Out" or LIFO Data Structure?

1. Linked List
2. Stack
3. Queue
4. Tree

159. A queue is a _____data structure, whereas a stack is a _____data structure.

1. FIFO, LIFO

160. Each node in doubly link list has

1. 2
2. 3

3. 4

161. Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function

1. First argument

2. Return address

3. Last argument

4. none of the above

162. _____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()

2. pop(), push()

3. isEmpty() , isFull()

163. Stack and Queue can be implemented using _____,

1. Singly Link List
2. Binary Tree
3. Binary Search Tree
4. AVL Tree

164. The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

1. Remove
2. next
3. start
4. back

165. In array list the worst case of removing an element is

1. To remove an element from the end of the list
2. To remove an element from the middle of the list
3. To remove an element from the start of the list
4. We cannot remove element from an array list

166. STL is a _____ that is a part of the official standard of C++.

1. C program file

2. .h file

167. When an executable program run, it is loaded in the memory and becomes a_____.

1. Thread

2. .h file

3. Process

4. None of the above

168. Each node in doubly link list has,

1. pointer

2. pointers

3. pointers

4. pointer

169. A queue is a _____ data structure, whereas a stack is a _____ data structure.

1. FIFO, LIFO

2. LIFO, FIFO

3. both of these

4. none of these

170. Which one of the following is TRUE about recursion?

1. Recursive function calls consume a lot of memory.

2. Recursion extensively uses stack memory.

3. Recursion extensively uses stack memory.

4. Iteration is more efficient than iteration.

To create a _____ we link the last node with the first node in the list.

1. Double linked list

2. Circularly-linked list

3. Linked list

4. None of the above

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()

2. pop(), push()

3. isEmpty() , isFull()

4. push(),pop()

The principal benefit of a linked list over a conventional array is that the order of the linked items may be_____ from the order that the data items are stored in memory.

1. Same
2. Identical
3. Different
4. Equivalent

In_____ the 'next' returns false when it reaches to the last node due to the fact that the next field of the last node is set to NULL.

1. Circular linked list
2. Triple linked list
3. Singly linked list
4. None of the above

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

1. 48
2. 4
3. 333
4. 5

Which boolean expression indicates whether the numbers in two nodes (p and q) are the same. Assume that neither p nor q is null.

1. `p == q`
2. `p.data == q.data`
3. `p.nextNode == q. nextNode`
4. `p.data == q`

Local variables of a function are stored in,

1. Binary Search Tree

2. Stack

3. Queue

4. AVL Tree

A queue is a data structure where elements are

1. inserted at the front and removed from the back.
2. inserted and removed from the top.
3. inserted at the back and removed from the front.
4. inserted and removed from both ends.

A kind of expressions where the operator is present between two operands called _____ expressions.

1. Infix

2. Postfix

3. Prefix
4. None of the above

A template is a function or class that is written with a _____data type.

1. Specific
2. Definite
3. Generic
4. None of the above.

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.

1. First argument
2. Return address
3. Last argument
4. None of the above

Which of the following can be used to reverse a string value,

1. Stack
2. Queue
3. Both of these
4. None of these

A queue is a _____data structure, whereas a stack is a _____data structure.

1. FIFO, LIFO
2. LIFO,FIFO
3. both of these
4. none of these

Stack and Queue can be implemented using _____,

1. Singly Link List
2. Binary Tree
3. Binary Search Tree
4. AVL Tree

Which of the following is not a data structure ?

1. Linked list
2. Stack
3. Queue
4. Memory cell

Compiler uses which one of the following in Function calls,

1. Stack

2. Queue
3. Binary Search Tree
4. AVL Tree

The principal benefit of a linked list over a conventional array is that the order of the linked items may be_____ from the order that the data items are stored in memory.

1. Same
2. Identical
3. Different
4. Equivalent

Doubly Linked List always has one NULL pointer.

1. True
2. False

We cannot remove items randomly from _____

1. Stack
2. Queue
3. Both of these
4. None of these

_____ is the maximum number of nodes that you can have on a stack-linked list ?

1. Zero
2. $2n$ (where n is the number of nodes in linked list)
3. Any Number
4. None of these

“+” is a _____ operator.

1. Unary
2. Binary

3. Ternary
4. None of the above

In_____, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node.

1. Linked list
2. doubly-link list
3. array
4. structure

A queue is a data structure where elements are

1. inserted at the front and removed from the back.
2. inserted and removed from the top.
3. inserted at the back and removed from the front.
4. inserted and removed from both ends.

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()
2. pop(), push()
3. isEmpty() , isFull()
4. push(),pop()

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

1. Remove
2. Next
3. Start
4. Back

Parameters in function call are passed using,

1. Stack
2. Queue
3. Binary Search Tree

4. AVL Tree

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

1. 48
2. 4
3. 333
4. 5

For a perfect binary tree of height 4, what will be the sum of highest of node

1. 31
2. 30
3. 27
4. 26

If Ahmed is cousin of Ali and Ali is cousin of Asad then Ahmed is also cousin of Asad. This statement has the following property

1. Reflexivity
2. Symmetry
3. Transitivity
4. All of the above

Which one of the following is NOT the property of equivalence relation?

1. Reflexive
2. Symmetric
3. Transitive
4. Associative

The main reason of using heap in priority queue is

1. Improve performance
2. Code readable
3. Less code
4. Heap can't be used in priority queues

The total numbers of nodes on 10th level of perfect binary tree are

1. 256

2. 512

3. 1024

4. Can't be determined

Suppose there are 100 elements in an equivalence class, so initially there will be 100 trees, the collection of these trees is called _____.

1. Cluster

2. Class

3. Forest

4. Bunch

The percolate Down procedure will move the smaller value and bigger value.

1. Left, right

2. Right, left

3. Down, up

4. Up, down

For a perfect binary tree of height h , having N nodes, the Sum of height of nodes is

1. $N - h - 1$
2. $N - 1$
3. $N - 1 + h$
4. $N - (h - 1)$

Which of the following method is helpful in creating the heap at once?

1. Insert
2. Add
3. Update
4. `percolateDown`

If ahmad is boss of ehsan and ehsan is boss of umer then ahmad is also boss of

umer, the above-mentioned relation is_____ .

1. Reflexive

2. Symmetry
3. Transitive
4. None of given

If we want to find median of 50 elements, then after applying builtHeap method, how manytime deleteMin method will be called?

1. 5
2. 25
3. 35
4. 50

Which of the following properties are satisfied by equivalence relationship?

1. Reflexive, symmetric
2. Reflexive, transitive
3. Symmetric, transitive
4. Reflexive, symmetric and transitive

Sorting procedure normally takes____ time.

1. $N \log N$

2. $2N$

3. $N * N * N$

4. N

15. The Expression

if (! heap->isFull())

Check

1. Heap is empty

2. Heap is full

3. Heap is not empty

4. Heap is not full

The Expression

if (! heap->isEmpty())

Check

1. Heap is empty

2. Heap is full

3. Heap is not empty

4. Not a valid expression

Given the values are the array representation of heap: 12 23 26 31 34 44
56 64 78

100 If we perform 4 deleteMin operation, the last element deleted
is_____ .

1. 31

2. 34

3. 44

4. 56

Which of the following heap method increase the value of key at position
'p' by the amount 'delta'?

1. increaseKey(p, delta)

2. decreaseKey(p, delta)

3. percolateDown(p, delta)

4. remove(p, delta)

Which of the following heap method lowers the value of key at position 'p' by the amount 'delta'?

1. increaseKey(p, delta)
2. decreaseKey(p, delta)
3. percolateDown(p, delta)
4. remove(p, delta)

Which property of equivalence relation is satisfied if we say:

Ahmad R(is related to)Ahmad

1. Reflexivity
2. Symmetry
3. Transitivity
4. All of Above

Given the values are the array representation of heap; 12 23 26 31 34 44 56 64 78 100 What is the 5th smallest element in the given heap?

1. 31
2. 34
3. 44

CS301 Grand Quiz Solution File

In AVL tree during insertion, a single rotation can fix the balance in cases _____ and 4.

1

If a node is inserted in outer side of a node in binary search tree then to make it AVL tree,

We may have to apply single rotation

A zigzag rotation is performed. In Left-Left case of rotation in AVL tree.

False

The principal benefit of a linked list over a conventional array is that the order of the linked items may be _____ from the order that the data items are stored in memory.

Same

Stack.push(15) will push 15 on _____.

Top of the stack

Suppose we have the following values to be inserted in constructing AVL tree,

10,13,15,5,7,8

Tell when first rotation will take place,

After inserting the node 15

All the objects created using _____ operator have to be explicitly destroyed using the delete operator.

New

During the execution of a process, operating system constructs four things for that process. Which of the following is not part of that process?

Linked list

There are _____ cases of Rotation in AVL tree.

4

Which of the following line of code is incorrect?

`Float *I = float new` (not confirmed)

In simple or singly linked list there is/are _____ pointer/s in each node.

One

Suppose there are three nodes tree with three numbers 14, 4, 15. Following is not a permutation, or combination for output them.

(4, 4, 15)

The type of expression in which operator succeeds its operands is _____ expression.

Postfix

The _____ of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

Balance

If both left and right nodes of a node are NULL then this type of node is called a _____ node.

Non leaf (not confirmed)

The function calls are made with the help of _____.

Stack

Which data structure is needed to convert infix expression to postfix expression?

Stack

In case of insertion of right inner node in BST,

Left rotation then right to make AVL

A BST generated from the data in ascending order is _____.

Linear

Which operation of queue data structure is used to get front element from the queue and then remove it from the queue?

Remove ()

In _____ various cells of memory are not located continuously.

Linked list

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

Start

Which operation of queue data structure is used to insert an element into the Queue?

Enqueue ()

From Operating System point of view, the recursive function calls are made with the help of _____.

Queue

We can make a lexicographic order of characters based on their _____.

ASCII values

What are the basic things associated with data structures?

All of the above

Consider the following push operations of a Stack:

Stack.push(4);

Stack.push(6);

Stack.push(5);

Stack.push(8);

If a user calls a pop() operation, then which value will be returned?

8 (stack is last in first out)

~BinarySearchTree() is a _____.

Destructor

Array cells are _____ in computer memory.

Contiguous

Leaf node of binary search tree contains _____

One Null Pointer

dequeue() operation of queue data structure is used to _____

remove an element from the front and return it

The depth of a binary tree is

Maximum level of a leaf

Which of the following operation returns but do not removes top value of the stack?

Top

Which of the following data structure is linear type?

All of above

In doubly linked list a node consists of three parts:

2 pointers and 1 object

Last node in circular linked list contains

No null pointer

A _____ model attempts to model a real-world phenomenon

Simulation

Factorial is an example of _____ function.

Recursive

The balance of a node in a binary tree is defined as the height of its _____ sub tree minus height of its right sub tree.

Left

In case of insertion of left outer node in BST,

We first apply left rotation and then rotation to make it an AVL Tree.

`int htdiff = height(root->getLeft()) - height(root->getRight());` The above line of code is taken from AVL insert method. Complete it by selecting an appropriate symbol.

Minus (-)

Each operator in a postfix expression refers to the previous _____ operand(s).

Two

For making Binary Search Tree for Strings we need, _____ data type.

Int

A _____ model attempts to model a real-world phenomenon

Simulation

The lifetime of a transient object cannot exceed that of the application.

True

Which one of the following calling method does not change the original value of the argument in the calling function?

Call by passing the value of the argument

In level-order traversal for Binary Search Tree, we visit the nodes at each level before proceeding to the next level, in a _____ order.

Left-to-right

In doubly linked list a node consists of three parts:

1 pointer and 2 objects

Binary Search Tree violates the condition of AVL tree when any node has balance equal to

1 or -1

To search an element in AVL tree, it takes maximum $1.88 \log_2 n$ time

True

HOW many cases of rotation are there in AVL tree?

4

_____ tree has been named after two persons Adelson-Velskii and Landis.

AVL

Which of the following is a nonlinear data structure?

Tree

AVL tree is nonlinear data structure.

True

_____ is when function is calling to itself.

Recursion

Which of the following function don't belongs to the stack class?

Crash ()

Which one is the core function call for the following function of calculating cube?

int cube(int& num)

cube (num)

Binary search algorithm cannot be applied to _____

None of the given

copy() method of list data structure _____

Set one list to be a copy of another

While implementing non-recursive traversal for Binary Search Tree, we need to implement

Stack

Following is true in case of using Recursive method calls

The code becomes very short

_____ is a self-balancing tree.

Binary Search Tree

While implementing stack with an array and to achieve LIFO behavior, we used push and pop elements at

The start of the array

Generalized code written for a class is called

Structure

Array cells are _____ in computer memory

Contiguous

In singly linked list a node comprises of _____ field/s.

Two

Which of the following statement is false?

Pointers store the next data element of a list

Leaf node of binary search tree contains _____

One null Pointer

What will be the result of evaluating following expression? $5+3*2/(6-3)$

7

Want and de-allocating memory for linked list nodes does take more than pre-allocated array

MORE

Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

Compiler will automatically generate a default constructor

Factorial is an example of _____ function.

Recursive

A binary tree is said to be a _____ binary tree if every non-leaf node in a binary tree has non-empty left and right subtrees.

Strictly

Memory address is stored in

Pointer

In level-order traversal for Binary Search Tree, _____ data structure is used.

Queue

Which of the following is the correct option for priority Queue?

The type of queues that is FIFO i.e. the person who comes first should leave first.

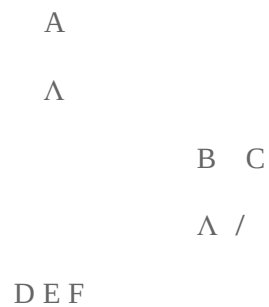
AVL tree is linear data structure.

False

AVL tree is a binary search tree

True

The post order traversal of a binary tree is DEBFCA. Find out the pre order traversal



ABDCEF

In a tree, we link the nodes in such a way that it _____ a linear structure.

Does not remain

add(12) method of linked list class will:

Add 12 as value in linked list

Which of the following is the correct conversion of infix to postfix expression?

$Z+B-(D-H)/K$

$ZB+DH-K/-$

In singly linked list “next” field of node contains:

Address of next node

Which of the following line of code is incorrect?

The lifetime of a transient object can exceed that of the application which is accessing it.

A list is the collection of items of the _____

same type

back() method of List class is used to:

Moves the “current” pointer to backward one element.

Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

_____ is the maximum height of the AVL tree.

$1.44\log_2 n$

length() method of List class is used to:

Return the length of the list

There are four cases of rotation in an _____ tree.

AVL

Allocating and de-allocating memory for linked list nodes does take _____ time than pre-allocated array.

More

_____ is utilized at the time of memory allocation in dynamic manner.

Heap

In which case of insertion we require double rotation to make the AVL tree balance.

None

The back() method decreases the value of variable current by _____.

One

In array list the worst case of removing an element is

To remove the item from start

Left, right, info, and parent are the operations of _____ data structure.

Tree

Which of the following is TRUE for search operations in a binary tree as compared to linked list or an array?

It increase the number of comparison

Which of the following is a correct statement?

An AVL tree is identical to BST except the height of left and right subtree can differ by at least 1

length() method of List class is used to:

Return the length of the list

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.

Return Address

Whenever we call a function, the compiler makes a _____ that it uses to fulfill this function call.

Stack

_____ is the major factor to see the efficiency of a program.

Time

_____ rule applies for evaluating operators of same precedence in an expression

None

The postfix form of the expression $A + B * C$ and $(A + B) * C$ will be _____.

Same

If we use array to implement list, then there is an issue that it gives difficulty when:

We will access value randomly

During in-order traversal using recursive calls, if we found a node is NULL. It means this node will satisfy following condition.

It will not have left child

Which one the following is more closer to AVL tree, (chose the best option)

Binary Search Tree

start() method of List class is used to:

Move the current pointer to first element

Consider the following infix expression.

$7/8 + 9$

If one converts the above expression into postfix, what would be the resultant expression?

78/9+

In internal memory organization of a process, there is some area of memory for static data that holds _____ variables.

Both Static and Global

In Left-Right case of rotation in _____ tree. A double rotation is performed.

AVL

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

5

Which one is not the property of binary tree?

Sibling node should be same parent (Not Sure)

Each node in singly linked list contains_____

One Pointers

What's wrong with following loop? while((i < 10) && (i > 24)) { }

The Condition is always false

Cs301 Quiz no 1 and 2 for midterm exam

My todays quizzz no 1

1. When an executable program run, it is loaded in the memory and becomes a_____.

Select correct option:

Thread

.h file

Process

None of the above

2. A kind of expressions where the operator is present between two operands called _____ expressions.

Select correct option:

Infix

Postfix

Prefix

None of the above

3. Which of the following operations returns top value of the stack?

Select correct option:

push

pop

top

first

4. To create a _____ we link the last node with the first node in the list.

Select correct option:

Double linked list

Circularly-linked list

Linked list

None of the above

5. In the calling function, after the execution of the function called, the program continues its execution from the _____ after the function call.

Select correct option:

Previous line

Next line

Beginning

None of the above

6. Doubly Linked List always has one NULL pointer.

Select correct option:

True

False

7. _____ only removes items in reverse order as they were entered.

Select correct option:

Queue

Stack

Both of these

None of these

8. In the linked list implementation of the stack class, where does the push member function place the new entry on the linked list?

Select correct option:

After all other entries that are greater than the new entry.

At the head

After all other entries that are smaller than the new entry.

At the tail

9. Which of the following is known as "Last-In, First-Out" or LIFO Data Structure?

Select correct option:

Linked List

Stack

Queue

Tree

10. Which boolean expression indicates whether the numbers in two nodes (p and q) are the same. Assume that neither p nor q is null.

Select correct option:

p == q

p.data == q.data

p.nextNode == q.nextNode

p.data == q

1. Local variables of a function are stored in,

Select correct option:

Binary Search Tree

Stack

Queue

AVL Tree

- **2.** Question # 2 of 5 (Start time: 11:41:31 AM)

When an executable program runs, it is loaded in the memory and becomes a_____.

Select correct option:

Thread

.h file

Process

None of the above

- 3. In _____ the 'next' returns false when it reaches to the last node due to the fact that the next field of the last node is set to NULL.

Select correct option:

Circular linked list

Triple linked list

Singly linked list

None of the above

- 4. A kind of expressions where the operator is present between two operands called _____ expressions.

Select correct option:

Infix

Postfix

Prefix

None of the above

- 5. Question # 1 of 5 (Start time: 11:44:44 AM) Total Marks: 1
Compiler uses which one of the following in Function calls,
Select correct option:

Stack

Queue

Binary Search Tree

AVL Tree

- 6. Stack and Queue can be implemented using _____,

Select correct option:

Singly Link List

Binary Tree

Binary Search Tree

AVL Tree

7. Which one of the following is TRUE about recursion?
Select correct option:

Recursive function calls consume a lot of memory.

Recursion extensively uses stack memory.

Recursion extensively uses stack memory.

Iteration is more efficient than iteration.

8. Doubly Linked List always has one NULL pointer.
Select correct option:

True

False

9. Which boolean expression indicates whether the numbers in two nodes (p and q) are the same.
Assume that neither p nor q is null.
Select correct option:

p == q

p.data == q.data

p.nextNode == q. nextNode

p.data == q

10. Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.
Select correct option:

First argument

Return address

Last argument

None of the above

11.

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

Select correct option:

isFull(),isEmpty()
pop(), push()
isEmpty(), isFull()
push(),pop()

12.

Each operator in a postfix expression refers to the previous _____ operand(s).

Select correct option:

one
two
three
four

13The next field in the last node in a singly-linked list is set to_____.

Select correct option:

0
1
NULL
false

-

14.

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

Select correct option:

Remove
Next
Start
Back

15.

_____ is the maximum number of nodes that you can have on a stack-linked list ?

Select correct option:

Zero

$2n$ (where n is the number of nodes in linked list)

Any Number

None of these

-

16.

In the linked list implementation of the stack class, where does the push member function places the new entry on the linked list?

Select correct option:

After all other entries that are greater than the new entry.

At the head

After all other entries that are smaller than the new entry.

At the tail

-

17. Which of the following operations returns top value of the stack?

Select correct option:

push

pop

top

first

-

18. it will be efficient to place stack elements at the start of the list because insertion and removal take _____ time.

Select correct option:

Variable

Constant

Inconsistent

None of the above

19. A template is a function or class that is written with a _____ data type.

Select correct option:

Specific

Definite

Generic

None of the above.

-

20. Local variables of a function are stored in,

Select correct option:

Binary Search Tree

Stack

Queue

AVL Tree

21. To create a _____ we link the last node with the first node in the list.

Select correct option:

Double linked list

Circularly-linked list

Linked list

None of the above

-

22. In the calling function, after the execution of the function called, the program continues its execution from the _____ after the function call.

Select correct option:

Previous line

Next line

Beginning

None of the above

23. Which of the following can be used to reverse a string value,

Select correct option:

Stack

Queue

Both of these

None of these

-

24. Question # 4 of 5 (Start time: 05:32:20 PM) Total Marks: 1

Each node in doubly link list has,

Select correct option:

1 pointer

2 pointers

3 pointers

4 pointers

25. In _____ the 'next' returns false when it reaches to the last node due to the fact that the next field of the last node is set to NULL.

Select correct option:

Circular linked list

Triple linked list

Singly linked list
None of the above

26. only removes items in reverse order as they were entered.
Select correct option:

Queue
Stack
Both of these
None of these

27. A queue is a _____ data structure, whereas a stack is a _____ data structure.
Select correct option:

FIFO, LIFO
LIFO, FIFO
both of these
none of these

-

28. The principal benefit of a linked list over a conventional array is that the order of the linked items may be _____ from the order that the data items are stored in memory.
Select correct option:

Same
Identical
Different
Equivalent

-

29. Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.
Select correct option:

First argument
Return address
Last argument
None of the above

-

30. The _____ method of list will position the current Node and lastCurrentNode at the start of the list.
Select correct option:

Remove
Next
Start
Back

See the below code and fill the appropriate answer for? void fastInorder(TreeNode* p) {
while((p=nexInorder(p)) != ?) cout p->getInfo(); }

Dummy
rootNode
LTH
RTH

If there are N external nodes in a binary tree then what will be the no. of the internal nodes in this binary tree?

N-1
N
N+1
N+2

An expression tree will always be a,

Complete binary tree
Binary search tree
Heap AVL tree

When a complete binary tree represented by an array then if right child is at position 5 then left child will be at position _____

2
3
4
6

Consider a binary tree, represented by the following array: A,B,C,D,E,F,G,I Is it a strictly binary tree ?

Yes
No

In a min heap the parent node has key smaller than or equal to

Left child
Right child
Both

None

The expression if (! heap->isEmpty()) checks

Heap is empty

Heap is full

Heap is not empty

Not a valid expression (not confirm)

We implement the heap by

Threaded Tree

AVL tree

Complete binary tree

Expression

For a perfect binary tree of height h, having N nodes, the sum of heights of nodes is

$n-(h-1)$

$n-(h+1)$

$n-h$

none

Traversing a binary tree can only be done using

Recursion

Iteration

Both

none

For a perfect binary tree of height 4. What will be the sum of heights of nodes?

[31](#)

3027

26

For a perfect binary tree of height h, having N nodes, the sum of heights of nodes is

_____.

$N - (h - 1)$

[\$N - \(h + 1\)\$](#)

$N - 1$

$N - 1 + h$

If we want to find median of 50 elements, then after applying buildHeap method, how

many times deleteMin method will be called ?

5

[25](#)

35

50

Which of the following heap method increase the value of key at position 'p' by the amount 'delta'?

[increaseKey\(p,delta\)](#)

decreaseKey(p,delta)

preculatDown(p,delta)

remove(p,delta)

The main reason of using heap in priority queue is

[improve performance](#)

code is readable

less code

heap can't be used in priority queues

The total number of nodes on 10th level of a perfect binary tree are :

256

512

1024

[Can't be determined](#)

Which property of equivalence relation is satisfied if we say: Ahmad R(is related to)

Ahmad

Reflexivity

[Symmetry](#)

Transitivity

All of the above

Which of the following heap method lowers the value of key at position 'p' by the amount 'delta'?

[increaseKey\(p,delta\)](#)

decreaseKey(p,delta)

preculatDown(p,delta)

remove(p,delta)

We can build a heap in _____ time.

Linear

Exponential

Polynomial

None of the given options

we can build a heap in linear time using n calls of `percolate_down()`

If a tree has 50 nodes, then the total edges/links in the tree will be :

55

51

50

$49 \text{ } N-1 = 49$

Consider a max heap, represented by the following array;

40,30,20,10,15,16,17,18,4 After inserting a nodes with value 35.Which of following is the updated max heap?

40,30,20,10,15,16,17,8,4,35

40,30,20,10,35,16,17,8,4,15

40,35,20,10,30,16,17,8,4,15

40,35,20,10,15,16,17,18,4,30

A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a Link) _____Successor.

Preorder

Inorder

Postorder

Leveloder

Which of the following is a property of binary tree?

A Binary tree with N internal nodes has $2+N$ links, $N-1$ links to internal nodes and $N+1$ links to external nodes

A Binary tree with N internal nodes has $2*N$ links, $N-1$ links to internal nodes and $N+1$ links to external nodes.

A Binary tree with N internal nodes has $2-N$ links, $N-1$ links to internal nodes and $N+1$ links to external nodes.

A Binary tree with N internal nodes has $2N$ links, $N+1$ links to internal nodes and $N-1$ links to external nodes.

A Threaded Binary tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link)_____ successor.

Preorder

Inorder

Postorder

Levelorder

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

54

55

56

57

Which of the following statement is correct?

A threaded Binary tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER successor.

A threaded Binary tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its PREORDER successor.

A threaded Binary tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER successor.

A threaded Binary tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its POSTORDER predecessor.

It is necessary for Huffman encoding tree to be,

AVL tree

Binary tree

Complete binary Tree

None of these

A binary tree with 45 internal nodes has _____ links to external nodes.

44

45

46

90

In which of the following tree, parent nodes has key greater than or equal to its both children?

Max heap

Binary search tree

Threaded Binary tree

Complete Binary tree

If one pointer of the nodes in a binary tree is NULL then it will be a/an

Inner node

Leaf node

External node

Root node

If there are N external nodes in a binary tree then what will be the no. of the internal nodes in this binary tree?

N-1

N

N+1

N+2

See the below code and fill the appropriate answer for? Void

fastInorder(TreeNod+p) {while((p+nextInorder(p)) != ?) cout << p->getInfo();}

Dummy

rootNode

LTH

RTH

In threaded binary tree, the NULL pointer are replaced by the.

Preorder successor or Predecessor

Inorder successor or predecessor

Postorder successor or predecessor

NULL pointer are not replaced

In which of the following tree, parent nodes has a key greater than or equal to its both children?

Max heap

Binary search tree

Threaded Binary tree

Complete Binary tree

In Complete binary tree the bottom level is filled from _____.

Left to right

Right to left

Not filled at all
None of the given options

If the bottom level of a binary tree is NOT completely filled, depicts that the tree is NOT a _____

Complete Binary tree

Threaded Binary Tree

Expression tree

Perfectly complete Binary tree

If an expression tree is correct then its root should have,

An operator

(

)

an operand

In threaded binary tree, the NULL pointers are replaced by the.

Preorder successor or predecessor

Inorder successor or predecessor

Postorder successor or predecessor

NULL pointer are not replaced

A complete binary tree is a tree that is _____ filled, with the possible exception of the bottom level.

Partially

Completely

Incompletely

Partly

If the bottom level of a binary tree is not completely filled, depicts that the tree is not a _____.

Expression tree

Threaded binary tree

Complete binary tree

Perfectly complete binary tree

An expression tree will always be a,

Complete binary tree

Binary search tree

Heap AVL tree

Which of the following is a property of binary tree?

A binary tree of N external nodes has N internal node

A Binary tree of N internal nodes has N+1 external node

A Binary tree of N external nodes has N+1 internal node

A Binary tree of N internal has N-1 external node

In a threaded binary tree which nodes have NULL child pointers,

All leaf nodes

Nodes other than leaf nodes

Root Node

None of the nodes

In threaded binary tree, the NULL pointers are replaced by the

preorder successor or predecessor

inorder successor or predecessor

postorder successor or predecessor

NULL pointers are not replaced

A complete binary tree is a tree that is _____ filled, with the possible exception of the bottom level.

partially

completely

incompletely

partly

Which one of the following is TRUE about iteration?

Iterative function calls consumes a lot of memory

Threaded Binary Trees use the concept of iteration

Iteration extensively uses stack memory

Recursion is more efficient than iteration

We implement the heap by _____ :

Threaded Tree

AVL tree

Complete binary tree

Expression

Which of the following statement concerning heaps is NOT true?

Traversing a heap in order provides access to the data in numeric or alphabetical

order.

Removing the item at the top provides immediate access to the key value with highest (or lowest) priority.

Inserting an item is always done at the end of the array, but requires maintaining the heap property.

A heap may be stored in an array.

Which of the following statement concerning heaps is NOT true?

A heap can be stored in a binary search tree.

A heap can be stored in an array.

A heap can be used to implement a priority queue.

A heap can be used to sort data.

A complete binary tree is a tree that is _____ filled, with the possible exception of the bottom level.

partially

completely

incompletely

partly

By using _____ we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

Binary tree only

Heap data structure

Huffman encoding

Which of the following statement is true about dummy node of threaded binary tree?

The left pointer of dummy node points to the itself while the right pointer points to the root of tree.

The left pointer of dummy node points to the root node of the tree while the right pointer points itself i.e. to dummy node.

The left pointer of dummy node points to the root node of the tree while the right pointer is always NULL.

The right pointer of dummy node points to the itself while the left pointer is always NULL.

Threaded binary tree

When a complete binary tree, represented by an array then for any array element at position i , the parent is at position _____ .

$2i-1$

$2i$

$2i+1$

$\text{floor}(i/2)$

When a complete binary tree represented by an array then if right child is at position 5 then left child will be at position _____

2

3

4

6

A binary tree with N internal nodes has _____ links, _____ links to internal nodes and _____ links to external nodes.

$2N, N-1, N+1$

$N-1, 2N, N+1$

$N+1, 2N, N-1$

$N+1, N-1, 2N$

If a binary tree has $N + 1$ external nodes then,

It has N internal nodes.

It has $N-1$ internal nodes.

It has $N/2$ internal nodes.

It has $N+2$ internal nodes.

A binary tree with 45 internal nodes has _____ links to external nodes.

44

45

46

90

Consider a binary tree, represented by the following array: 10,7,9,5,2,1,6,3,4 This is a _____.

Min heap

Max heap (Not Sure)

Threaded binary tree

Binary Search tree

Consider a binary tree, represented by the following array: A,B,C,D,E,F,G,I Is it a strictly binary tree ?

Yes

No

**In threaded binary tree the NULL pointers are replaced by the
preorder successor or predecessor
inorder successor or predecessor**

inorder successor or predecessor

NULL pointers are not replaced

Consider a binary tree, represented by the following array: A,B,C,D,E,F,G,H,I,J,K,L

Is it a strictly binary tree?

Yes

No

We implement the heap by _____ .

Threaded Tree

AVL tree

Complete binary tree

Expression

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

- ▶ 54
- ▶ 55
- ▶ 56
- ▶ 57

Which of the following statements is correct property of binary trees?

- ▶ A binary tree with N internal nodes has N+1 internal links.
- ▶ A binary tree with N external nodes has 2N internal nodes.
- ▶ **A binary tree with N internal nodes has N+1 external nodes.**
- ▶ None of above statement is a property of the binary tree.

Which of the following is a property of binary tree?

- ▶ A binary tree of N external nodes has N internal node.
- ▶ **A binary tree of N internal nodes has N+ 1 external node.**
- ▶ A binary tree of N external nodes has N+ 1 internal node.
- ▶ A binary tree of N internal nodes has N- 1 external node.

Which of the following statement is true about dummy node of threaded binary tree?

▶ The left pointer of dummy node points to the itself while the right pointer points to the root of tree.

▶ **The left pointer of dummy node points to the root node of the tree while the right pointer points itself i.e. to *dummy* node**

▶ The left pointer of dummy node points to the root node of the tree while the right pointer is always NULL.

▶ The right pointer of dummy node points to the itself while the left pointer is always NULL.

If the bottom level of a binary tree is NOT completely filled, depicts that the tree is NOT a

- ▶ Expression tree
- ▶ Threaded binary tree
- ▶ **complete Binary tree**
- ▶ Perfectly complete Binary tree

Which of the following statement is correct about find(x) operation:

▶ A find(x) on element x is performed by returning exactly the same node that is found.

- ▶ A find(x) on element x is performed by returning the root of the tree containing x.
- ▶ A find(x) on element x is performed by returning the whole tree itself containing x.
- ▶ A find(x) on element x is performed by returning TRUE.

If there are 23 external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ 23
- ▶ 2
- ▶ 21
- ▶ **22**

If there are N external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

- ▶ **N -1**
- ▶ N+1
- ▶ N+2
- ▶ N

Which of the following statement is correct?

▶ A Threaded Binary Tree is a binary tree in which every node that does not have a left child has a THREAD (in actual sense, a link) to its INORDER successor.

▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its PREORDER successor.

▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor.

▶ A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its POSTORDER successor.

- By using _____ we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

- ▶ Binary tree only
- ▶ **Threaded binary tree**
- ▶ Heap data structure
- ▶ Huffman encoding

Consider a min heap, represented by the following array:

10,30,20,70,40,50,80,60

After inserting a node with value 31. Which of the following is the updated min heap?

- ▶ 10,30,20,31,40,50,80,60,70
- ▶ 10,30,20,70,40,50,80,60,31
- ▶ 10,31,20,30,40,50,80,60,31
- ▶ 31,10,30,20,70,40,50,80,60

In complete binary tree the bottom level is filled from _____.

- ▶ **Left to right**
- ▶ Right to left
- ▶ Not filled at all
- ▶ None of the given options

In case of deleting a node from AVL tree, rotation could be prolong to the *root* node.

- ▶ **Yes**
- ▶ No

When an array of object is created dynamically then there is no way to provide parameterized constructors for array of objects.

True

Flase

Which of the following method is helpful in creating the heap at once?

insert

add

update
preculateDown

Which of the following calling method does not change the original value of variable passed to a function as an argument?

Call by passing as value

What is the maximum of recursive calls a function may make?

n (where n is the argument)

Here is a small function definition:

```
void f(int i, int &k)
{
    i = 1;
    k = 2;
}
```

Suppose that a main program has two integer variables x and y, which are given the value 0. Then the main program calls f(x,y); What are the values of x and y after the function f finishes?

x is still 0, but y is now 2.

Write in postfix form $5 * (9 - 7)$

5 9 7 - *

Which of the following is NOT true regarding the maze generation?

Remove a randomly chosen wall if the cells it separates are already in the same set

An array x of 100 integers is declared as

int x[100];

Local variables defined inside function body are _____ automatically at the end of function execution.

Created

Which of the following statement is NOT correct regarding Table ADT?

A table consists of several columns known as entities

In doubly linked list there is/are _____ pointer/s in each node.

Two

"new int[11]" will allocate memory for _____ integers.

11

If a tree has 20 edges/links, then the total number of nodes in the tree will be:

21

In the statement `int& a=b;`

`b` hold the address of variable `a`

Which of the following is NOT an open addressing technique to resolve collisions?

Double hashing

Consider the function X as under

```
int X (int& Value)
```

```
{
```

```
return Value;
```

```
}
```

Now `a` and `b` are integers in a calling function. Which one of the following is a valid call to the above function X?

`a = X(*b);`

For a perfect binary tree of height 4, What will be the sum of heights of nodes?

26

If unions are done by weight (size), the depth of any element is never greater than

$\log_2 n$

Which of the following statement is NOT true for reference variable?

References can be uninitialized.

The tree data structure is a

Non-linear data structure

In singly linked list a node consists of two parts:

Object and pointer

The simplest case in a BST to delete a node is:

When the node, that is to be deleted is a leaf node

A binary tree with 24 internal nodes has _____ external nodes.

25

$a * (b+c)-d$ is an example of _____ expression.

infix

given the values are the array representation of heap:

12 23 26 31 34 44 56 64 78 100

If we perform 4 deleteMin operations, the last element deleted is _____.

31

In _____, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node.

Linked list

Trying to remove an element from an empty stack is called _____ .

Underflow of stack

For reference variables, _____ sign is used.

ampersand

Which property of equivalence relation is satisfied if we say:
Ahmad R(is related to) Ahmad

Reflexivity

Suppose there is an image segmented into pixels. Each pixel has _____ neighbour(s).

8

The most difficult case for deleting a node from Binary Search Tree is when the node to be deleted _____.

is Leaf node

When an executable program runs, it is loaded in the computer memory and becomes a _____ .

Process

If class A defines class B as its friend, then:

Class B can access private members of Class A

In the post-order traversal of a binary search tree, nodes process as:

Left-subtree, Right-subtree, Root

The expression $AB+C^*$ is called?

Postfix expression

Binary Search can be categorized into which of following?

Divide and conquer

The main use of AVL tree is:

Storing of data

Select the one FALSE statement about binary trees:

Every binary tree has at least one node.

A table consists of several columns, known as

Fields

Each operator in a postfix expression refers to the previous _____ operand(s).

One

If there are 56 internal nodes in a binary tree then how many external nodes this binary tree will have?

57

Which of the following function don't belong to the stack class?

crash()

Which of the following statement is false?

Arrays are dense lists and static data structure

During union by size method, which data structure is used to improve the balancing of tree?

Array

Non-recursive calls are faster than which of the following calls?

Recursive

Binary Search Tree violates the condition of AVL tree when any node has balance equal to

0

Each node in Binary Search Tree has

2 pointers

What will be postfix expression of the following infix expression?

Infix Expression: $a + b * c - d$

$abc * + d -$

The balance of a node is the result of "height of left subtree" _____ "height of right subtree".

Minus

A complete binary tree having "N" nodes consists of _____ Levels.

$\log_2 (N+1) - 1$

Consider a hash table of size seven, with starting index zero, and a hash function $(3x + 4) \bmod 7$. Assuming the hash table is initially empty, which of the following is the contents of table when the sequence 1, 3, 8,

10 is inserted into the table using closed hashing? Note that '_' denotes an empty location in the table.

1, 8, 10, _, _, _, 3

Which one of the following is known as "Last-In, First-Out" or LIFO Data Structure?

Stack

Parameters in function call are passed using,

Stack

If we singly linked list to implement list, then there is an issue that it gives difficulty when we:

Move backward in the list

Which of the following is correct about AVL Tree?

It is identical to BST except height of the left and right subtrees can differ by at most 1.

Insertion in a linked list can be done at

Front, Back and somewhere in the middle

If the height of a perfect binary tree is 4. What will be the total number of nodes in it?

31

AVL tree is linear data structure.

TRUE

During the union by size method, all the array element initialized to -1 shows:

Initial condition of tree

Consider the following pseudo code

declare a stack of characters

while (there are more characters in the word to read)

{

read a character

push the character on the stack

}

while (the stack is not empty)

{

pop a character off the stack

write the character to the screen

}

What is written to the screen for the input "apples"?

selppa

Consider the linked list having data [6, 72, 35, 65, 25] stored in it. While current pointer is pointing to memory location having 72 stored in it.

After calling add(4) function on the following linked list current point will point to memory location having value?

4

One difference between a queue and a stack is

Queues use two ends of the structure, stacks use only one.

next() method of List class is used to:

Moves the current position forward one element

get(?) method of list class is used to:

Get element at the given position

If we want to find median of 50 elements, then after applying buildHeap methd, how many times deleteMin method will be called?

25

In simple or singly linked list there is/are _____ pointer/s in each node.

One

If we have 1000 sets each containing a single different person. Which of the following relation will be true on each set:

Reflexive

The total number of nodes on 10th level of a perfect binary tree are :

1024

If there are 100 elements in a heap, and 100 deleteMin operation are performed, will get _____ list.

Sorted

In simple implementation of stack, isFull() method is used due to _____.

Limitation of array

If a tree has 50 nodes, then the total edges/links in the tree will be:

49

Following is a keyword of C++ _____ .

Delete

In level-order traversal for Binary Search Tree, at each level, we visit the nodes in _____ order.

Left-to-right

To search an element in AVL tree, it takes maximum $1.88 \log_2 n$ time.

FALSE

If we want to find 3rd minimum element from an array of elements, then after applying buildHeap method, how many times deleteMin method will be called?

3

If there are N elements in an array then the number of maximum steps needed to find an element using Binary Search is _____.

$\log_2 N$

The _____ method of list data structure removes the element residing at the current position

Remove

Allocating and de-allocating memory for linked list nodes take _____ time than pre-allocated array.

More

In a tree, we link the nodes in such a way that it _____ a linear structure.

Forms

Longest path from root node to farthest leaf node is called _____ of tree.

Depth

In which of the following function signatures, the value of variable "num" cannot be changed in function body?

`int cube(int num)`

Binary search algorithm cannot be applied to _____ .

`sorted linked list`

The expression
if (! heap ->isFull())
check

`Heap is not full`

Which of the following is correct statement?

`An AVL tree is identical to BST except height of the left and right subtree can differ by at most 1.`

The easiest case of deleting a node from BST is the case in which the node to be deleted _____.

`Is a leaf node`

Every AVL is _____.

`Binary Search Tree`

_____ is when function is calling to itself.

`Recursion`

For a perfect binary tree of height h, having N nodes, the sum of heights of nodes is _____.

`N - h - 1`

The binary tree is the maximum level of its leaves (also called the depth).

Height

Which one of the following is NOT true regarding the skip list?

List Sh contains only the n special keys

Suppose we have been given the following data set for a Queue.

7 5 2 4

What will be the resultant Queue if we call enqueue(3) method?

Note that 7 is the front element whereas 4 is rear element of queue.

7 5 2 4 3

If there are N external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

N -1

int htdiff = height(root->getLeft()) _____ height(root->getRight());

The above line of code is taken from AVL insert method. Complete it by selecting an appropriate symbol.

-

Which of the following can be used to reverse a string value?

Stack

The next field in the last node of a singly-linked list is set to _____.

NULL

If there are N internal nodes in a binary tree then what will be the no. of external nodes in this binary tree?

N + 1

A hash function returns a _____ value.

Integer

The union operation is based on size or weight but the reducing the in-between links or path compression from _____ to the _____ is done by the find method.

nodes, root

If there are 23 external nodes in a binary tree then what will be the no. of internal nodes in this binary tree?

22

_____ in AVL is logarithmic.

Searching

In case of insertion of right inner node in BST,

we need to apply single left rotation to make it AVL tree.

Consider the following infix expression.

$7/8 + 9$

If one converts the above expression into postfix. What would be the resultant expression?

7 8 9 / +

What are the basic things associated with data structures?

All of the given

Which of the following properties are satisfied by Equivalence relationship?

Reflexive, Symmetric and Transitive

For compiler, a postfix expression is easier to evaluate than infix expression?

True

The expression $DE+H^*$ is called _____.

Hybrid expression

Consider the following expression:

$x - y * a + b / c$

Which of the following is a correct equivalent expression for the above?

$x y a * - b c / +$

In the call by _____ methodology, a copy of the object is passed to the called function.

Value

There are four cases of rotation in an _____ tree.

AVL

In singly linked list which node will keep track of starting position of the list.

Next Node

The depth of a binary tree is

Maximum level of a leaf

What will be the value of root of an AVL and BST if built from the same data?

Root of AVL is square of root of BST

In 1990, Bill pugh proposed an enhancement on linked lists and the new data structure was termed as

Skip list

Avl tree takes maximum _____ time to search an element.

1.44 Log₂n

Can we store elements with different data types in a single array?

Yes

The _____ of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

Balance

What is the depth of any tree if the union operation is performed by height?

O(log N)

If a hash function returns 4 for more than one value, this condition is called _____.

Collision

Which one of the following calling methods does not change the original value of the argument in the calling function?

Call by passing the value of the argument

Every AVL tree is a binary search tree.

TRUE

The difference between a "Binary Tree (BT)" and a "Binary Search Tree (BST)" is that,

In BST nodes are inserted based on the values they contain

The worst case of searching in binary search tree (BST) is:

When the data inserted in BST is sorted

Suppose there are 100 elements in an equivalence class, so initially there will be 100 trees. The collection of these trees is called

Forest

Factorial is an example of _____ function.

Recursive

Suppose a hash function returns 6 for the given value. At which index of array the value will be saved?

6

A binary tree of N nodes has _____ .

$\log_2 N$ levels

Every AVL is _____.

Binary Search Tree

If Ahmad is cousin of Ali and Ali is cousin of Asad then Ahmad is also cousin of Asad. This statement has the following property

Transitivity

What is the time complexity of binary search with iteration?

$O(\log n)$

A solution is said to be efficient if it solves the problem within its resource constraints i.e. hardware and time.

True

Searching an element in an AVL tree takes maximum _____ time (where n is number of nodes in AVL tree).

1.44 $\log_2 n$

Which of the following is not a form of expression?

Pastfix

Heap can be used to implement

Priority Queue

Sorting procedure normally takes _____ time.

$N \log N$

Consider we have performed the following operations on stack of size 5.

Push(10);

Push(20);

Push(30);

Pop();

Pop();

Push(40);

20

Which one of the following is NOT the property of equivalence relation?

Associative

What is the hash function used in linear probing?

$h_i(x) = (\text{hash}(x) + f(i)) \bmod \text{table size}$

Recursive call of a function use _____ data structure.

Stack

Left, right, info, and parent are the operation of _____ data structure.

Tree

For Binary Search Tree, we call the findMax() method as _____.

findMin(tree->getLeft())

From Operating System point of view, the recursive function calls are made with the help of _____.

Stack

Which of the following is true regarding the maze generation?

All of the given

If there are 100 elements in an equivalence class then we will have _____ sets initially.

100

given the values are the array representation of heap: 12 23 26 31 34 44
56 64 78 100

What is the 5th smallest element in the given heap?

34

Which operation of queue data structure is used to get front element from the queue and then remove it from the queue?

enqueue()

_____ is an area in computer memory that is allocated dynamically.

Heap

The tree structure is a

Non-linear data structure

Binary Tree traversal can be performed with the help of _____ .

Both Recursive and Non-recursive calls

New items are added at the _____ of the stack.

Top

Leaf node of binary search tree contains _____.

All of the given

Suppose there are a set of fruits and a set of vegetables. Both sets are _____ sets.

Disjoint

Which of the following operations returns "most recently entered value" from the stack?

Top

back() method of list class is used to:

Moves the "current" pointer to backward one element.

Stack and Queue can be implemented using _____.

Singly Link List

The array in binary search is sub divided _____.

Untill a sublist is no more divisible

Which of the following heap method lowers the value of key at position 'p' by the amount 'delta'?

`decreaseKey(p,delta)`

What will be the postfix expression of following infix expression?
 $A*B/C+D-E$

$A\ B\ +\ C\ D\ /\ +\ E\ -$

An efficient program executes faster and helps in _____ the usage of resources like memory and disk.

Maximizing

Array cells are _____ in computer memory.

Contiguous

Suppose there is an image of $7 * 7$, now we will have matrix of _____ rows and _____ columns.

7, 7

An array is a group of _____ memory locations.

Consecutive

for every process executing, the last part of the memory is for _____ of the program.

Heap

What will be the postfix expression of following infix expression?
 $D+E*F/G$

$DEF*G/+$

The expression
if (! heap ->isEmpty())
checks

Heap is not empty

_____ is used for Reference variables in C++.

&

For searching a particular number in Binary Search Tree (if it is not present), the maximum number of comparisons will be _____ comparison at each level.

2

Suppose n is the number of nodes in a complete Binary Tree, then maximum steps required for a search operation are

$\log_2(n+1) - 1$

If we use array to implement list, then there is an issue that it gives difficulty when:

We will increase its size

Consider the following function:

```
void test_a(int n)
{
    cout << n << " ";
    if (n>0)
        test_a(n-2);
}
```

What is printed by the call test_a(4)?

4 2 0

Doubly Linked List always has _____ NULL pointer/s in a node.

Two

The percolateDown procedure will move the smaller value _____ and bigger value _____.

up, down

What will be the result of evaluation following expression?

$5+3*2/(6-3)$

7

_____ only removes items in reverse order as they were entered.

Stack

Linked list always contains elements that can be described as,

Self-referential

What will be postfix expression of the following infix expression?

Infix Expression: $a+b*c-d$

$abc*+d-$

In tree, the search operation is _____ as compared to the linked list.

Very fast

Each node in doubly link list has

1 pointer

Which of the is NOT true regarding the maze generation?

Remove a randomly chosen wall if the cells it separates are already in same set

The new operation in C++ for dynamically allocating memory returns,

Pointer to the memory it has allocated

Which of the following applications may use a stack?

Parentheses balancing program

Which one of the following statement is correct?

Array size is fixed once it is created

What will be result of following postfix expression?

1 2 3 * + 2 -

4

When we compare recursive method calls and non-recursive method calls, following statement is true.

Recursion is implemented in the same way as other function calls are implemented

A queue where the dequeue operation does not depend upon FIFO, is called:

Priority queue

A Linear Data Structure is the data structure in which data elements are arranged in a sequence or a linear list. Which of the following is Non Linear Data Structure?

Binary Search Trees

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

isEmpty(), isFull()

`Y = &x[0];`

In the above statement, we get address of the first location of the array x and store it in y. Here "y" is:

lvalue

Each node in singly link list has,

1 pointer

If ahmad is boss of ehsan and ehsan is boss of umer then ahmad is also boss of umer.

The above mentioned relation is _____.

Transitive

Which of the following possible operation are performed on Table ADT?

Insert, Find, Remove

Which of the following operation returns but do not removes top value of the stack?

top

While implementing non-recursive traversal for Binary SearchTree, we need to implement _____ .

Stack

~BinarySearchTree() is a _____ .

Constructor

Which property of equivalence relation is satisfied if we say:
Ahmad is cousin of Ali and Ali is also cousin of Ahmad

Symmetry

The total number of nodes on 5th level of a perfect binary tree are :

32

In C++, we place the class interface in _____ file.

.cpp

The variables which are destroyed automatically when a function's execution ends are:

Local variables defined inside function body

If the elements are 3, 5, 1, 6 in which order they will be removed from the queue one by one

3516

The.....of a binary tree is a maximum level of its leaves (also the depth).

height

.....of the an empty AVL tree is define to be

Height

Following is the keyword of c++

Delete

In the perspective of memory organization every process expecting the first part of memory is for.....of the program.

code

The.....of a node in a binary tree is define as the height of its left subtree minus height of its right subtree.

balance

If p is pointing to a node in an existing tree then info (p)means

Return the content of node p.

Following is a keyword of c++......

Delete

Every.....tree is a binary search tree

AVL

If a function has recursive call as last statement it is known as.....

Tail recursive

During in a order traversal call if we found a node is null it means this node will satisfy following condition

it will not have left child

In binary search tree deleting a node is easy if it is a..... node

Root

Which of the following operation returns the content of binary tree node?

Info (p)

1 2 6

5 8

3

4

1 2 3 4 5 6 8

in a complete binary tree the number of node of level 5 are

30

One should be careful about transient.....that are stored by reference I
data structure

Objects

In a tree we link the node in such a way that it.....a linear structure

does not remain

the function call are mode with the help of.....

stack

18. 15 25 45

35 75

15 35 25 75 45

A binary is said to be a.....binary tree if every non leaf node in binary
tree has non empty left and right structure

Strictly

In which of the following function signature the value of variable num
cannot be changed in function body

Binary search tree

Two common method of simulations are.....are.....

Time based simulation and event based simulation

Generally there are case (s)to delete a node from BST

1

When a function calls another function the parameters and address of the calling function are put in...

Stack

Binary search tree ()is a.....

Destructor

We can calculate the.....of subtree by counting its level from the bottom

Nodes

If there is a strictly complete binary tree of depth 3 the total number of nodes it will be

7

A model attempt to model real word phenomenon

Simulation

The process of getting the value of a variable using a pointer is called

Dereferencing

When a function calling itself is called as.....

Recursion

A.....is a tree in which every except possibly the last, is completely filled

Strict binary tree

In AVL tree during insertion, a single rotation can fix the balance in cases _____ and 4.

1

If a node is inserted in outer side of a node in binary search tree then to make it AVL tree,

We may have to apply single rotation

A zigzag rotation is performed. In Left-Left case of rotation in AVL tree.

False

The principal benefit of a linked list over a conventional array is that the order of the linked items may be _____ from the order that the data items are stored in memory.

Same

Stack.push(15) will push 15 on _____.

Top of the stack

Suppose we have the following values to be inserted in constructing AVL tree,

10,13,15,5,7,8

Tell when first rotation will take place,

After inserting the node 15

All the objects created using _____ operator have to be explicitly destroyed using the delete operator.

New

During the execution of a process, operating system constructs four things for that process. Which of the following is not part of that process?

Linked list

There are _____ cases of Rotation in AVL tree.

4

Which of the following line of code is incorrect?

Float *I = float new (not confirmed)

In simple or singly linked list there is/are _____ pointer/s in each node.

One

Suppose there are three nodes tree with three numbers 14, 4, 15. Following is not a permutation, or combination for output them.

(4, 4, 15)

The type of expression in which operator succeeds its operands is _____ expression.

Postfix

The _____ of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

Balance

If both left and right nodes of a node are NULL then this type of node is called a _____ node.

Non leaf (not confirmed)

The function calls are made with the help of_____.

Stack

Which data structure is needed to convert infix expression to postfix expression?

Stack

In case of insertion of right inner node in BST,

Left rotation then right to make AVL

A BST generated from the data in ascending order is _____.

Linear

Which operation of queue data structure is used to get front element from the queue and then remove it from the queue?

Remove ()

In _____ various cells of memory are not located continuously.

Linked list

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

Start

Which operation of queue data structure is used to insert an element into the Queue?

Enqueueer ()

From Operating System point of view, the recursive function calls are made with the help of _____.

Queue

We can make a lexicographic order of characters based on their _____.

ASCII values

What are the basic things associated with data structures?

All of the above

Consider the following push operations of a Stack:

Stack.push(4);

Stack.push(6);

Stack.push(5);

Stack.push(8);

If a user calls a pop() operation, then which value will be returned?

8 (stack is last in first out)

~BinarySearchTree() is a _____.

Destructor

Array cells are _____ in computer memory.

Contiguous

Leaf node of binary search tree contains _____

One Null Pointer

dequeue() operation of queue data structure is used to_____

remove an element from the front and return it

The depth of a binary tree is

Maximum level of a leaf

Which of the following operation returns but do not removes top value of the stack?

Top

Which of the following data structure is linear type?

All of above

In doubly linked list a node consists of three parts:

2 pointers and 1 object

Last node in circular linked list contains

No null pointer

A_____ model attempts to model a real-world phenomenon

Simulation

Factorial is an example of _____function.

Recursive

The balance of a node in a binary tree is defined as the height of its _____ sub tree minus height of its right sub tree.

Left

In case of insertion of left outer node in BST,

We first apply left rotation and then rotation to make it an AVL Tree.

`int htdiff = height(root->getLeft()) height(root->getRight());` The above line of code is taken from AVL insert method. Complete it by selecting an appropriate symbol.

Minus (-)

Each operator in a postfix expression refers to the previous _____ operand(s).

Two

For making Binary Search Tree for Strings we need, _____ data type.

Int

A _____ model attempts to model a real-world phenomenon

Simulation

The lifetime of a transient object cannot exceed that of the application.

True

Which one of the following calling method does not change the original value of the argument in the calling function?

Call by passing the value of the argument

In level-order traversal for Binary Search Tree, we visit the nodes at each level before proceeding to the next level, in a _____ order.

Left-to-right

In doubly linked list a node consists of three parts:

1 pointer and 2 objects

Binary Search Tree violates the condition of AVL tree when any node has balance equal to

1 or -1

To search an element in AVL tree, it takes maximum $1.88 \log_2 n$ time

True

How many cases of rotation are there in AVL tree?

4

_____ tree has been named after two persons Adelson-Velskii and Landis.

AVL

Which of the following is a nonlinear data structure?

Tree

AVL tree is nonlinear data structure.

True

_____ is when function is calling to itself.

Recursion

Which of the following function doesn't belong to the stack class?

Crash ()

Which one is the correct function call for the following function of calculating cube?

int cube(int& num)

cube (num)

Binary search algorithm cannot be applied to ____

None of the given

copy() method of list data structure _____

Set one list to be a copy of another

While implementing non-recursive traversal for Binary Search Tree, we need to implement

Stack

Following is true in case of using Recursive method calls

The code becomes very short

_____is a self-balancing tree.

Binary Search Tree

While implementing stack with an array and to achieve LIFO behavior, we used push and pop elements at

The start of the array

Generalized code written for a class is called

Structure

Army cells are _____ in computer memory

Contiguous

In singly linked list a node comprises of _____ field/s.

Two

Which of the following statement is false?

Pointers store the next data element of a list

Leaf node of binary search tree contains _____

One null Pointer

What will be the result of evaluating following expression? $5+3*2/(6-3)$

7

Want and de-allocating memory for linked list nodes does take more than pre-allocated array

MORE

Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

Compiler will automatically generate a default constructor

Factorial is an example of _____ function.

Recursive

A binary tree is said to be a _____ binary tree if every non-leaf node in a binary tree has non-empty left and right subtrees.

Strictly

Memory address is stored in

Pointer

In level-order traversal for Binary Search Tree, _____ data structure is used.

Queue

Which of the following is the correct option for priority Queue?

The type of queues that is FIFO i.e. the person who comes first should leave first.

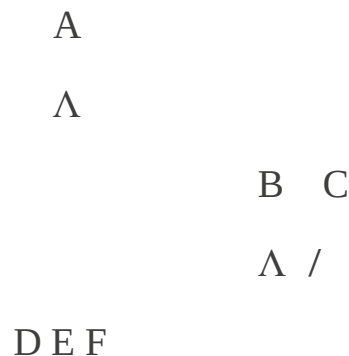
AVL tree is linear data structure.

False

AVL tree is a binary search tree

True

The post order traversal of a binary tree is DEBFCA. Find out the pre order traversal



ABDCEF

In a tree, we link the nodes in such a way that it _____ a linear structure.

Does not remain

add(12) method of linked list class will:

Add 12 as value in linked list

Which of the following is the correct conversion of infix to postfix expression?

$Z+B-(D-H)/K$

$ZB+DH-K/-$

In singly linked list “next” field of node contains:

Address of next node

Which of the following line of code is incorrect?

The lifetime of a transient object can exceed that of the application which is accessing it.

A list is the collection of items of the _____

same type

back() method of List class is used to:

Moves the “current” pointer to backward one element.

Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

_____ is the maximum height of the AVL tree.

$1.44\log_2 n$

length() method of List class is used to:

Return the length of the list

There are four cases of rotation in an _____ tree.

AVL

Allocating and de-allocating memory for linked list nodes does take _____ time than pre-allocated array.

More

_____ is utilized at the time of memory allocation in dynamic manner.

Heap

In which case of insertion we require double rotation to make the AVL tree balance.

None

The back() method decreases the value of variable current by _____.

One

In array list the worst case of removing an element is

To remove the item from start

Left, right, info, and parent are the operations of _____ data structure.

Tree

Which of the following is TRUE for search operations in a binary tree as compared to linked list or an array?

It increase the number of comparison

Which of the following is a correct statement?

An AVL tree is identical to BST except the height of left and right subtree can differ by at least 1

length() method of List class is used to:

Return the length of the list

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.

Return Address

Whenever we call a function, the compiler makes a _____ that it uses to fulfill this function call.

Stack

_____ is the major factor to see the efficiency of a program.

Time

_____ rule applies for evaluating operators of same precedence in an expression

None

The postfix form of the expression $A + B * C$ and $(A + B) * C$ will be _____.

Same

If we use array to implement list, then there is an issue that it gives difficulty when:

We will access value randomly

During in-order traversal using recursive calls, if we found a node is NULL. It means this node will satisfy following condition.

It will not have left child

Which one the following is more closer to AVL tree, (chose the best option)

Binary Search Tree

start() method of List class is used to:

Move the current pointer to first element

Consider the following infix expression. $7/8 + 9$

If one converts the above expression into postfix, what would be the resultant expression?

$78/9+$

In internal memory organization of a process, there is some area of memory for static data that holds _____ variables.

Both Static and Global

In Left-Right case of rotation in _____ tree. A double rotation is performed.

AVL

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

5

Which one is not the property of binary tree?

Sibling node should be same parent (Not Sure)

Each node in singly linked list contains _____

One Pointers

What's wrong with following loop? `while( (i < 10) && (i > 24)) { }`

The Condition is always false

Which one of the following is TRUE about recursion?

1. Recursive function calls consume a lot of memory.
2. Recursion extensively uses stack memory.
3. Recursion extensively uses stack memory.
4. Iteration is more efficient than iteration.

Which of the following is not a data structure ?

1. Linked list
2. Queue
3. stack
4. Memory cell

“+” is a _____ operator.

1. Unary
2. Binary
3. Ternary
4. None of the above

When an executable program run, it is loaded in the memory and becomes a_____.

1. Thread
2. .h file
3. Process
4. None of the above

www.vustudents.ning.com

In array list the worst case of removing an element is

1. To remove an element from the end of the list
2. To remove an element from the middle of the list
3. To remove an element from the start of the list
4. We cannot remove element from an array list

The tree data structure is a

1. Linear data structure
2. Graphical data structure
3. Non-linear data structure (Not Confirm)
4. Data structure like queue

In_____, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node.

1. Linked list
2. doubly-link list
3. array
4. Structure

Each operator in a postfix expression refers to the previous _____ operand(s)

1. one
2. two
3. three
4. four

Which of the following operations returns top value of the stack?

1. push
2. pop
3. top
4. First

In the calling function, after the execution of the function called, the program continues its execution from the _____ after the function call.

1. Previous line
2. Next line
3. Beginning
4. None of the above

_____ only removes items in reverse order as they were entered.

1. Queue
2. Stack
3. Both of these
4. None of these

In the linked list implementation of the stack class, where does the push member function places the new entry on the linked list?

1. After all other entries that are greater than the new entry.
2. At the head
3. After all other entries that are smaller than the new entry.
4. At the tail

Which of the following is known as "Last-In, First-Out" or LIFO Data Structure?

1. Linked List
2. Stack

3. Queue

4. Tree

A queue is a _____ data structure, whereas a stack is a _____ data structure.

1. FIFO, LIFO

Each node in doubly link list has

1. 2

2. 3

3. 4

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function

1. First argument

2. Return address

3. Last argument

4. none of the above

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()

2. pop(), push()

3. isEmpty() , isFull()

Stack and Queue can be implemented using _____,

1. Singly Link List

2. Binary Tree

3. Binary Search Tree

4. AVL Tree

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

1. Remove
2. next
3. start
4. back

In array list the worst case of removing an element is

1. To remove an element from the end of the list
2. To remove an element from the middle of the list
3. To remove an element from the start of the list
4. We cannot remove element from an array list

STL is a _____ that is a part of the official standard of C++.

1. C program file
2. .h file

When an executable program run, it is loaded in the memory and becomes a_____.

1. Thread
2. .h file
3. Process
4. None of the above

Each node in doubly link list has,

1. pointer
2. pointers
3. pointers
4. pointer

A queue is a _____data structure, whereas a stack is a _____data structure.

1. FIFO, LIFO

2. LIFO,FIFO
3. both of these
4. none of these

Which one of the following is TRUE about recursion?

1. Recursive function calls consume a lot of memory.
2. Recursion extensively uses stack memory.
3. Recursion extensively uses stack memory.
4. Iteration is more efficient than iteration.

To create a _____ we link the last node with the first node in the list.

1. Double linked list

2. Circularly-linked list
3. Linked list
4. None of the above

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()
2. pop(), push()
3. isEmpty() , isFull()
4. push(),pop()

The principal benefit of a linked list over a conventional array is that the order of the linked items may be_____ from the order that the data items are stored in memory.

1. Same

2. Identical

3. Different

4. Equivalent

In_____ the 'next' returns false when it reaches to the last node due to the fact that the next field of the last node is set to NULL.

1. Circular linked list

2. Triple linked list

3. Singly linked list

4. None of the above

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

1. 48

2. 4

3. 333

4. 5

Which boolean expression indicates whether the numbers in two nodes (p and q) are the same. Assume that neither p nor q is null.

1. `p == q`
2. `p.data == q.data`
3. `p.nextNode == q. nextNode`
4. `p.data == q`

Local variables of a function are stored in,

1. Binary Search Tree
2. Stack
3. Queue
4. AVL Tree

A queue is a data structure where elements are

1. inserted at the front and removed from the back.
2. inserted and removed from the top.
3. inserted at the back and removed from the front.
4. inserted and removed from both ends.

A kind of expressions where the operator is present between two operands called _____ expressions.

1. Infix
2. Postfix
3. Prefix
4. None of the above

A template is a function or class that is written with a _____ data type.

1. Specific
2. Definite
3. Generic
4. None of the above.

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.

1. First argument
2. Return address
3. Last argument
4. None of the above

Which of the following can be used to reverse a string value,

1. Stack
2. Queue
3. Both of these
4. None of these

A queue is a _____data structure, whereas a stack is a _____data structure.

1. FIFO, LIFO
2. LIFO,FIFO
3. both of these
4. none of these

Stack and Queue can be implemented using _____,

1. Singly Link List
2. Binary Tree

3. Binary Search Tree

4. AVL Tree

Which of the following is not a data structure ?

1. Linked list
2. Stack
3. Queue
4. Memory cell

Compiler uses which one of the following in Function calls,

1. Stack
2. Queue
3. Binary Search Tree
4. AVL Tree

The principal benefit of a linked list over a conventional array is that the order of the linked items may be_____ from the order that the data items are stored in memory.

1. Same
2. Identical
3. Different
4. Equivalent

Doubly Linked List always has one NULL pointer.

1. True
2. False

We cannot remove items randomly from _____

1. Stack
2. Queue
3. Both of these
4. None of these

_____ is the maximum number of nodes that you can have on a stack-linked list ?

1. Zero
2. $2n$ (where n is the number of nodes in linked list)
3. Any Number
4. None of these

“+” is a _____ operator.

1. Unary
2. Binary
3. Ternary
4. None of the above

In_____, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node.

1. Linked list

2. doubly-link list

3. array

4. structure

A queue is a data structure where elements are

1. inserted at the front and removed from the back.

2. inserted and removed from the top.

3. inserted at the back and removed from the front.

4. inserted and removed from both ends.

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()

2. pop(), push()

3. isEmpty() , isFull()

4. push(),pop()

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

1. Remove
2. Next
3. Start
4. Back

Parameters in function call are passed using,

1. Stack
2. Queue
3. Binary Search Tree
4. AVL Tree

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

1. 48
2. 4
3. 333
4. 5

For a perfect binary tree of height 4, what will be the sum of highest of node

1. 31
2. 30
3. 27
4. 26

If Ahmed is cousin of Ali and Ali is cousin of Asad then Ahmed is also cousin of Asad. This statement has the following property

1. Reflexivity
2. Symmetry
3. Transitivity
4. All of the above

Which one of the following is NOT the property of equivalence relation?

1. Reflexive
2. Symmetric
3. Transitive
4. Associative

The main reason of using heap in priority queue is

1. Improve performance
2. Code readable
3. Less code
4. Heap can't be used in priority queues

The total numbers of nodes on 10th level of perfect binary tree are

1. 256
2. 512
3. 1024

4. Can't be determined

Suppose there are 100 elements in an equivalence class, so initially there will be 100 trees, the collection of these trees is called _____.

1. Cluster

2. Class

3. Forest

4. Bunch

The percolate Down procedure will move the smaller value and bigger value.

1. Left, right

2. Right, left

3. Down, up

4. Up, down

For a perfect binary tree of height h , having N nodes, the Sum of height of nodes is

1. $N - h - 1$

2. $N - 1$
3. $N - 1 + h$
4. $N - (h - 1)$

Which of the following method is helpful in creating the heap at once?

1. Insert
2. Add
3. Update
4. **percolateDown**

If ahmad is boss of ehsan and ehsan is boss of umer then ahmad is also boss of

umer, the above-mentioned relation is_____ .

1. Reflexive
2. Symmetry
3. **Transitive**
4. None of given

If we want to find median of 50 elements, then after applying builtHeap method, how manytime deleteMin method will be called?

1. 5
2. 25
3. 35
4. 50

Which of the following properties are satisfied by equivalence relationship?

1. Reflexive, symmetric
2. Reflexive, transitive
3. Symmetric, transitive
4. Reflexive, symmetric and transitive

Sorting procedure normally takes____ time.

1. $N \log N$
2. $2N$
3. $N * N * N$

4. N

15. The Expression

if (! heap->isFull())

Check

1. Heap is empty
2. Heap is full
3. Heap is not empty
4. Heap is not full

The Expression

if (! heap->isEmpty())

Check

1. Heap is empty
2. Heap is full
3. Heap is not empty
4. Not a valid expression

Given the values are the array representation of heap: 12 23 26 31 34 44
56 64 78

100 If we perform 4 deleteMin operation, the last element deleted is_____ .

1. 31

2. 34

3. 44

4. 56

Which of the following heap method increase the value of key at position 'p' by the amount 'delta'?

1. increaseKey(p, delta)

2. decreaseKey(p, delta)

3. percolateDown(p, delta)

4. remove(p, delta)

Which of the following heap method lowers the value of key at position 'p' by the amount 'delta'?

1. increaseKey(p, delta)

2. decreaseKey(p, delta)

3. percolateDown(p, delta)
4. remove(p, delta)

Which property of equivalence relation is satisfied if we say:

Ahmad R(is related to)Ahmad

1. Reflexivity
2. Symmetry
3. Transitivity
4. All of Above

Given the values are the array representation of heap;12 23 26 31 34 44
56 64 78 100 What is the 5th smallest element in the given heap?

1. 31
2. 34
3. 44
4. 56

The _____ of a binary tree is the maximum level of its leaves (also called the depth).

Select correct option:

Level

Width

Height

None of the above

Question # 2 of 10 (Start time: 06:40:41 PM) Total Marks: 1

If we write functions for recursive and non recursive inorder traversal method of BST, what will be the difference between its functions prototypes?

Select correct option:

Different return types

Different function names

Different arguments list

Nothing will be different

Question # 3 of 10 (Start time: 06:40:41 PM) Total Marks: 1

binary tree whose every node has either zero or two children is called

Select correct option:

Complete binary tree

Binary search tree

Strictly binary tree

None of above

Question # 4 of 10 (Start time: 06:40:41 PM) Total Marks: 1

Which one of the following calling method does not change the original value of the argument in the calling function?

Select correct option:

Call by passing reference of the argument

Call by passing the address of the argument

Call by passing the value of the argument

None of the given options

Question # 5 of 10 (Start time: 06:40:41 PM) Total Marks: 1

Longest path from root node to farthest leaf node is called _____ of tree

Select correct option:

Level

Length

Depth

Node level

Question # 6 of 10 (Start time: 06:40:41 PM) Total Marks: 1

Which of the following statement is false?

Select correct option:

Arrays are dense lists and static data structure

data elements in linked list need not be stored in adjacent space in memory

pointers store the next data element of a list

linked lists are collection of the nodes that contain information part and next pointer

The _____ of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

Select correct option:

Height

Balance

Width

None of the above

A tree is an AVL tree if

Select correct option:

Any one node fulfills the AVL condition

At least half of the nodes fulfill the AVL condition

All the nodes fulfill the AVL condition

None of the given options

In a program a reference variable, say x, can be declared as

Select correct option:

`int &x ;`

`int *x ;`

`int x ;`

None of the given options

In which of the following functions, the value of variable cannot be changed in function body?

Select correct option:

`int cube( int num)`

`int cube(int& num)`

`int cube(const int& num)`

`int cube(int* num)`

CS301 quiz(1) file

By

<https://vuonlinehelp.blogspot.com/>

Compose by (ALI HASSAN)

If the elements are 3, 5, 1, 6 in which order they will be removed from the queue one by one

3516

The.....of a binary tree is a maximum level of its leaves (also the depth).

height

.....of the an empty AVL tree is define to be

Height

Following is the keyword of c++

Delete

In the perspective of memory organization every process expecting the first part of memory is for.....of the program.

code

The.....of a node in a binary tree is define as the height of its left

subtree minus height of its right subtree.

balance

If p is pointing to a node in an existing tree then info (p) means

Return the content of node p.

Following is a keyword of c++.....

Delete

Every.....tree is a binary search tree

AVL

If a function has recursive call as last statement it is known as.....

Tail recursive

During in a order traversal call if we found a node is null it means this node will satisfy following condition

it will not have left child

In binary search tree deleting a node is easy if it is a..... node

Root

Which of the following operation returns the content of binary tree node?

Info (p)

1 2 6

5 8

3

4

1 2 3 4 5 6 8

in a complete binary tree the number of node of level 5 are

30

One should be careful about transient.....that are stored by reference I data structure

Objects

In a tree we link the node in such a way that it.....a linear structure

does not remain

the function call are mode with the help of.....

stack

18. 15 25 45

35 75

15 35 25 75 45

A binary is said to be a.....binary tree if every non leaf node in binary tree has non empty left and right structure

Strictly

In which of the following function signature the value of variable num cannot be changed in function body

Binary search tree

Two common method of simulations are.....are.....

Time based simulation and event based simulation

Generally there are case (s)to delete a node from BST

1

When a function calls another function the parameters and address of the calling function are put in...

Stock

Binary search tree ()is a.....

Destructor

We can calculate the.....of subtree by counting its level from the bottom

Nodes

If there is a strictly complete binary tree of depth 3 the total number of nodes it will be

7

A model attempt to model real word phenomenon

Simulation

The process of getting the value of a variable using a pointer is called

Dereferencing

When a function calling itself is called as.....

Recursion

A.....is a tree in which every except possibly the last, is completely filled

Strict binary tree

In AVL tree during insertion, a single rotation can fix the balance in cases _____ and 4.

1

If a node is inserted in outer side of a node in binary search tree then to make it AVL tree,

We may have to apply single rotation

A zigzag rotation is performed. In Left-Left case of rotation in AVL tree.

False

The principal benefit of a linked list over a conventional array is that the order of the linked items may be_____ from the order that the data items are stored in memory.

Same

Stack.push(15) will push 15 on _____.

Top of the stack

Suppose we have the following values to be inserted in constructing AVL tree,

10,13,15,5,7,8

Tell when first rotation will take place,

After inserting the node 15

All the objects created using _____ operator have to be explicitly destroyed using the delete operator.

New

During the execution of a process, operating system constructs four things for that process. Which of the following is not part of that process?

Linked list

There are _____ cases of Rotation in AVL tree.

4

Which of the following line of code is incorrect?

Float *l = float new (not confirmed)

In simple or singly linked list there is/are _____ pointer/s in each node.

One

Suppose there are three nodes tree with three numbers 14, 4, 15. Following is not a permutation, or combination for output them.

(4, 4, 15)

The type of expression in which operator succeeds its operands is _____ expression.

Postfix

The _____ of a node in a binary tree is defined as the height of its left subtree minus height of its right subtree.

Balance

If both left and right nodes of a node are NULL then this type of node is called a _____ node.

Non leaf (not confirmed)

The function calls are made with the help of_____.

Stack

Which data structure is needed to convert infix expression to postfix expression?

Stack

In case of insertion of right inner node in BST,

Left rotation then right to make AVL

A BST generated from the data in ascending order is _____.

Linear

Which operation of queue data structure is used to get front element from the queue and then remove it from the queue?

Remove ()

In _____ various cells of memory are not located continuously.

Linked list

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

Start

Which operation of queue data structure is used to insert an element into the Queue?

Enqueueer ()

From Operating System point of view, the recursive function calls are made with the help of _____.

Queue

We can make a lexicographic order of characters based on their _____.

ASCII values

What are the basic things associated with data structures?

All of the above

Consider the following push operations of a Stack:

Stack.push(4);

Stack.push(6);

Stack.push(5);

Stack.push(8);

If a user calls a pop() operation, then which value will be returned?

8 (stack is last in first out)

~BinarySearchTree() is a _____.

Destructor

Array cells are _____ in computer memory.

Contiguous

Leaf node of binary search tree contains _____

One Null Pointer

dequeue() operation of queue data structure is used to _____

remove an element from the front and return it

The depth of a binary tree is

Maximum level of a leaf

Which of the following operation returns but do not removes top value of the stack?

Top

Which of the following data structure is linear type?

All of above

In doubly linked list a node consists of three parts:

2 pointers and 1 object

Last node in circular linked list contains

No null pointer

A _____ model attempts to model a real-world phenomenon

Simulation

Factorial is an example of _____ function.

Recursive

The balance of a node in a binary tree is defined as the height of its _____ sub tree minus height of its right sub tree.

Left

In case of insertion of left outer node in BST,

We first apply left rotation and then rotation to make it an AVL Tree.

`int htdiff = height(root->getLeft()) - height(root->getRight());` The above line of code is taken from AVL insert method. Complete it by selecting an appropriate symbol.

Minus (-)

Each operator in a postfix expression refers to the previous _____ operand(s).

Two

For making Binary Search Tree for Strings we need, _____ data type.

Int

A _____ model attempts to model a real-world phenomenon

Simulation

The lifetime of a transient object cannot exceed that of the application.

True

Which one of the following calling method does not change the original value of the argument in the calling function?

Call by passing the value of the argument

In level-order traversal for Binary Search Tree, we visit the nodes at each level before proceeding to the next level, in a _____ order.

Left-to-right

In doubly linked list a node consists of three parts:

1 pointer and 2 objects

Binary Search Tree violates the condition of AVL tree when any node has balance equal to

1 or -1

To search an element in AVL tree, it takes maximum $1.88 \log_2 n$ time

True

How many cases of rotation are there in AVL tree?

4

_____ tree has been named after two persons Adelson-Velskii and Landis.

AVL

Which of the following is a nonlinear data structure?

Tree

AVL tree is nonlinear data structure.

True

_____ is when function is calling to itself.

Recursion

Which of the following function don't belongs to the stack class?

Crash ()

Which one is the cored function call for the following function of calculating cube?

```
int cube(int& num)
```

cube (num)

Binary search algorithm cannot be applied to ____

None of the given

copy() method of list data structure _____

Set one list to be a copy of another

While implementing non-recursive traversal for Binary Search Tree, we need to implement

Stack

Following is true in case of using Recursive method calls

The code becomes very short

_____is a self-balancing tree.

Binary Search Tree

While implementing stack with an array and to achieve LIFO behavior, we used push and pop elements at

The start of the array

Generalized code written for a class is called

Structure

Array cells are _____ in computer memory

Contiguous

In singly linked list a node comprises of _____ field/s.

Two

Which of the following statement is false?

Pointers store the next data element of a list

Leaf node of binary search tree contains _____

One null Pointer

What will be the result of evaluating following expression?

$5 + 3 * 2 / (6 - 3)$

7

Want and de-allocating memory for linked list nodes does take less time than pre-allocated array

MORE

Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

Compiler will automatically generate a default constructor

Factorial is an example of _____ function.

Recursive

A binary tree is said to be a _____ binary tree if every non-leaf node in a binary tree has non-empty left and right subtrees.

Strictly

Memory address is stored in

Pointer

In level-order traversal for Binary Search Tree, _____ data structure is used.

Queue

Which of the following is the correct option for priority Queue?

The type of queues that is FIFO i.e. the person who comes first should leave first.

AVL tree is linear data structure.

False

AVL tree is a binary search tree

True

The post order traversal of a binary tree is DEBFCA. Find out the pre order traversal

A

Λ

B C

$\wedge /$

D E F

ABDCEF

In a tree, we link the nodes in such a way that it _____ a linear structure.

Does not remain

add(12) method of linked list class will:

Add 12 as value in linked list

Which of the following is the correct conversion of infix to postfix expression?

$Z+B-(D-H)/K$

ZB+DH-K/-

In singly linked list "next" field of node contains:

Address of next node

Which of the following line of code is incorrect?

The lifetime of a transient object can exceed that of the application which is accessing it.

A list is the collection of items of the _____

same type

back() method of List class is used to:

Moves the "current" pointer to backward one element.

Suppose you are writing a class for Node class and forgot to write the constructor of the class, then

_____ is the maximum height of the AVL tree.

$1.44\log_2 n$

length() method of List class is used to:

Return the length of the list

There are four cases of rotation in an _____ tree.

AVL

Allocating and de-allocating memory for linked list nodes does take _____ time than pre-allocated array.

More

_____ is utilized at the time of memory allocation in dynamic manner.

Heap

In which case of insertion we require double rotation to make the AVL tree balance.

None

The back() method decreases the value of variable current by _____.

One

In array list the worst case of removing an element is

To remove the item from start

Left, right, info, and parent are the operations of _____ data structure.

Tree

Which of the following is TRUE for search operations in a binary tree as compared to linked list or an array?

It increase the number of comparison

Which of the following is a correct statement?

An AVL tree is identical to BST except the height of left and right subtree can differ by at least 1

length() method of List class is used to:

Return the length of the list

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.

Return Address

Whenever we call a function, the compiler makes a _____ that it uses to fulfill this function call.

Stack

_____ is the major factor to see the efficiency of a program.

Time

_____ rule applies for evaluating operators of same precedence in an expression

None

The postfix form of the expression $A + B * C$ and $(A + B) * C$ will be _____.

Same

If we use array to implement list, then there is an issue that it gives difficulty when:

We will access value randomly

During in-order traversal using recursive calls, if we found a node is NULL. It means this node will satisfy following condition.

It will not have left child

Which one the following is more closer to AVL tree, (chose the best option)

Binary Search Tree

start() method of List class is used to:

Move the current pointer to first element

Consider the following infix expression. $7/8 + 9$

If one converts the above expression into postfix, what would be the resultant expression?

78/9+

In internal memory organization of a process, there is some area of memory for static data that holds _____ variables.

Both Static and Global

In Left-Right case of rotation in _____ tree. A double rotation is performed.

AVL

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

5

Which one is not the property of binary tree?

Sibling node should be same parent (Not Sure)

Each node in singly linked list contains_____

One Pointers

What's wrong with following loop? while((i < 10) && (i > 24)) { }

The Condition is always false

Which one of the following is TRUE about recursion?

1. Recursive function calls consume a lot of memory.
2. Recursion extensively uses stack memory.
3. Recursion extensively uses stack memory.
4. Iteration is more efficient than iteration.

Which of the following is not a data structure ?

1. Linked list

2. Queue
3. stack
4. Memory cell

“+” is a _____operator.

1. Unary
2. Binary
3. Ternary
4. None of the above

When an executable program run, it is loaded in the memory and becomes a_____.

1. Thread
2. .h file
3. Process
4. None of the above

www.vustudents.ning.com

In array list the worst case of removing an element is

1. To remove an element from the end of the list
2. To remove an element from the middle of the list
3. To remove an element from the start of the list
4. We cannot remove element from an array list

The tree data structure is a

1. Linear data structure

2. Graphical data structure
3. Non-linear data structure (Not Confirm)
4. Data structure like queue

In_____, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node.

1. Linked list
2. doubly-link list
3. array
4. Structure

Each operator in a postfix expression refers to the previous _____ operand(s)

1. one
2. two
3. three
4. four

Which of the following operations returns top value of the stack?

1. push
2. pop
3. top
4. First

In the calling function, after the execution of the function called, the program continues its execution from the _____ after the function call.

1. Previous line
2. Next line
3. Beginning
4. None of the above

_____ only removes items in reverse order as they were entered.

1. Queue
2. Stack
3. Both of these
4. None of these

In the linked list implementation of the stack class, where does the push member function place the new entry on the linked list?

1. After all other entries that are greater than the new entry.
2. At the head
3. After all other entries that are smaller than the new entry.
4. At the tail

Which of the following is known as "Last-In, First-Out" or LIFO Data Structure?

1. Linked List
2. Stack
3. Queue
4. Tree

A queue is a _____data structure, whereas a stack is a _____data structure.

1. FIFO, LIFO

Each node in doubly link list has

1. 2
2. 3
3. 4

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function

1. First argument
2. Return address
3. Last argument
4. none of the above

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()

2. pop(), push()
3. isEmpty() , isFull()

Stack and Queue can be implemented using _____,

1. Singly Link List
2. Binary Tree
3. Binary Search Tree
4. AVL Tree

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

1. Remove
2. next
3. start
4. back

In array list the worst case of removing an element is

1. To remove an element from the end of the list

2. To remove an element from the middle of the list
3. To remove an element from the start of the list
4. We cannot remove element from an array list

STL is a _____ that is a part of the official standard of C++.

1. C program file

2. .h file

When an executable program run, it is loaded in the memory and becomes a_____.

1. Thread

2. .h file

3. Process

4. None of the above

Each node in doubly link list has,

1. pointer

2. pointers

3. pointers

4. pointer

A queue is a _____data structure, whereas a stack is a _____data structure.

1. FIFO, LIFO

2. LIFO,FIFO

3. both of these

4. none of these

Which one of the following is TRUE about recursion?

1. Recursive function calls consume a lot of memory.

2. Recursion extensively uses stack memory.

3. Recursion extensively uses stack memory.
4. Iteration is more efficient than iteration.

To create a _____ we link the last node with the first node in the list.

1. Double linked list
2. Circularly-linked list
3. Linked list
4. None of the above

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()
2. pop(), push()

3. isEmpty() , isFull()

4. push(),pop()

The principal benefit of a linked list over a conventional array is that the order of the linked items may be_____ from the order that the data items are stored in memory.

1. Same

2. Identical

3. Different

4. Equivalent

In_____ the 'next' returns false when it reaches to the last node due to the fact that the next field of the last node is set to NULL.

1. Circular linked list

2. Triple linked list

3. Singly linked list

4. None of the above

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

1. 48

2. 4

3. 333

4. 5

Which boolean expression indicates whether the numbers in two nodes (p and q) are the same. Assume that neither p nor q is null.

1. `p == q`

2. `p.data == q.data`

3. `p.nextNode == q.nextNode`

4. `p.data == q`

Local variables of a function are stored in,

1. Binary Search Tree

2. Stack

3. Queue

4. AVL Tree

A queue is a data structure where elements are

1. inserted at the front and removed from the back.

2. inserted and removed from the top.

3. inserted at the back and removed from the front.

4. inserted and removed from both ends.

A kind of expressions where the operator is present between two

operands called _____expressions.

1. Infix

2. Postfix

3. Prefix

4. None of the above

A template is a function or class that is written with a _____data type.

1. Specific

2. Definite

3. Generic

4. None of the above.

Whenever we call a function, the compiler makes a stack, the top element of the stack is _____ of the function.

1. First argument

2. Return address

3. Last argument

4. None of the above

Which of the following can be used to reverse a string value,

1. Stack

2. Queue

3. Both of these

4. None of these

A queue is a _____data structure, whereas a stack is a _____data structure.

1. FIFO, LIFO

2. LIFO,FIFO
3. both of these
4. none of these

Stack and Queue can be implemented using _____,

1. Singly Link List
2. Binary Tree
3. Binary Search Tree
4. AVL Tree

Which of the following is not a data structure ?

1. Linked list
2. Stack
3. Queue
4. Memory cell

Compiler uses which one of the following in Function calls,

1. Stack
2. Queue
3. Binary Search Tree
4. AVL Tree

The principal benefit of a linked list over a conventional array is that the order of the linked items may be_____ from the order that the data items are stored in memory.

1. Same
2. Identical
3. Different
4. Equivalent

Doubly Linked List always has one NULL pointer.

1. True
2. False

We cannot remove items randomly from _____

1. Stack
2. Queue
3. Both of these
4. None of these

_____ is the maximum number of nodes that you can have on a stack-linked list ?

1. Zero
2. $2n$ (where n is the number of nodes in linked list)
3. Any Number
4. None of these

“+” is a _____operator.

1. Unary
2. Binary
3. Ternary
4. None of the above

In_____, a programmer uses two pointers in the node, i.e. one to point to next node and the other to point to the previous node.

1. Linked list
2. doubly-link list
3. array
4. structure

A queue is a data structure where elements are

1. inserted at the front and removed from the back.
2. inserted and removed from the top.

3. inserted at the back and removed from the front.

4. inserted and removed from both ends.

_____ is the stack characteristic but _____ was implemented because of the size limitation of the array.

1. isFull(),isEmpty()

2. pop(), push()

3. isEmpty() , isFull()

4. push(),pop()

The _____ method of list will position the currentNode and lastCurrentNode at the start of the list.

1. Remove

2. Next

3. Start

4. Back

Parameters in function call are passed using,

-
1. Stack
 2. Queue
 3. Binary Search Tree
 4. AVL Tree

If numbers 5, 222, 4, 48 are inserted in a queue, which one will be removed first?

1. 48
2. 4
3. 333
4. 5

For a perfect binary tree of height 4, what will be the sum of highest of node

1. 31
2. 30
3. 27

4. 26

If Ahmed is cousin of Ali and Ali is cousin of Asad then Ahmed is also cousin of Asad. This statement has the following property

1. Reflexivity
2. Symmetry
3. Transitivity
4. All of the above

Which one of the following is NOT the property of equivalence relation?

1. Reflexive
2. Symmetric
3. Transitive
4. Associative

The main reason of using heap in priority queue is

1. Improve performance

2. Code readable
3. Less code
4. Heap can't be used in priority queues

The total numbers of nodes on 10th level of perfect binary tree are

1. 256
2. 512
3. 1024
4. Can't be determined

Suppose there are 100 elements in an equivalence class, so initially there will be 100 trees, the collection of these trees is called _____.

1. Cluster
2. Class
3. Forest
4. Bunch

The percolate Down procedure will move the smaller value and

bigger value.

1. Left, right
2. Right, left
3. Down, up
4. Up, down

For a perfect binary tree of height h , having N nodes, the Sum of height of nodes is

1. $N - h - 1$
2. $N - 1$
3. $N - 1 + h$
4. $N - (h - 1)$

Which of the following method is helpful in creating the heap at once?

1. Insert
2. Add
3. Update

4. percolateDown

If ahmad is boss of ehsan and ehsan is boss of umer then ahmad is also boss of umer, the above-mentioned relation is_____ .

1. Reflexive
2. Symmetry
3. Transitive
4. None of given

If we want to find median of 50 elements, then after applying builtHeap method, how many times deleteMin method will be called?

1. 5
2. 25
3. 35
4. 50

Which of the following properties are satisfied by equivalence relationship?

1. Reflexive, symmetric

2. Reflexive, transitive

3. Symmetric, transitive

4. Reflexive, symmetric and transitive

Sorting procedure normally takes___ time.

1. $N \log N$

2. $2N$

3. $N * N * N$

4. N

15. The Expression

if (! heap->isFull())

Check

1. Heap is empty

2. Heap is full

3. Heap is not empty

4. Heap is not full

The Expression

if (! heap->isEmpty())

Check

1. Heap is empty
2. Heap is full
3. Heap is not empty
4. Not a valid expression

Given the values are the array representation of heap: 12 23 26 31
34 44 56 64 78

100 If we perform 4 deleteMin operation, the last element deleted is_____ .

1. 31

2. 34

3. 44

4. 56

Which of the following heap method increase the value if key at

position 'p' by the amount 'delta'?

1. increaseKey(p, delta)

2. decreaseKey(p, delta)

3. percolateDown(p, delta)

4. remove(p, delta)

Which of the following heap method lowers the value of key at position 'p' by the amount 'delta'?

1. increaseKey(p, delta)

2. decreaseKey(p, delta)

3. percolateDown(p, delta)

4. remove(p, delta)

Which property of equivalence relation is satisfied if we say:

Ahmad R(is related to)Ahmad

1. Reflexivity

2. Symmetry

3. Transitivity

4. All of Above

Given the values are the array representation of heap; 12 23 26 31 34 44 56 64 78 100 What is the 5th smallest element in the given heap?

1. 31

2. 34

3. 44

4. 56

SOHAIL ZUBAIR

PLEASE SUBSCRIBE MY CHANNEL ON YOUTUBE "TECH SOLO SOFT"

CS301 –DATA STRUCTURE GRAND QUIZ PREPARATION FILE

1. An efficient program executes _____ and helps _____ the usage of resources like memory, disk.
 - a. **Faster , minimize**
 - b. Slower, maximize
 - c. Faster, maximize
 - d. Minimize, faster
2. As computer applications are becoming_____, so there is need for more resources
 - a. Efficient
 - b. **Complex**
 - c. Simple
 - d. None of given
3. Within an organization data should be arranged in a way that it is easily_____.
 - a. Control
 - b. Secure
 - c. **Accessible**
 - d. Destroyed
4. The correct syntax to declare of array of 06 elements is?
 - a. **int x[6];**
 - b. int x[5];
 - c. int x{6};
 - d. int x{5};
5. An array is collection of cells of the _____ type.
 - a. Different
 - b. **Same**
 - c. Integer
 - d. String
6. Array occupies _____ memory area in the computer.
 - a. Separate
 - b. Collective
 - c. Different
 - d. **Contiguous**
7. Which of these best describes an array?
 - a) A data structure that shows a hierarchical behavior
 - b) **Container of objects of similar types**
 - c) Arrays are immutable once initialized
 - d) Array is not a data structure

8. How do you initialize an array in C?
- a) `int arr[3] = (1,2,3);`
 - b) `int arr(3) = {1,2,3};`
 - c) **`int arr[3] = {1,2,3};`**
 - d) `int arr(3) = (1,2,3);`
9. The _____ data structure is among the most generic of data structures.
- a. Array
 - b. **List**
 - c. Stacks
 - d. Queues
10. List is a set of elements in an/a _____ order.
- a. Exponential
 - b. Static
 - c. Dynamic
 - d. **Linear**
11. A List is collection of item of the _____ type.
- a. Different
 - b. **Same**
 - c. Integer
 - d. String
12. The function of `clear()`; perform what in List data structure ?
- a. Remove element at some position in the list
 - b. Set one list to be a copy of another
 - c. **Clear a list (remove all elements)**
 - d. None of the given
13. It is not necessary to always start the indexing from zero. Sometimes, it is required to start the indexing from 1.
- a. **True**
 - b. False
14. In this method, we do not add a new element to the list but simply move the pointer one element ahead.
- a. `start()`
 - b. `tail()`
 - c. **`next()`**
 - d. `back()`
15. When does the `ArrayIndexOutOfBoundsException` occur?
- a) Compile-time
 - b) **Run-time**
 - c) Not an error
 - d) Not an exception at all
16. A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as a ?
- a) **Queue**
 - b) Stack

- c) Tree
 - d) Linked list
17. Process of inserting an element in stack is called _____
- a) Create
 - b) Push**
 - c) Evaluation
 - d) Pop
18. The data structure required for Breadth First Traversal on a graph is?
- a) Stack
 - b) Array
 - c) Queue**
 - d) Tree
19. What does 'stack underflow' refer to?
- a) accessing item from an undefined stack
 - b) adding items to a full stack
 - c) removing items from an empty stack**
 - d) index out of bounds exception
20. What is the time complexity of pop() operation when the stack is implemented using an array?
- a) O(1)**
 - b) O(n)
 - c) O(logn)
 - d) O(nlogn)
21. It is impossible that one element of the array is located at a memory location while the other element is located somewhere far from it in the memory.
- a. True**
 - b. False
22. When we have declared the size of the array, it is _____ to increase or decrease it during the execution of the program.
- a. Possible
 - b. Not possible**
 - c. Depend on the programming language
 - d. Depend on the compiler
23. While calling a function, we pass values of variables to it. Such functions are known as
- a. Call by value**
 - b. Call by reference
 - c. Call by address
 - d. Call by location
24. A node comprises _____ fields.
- a. Two**
 - b. One
 - c. Three
 - d. Four
25. While calling a function, instead of passing the values of variables, we pass address of variables (location of variables) to the function known as

- a. Call by value
 - b. Call by reference**
 - c. Call by address
 - d. Call by location
26. The local variables of a function are created on _____.
- a. Heap
 - b. Array
 - c. call stack**
 - d. Linked List
27. All the variables or objects created in a function that we want to access later are created on memory heap, sometimes called?
- a. Memory bank
 - b. Heap store
 - c. Stack
 - d. free store**
28. Heap is an area in computer memory that is allocated _____.
- a. Statically
 - b. Manually
 - c. Dynamically**
 - d. Randomly
29. All the objects created using new operator have to be explicitly destroyed using the _____.
- a. delete operator**
 - b. kill operator
 - c. remove operator
 - d. clear operator
30. Nameless objects (objects accessed by pointers) are called _____.
- a. Free object
 - b. Pointer object
 - c. anonymous objects**
 - d. none of given
31. A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as a ?
- a) Queue**
 - b) Stack
 - c) Tree
 - d) Linked list
32. Which one of the following is an application of Queue Data Structure?
- (A) When a resource is shared among multiple consumers.
 - (B) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes
 - (C) Load Balancing
 - (D) All of the above**
33. The data structure required for Breadth First Traversal on a graph is?
- a) Stack
 - b) Array
 - c) Queue**
 - d) Tree

34. A queue follows _____
a) **FIFO (First In First Out) principle**
b) LIFO (Last In First Out) principle
c) Ordered array
d) Linear tree
35. Circular Queue is also known as _____
a) **Ring Buffer**
b) Square Buffer
c) Rectangle Buffer
d) Curve Buffer
36. If the elements "A", "B", "C" and "D" are placed in a queue and are deleted one at a time, in what order will they be removed?
a) **ABCD**
b) DCBA
c) DCAB
d) ABDC
37. A data structure in which elements can be inserted or deleted at/from both the ends but not in the middle is?
a) Queue
b) Circular queue
c) **Deque**
d) Priority queue
38. A normal queue, if implemented using an array of size MAX_SIZE, gets full when
a) **Rear = MAX_SIZE - 1**
b) Front = (rear + 1) mod MAX_SIZE
c) Front = rear + 1
d) Rear = front
39. Queues serve major role in _____
a) Simulation of recursion
b) Simulation of arbitrary linked list
c) **Simulation of limited resource allocation**
d) Simulation of heap sort
40. Which of the following is not the type of queue?
a) Ordinary queue
b) **Single ended queue**
c) Circular queue
d) Priority queue
41. How many stacks are needed to implement a queue? Consider the situation where no other data structure like arrays, linked list is available to you.
(A) 1
(B) **2**
(C) 3
(D) 4
42. Choose correct output for the following sequence of operations.

```
push(5)
```

```
push(8)
```

```
pop
push(2)
push(5)
pop
pop
pop
push(1)
pop
```

- a. 85251
 - b. 85521
 - c. 82551
 - d. 81255
43. Stack can be implemented using _____ and _____?
- a. Array & Binary Tree
 - b. Linked List & Graph
 - c. **Array & Linked list**
 - d. Queue & linked list
44. Postfix form of following expression.
- a. EF^*D+
 - b. **DEF^*+**
 - c. $DEF+^*$
 - d. EFD^*+
45. When function calls another function then the details of previous function are stored in Stack?
- a. **True**
 - b. False
46. Insertion and Deletion operation in Queue is known as
- a. Push & Pop
 - b. **Enqueue & Dequeue**
 - c. Insert & Delete
 - d. None of given
47. Stack data structure cannot be used for
- a. Implementation of recursive function
 - b. **Allocation resource & scheduling**
 - c. Reversing string
 - d. Evaluations of string in postfix form
48. What is the advantage of recursive approach than an iterative approach?
- a) Consumes less memory
 - b) **Less code and easy to implement**
 - c) Consumes more memory
 - d) More code has to be written

49. In which of the cases uniform binary search fails compared to binary search?
- a) A table lookup is generally faster than an addition and a shift
 - b) Many searches will be performed on the same array
 - c) Many searches will be performed on several arrays of the same length
 - d) Complexity of code**
50. Given an input arr = {2,5,7,99,899}; key = 899; What is the level of recursion?
- a) 5
 - b) 2
 - c) 3**
 - d) 4
51. What is the worst case complexity of binary search using recursion?
- a) $O(n \log n)$
 - b) $O(\log n)$**
 - c) $O(n)$
 - d) $O(n^2)$
52. What is the average case time complexity of binary search using recursion?
- a) $O(n \log n)$
 - b) $O(\log n)$**
 - c) $O(n)$
 - d) $O(n^2)$
- 53.** Which of the following is not an application of binary search?
- a) To find the lower/upper bound in an ordered sequence
 - b) Union of intervals
 - c) Debugging
 - d) To search in unordered list**
54. Binary Search can be categorized into which of the following?
- a) Brute Force technique
 - b) Divide and conquer**
 - c) Greedy algorithm
 - d) Dynamic programming
- 55.** Given an array arr = {5,6,77,88,99} and key = 88; How many iterations are done until the element is found?
- a) 1
 - b) 3
 - c) 4
 - d) 2**
56. Each element of a binary tree is called a _____ of the tree.
- a. Node**
 - b. Leaf
 - c. Mode
 - d. Child
57. We make (draw) a tree by joining different nodes with lines. But we _____ join any nodes whichever we want to each other.
- a. Can also
 - b. Cannot**
 - c. Depend on the nodes
 - d. None of given
58. In a tree there is always _____ path to go (from root) to a node.

- a. Two
 - b. One**
 - c. No path
 - d. None of the given
59. We can use the words descendant and child interchangeably.
- a. True**
 - b. False
60. A binary tree is said to be a strictly binary tree if every non-leaf node in a binary tree has non-empty left and right sub trees.
- a. True**
 - b. False
61. Level of any other node is one more than the level its_____.
- a. Child
 - b. Leaf
 - c. Descendant
 - d. Parent**
62. A complete binary tree of depth d is the strictly binary tree all of whose leaves are at level d".
- a. True**
 - b. False
63. We can say that in a complete binary tree at a particular level k, the number of nodes is equal to _____?
- a. $2K$
 - b. $2*2^k$
 - c. $2\log k$
 - d. 2^k**
64. Which of the following is not an operation on binary tree?
- a. `right(p)` returns pointer to right sub tree
 - b. `parent(p)` returns the father of p
 - c. `brother(p)` returns brother of p.
 - d. all of the given
65. A complete binary tree is necessarily a strictly binary tree but not vice versa.
- a. True**
 - b. False
66. Binary tree is useful structure when two-way decisions are made at each point.
- a. True**
 - b. False
67. What is an AVL tree?
- a) a tree which is balanced and is a height balanced tree**
 - b) a tree which is unbalanced and is a height balanced tree
 - c) a tree with three children
 - d) a tree with at most 3 children
68. Why we need to a binary tree which is height balanced?
- a) to avoid formation of skew trees**
 - b) to save memory
 - c) to attain faster memory access
 - d) to simplify storing

69. In post order traversal of binary tree right sub tree is traversed before visiting root.
a) **True**
b) False
70. What is the possible number of binary trees that can be created with 3 nodes, giving the sequence N, M, L when traversed in post-order.
a) 15
b) 3
c) **5**
d) 8
71. Which of the following pair's traversals on a binary tree can build the tree uniquely?
a) post-order and pre-order
b) **post-order and in-order**
c) post-order and level order
d) level order and preorder
72. To obtain a prefix expression, which of the tree traversals is used?
a) Level-order traversal
b) **Pre-order traversal**
c) Post-order traversal
d) In-order traversal
73. Which one is a self- referential data type?
a. Stack
b. Queue
c. Link list
d. **All of these**
74. Each node in doubly link list has,
a. 1 pointer
b. **2 pointers (Page 39)**
c. 3 pointers
d. 4 pointers
75. I have implemented the queue with a linked list, keeping track of a front pointer and a rear pointer. Which of these pointers will change during an insertion into an EMPTY queue?
a. Neither changes
b. Only front pointer changes.
c. **Only rear pointer changes.**
d. Both change.
76. The nodes with no successor are called _____
a. Root Nodes
b. **Leaf Nodes**
c. Both of these
d. None of these
77. Doubly Linked List always has one NULL pointer.
a. True
b. **False (Page 43)**
78. In which of the traversal method, the recursive calls can be used to traverse a binary tree ?
a. **In preorder traversal only (Page 143)**
b. In inorder traversal only
c. In postorder traversal only
d. All of the given options

79. What is the maximum height of an AVL tree with p nodes?
- a) p
 - b) $\log(p)$**
 - c) $\log(p)/2$
 - d) $p/2$
80. A tree is an AVL tree if
- a. Any one node fulfills the AVL condition
 - b. At least half of the nodes fulfill the AVL condition
 - c. All the nodes fulfill the AVL condition (Page 213)**
 - d. None of the given options
81. Consider the following sequence of push operations in a stack:
stack.push('7');
stack.push('8');
stack.push('9');
stack.push('10');
stack.push('11');
stack.push('12');
- a. 7 8 9 10 11 12**
 - b. 9 8 11 10 7 12
 - c. 9 10 8 11 12 7
 - d. 9 10 8 12 7 11
82. Which one of the following operators has higher priority than all of others?
- a. Multiplication operator**
 - b. Minus operator
 - c. Plus operator
 - d. Exponentiation operator
83. Which of the following is "TRUE" about arrays,
- a. We can increase the size of arrays after their creation.
 - b. We can decrease the size of arrays after their creation.
 - c. We can increase but can't decrease the size of arrays after their creation.
 - d. We can neither increase nor decrease the array size after their creation.**
84. Next item in a linked list is known as
- a. Index
 - b. Item
 - c. Node**
 - d. Child
85. Each operator in a postfix expression refers to the previous _____ operand(s).
- a. One
 - b. Two**
 - c. Three
 - d. Four
86. We access elements in AVL Tree in,
- a. Linear way only
 - b. Non Linear way only**
 - c. Both linear and non linear ways
 - d. None of the given options.
87. $A + B * C$ the postfix expression is _____?
- a. $+ A * B C$

- b. $+*ABC$
 - c. **$ABC*+$**
 - d. $A+*BC$
88. Data compression plays a significant role in _____?
- a. Application software
 - b. Database
 - c. Encoding scheme
 - d. **computer networks**
89. Huffman code is method for the compression of standard text documents.
- a. **True**
 - b. False
90. _____ is a data structure that can grow easily dynamically at run time without having to copy existing elements.
- a. Array
 - b. **List**
 - c. Both of these
 - d. None of these
91. "+" is a _____ operator.
- a. Unary
 - b. **Binary**
 - c. Ternary
 - d. None of the above
92. A _____ is a FIFO structure, which can make the level-order traversal easier.
- a. Array
 - b. List
 - c. **Queue**
 - d. Linked List
93. We can make a lexicographic order of characters based on their ASCII values.
- a. **True**
 - b. False
94. A queue is a _____ data structure, whereas a stack is a _____ data structure.
- a. **FIFO, LIFO**
 - b. LIFO, FIFO
 - c. none of these
 - d. both of these
95. Searching an element in an AVL tree take maximum _____ time (where n is no. of nodes in AVL tree),
- a. $\log_2(n+1)$
 - b. $\log_2(n+1) - 1$
 - c. **$1.44 \log_2 n$**
 - d. $1.66 \log_2 n$
96. Consider the following infix expression.
- $5 + 6/2$
- If one converts the above expression into postfix, what would be the resultant expression?
- a) $56/ + 2$
 - b) **$5 6 2 / +$** not confirmed
 - c) $5 6 / 2 +$
 - d) $/62 + 5$

97. Which of the following is a nonlinear data structure?

- a. Linked List
- b. Stack
- c. Queue
- d. **Tree**

98. In an AVL tree to delete a parent with two childs in a straight line following rotations will be required:-

- a. Single
- b. **Double**
- c. Triple
- d. None.

99. Each node in a BST has _____ Pointers:-

- a. 1
- b. **2**
- c. 3
- d. 4

100. When an operator is used in between two operands this is which type of notation

- a. Prefix
- b. Postfix
- c. **Infix**
- d. None of the Above



CS301- Data Structures
LATEST SOLVED SUBJECTIVES
FROM FINAL TERM PAPERS

Feb 06, 2012

Cs301 - subjective

Mc100401285

moaaz.pk@gmail.com

Final Term

FINAL TERM EXAMINATION 2010

Question No:1

name of two divide and conquer algorithm (2 marks)

Answer:- (Page 488)

- ❖ Merge Sort
- ❖ Quick Sort
- ❖ Heap Sort

These three algorithms fall under 'divide and conquer category'.

Question No:2

difference between call by value n call by reference (2 marks)

Answer:- (Page 202)

In case of call by value, a copy of object is made and placed at the time of function calling in the activation record. By using the references, we are not making the copy but send the address of the variable that function can use the original value.

Question No:3

heap and two types of heap(3)

Answer:- (Page 333)

Heap is a data structure of big use and benefit. It is used in priority queue. "The definition of heap is that it is a complete binary tree that conforms to the heap order". Here heap order means the min and max heap. Max heap: In max heap, each node has a value greater than the value of its left and right child nodes. Moreover, the value of the root will be largest and will become lesser at downward levels. Min heap: in a (min) heap for every node X, the key in the parent is smaller than (or equal to) the key in X. This means that the value in the parent node is less than the value on its children nodes.

Question No:4

height of a tree is 5 find sum of heights (2)

Answer:-

Height = 5

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari

Nodes = $2^h = 2^5 = 32$

Sum of Nodes = $N - (h+1) = 32 - (5+1) = 32 - 6 = 26$

Question No: 5

Here is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array. (Marks: 5)

Answer:- [Click here for detail](#)

If there is an array of 15 elements, {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}

it wants to know which elements can be discovered by examining two or fewer elements. I want assume the divide and conquer aspect of a binary search, so that leads me to think these elements will be found
1, 2, 14, 15, 7, 3

rather; 1, 2, 3, 4, 6, 8, 14, 15

1 and 15 for heads and tails

2, and 14 for next elements (following or preceding)

1/2 of 15 is either 6, or 8, from 7.5

and 1/2 of 6 = 3

1/2 8 = 4

Note: - I am not sure about answer; I feel Question logic is not clear.

FINAL TERM EXAMINATION 2010

Q1 Write one example of hashing?

Answer: (page 459)

Hashing can be used to store the data. Suppose we have a list of fruits. The names of Fruits are in string. The key is the name of the fruit. We will pass it to the hash Function to get the hash key.

HashCode ("apple") = 5

We pass it to the string "apple". Resultantly, it returns a number 5.

Q2 Height of a tree is 5 find sum of heights

Answer:- rep

Q3 How do we carry out degeneration of Complete Binary Tree?

Answer: [Click here for detail](#)

Degeneration of the (original) binary tree is mainly reflected by degeneration of the internal infix trees and less

by degeneration of the external height of the PPbin tree.

Q4 What is a skip List?

Answer:- (Page 446)

A skip list for a set S of distinct (key, element) items is a series of lists $S_0, S_1, \dots, S_h$ such that

- Each list S_i contains the special keys $+\infty$ and $-\infty$
- List S_0 contains the keys of S in non-decreasing order
- Each list is a subsequence of the previous one, i.e.,
$$S_0 \supseteq S_1 \supseteq \dots \supseteq S_h$$
- List S_h contains only the two special keys

Q5 How we can generate a maze with the help of union

Answer:- (Page 424)

How can we generate maze? The algorithm is as follows:

- ❖ Randomly remove walls until the entrance and exit cells are in the same set.
- ❖ Removal of the wall is the same as doing a union operation.
- ❖ Do not remove a randomly chosen wall if the cells it separates are already in the same set.

Q7 Following is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array. 5

Answer:- Repeat

FINAL TERM EXAMINATION 2010

(Marks: 10)

Write down the C++ code to implement insertion sort algorithm.

Answer:- (Page 483)

Following is the code of the insertion sort in C++.

```
void insertionSort(int *arr, int N)
{
    int pos, count, val;
    for(count=1; count < N; count++)
    {
        val = arr[count];
        for(pos=count-1; pos >= 0; pos--)
            if (arr[pos] > val)
                arr[pos+1]=arr[pos];
        else break;
        arr[pos+1] = val;
    }
}
```

```
}  
}
```

Q2. What is Table ADT? Discuss any two implementations of table ADT. (Marks: 5)

Answer:- (Page 427, 428)

The table, an abstract data type, is a collection of rows and columns of information.

Implementation of Table

o Unsorted Sequential Array

o Sorted Sequential Array

Question No: 3

Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right child nodes and Parent node of the node D (Marks: 5)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Answer:-

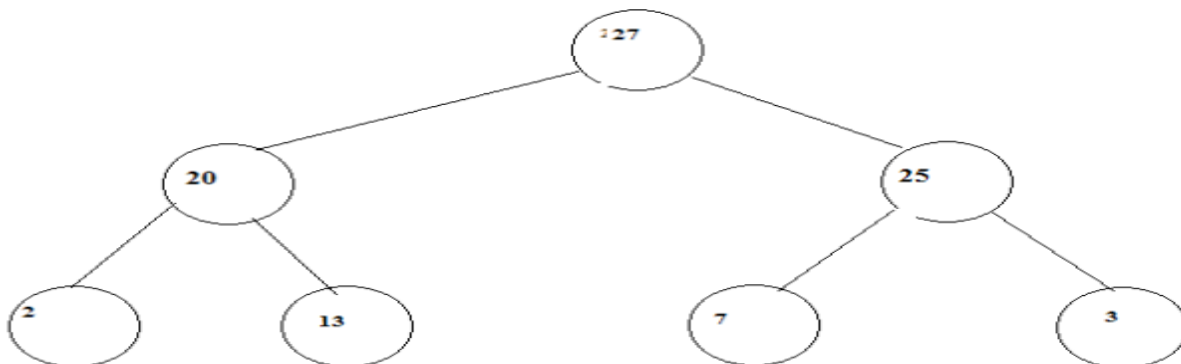
parent: B using formula($i/2$)

left child:-H $2i=2*4=8=H$

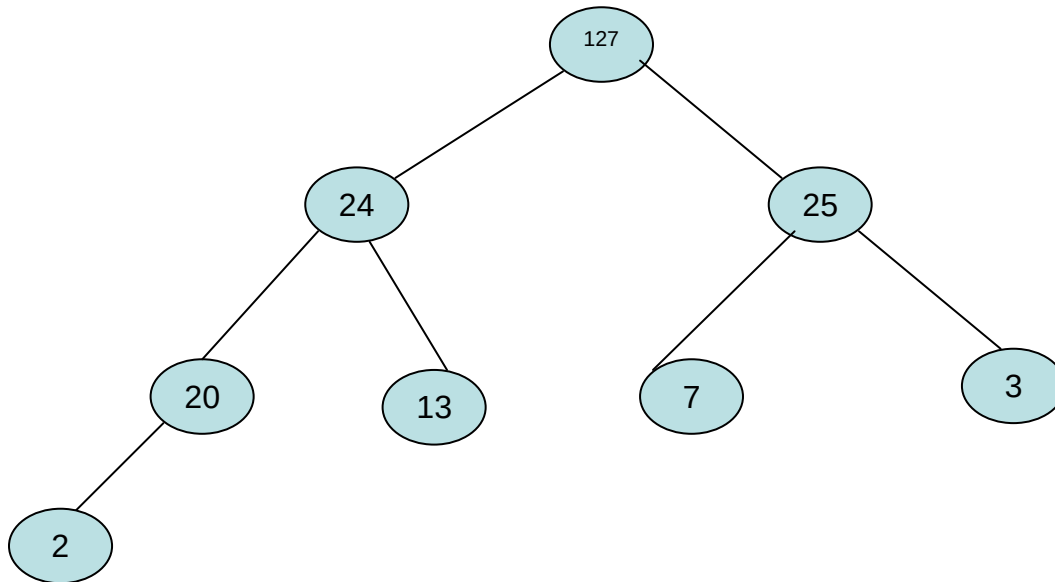
Right child:-I $2i+1=8+1=9=I$

Question No: 4

.Consider the following max heap. Add node 24 in it and show the resultant Heap. (Marks: 3)



Answer:-



Question No: 5 (Marks: 3)

Here is an array of 10 integers.

5 3 8 1 7 0 2 6 4.

Draw this array after the first iteration of the large loop in a selection sort (Sorting from smallest to largest)

Answer:-

	5	3	8	9	1	7	0	2	6	4
1 st step	0	3	8	9	1	7	5	2	6	4
2 nd step	0	1	8	9	3	7	5	2	6	4
3 rd step	0	1	2	9	3	7	5	8	6	4
4 th step	0	1	2	3	9	7	5	8	6	4
5 th step	0	1	2	3	4	7	5	8	6	9
6 th step	0	1	2	3	4	5	7	8	6	9
7 th step	0	1	2	3	4	5	6	8	7	9
8 th step	0	1	2	3	4	5	6	7	8	9

Question No: 6 (Marks: 3)

What is an Equivalent relation? Give any two examples.

Answer : (page 387)

‘A binary relation R over a set S is called an *equivalence relation* if it has following properties’:

1. Reflexivity: for all element $x \in S$, $x R x$
2. Symmetry: for all elements x and y , $x R y$ if and only if $y R x$
3. Transitivity: for all elements x , y and z , if $x R y$ and $y R z$ then $x R z$

Example; sets of peoples ,electric circuit domain

Q7. What are the properties of equivalence class? 3marks

Answer:- (Page 385)

A binary relation R over a set S is called an *equivalence relation* if it has following Properties

1. Reflexivity: for all element $x \in S$, $x R x$
2. Symmetry: for all elements x and y , $x R y$ if and only if $y R x$
3. Transitivity: for all elements x , y and z , if $x R y$ and $y R z$ then $x R z$

Q8. How heap sort works to set a set of data. 2Marks

Answer:- (Page 334)

Min heap and max heap are used to sort the data. Min heap convert the data in ascending sorted order and max heap convert the in descending order.

Q9. How can we search an element in skip list? 2Marks

Answer:- (Page 447)

We search for a key x in the following fashion:

We start at the first position of the top list

- At the current position p , we compare x with $y \sqcap \text{key}(\text{after}(p))$
- $x = y$: we return $\text{element}(\text{after}(p))$
- $x > y$: we “scan forward”
- $x < y$: we “drop down”
- If we try to drop down past the bottom list, we return `NO_SUCH_KEY`

Q10. What is an ADT?

Answer:- (Page 428)

The table, an abstract data type, is a collection of rows and columns of information.

FINAL TERM EXAMINATION 2010

Question No: 27 (Marks: 2)

Give one example of Hashing

Answer:- Repeated

Question No: 28 (Marks: 2)

How heap sort works to sort a set of data.

Answer:- Repeated

Question No: 29 (Marks: 2)

How we can implement Table ADT using Linked?

Answer:- (Page 441)

List linked list is one choice to implement table abstract data type. For unsorted elements, the insertion at *front* operation will take constant time. But if the data inside the list is kept in sorted order then to insert a new element in the list, the entire linked list is traversed through to find the appropriate position for the element.

Question No: 30 (Marks: 2)

If we allow assignment to constants what will happen?

Answer:- (Page 8)

Suppose 'a' has the value number 3. Now we assigned number 2 the number 3 i.e. all the number 2 will become number 3 and the result of $2 + 2$ will become 6. Therefore it is not allowed.

Question No: 31 (Marks: 3)

Explain the process of Deletion in a Min-Heap

Answer:- (Page 350)

We have to write a function *deleteMin()*, which will find and delete the minimum number from the tree. Finding the minimum is easy; it is at the top of the heap.

Question No: 32 (Marks: 3)

Give any three characteristics of Union by Weight method.

Answer:- (Page 408)

Following are the salient characteristics of this method:

- ❖ Maintain sizes (number of nodes) of all trees, and during *union*.
- ❖ Make smaller tree, the sub-tree of the larger one.
- ❖ Implementation: for each root node *i*, instead of setting *parent[i]* to -1, set it to -*k* if tree rooted at *i* has *k* nodes.

Question No: 33 (Marks: 3)

"For smaller lists, linear insertion sort performs well, but for larger lists, quick sort is suitable to apply." Justify why?

Answer:-

Since for smaller lists number of comparisons and interchange (if needed) decreases radically in insertion sort than quick sort. For larger lists, opposite occurs in both sorting techniques.

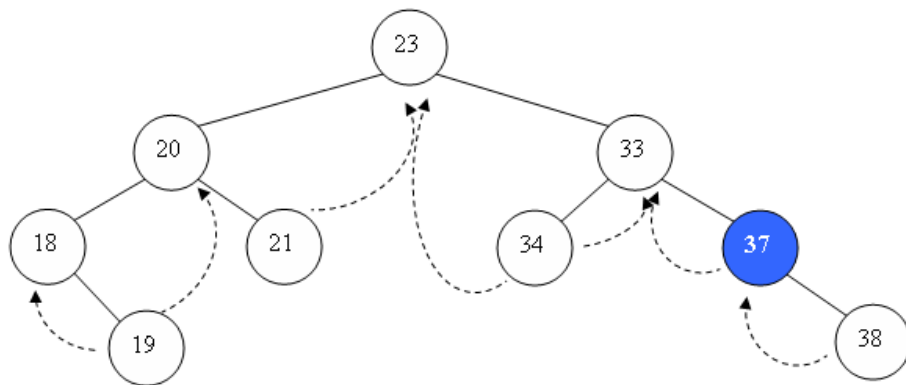
Question No: 34 (Marks: 5)

Write down the C++ code to implement Insertion Sort Algorithm.

Answer:- repeat

Question No: 35 (Marks: 5)

Consider the following Threaded Binary Tree,



You have to give the values that should be in the four variables given below, for the node 37

1. LTH (Left flag)
2. RTH (Right flag)
3. Left node pointer ($\rightarrow L$)
4. Right node pointer ($\rightarrow R$)

Question No: 36 (Marks: 5)

What is Disjoint Sets? Explain with an example.

Answer:- (Page 388)

Disjoint sets - Two sets are disjoint if they have no elements in common.

Example:- Suppose there are many people around you. How can we separate those who are related to each other in this gathering?

We make groups of people who are related to each other. The people in one group will say that they are related to each other. Similarly we can have many groups. So every group will say that they are related to each other with family relationship and there is no group that is related to any other group. The people in the group will say that there is not a single person in the other group who is related to them.

FINAL TERM EXAMINATION 2010

Question No: 27 (Marks: 2)

Give the difference between strict and complete binary tree.

Answer:- (Page 121,123)

A binary tree said to be a strictly binary tree if every non-leaf node in a binary has non-empty left and right sub trees. If there are left and right sub trees for each node in the binary tree as shown a complete binary tree.

Question No: 28 (Marks: 2)

A complete binary tree can be stored in an array. While storing the tree in an array we leave the first position (0th index) of the array empty. Why?

Answer:- (page 325)

In case of having a node with left and right children, stored at position i in the array, the left child will be at position $2i$ and the right child will be at $2i+1$ position. If the value of i is 2, the parent will be at position 2 and the left child will be at position $2i$ i.e. 4. The right child will be at position $2i+1$ i.e. 5. we have not started the 0th position. It is simply due to the fact if the position is 0, $2i$ will also become 0. So we will start from the 1st position, ignoring the 0th.

Question No: 29 (Marks: 2)

Give the name of two Divide and Conquer algorithms.

Answer:- Repeated

Question No: 30 (Marks: 2)

Give the effect of sorted data on Binary Search.

Answer:- <http://www.cprogramming.com/tutorial/lesson18.html>

The binary tree is a fundamental data structure used in computer science. The binary tree is a useful data structure for rapidly storing sorted data and rapidly retrieving stored data.

Question No: 31 (Marks: 3)

Give any three characteristics of Union by Weight method.

Answer:- Repeated

Question No: 32 (Marks: 3)

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

Draw this array after the FIRST iteration of the large loop in an insertion sort (sorting from smallest to largest). This iteration has shifted at least one item in the array!

Answer:-

	5	3	8	9	1	7	0	2	6	4
1 st step	3	5	8	9	1	7	0	2	6	4
2 nd step	1	3	5	8	9	7	0	2	6	4
3 rd step	1	3	5	7	8	9	0	2	6	4
4 th step	0	1	3	5	7	8	9	2	6	4
5 th step	0	1	2	3	5	7	8	9	6	4
6 th step	0	1	2	3	5	6	7	8	9	4
7 th step	0	1	2	3	4	5	6	7	8	9

Question No: 33 (Marks: 3)

Give your comment on the statement that heap uses least memory in array representation of binary trees. Justify your answer in either case.

Answer:-

A heap is a complete binary tree, so it is easy to be implemented using an array representation

It is a binary tree with the following properties:-

- ❖ Property 1: it is a complete binary tree
- ❖ Property 2: the value stored at a node is greater or equal to the values stored at the children
- ❖ From property 2 the largest value of the heap is always stored at the root

Question No: 34 (Marks: 5)

Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right child nodes and Parent node of the node D

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Answer: - Repeated

Question No: 35 (Marks: 5)

Explain the following terms:

1. Collision
2. Linear Probing
3. Quadratic Probing

Answer:- (Page 464-470)

Collision:

When two values hash to the same array location, this is called a *collision*..

Linear Probing

When there is a collision, some other location in the array is found. This is known as linear probing. In linear probing, at the time of collisions, we add one to the index and check that location. If it is also not empty, we add 2 and check that position. Suppose we keep on incrementing the array index and reach at the end of the table. We were unable to find the space and reached the last location of the array.

Quadratic Probing

In the quadratic probing when a collision happens we try to find the empty location at index + 1^2 . If it is filled then we add 2^2 and so on.

Quadratic probing uses different formula:

1. Use $F(i) = i^2$ (square of i) to resolve collisions
2. If hash function resolves to H and a search in cell H is inconclusive, try $H + 1^2$, $H + 2^2$, $H + 3^2$

Question No: 36 (Marks: 5)

Here is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

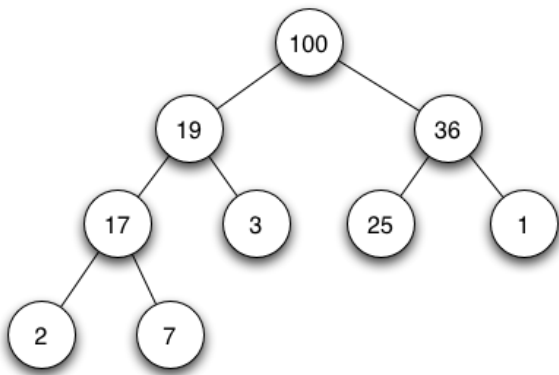
Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array.

Answer:- Repeat

FINALTERM EXAMINATION 2010

Question No: 27 (Marks: 2)

Convert this tree representation of a heap into the corresponding array representation



Answer:-

100	19	36	17	3	25	1	2	7

Question No: 28 (Marks: 2)

What are different applications of Hashing?

Answer (page 474)

- ❖ Compilers use hash tables to keep track of declared variables (symbol table).
- ❖ A hash table can be used for on-line spelling checkers — if misspelling detection (rather than correction) is important, an entire dictionary can be hashed and words checked in constant time.
- ❖ Game playing programs use hash tables to store seen positions, thereby saving computation time if the position is encountered again.

Question No: 29 (Marks: 2)

Give the operation names that we can perform on Table abstract data type.

Answer:- (page 429)

Insert
Find
Remove

Question No: 30 (Marks: 2)

Give your comments on the statement:

"Efficiently developed data structures decrease programming effort"

Answer:- [Click here for detail](#)

Reduces programming effort by providing useful data structures and algorithms so you don't have to write them yourself.

Increases performance by providing high-performance implementations of useful data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be easily tuned by switching implementations.

Question No: 31 (Marks: 3)

When Hashing is NOT suitable?

Answer: - (Page 475)

Hash tables are not so good if there are many insertions and deletions, or if table traversals are needed — in this case, AVL trees are better.

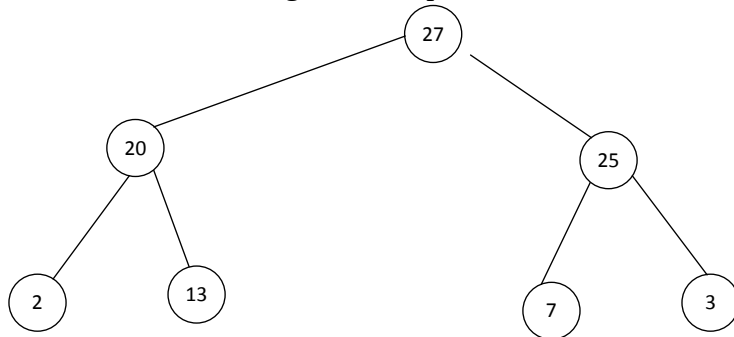
Question No: 32 (Marks: 3)

Give any three characteristics of Union by Weight method.

Answer: - Repeated

Question No: 33 (Marks: 3)

Consider the following Max Heap add node 24 in it and show the resultant Heap,



Answer: - Repeated

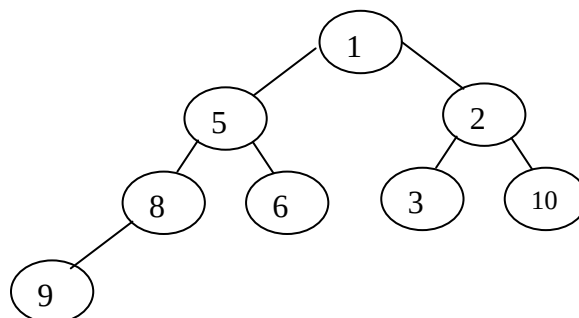
Question No: 34 (Marks: 5)

Heapify the elements of the following array (reading from left to right) into a Min Heap and show that Min Heap contents in the form of array as shown below,

original array	6	5	3	9	1	2	10	8	-

Answer(page 334)

Min heap



Heapified array	1	5	2	8	6	3	10	9	

Question No: 35 (Marks: 5)

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

Show the first three merging steps for Merge sort on this array.

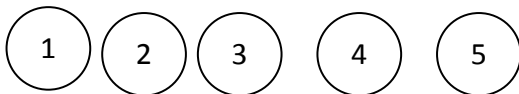
Answer:-

	5	3	8	9	1	7	0	2	6	4
1 st step	5	3	8	9	1	7	0	2	6	4
2 nd step	3	5	8	9	1	7	0	2	4	6
3 rd step	3	5	8	9	1	7	0	2	4	6
4 th step	1	3	8	5	9	2	6	7	0	4
5 th step	1	3	8	5	9	2	6	7	0	4
6 th step	1	2	3	6	7	8	0	5	4	9

Question No: 36 (Marks: 5)

Consider the following sequence of union commands on the set of elements

{1,2,3,4, 5}:



union(4,2)

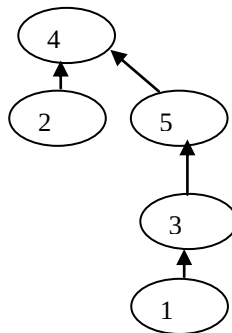
union(3,1)

union(5,4)

union(5,3)

Show the result when the unions are performed

Answer: (Page 396 to398)



FINAL TERM EXAMINATION 2010

Question No: 27 (Marks: 2)

Let's call the node as a that requires re-balancing. Consider two cases below:

- 1) An insertion into right subtree of the left child of a.
- 2) An insertion into left subtree of the right child of a.

Which of the following statement is correct about the above two cases.

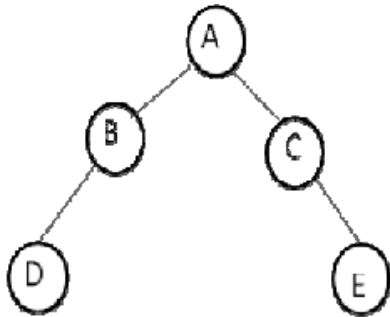
- 1) The insertion occurs outside (i.e., right to left or left to right) in the above two cases, single rotation can fix the balance in these two cases.
- 2) The insertion occurs inside ((i.e., right to left or left to right) in these two cases, single rotation cannot fix the balance in these two cases.

Answer: (page 230)

Insertion occurs on the *inside* in cases 2 and 3 which a single rotation cannot fix.

Question No: 28 (Marks: 2)

Represent the following Binary tree using array representation.



Answer:-

A	B	C	D	E

Question No: 29 (Marks: 2)

Give one example of Hashing

Answer:- repeat

Question No: 30 (Marks: 2)

How we can apply Find operation on elements combined through Union operation.

Answer:- (Page 392)

We give an element to the *find* method and it returns the name of the set.

Question No: 31 (Marks: 3)

Here is an array of ten integers:

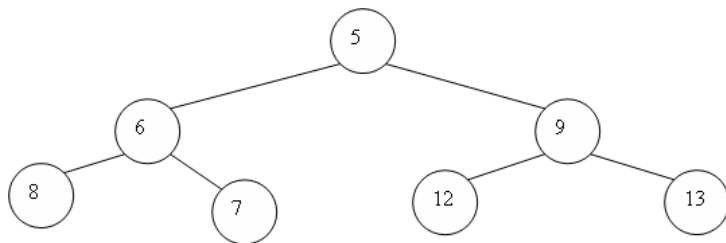
5 3 8 9 1 7 0 2 6 4

Draw this array after the FIRST iteration of the large loop in an insertion sort (sorting from smallest to largest). This iteration has shifted at least one item in the array!

Answer:- repeat

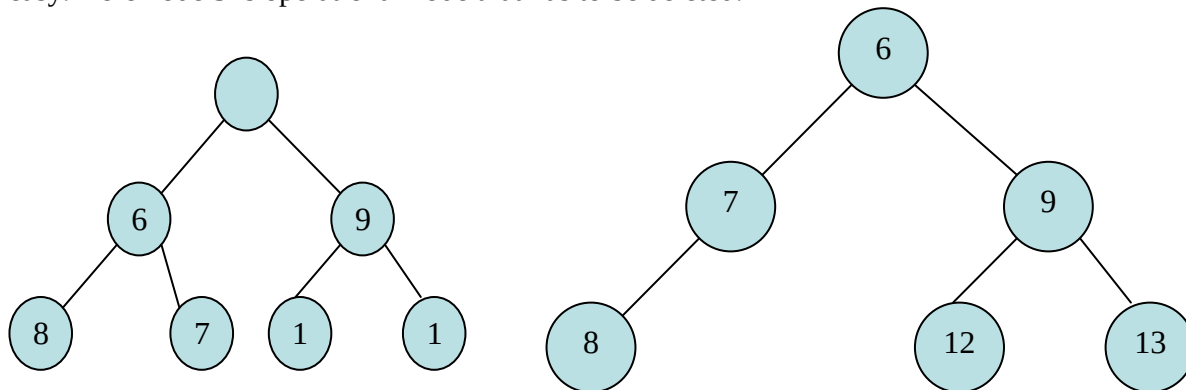
Question No: 32 (Marks: 3)

Consider the following Min Heap apply operation delMin on it and show the resultant Heap,



Answer:- (page 349)

It is clear from the definition of min-heap that it should be in the *root* of the tree. So finding the minimum is very easy. Here node 5 is operational node that has to be deleted.



Question No: 33 (Marks: 3)

Which implementation of disjoint set is better with respect space and why?

Answer:- (page 405)

An array that is a single dimensional structure and seems better with respect to space. Suppose we have 1000 set members i.e. names of people. If we make a Boolean matrix for 1000 items, its size will be 1000 x 1000. Thus we need 1000000 locations for Boolean items, its size will be 1000 x 1000. Thus we need 1000000 locations for Boolean values. in case of an array the number of location will be 1000. Thus this use of array i.e. tree like structure is better then two dimensional arrays in term of space to keep disjoint sets and doing union and find operation.

Question No: 34 (Marks: 5)

What is Table ADT? Discuss any two implementations of table ADT.

Answer (page 427,728)

The table, an abstract data type, is a collection of rows and columns of information.

Implementation of Table
 o Unsorted Sequential Array
 o Sorted Sequential Array

Question No: 35 (Marks: 5)

Suppose the following numbers are stored in an array 'BS':

32,51,27,85,66,23,13,57

We apply Bubble sort to array 'BS'. Show step by step sorting. You have to write only two passes that is passing 1 and pass 2.

Hint; while concern step by step sorting there are 6 passes involved to sort array nums. You have to show all passes

Answer:-

Pass one:-

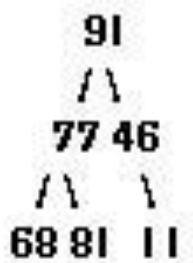
	32	51	27	85	66	23	13	57
Step1	32	51	27	85	66	23	13	57
Step2	32	27	51	85	66	23	13	57
Step3	32	27	51	66	85	23	13	57
Step4	32	27	51	66	23	85	13	57
Step5	32	27	51	66	23	13	85	57
Step6	32	27	51	66	23	13	57	85

Pass two:-

	32	27	51	66	23	13	57	85
Step1	32	27	51	66	23	13	57	85
Step2	27	32	51	66	23	13	57	85
Step3	32	27	66	23	66	13	57	85
Step4	32	27	66	23	13	66	57	85
Step5	32	27	66	23	13	57	66	85
Step6	32	27	66	23	13	57	66	85

Question No: 36 (Marks: 5)

Give two different reasons to explain why the following binary tree is not a heap:



Answer:- (Page 335)

Above array is not min heap or max heap because every parent node has not minimum or maximum value. In max heap, each parent node has a value greater than the value of its left and right child nodes. In min heap, each parent node has a value less than the value of its left and right child nodes.

FINAL TERM EXAMINATION

Fall 2009

Question No: 31 (Marks: 1)

Describe the conditions for second case of deletion in AVL Trees.

Answer:- (Page 266)

This is the case where the parent of the deleted node had a balance of 1 and the node was deleted in the parent's *left* sub tree.

Question No: 32 (Marks: 1)

What is Table abstract data type.

Answer:- repeat

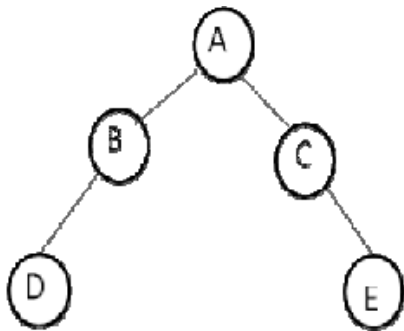
Question No: 33 (Marks: 2)

How we can generate a maze .Give an algorithm.

Answer:- repeat

Question No: 34 (Marks: 2)

Represent the following Binary tree using array representation.



Answer:- repeat

Question No: 35 (Marks: 3)

What is an Equivalent relation? Give any two examples.

Answer:- repeat

Question No: 36 (Marks: 3)

"For smaller lists, linear insertion sort performs well, but for larger lists, quick sort is suitable to apply." Justify why?

Answer:- repeat

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 37 (Marks: 3)

How many leaf and non-leaf nodes are present in a complete binary tree if its depth is 7?

Answer:- (Page 125)

Leaf Node = $2^d = 2^7 = 128$

Non leaf Nodes = $2^d - 1 = (2^7) - 1 = 128 - 1 = 127$

Question No: 38 (Marks: 5)

Remove the smallest element from the following array which represents a min-heap.

original min-heap	1	3	2	5	4	8	9	10	7

and show the resultant heap in the form of array as shown below,

Answer:- repeat

Question No: 39 (Marks: 5)

Here is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array.

Answer:- repeat

Question No: 40 (Marks: 10)

(a) Write C++ algorithm for Binary Search

Answer:- (Page 125)

```
int isPresent(int *arr, int val, int N)
```

```
{
int low = 0;
int high = N - 1;
int mid;
while ( low <= high )
{
mid = ( low + high )/2;
if (arr[mid] == val)
return 1; // found!
else if (arr[mid] < val)
low = mid + 1;
else
high = mid - 1;
}
return 0; // not found
```

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

b) Consider the following array of values; show how the binary search algorithm would find the value -5. Clearly show/explain your work; that is, show the values of start, end, etc. for each step of the algorithm.

0		-15	
1		-5	
2		0	
3		7	
4		13	
5		16	
6		27	
7		30	
8		42	

Answer : (page 437)

Value < mid

Value = -5

Mid=(0+8)/2=4

Low=0

High=mid -1=4-1

-15	-5	0	7	13	16	27	30	42
0	1	2	3	4	5	6	7	8
low			high	mid				

Starting from the low which is -15.now compare the -5 form the next index which is at position 1.



-15	-5	0	7	13	16	27	30	42
0	1	2	3	4	5	6	7	8
low			high	mid				



-15	-5	0	7	13	16	27	30	42
0	1	2	3	4	5	6	7	8
low			high	mid				

Question No: 41 (Marks: 10)

Show the result of following sequence of instructions

Union(1,2)
Union(3,4)
Union(3,5)
Union(1,7)
Union(3,6)
Union(8,9)
Union(1,8)
Union(3,10)
Union(3,11)
Union(3,12)
Union(3,13)
Union(14,15)
Union(16,17)
Union(14,16)
Union(1,3)
Union(1,14)

When the unions are performed by height

Note: You have to show only Final tree, No need to show all steps.

Answer:-

Solve it yourself; take help from previous same question that I have solved in page 12 of this file.

FINALTERM EXAMINATION

Fall 2009

Question No: 31 (Marks: 1)

If a Binary Tree has N internal nodes what are the no. of external nodes in it.

Answer:- (Page 304)

The No. of external nodes will be N+1

Question No: 32 (Marks: 1)

What is meant by Symmetry in equivalence relations?

Answer:- (Page 387)

Symmetry in equivalence relations mean for all elements x and y, $x R y$ if and only if $y R x$

Question No: 33 (Marks: 2)

How heap sort works to sort a set of data.

Answer:- Repeated

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 34 (Marks: 2)

How we can apply Find operation on elements combined through Union operation.

Answer:-Repeated

Question No: 35 (Marks: 3)

How we can use concept of equivalence relations to generate a Maze.

Answer:-Repeated

Question No: 36 (Marks: 3)

Suppose we are sorting an array of eight integers using a some quadratic sorting algorithm. After four iterations of the algorithm's main loop, the array elements are ordered as shown here:

2 4 5 7 8 1 3 6

Which statement is correct? (Note: Our selectionsort picks largest items first.)

- A. The algorithm might be either selection sort or insertion sort.
- B. The algorithm might be selection sort, but it is not insertion sort.
- C. The algorithm is not selection sort, but it might be insertion sort.
- D. The algorithm is neither selection sort nor insertion sort.
- E. None of these.

Answer:-

C. The algorithm is not selection sort, but it might be insertion sort. (Correct)

Question No: 37 (Marks: 3)

How many leaf and non-leaf nodes are present in a complete binary tree if its depth is 7 ?

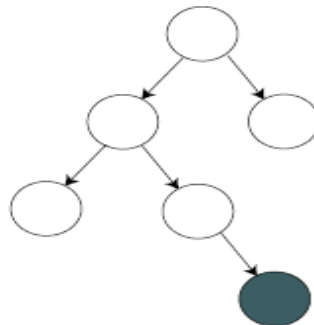
Answer:- repeat

Question No: 38 (Marks: 5)

If we insert a new element into an AVL tree of height 4, is one rotation sufficient to re-establish balance? Justify your answer.

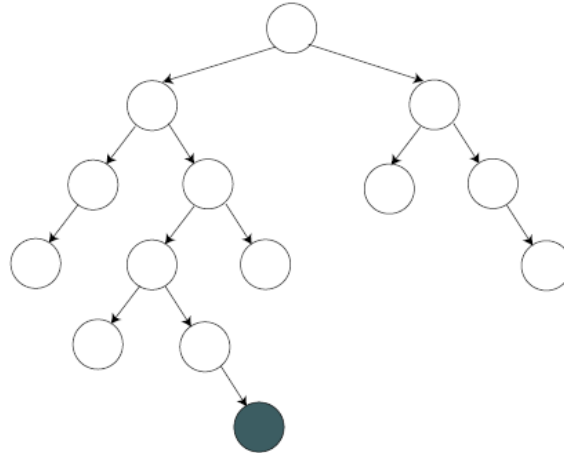
Answer:-

No, one rotation is not always sufficient to re-establish balance. For example, consider the insertion of the shaded node in the following AVL tree:



Though the original tree was balanced, more than one rotation is needed to restore balance following the insertion.

The problem asks for a tree of height 4, so we can extend the above example into a larger tree:



Write down the C++ code from Selection Sort Algorithm.

```
void selectionSort(int *arr, int N)
```

FINAL TERM EXAMINATION 2011 (Feb)

Answer ; (page 442)

Search and update require linear time
Fast Searching of Sorted Chain
Provide alternative to BST (binary search trees) and related tree structures.
Balancing can be expensive.
Relatively recent data structure: Bill Pugh proposed it in 1990

2- Drawback of using array to store BST.

Answer:- (Page 21)

The drawback of using is the limitations that array being of fixed size can only store a fixed number of elements. Therefore, no more elements can be stored after the size of the array is reached.

3-examples of hashing.

Answer:- repeat

4-c++ code to add node in doubly link list.

Answer ; (page 40)

```
class Node {
public:
int get() { return object; }; // returns the value of the element
void set(int object) { this->object = object; }; // set the value of the element
Node* getNext() { return nextNode; }; // get the address of the next node
void setNext(Node* nextNode) // set the address of the next node
{ this->nextNode = nextNode; };
Node* getPrev() { return prevNode; }; // get the address of the prev node
void setPrev(Node* prevNode) // set the address of the prev node
{ this->prevNode = prevNode; };
private:
int object; // it stores the actual value of the element
Node* nextNode; // this points to the next node
Node* prevNode; // this points to the previous node
};
```

5-What you conclude from the running time analysis of disjoint sets.

Answer (page 405)

We conclude that:

- ❖ union is clearly a constant time operation
- ❖ Running time of find(i) is proportional to the height of the tree containing node i.
- ❖ This can be proportional to n in the worst case (but not always).
- ❖ Goal: Modify union to ensure that heights stay small

6-explain how heap sort work.

Answer:- repeat

7-what is c++ template.

Answer:- (page 75)

A template is a function or class that is written with a generic data type. when a programmer uses this function or class the generic data type is replaced with the data type needed to be used in the template function or in the template class. We only give the data type of our choice while calling a template function or creating an object of the template class. The compiler automatically creates a version of that function or class with that specified data type.

8-explain and write the code for union and find operation of parent array in disjoint sets.

Answer:- (page 399)

Find (i)

This loop is used to find the parent of an element or the name of the set that contains that element.

```
// traverse to the root (-1)
for(j=i; parent[j] >= 0; j=parent[j])
    ;
return j;
```

Union (i, j)

Now let's see the code for the function of union. Here we pass two elements to the function union. The union finds the roots of i and j. If i and j are disjoint sets, it will merge them. Following is the code of this function.

```
root_i = find(i);
root_j = find(j);
if (root_i != root_j)
    parent[root_j] = root_i;
```

In the code, we see that at first it finds the root of tree in which i exists by the find(i) method and similarly finds the root of the set containing j by find(j). Then there is a check in if statement to see whether these sets (roots) are same or not. If these are not the same, it merges them in such a fashion that the root i is set as the parent of root j. Thus, in the array where the value of root j exists, the value of root i becomes there.

FINAL TERM EXAMINATION 2011 (Feb)

Question No: 41 (Marks: 2)

Convert the given infix form to postfix form.

A+B/C-D^E-F

Answer:- (Page 66)

AB+C/D-EF -^

Question No: 42 (Marks: 2)

How we can implement Table ADT using Linked List

Answer:- repeat

Question No: 43 (Marks: 2)

If we allow assignment to constants what will happen?

Answer:- repeat

Question No: 44 (Marks: 2)

How heap sort works to sort a set of data.

Answer:- repeat

Question No: 46 (Marks: 3)

Give your comment on the statement that heap uses least memory in array representation of binary trees. Justify your answer in either case.

Answer: - repeat

Question No: 47 (Marks: 3)

How we can use concept of equivalence relations to generate a Maze.

Answer: - repeat

Question No: 48 (Marks: 3)

"For smaller lists, linear insertion sort performs well, but for larger lists, quick sort is suitable to apply." Justify why?

Answer: - repeat

Question No: 49 (Marks: 5)

6 9 5 0 4 1 7 3 2

Show the first five merging steps for Merge sort on this array.

Answer:- repeat (Also see the same solution above)

Question No: 50 (Marks: 5)

What is Disjoint Sets? Explain with an example.

Answer:- (Page 388)

Disjoint sets - Two sets are disjoint if they have no elements in common.

Example:-Suppose there are many people around you. How can we separate those who are related to each other in this gathering?

We make groups of people who are related to each other. The people in one group will say that they are related to each other. Similarly we can have many groups. So every group will say that they are related to each other with family relationship and there is no group that is related to any other group. The people in the group will say that there is not a single person in the other group who is related to them.

Muhammad Moaaz Siddiq – MCS (2nd)

mc100401285@Gmail.com

**Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

Question No: 51 (Marks: 5)

Write the code of the percolateDown() function and also comment it.

Answer:- (page 369)

```
template <class eType>
void Heap<eType>::percolateDown( int hole )
{
    int child;
    eType tmp = array[ hole ];
    for( ; hole * 2 <= currentSize; hole = child )
    {
        child = hole * 2;
        if( child != currentSize && array[child+1] < array[ child ] )
            child++; // right child is smaller
        if( array[ child ] < tmp )
            array[ hole ] = array[ child ];
        else break;
    }
    array[ hole ] = tmp;
}
```

Question No: 52 (Marks: 5)

Here is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array.

Answer:- repeat

FINALTERM EXAMINATION 2011

Question No:1

How we can implement Table ADT using Linked List (marks 2)

Answer: - repeat

Question No:2

which three basic operations are performed on AVL trees (mark 3)

Answer:- (page 457)

- ❖ Insert
- ❖ Delete
- ❖ search

Question No:3

write down the c++ code to implement insertion sort algorithm (5)

Answer: - repeat

Question No:4

Give the name of three Divide and Conquer algorithms. (marks 3)

Answer: - repeat

Question No:5

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

Draw this array after the FIRST iteration of the large loop in an insertion sort (sorting from smallest to largest). This iteration has shifted at least one item in the array! (marks 3)

Answer: - repeat

FINAL TERM EXAMINATION 2011

Question No: 1

For smaller list linear insertion sort is perform well but for large list Quick sort suitable to apply/justify why?????????

Answer:-

Since for smaller lists number of comparisons and interchange (if needed) decreases radically in insertion sort than quick sort. For larger lists, opposite occurs in both sorting techniques.

Question No: 2

Here is a array of exactly 15 elements

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Suppose that we are doing a binary search for an elemnt .Indicate any element that will be found by examining two or fewer number from the array (5 mrks)

Answer:- repeat

Question No: 3

Write the applications of heap.

Answer:- (page 84)

The stack and heap are used in function calls. When we allocate memory dynamically in our programs, it is allocated from the heap. The use of priority queue with the help of heap is a major application. The priority queue is itself a major data structure, much-used in operating systems. Similarly priority queue data structure is used in network devices especially in routers. Heap data structure is also employed in sorting algorithms.

Question No: 4

Write down the c++ code for selection sort (5 marks)

Answer: - repeat

Question No: 5

Write the application of binary search tree

Answer: - (page 369)

The searching and sorting operations are very common in computer science with the help of binary search tree.

Question No: 6

Write the about disjoint set?

Answer: - repeat

Question No: 7

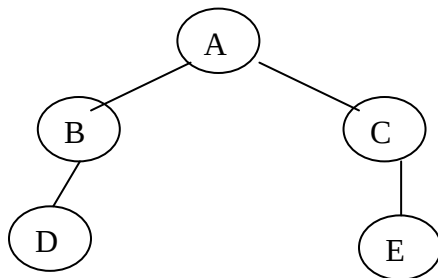
Heapify the elements of the following array (reading from left to right) into a Min Heap and show that Min Heap contents in the form of array as shown below,

original array	6	5	3	9	1	2	10	8	-

Answer: - repeat

Question No: 8

Represent the following binary tree in array



Answer: - repeat

FINAL TERM EXAMINATION 2011

Question No: 1

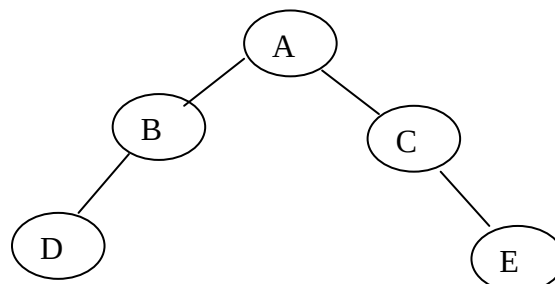
Write the any two operation of linked list that perform their function in one step. 2Marks

Answer: - (page 38)

- ❖ Add
- ❖ Remove
- ❖ Find

Question No: 2

Represent the following binary tree in array. 2Marks



Answer: - repeat

Question No: 3**Write different type of hashing. 2 Marks****Answer: -** [Click here for detail](#)

Data structures that employ hashing methods to store keys (and information possibly associated to it) are denoted as hashing types.

Question No: 4**Give the table name that that we can on table abstract data type. 3mark****Question No: 5****Explain the process of Deletion in a Min-Heap 3 Marks****Answer: - repeat****Question No: 6****What is skip list? 3 Marks****Answer: - repeat****Question No: 7****What we can conclude from the running time array of disjoint set. 5 Marks****Answer: - repeat****Question No: 8****77, 33,44,11,88,22,66,55 sort the following array using insertion algorithm. 5 Marks****Answer: - See the same solution above****Question No: 9****What is disjoint set? 5 Marks****Answer: - repeat****Question No: 10****Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right child nodes and Parent node of the node D (Marks: 5)**

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Answer: - repeat**FINAL TERM EXAMINATION 2011****1. Give one example of Hashing. (2)****Answer: - repeat****Muhammad Moaaz Siddiq – MCS (2nd)****mc100401285@Gmail.com****Campus:- Institute of E-Learning & Modern
Studies (IEMS) Samundari**

2. Give the operation names that we can perform on Table abstract data type.

(2)

Answer: - repeat

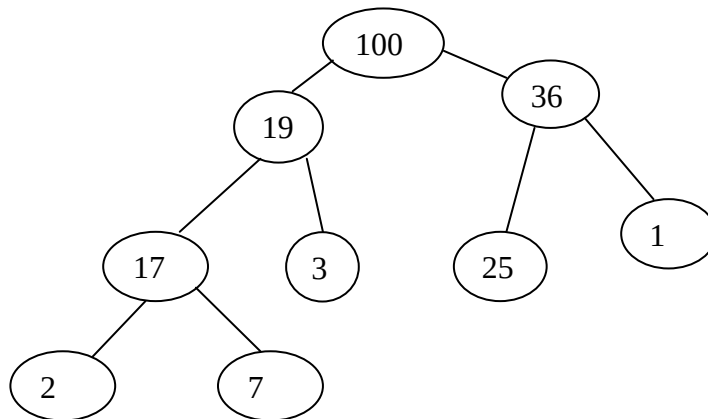
3. Define Linked List in your own words.

(2)

Answer: - repeat

4. Convert this tree representation of a heap into the corresponding array representation.

(2)



Answer: - repeat

5. Here is an array of ten integers:

5 3 8 9 1 7 2 6 4

Draw this array after the FIRST iteration of the large loop in an insertion sort (Sorting from smallest to largest). This iteration has shifted at least one item in the array.

(3)

Answer:- repeat

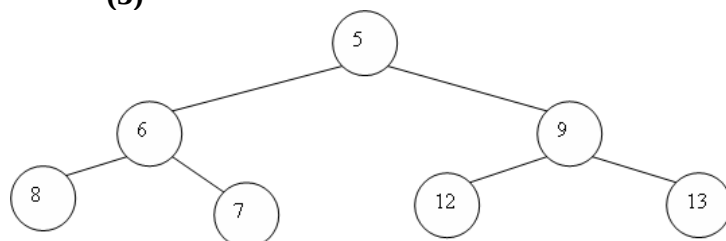
6. Which implementation of disjoint set is better with respect space and why?

(3)

Answer: - repeat

7. Consider the following Min Heap apply operation delMin on it and show resultant Heap.

(3)



Answer: - repeat

8. Which is Dynamic Equivalence Problem with an example? (5)

Answer: - (page 395)

We are using sets to store the elements. For this purpose, it is essential to remember which element belongs to which set. We know that the elements in a set are unique and a tree is used to represent a set. Each element in a tree has the same root, so the root can be used to name the set. The item (number) in the root will be unique as the set has unique values of items. We use this root as the name of set for our convenience. Otherwise, we can use any name of our choice. So the element in the root (the number) is used as the name of the set. The *find* operation will return this name. Due to the presence of many sets, there will be a collection of trees. In this collection, each tree will be representing one set. If there are ten elements, we will have ten sets initially. Thus there will be ten trees at the beginning. In general, we have N elements and initially there will be N trees. Each tree will have one element. Thus there will be N trees of one node. Now here comes a definition for this collection of trees, which states that a collection of trees is called a *forest*.

The trees used for the sets are not necessarily binary trees. That means it is necessary that each node should have a maximum of two children nodes. Here a n may have more than two children nodes.

To execute the union operation in two sets, we merge the two trees of these sets in such a manner that the root of one tree points to the root of other. So there will be one root, resulting in the merger of the trees.

In the find operation, when we call *find (x)*, it helps us to know which set this x belongs to. Internally, we find this x in a tree in the *forest*. When this x is found in a tree the *find* returns the number at root node (the name of the set) of that tree.

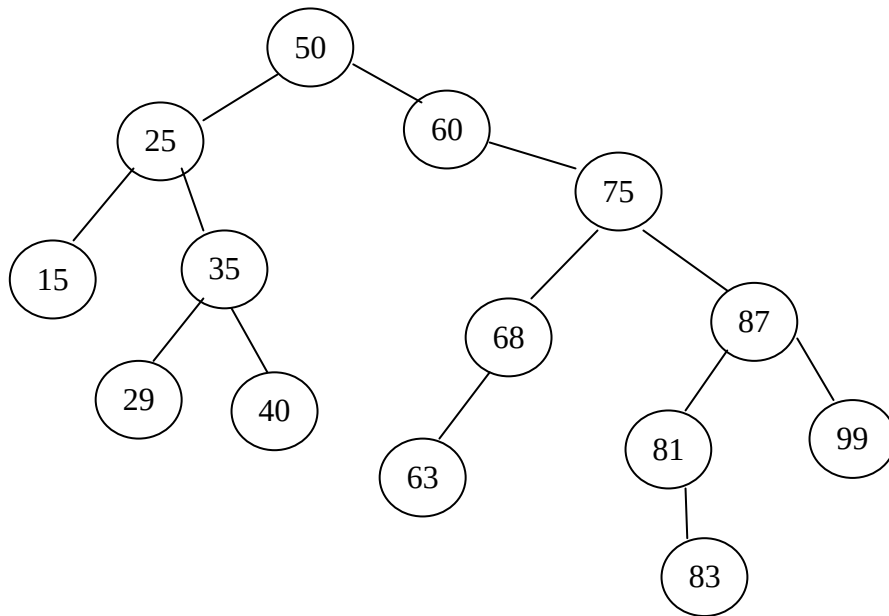
9. Here is an array of ten integers

5 3 8 9 1 7 0 2 6 4

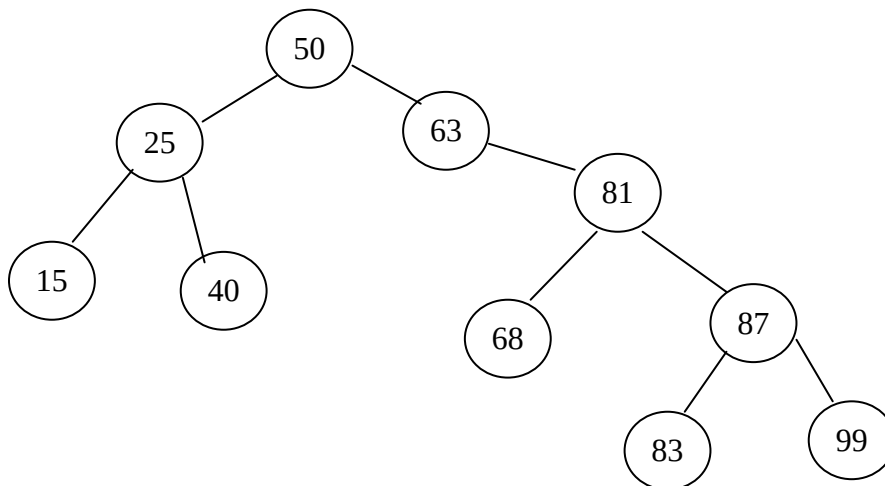
Sort the array by using selection sort algorithm and show content of array after each step. (5)

Answer: - repeat

10. Consider the following Binary tree and Delete the key 29, 35, 75 and 60 only. Show the resultant tree after deleting these keys. (5)



Answer:-



11. Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right child nodes and Parent node of the Node **D**

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Answer: - repeat

FINAL TERM EXAMINATION 2011

1) For a perfect binary tree of height 5, what will be the sum of heights of nodes? 2marks

Answer: - repeat

2) Properties of equivalence class.

Answer: - repeat

3) Two ways for the implementation of Queue data structure.

Answer: - (Page 89, 91)

1. Implementation of queue using linked list
2. Implementation of queue using Array.

4) Worst case of selection sort

Answer: - [Click here for detail](#)

The worst case occurs if the array is already sorted in descending order. Nonetheless, the time required by selection sort algorithm is not very sensitive to the original order of the array to be sorted.

6) “is a sibling of” set of all human beings is equivalence relation or not? Explain.

Answer: (Page 387)

Yes it is an equivalence relation

First rule is reflexive i.e. for all element $x \in S$, $x R x$. Suppose that x is Haris so $Haris R Haris$. This is true because everyone is related to each other. Second is Symmetry: for all elements x and y , $x R y$ if and only if $y R x$. Suppose that y is Saad. According to the rule, $Haris R Saad$ if and only if $Saad R Haris$. If two persons are related, the relationship is symmetric i.e. if I am cousin of someone so is he. Therefore if Haris is brother of Saad, then Saad is certainly the brother of Haris. The family relationship is symmetric. This is not the symmetric in terms of respect but in terms of relationship. The transitivity is: ‘for all elements x , y and z . If $x R y$ and $y R z$, then $x R z$ ’. Suppose x is Haris, y is Saad and z is Ahmed. If Haris “is related to” Saad, Saad “is related to” Ahmed. We can deduce that Haris “is related to” Ahmed. This is also true in relationships. If you are cousin of someone, the cousin of that person is also related to you. He may not be your first cousin but is related to you.

7) Linked memory

(Answer: Page 18)

Linked memory is a memory in which the various cells of memory are not located continuously. In this process, each cell of the memory not only contains the value of the element but also the information where the next element of the list is residing in the memory. It is not necessary that the next element is at the next location in the memory. It may be anywhere in the memory. We have to keep a track of it.

Thus, in this way, the first element must explicitly have the information about the location of the second element.

Similarly, the second element must know where the third element is located and the third should know the position of the fourth element and so on

8) 3 5 9 8 1 7 0 2 6 4 , draw after the 1st iteration of large loop in a selection sort (smallest to largest)

Answer: - repeat

9) Union commands 1, 2, 3, 4, 5 union(4,2),union(3,1),union(5,4) 5 marks

Answer: - repeat

10) Code for removing an element from a linked list data structure if it is implementation by using linked memory methods.

Answer: Page 34

```
void remove() {  
if( currentNode != NULL && currentNode != headNode) {  
lastCurrentNode->setNext(currentNode->getNext());  
delete currentNode;  
currentNode = lastCurrentNode->getNext();  
size--;  
}  
};
```



CS301-Data Structure
(Solved Subjective)
LECTURE FROM
(23 to 45)



Junaidfazal08@gmail.com
Bc190202640@vu.edu.pk

FOR MORE VISIT
VULMSHELP.COM

JUNAID MALIK
0304-1659294

AL-JUNAID TECH INSTITUTE



www.vulmshelp.com



Language Courses Training Available

I'm providing paid courses in different languages within 3 Months, Certificate will be awarded after completion.

- HTML
- CSS
- JAVASCRIPT
- BOOTSTRAPS
- IQUERY
- PHP MYSQL
- NODES.JS
- REACT JS

LMS Handling Services

LMS Activities Paid Task

Assignments 95% Results

Quizes 95% Results

GDB 95% Results

For CS619 Project Feel Free To Contact With Me

Ph# 0304-1659294
Email: junaidfazal08@gmail.com

Question No:1

name of two divide and conquer algorithm (2 marks)

Answer:- (Page 488)

- ❖ Merge Sort
- ❖ Quick Sort
- ❖ Heap Sort

These three algorithms fall under 'divide and conquer' category.

Question No:2

difference between call by value and call by reference (2 marks)

Answer:- (Page 202)

In case of call by value, a copy of object is made and placed at the time of function calling in the activation record. By using the references, we are not making the copy but send the address of the variable that function can use the original value.

Question No:3

heap and two types of heap(3)

Answer:- (Page 333)

Heap is a data structure of big use and benefit. It is used in priority queue. —The definition of heap is that it is a complete binary tree that conforms to the heap order. Here heap order means the min and max heap. Max heap: In max heap, each node has a value greater than the value of its left and right child nodes. Moreover, the value of the root will be largest and will become lesser at downward levels. Min heap: in a (min) heap for every node X, the key in the parent is smaller than (or equal to) the key in X. This means that the value in the parent node is less than the value on its children nodes.

Question No:4

height of a tree is 5 find sum of heights (2)

Answer:-

Height = 5

Nodes = $2^h = 2^5 = 32$

Sum of Nodes = $N - (h + 1) = 32 - (5 + 1) = 32 - 6 = 26$

Question No: 5

Here is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by

examining two or fewer numbers from the array. (Marks: 5)

Answer:-

If there is an array of 15 elements, {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15} it wants to know which elements can be discovered by examining two or fewer elements. I want assume the divide and conquer aspect of a binary search, so that leads me to think these elements will be found 1, 2, 14, 15, 7, 3

rather; 1, 2, 3, 4, 6, 8, 14, 15

1 and 15 for heads and tails

2, and 14 for next elements (following or preceding)

$1/2$ of 15 is either 6, or 8, from 7.5

and $1/2$ of 6 = 3

$1/28=4$

Question No.6

Write one example of hashing?

Answer: (page 459)

Hashing can be used to store the data. Suppose we have a list of fruits. The names of Fruits are in string. The key is the name of the fruit. We will pass it to the hash Function to get the hash key.

HashCode ("apple") = 5

We pass it to the string —apple. Resultantly, it returns a number 5.

Question No.7

Height of a tree is 5 find sum of heights

Answer:- rep

Question No.8

How do we carry out degeneration of Complete Binary Tree?

Answer:

Degeneration of the (original) binary tree is mainly reflected by degeneration of the internal infix trees and less by degeneration of the external height of the PPbin tree.

Question No.9

What is a skip List?

Answer:- (Page 446)

A skip list for a set S of distinct (key, element) items is a series of lists $S_0, S_1, \dots, S_h$ such that

Each list S_i contains the special keys $+\infty$ and $-\infty$

- List S_0 contains the keys of S in non-decreasing order
- Each list is a subsequence of the previous one, i.e.,
 $S_0 \supseteq S_1 \supseteq \dots \supseteq S_h$
- List S_h contains only the two special keys

Question No.10

How we can generate a maze with the help of union

Answer:- (Page 424)

How can we generate maze? The algorithm is as follows:

- ❖ Randomly remove walls until the entrance and exit cells are in the same set.
- ❖ Removal of the wall is the same as doing a union operation.
- ❖ Do not remove a randomly chosen wall if the cells it separates are already in the same set.

Question No.11

Following is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array. 5

Answer:- Repeat

Question No.12 (Marks: 10)

Write down the C++ code to implement insertion sort algorithm.

Answer:- (Page 483)

Following is the code of the insertion sort in C++.

```
void insertionSort(int *arr, int N)
{
    int pos, count, val;
    for(count=1; count < N; count++)
    {
        val = arr[count];
        for(pos=count-1; pos >= 0; pos--)
            if (arr[pos] > val)
                arr[pos+1]=arr[pos];
            else break;
        arr[pos+1] = val;
    }
}
```

Question No.13

What is Table ADT? Discuss any two implementations of table ADT. (Marks: 5)

Answer:- (Page 427, 428)

The table, an abstract data type, is a collection of rows and columns of information.

Implementation of Table

- o Unsorted Sequential Array
- o Sorted Sequential Array

Question No: 14

Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right child nodes and Parent node of the node D (Marks: 5)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Answer:-

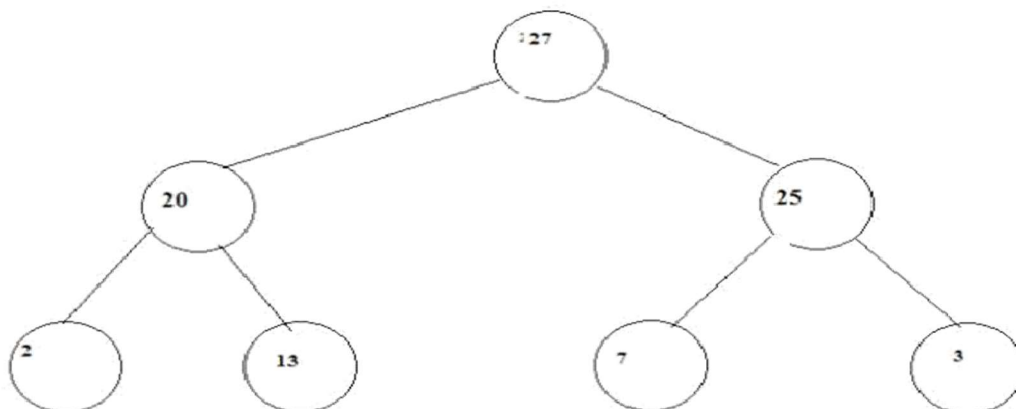
parent: B using formula($i/2$)

left child:-H $2i=2*4=8=H$

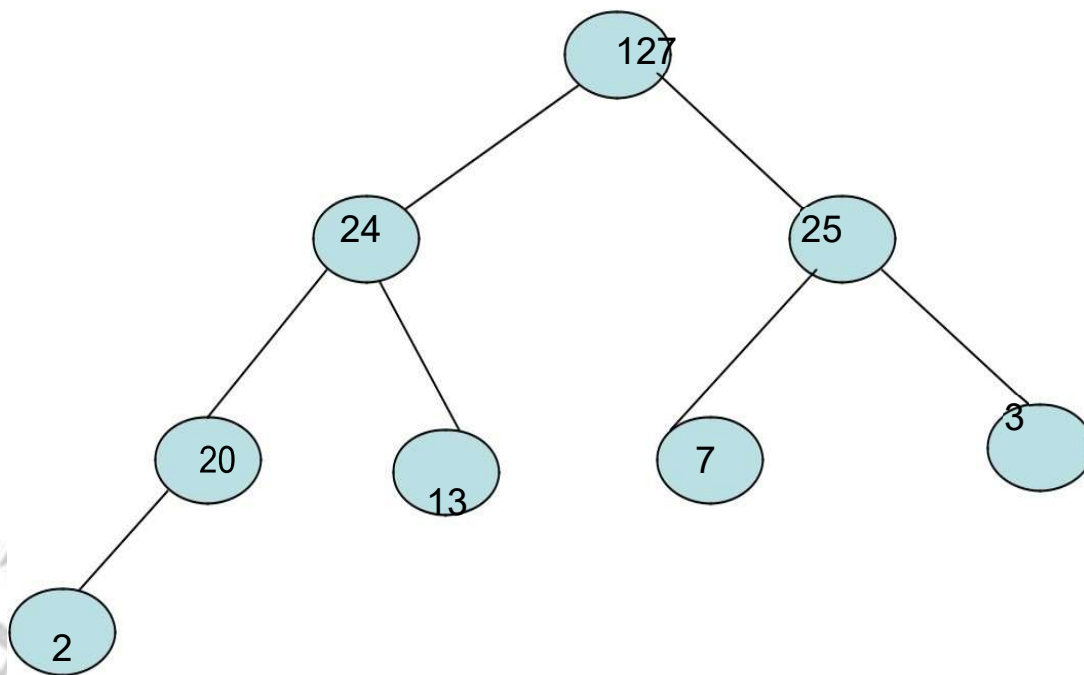
Right child:-I $2i+1=8+1=9=I$

Question No: 15

Consider the following max heap. Add node 24 in it and show the resultant Heap. (Marks: 3)



Answer:-



Question No: 16 (Marks: 3)

Here is an array of 10 integers.

5 3 8 1 7 0 2 6 4.

Draw this array after the first iteration of the large loop in a selection sort (Sorting from smallest to largest)

Answer:-

	5	3	8	9	1	7	0	2	6	4
1 st step	0	3	8	9	1	7	5	2	6	4
2 nd step	0	1	8	9	3	7	5	2	6	4
3 rd step	0	1	2	9	3	7	5	8	6	4
4 th step	0	1	2	3	9	7	5	8	6	4
5 th step	0	1	2	3	4	7	5	8	6	9
6 th step	0	1	2	3	4	5	7	8	6	9
7 th step	0	1	2	3	4	5	6	8	7	9
8 th step	0	1	2	3	4	5	6	7	8	9

Question No: 17 (Marks: 3)

What is an Equivalent relation? Give any two examples.

Answer : (page 387)

A binary relation R over a set S is called an *equivalence relation* if it has following properties:

1. Reflexivity: for all element $x \in S$, $x R x$
 2. Symmetry: for all elements x and y , $x R y$ if and only if $y R x$
 3. Transitivity: for all elements x , y and z , if $x R y$ and $y R z$ then $x R z$
- Example; sets of peoples ,electric circuit domain

Question No.18

What are the properties of equivalence class? 3marks

Answer:- (Page 385)

A binary relation R over a set S is called an *equivalence relation* if it has following Properties

1. Reflexivity: for all element $x \in S$, $x R x$
2. Symmetry: for all elements x and y , $x R y$ if and only if $y R x$
3. Transitivity: for all elements x , y and z , if $x R y$ and $y R z$ then $x R z$

Question No.19

How heap sort works to set a set of data.

Answer:- (Page 334)

Min heap and max heap are used to sort the data. Min heap convert the data in ascending sorted order and max heap convert the in descending order.

Question No.20

How can we search an element in skip list? 2Marks

Answer:- (Page 447)

We search for a key x in the following fashion:

We start at the first position of the top list

- At the current position p , we compare x with $y = \text{key}(\text{after}(p))$
- $x = y$: we return $\text{element}(\text{after}(p))$
- $x > y$: we —scan forward
- $x < y$: we —drop down
- If we try to drop down past the bottom list, we return NO_SUCH_KEY

Question No.21

What is an ADT?

Answer:- (Page 428)

The table, an abstract data type, is a collection of rows and columns of information.

Question No: 22 (Marks: 2)

Give one example of Hashing

Answer:- Repeated

Question No: 23 (Marks: 2)

How heap sort works to sort a set of data.

Answer:- Repeated

Question No: 24 (Marks: 2)

How we can implement Table ADT using Linked?

Answer:- (Page 441)

List linked list is one choice to implement table abstract data type. For unsorted elements, the insertion at *front* operation will take constant time. But if the data inside the list is kept in sorted order then to insert a new element in the list, the entire linked list is traversed through to find the appropriate position for the element.

Question No: 25 (Marks: 2)

If we allow assignment to constants what will happen?

Answer:- (Page 8)

Suppose `_a` has the value number 3. Now we assigned number 2 the number 3 i.e. all the number 2 will become number 3 and the result of $2 + 2$ will become 6. Therefore it is not allowed.

Question No: 26 (Marks: 3)

Explain the process of Deletion in a Min-Heap

Answer:- (Page 350)

We have to write a function `deleteMin()`, which will find and delete the minimum number from the tree.

Finding the minimum is easy; it is at the top of the heap.

Question No: 27 (Marks: 3)

Give any three characteristics of Union by Weight method.

Answer:- (Page 408)

Following are the salient characteristics of this method:

- ❖ Maintain sizes (number of nodes) of all trees, and during *union*.
- ❖ Make smaller tree, the sub-tree of the larger one.
- ❖ Implementation: for each root node i , instead of setting $\text{parent}[i]$ to -1 , set it to $-k$ if tree rooted at i has k nodes.

Question No: 28 (Marks: 3)

"For smaller lists, linear insertion sort performs well, but for larger lists, quick sort is suitable to apply." Justify why?

Answer:-

Since for smaller lists number of comparisons and interchange (if needed) decreases radically in insertion sort than quick sort. For larger lists, opposite occurs in both sorting techniques.

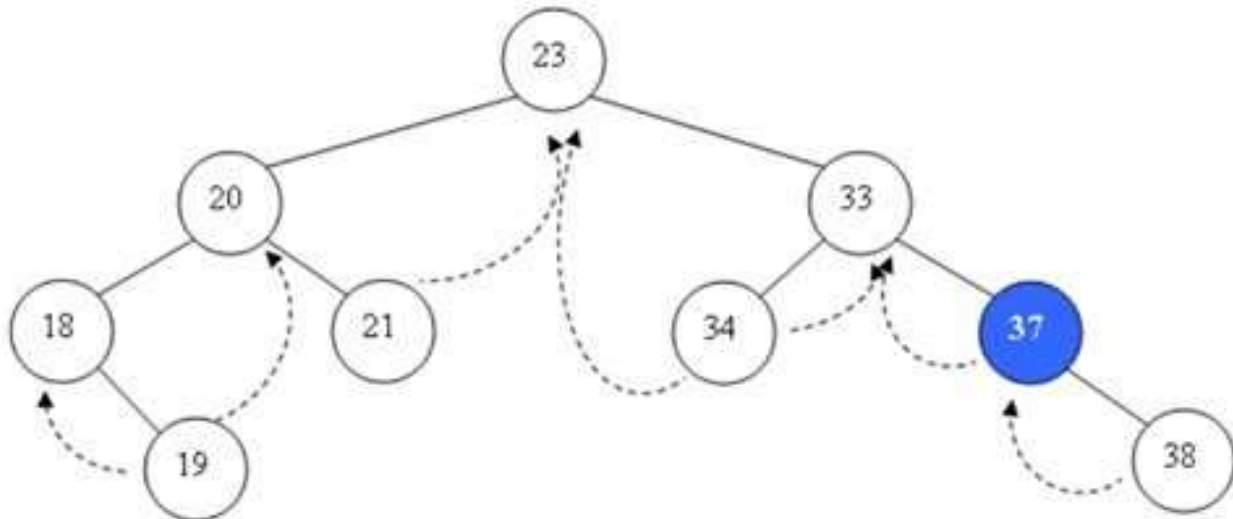
Question No: 29 (Marks: 5)

Write down the C++ code to implement Insertion Sort Algorithm.

Answer:- repeat

Question No: 30 (Marks: 5)

Consider the following Threaded Binary Tree,



You have to give the values that should be in the four variables given below, for the node 37

1. LTH (Left flag)
2. RTH (Right flag)
3. Left node pointer (->L)
4. Right node pointer (->R)

Question No: 31 (Marks: 5)

What is Disjoint Sets? Explain with an example.

Answer:- (Page 388)

Disjoint sets - Two sets are disjoint if they have no elements in common.

Example:- Suppose there are many people around you. How can we separate those who are related to each other in this gathering?

We make groups of people who are related to each other. The people in one group will say that they are related to each other. Similarly we can have many groups. So every group will say that they are related to each other with family relationship and there is no group that is related to any other group. The people in the group will say that there is not a single person in the other group who is related to them.

Question No: 32 (Marks: 2)

Give the difference between strict and complete binary tree.

Answer:- (Page 121,123)

A binary tree said to be a strictly binary tree if every non-leaf node in a binary has non-empty left and right sub trees. If there are left and right sub trees for each node in the binary tree as shown a complete binary tree.

Question No: 33 (Marks: 2)

A complete binary tree can be stored in an array. While storing the tree in an array we leave the first position (0th index) of the array empty. Why?

Answer:- (page 325)

In case of having a node with left and right children, stored at position i in the array, the left child will be at position $2i$ and the right child will be at $2i+1$ position. If the value of i is 2, the parent will be at position 2 and the left child will be at position $2i$ i.e. 4 .The right child will be at position $2i+1$ i.e. 5. we have not started the 0th position. It is simply due to the fact if the position is 0, $2i$ will also become 0. So we will start from the 1st position, ignoring the 0th.

Question No: 34 (Marks: 2)

Give the name of two Divide and Conquer algorithms.

Answer:- Repeated

Question No: 35 (Marks: 2)

Give the effect of sorted data on Binary Search.

Answer:-

The binary tree is a fundamental data structure used in computer science. The binary tree is a useful data structure for rapidly storing sorted data and rapidly retrieving stored data.

Question No: 36 (Marks: 3)

Give any three characteristics of Union by Weight method.

Answer:- Repeated

Question No: 37 (Marks: 3)

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

Draw this array after the FIRST iteration of the large loop in an insertion sort (sorting from smallest to largest). This iteration has shifted at least one item in the array!

Answer:-

	5	3	8	9	1	7	0	2	6	4
1 st step	3	5	8	9	1	7	0	2	6	4
2 nd step	1	3	5	8	9	7	0	2	6	4
3 rd step	1	3	5	7	8	9	0	2	6	4

4 th step	0	1	3	5	7	8	9	2	6	4
5 th step	0	1	2	3	5	7	8	9	6	4
6 th step	0	1	2	3	5	6	7	8	9	4
7 th step	0	1	2	3	4	5	6	7	8	9

Question No: 38 (Marks: 3)

Give your comment on the statement that heap uses least memory in array representation of binary trees. Justify your answer in either case.

Answer:-

A heap is a complete binary tree, so it is easy to be implemented using an array representation. It is a binary tree with the following properties:-

- ❖ Property 1: it is a complete binary tree
- ❖ Property 2: the value stored at a node is greater or equal to the values stored at the children
- ❖ From property 2 the largest value of the heap is always stored at the root

Question No: 39 (Marks: 5)

Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right child nodes and Parent node of the node D

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Answer: - Repeated

Question No: 40 (Marks: 5)

Explain the following terms:

1. Collision
2. Linear Probing
3. Quadratic Probing

Answer:- (Page 464-470)

Collision:

When two values hash to the same array location, this is called a *collision*..

Linear Probing

When there is a collision, some other location in the array is found. This is known as linear probing. In linear probing, at the time of collisions, we add one to the index and check that

location. If it is also not empty, we add 2 and check that position. Suppose we keep on incrementing the array index and reach at the end of the table. We were unable to find the space and reached the last location of the array.

Quadratic Probing

In the quadratic probing when a collision happens we try to find the empty location at index $+ 1^2$. If it is filled then we add 2^2 and so on.

Quadratic probing uses different formula:

1. Use $F(i) = i^2$ (square of i) to resolve collisions
2. If hash function resolves to H and a search in cell H is inconclusive, try $H+1^2, H+2^2, H+3^2$

Question No: 41

(Marks: 5)

Here is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

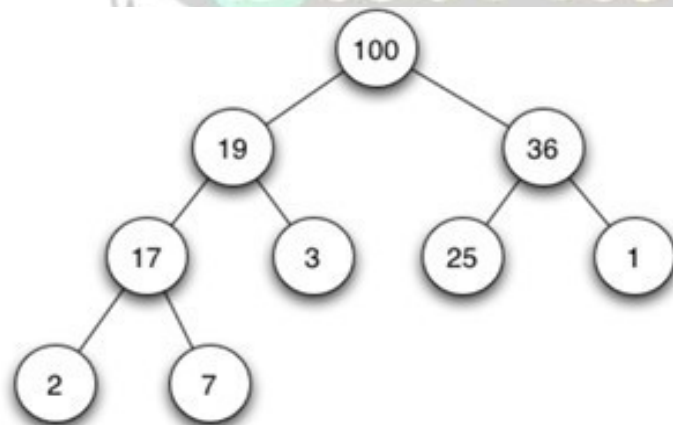
Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array.

Answer:- Repeat

Question No: 42

(Marks: 2)

Convert this tree representation of a heap into the corresponding array representation



Answer:-

100	19	36	17	3	25	1	2	7

Question No: 43 (Marks: 2)

What are different applications of Hashing?

Answer (page 474)

- ❖ Compilers use hash tables to keep track of declared variables (symbol table).
- ❖ A hash table can be used for on-line spelling checkers — if misspelling detection (rather than correction) is important, an entire dictionary can be hashed and words checked in constant time.
- ❖ Game playing programs use hash tables to store seen positions, thereby saving computation time if the position is encountered again.

Question No: 44 (Marks: 2)

Give the operation names that we can perform on Table abstract data type.

Answer:- (page 429)

Insert

Find

Remove

Question No: 45 (Marks: 2)

Give your comments on the statement:

"Efficiently developed data structures decrease programming effort"

Answer:-

Reduces programming effort by providing useful data structures and algorithms so you don't have to write them yourself.

Increases performance by providing high-performance implementations of useful data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be easily tuned by switching implementations.

Question No: 46 (Marks: 3)

When Hashing is NOT suitable?

Answer: - (Page 475)

Hash tables are not so good if there are many insertions and deletions, or if table traversals are needed — in this case, AVL trees are better.

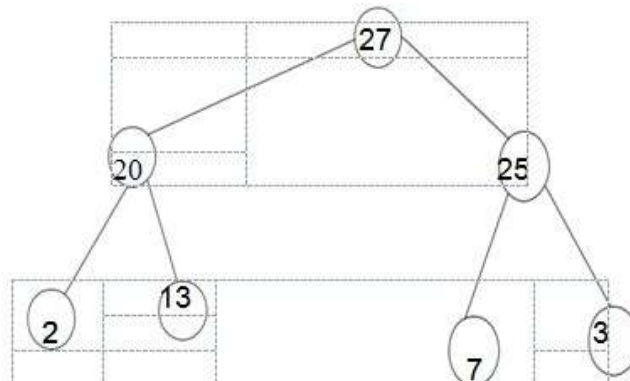
Question No: 47 (Marks: 3)

Give any three characteristics of Union by Weight method.

Answer: - Repeated

Question No: 48 (Marks: 3)

Consider the following Max Heap add node 24 in it and show the resultant Heap,



Answer: - Repeated

Question No: 49

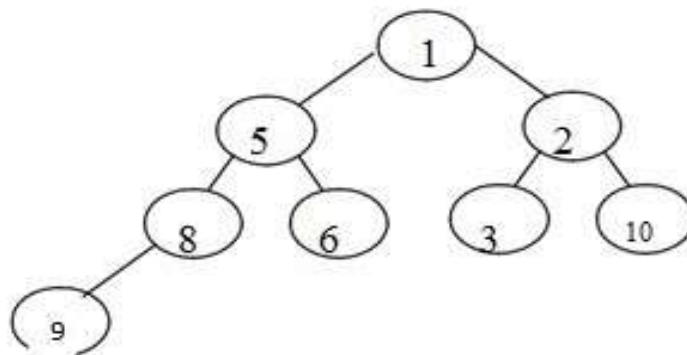
(Marks: 5)

Heapify the elements of the following array (reading from left to right) into a Min Heap and show that Min Heap contents in the form of array as shown below,

Original array	6	5	3	9	1	2	10	8

Answer:

Min heap



Question no. 50

Here is an array of ten integers

5389170264

Show the first three merging steps for Merge sort on this array.

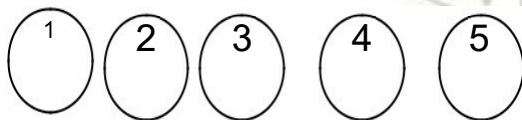
Answer:-

	5	3	8	9	1	7	0	2	6	4
1 st step	5	3	8	9	1	7	0	2	6	4
2 nd step	3	5	8	9	1	7	0	2	4	6
3 rd step	3	5	8	9	1	7	0	2	4	6
4 th step	1	3	8	5	9	2	6	7	0	4
5 th step	1	3	8	5	9	2	6	7	0	4
6 th step	1	2	3	6	7	8	0	5	4	9

Question No: 51 (Marks:5)

Consider the following sequence of union commands on the set of elements

{1,2,3,4, 5}:



union(4,2)

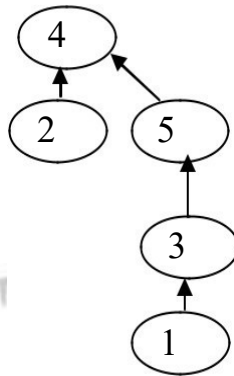
union(3,1)

union(5,4)

union(5,3)

Show the result when the unions are performed

Answer: (Page 396 to398)



Question No: 52 (Marks: 2)

Let's call the node as a that requires re-balancing. Consider two cases below:

- 1) An insertion into right subtree of the left child of a.
- 2) An insertion into left subtree of the right child of a.

Which of the following statement is correct about the above two cases.

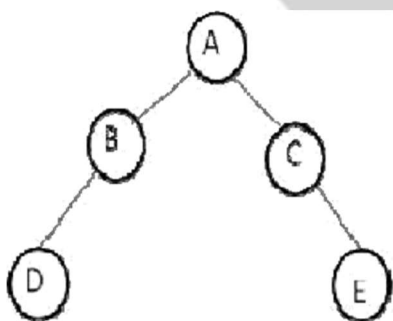
- 1) The insertion occurs outside (i.e., right to left or left to right) in the above two cases, single rotation can fix the balance in these two cases.
- 2) The insertion occurs inside ((i.e., right to left or left to right) in these two cases, single rotation cannot fix the balance in these two cases.

Answer: (page 230)

Insertion occurs on the *inside* in cases 2 and 3 which a single rotation cannot fix.

Question No: 53 (Marks: 2)

Represent the following Binary tree using array representation.



Answer:

A	B	C	D	E

Answer:-

Question No: 54

(Marks: 2)

Give one example of Hashing

Answer:- repeat

Question No: 55

(Marks: 2)

How we can apply Find operation on elements combined through Union operation.

Answer:- (Page 392)

We give an element to the *find* method and it returns the name of the set.

Question No: 56

(Marks: 3)

Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

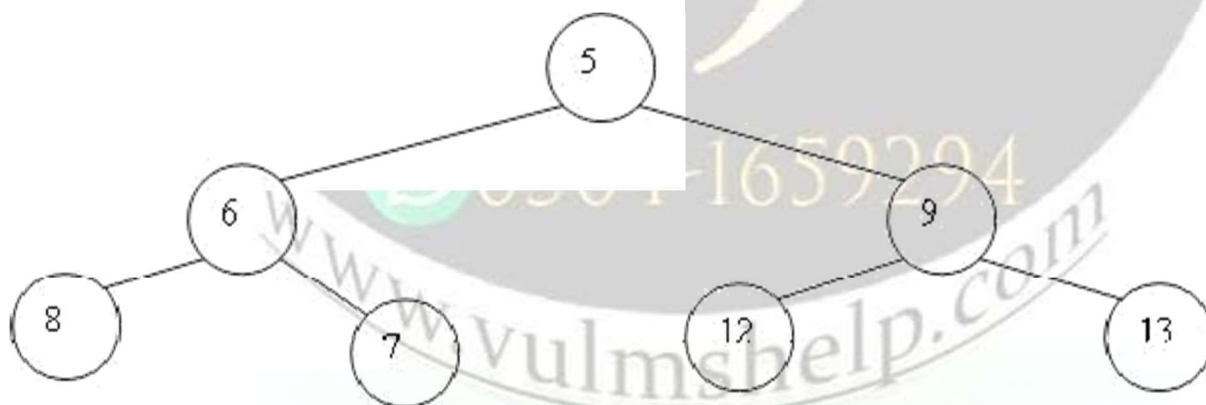
Draw this array after the FIRST iteration of the large loop in an insertion sort (sorting from smallest to largest). This iteration has shifted at least one item in the array!

Answer:- repeat

Question No: 57

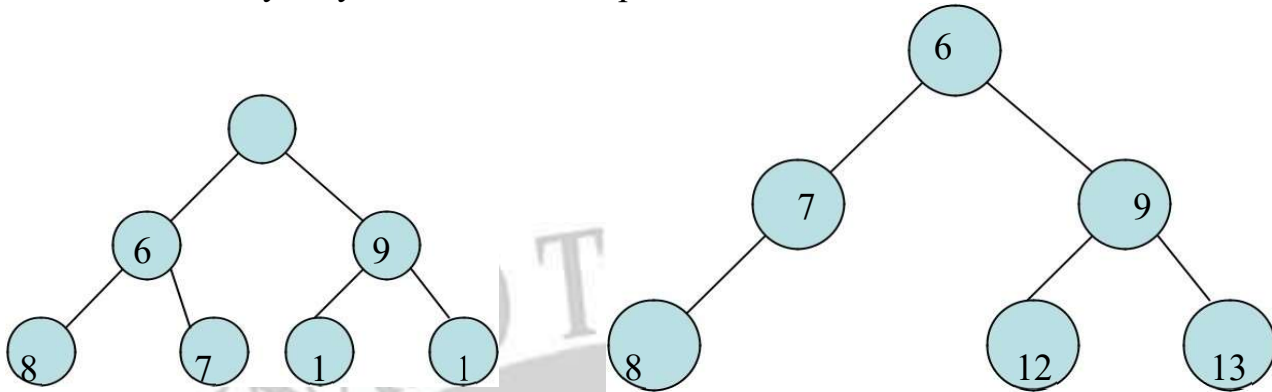
(Marks: 3)

Consider the following Min Heap apply operation delMin on it and show the resultant Heap,



Answer:- (page 349)

It is clear from the definition of min-heap that it should be in the *root* of the tree. So finding the minimum is very easy. Here node 5 is operational node that has to be deleted.



Question No: 58 (Marks: 3)

Which implementation of disjoint set is better with respect space and why?

Answer:- (page 405)

An array that is a single dimensional structure and seems better with respect to space. Suppose we have 1000 set members i.e. names of people. If we make a Boolean matrix for 1000 items, its size will be 1000 x 1000. Thus we need 1000000 locations for Boolean items, its size will be 1000 x 1000. Thus we need 1000000 locations for Boolean values. in case of an array the number of location will be 1000. Thus this use of array i.e. tree like structure is better then two dimensional arrays in term of space to keep disjoint sets and doing union and find operation.

Question No: 59 (Marks: 5)

What is Table ADT? Discuss any two implementations of table ADT.

Answer (page 427,728)

The table, an abstract data type, is a collection of rows and columns of information.

Implementation of Table

- o Unsorted Sequential Array
- o Sorted Sequential Array

Question No: 60 (Marks: 5)

Suppose the following numbers are stored in an array „BS“:

32,51,27,85,66,23,13,57

We apply Bubble sort to array „BS“. Show step by step sorting. You have to write only two passes that is passing 1 and pass 2.

Hint; while concern step by step sorting there are 6 passes involved to sort array nums. You have to show all passes

Answer:-

Pass one:-

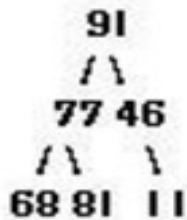
	32	51	27	85	66	23	13	57
Step1	32	51	27	85	66	23	13	57
Step2	32	27	51	85	66	23	13	57
Step3	32	27	51	66	85	23	13	57
Step4	32	27	51	66	23	85	13	57
Step5	32	27	51	66	23	13	85	57
Step6	32	27	51	66	23	13	57	85

Pass two:-

	32	27	51	66	23	13	57	85
Step1	32	27	51	66	23	13	57	85
Step2	27	32	51	66	23	13	57	85
Step3	32	27	66	23	66	13	57	85
Step4	32	27	66	23	13	66	57	85
Step5	32	27	66	23	13	57	66	85
Step6	32	27	66	23	13	57	66	85

Question No: 61 (Marks: 5)

Give two different reasons to explain why the following binary tree is not a heap:



Answer:- (Page 335)

Above array is not min heap or max heap because every parent node has not minimum or maximum value.

In max heap, each parent node has a value greater than the value of its left and right child nodes.

In min heap, each parent node has a value less than the value of its left and right child nodes.

Question No: 62 (Marks: 1)

Describe the conditions for second case of deletion in AVL Trees.

Answer:- (Page 266)

This is the case where the parent of the deleted node had a balance of 1 and the node was deleted in the parent's *left* sub tree.

Question No: 63

(Marks: 1)

What is Table abstract data type.

Answer:- repeat

Question No: 64

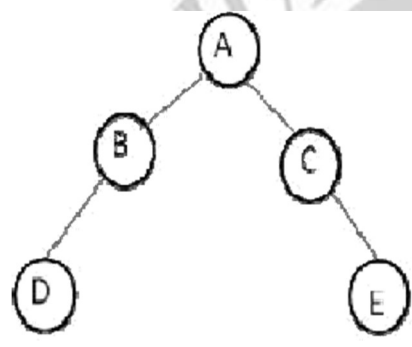
(Marks: 2)

How we can generate a maze .Give an algorithm.

Answer:- repeat

Question No: 65 (Marks: 2)

Represent the following Binary tree using array representation.



Answer:- repeat

Question No: 66 (Marks: 3)

What is an Equivalent relation? Give any two examples.

Answer:- repeat

Question No: 67 (Marks: 3)

"For smaller lists, linear insertion sort performs well, but for larger lists, quick sort is suitable to apply." Justify why?

Answer:- repeat

Question No: 68 (Marks: 3)

How many leaf and non-leaf nodes are present in a complete binary tree if its depth is 7?

Answer:- (Page 125)

Leaf Node = $2^d = 2^7 = 128$

Non leaf Nodes = $2^d - 1 = (2^7) - 1 = 128 - 1 = 127$

Question No: 69 (Marks: 5)

Remove the smallest element from the following array which represents a min-heap.

Original min heap	1	3	2	5	4	8	9	10	7

and show the resultant heap in the form of array as shown below, **Answer:- repeat**

Question No: 70 (Marks: 5)

Here is an array with exactly 15 elements:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array.

Answer: repeat

Question No: 71 (Marks: 10)

(a) Write C++ algorithm for Binary Search

Answer:- (Page 125)

```
int isPresent(int *arr, int val, int N)
{
    int low = 0;
    int high = N - 1;
    int mid;
    while ( low <= high )
    {
        mid = ( low + high )/2;
        if (arr[mid] == val)
            return 1; // found!
        else if (arr[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return 0; // not found
```

b) Consider the following array of values; show how the binary search algorithm would find the value -5. Clearly show/explain your work; that is, show the values of start, end, etc. for each step of the algorithm.

1		-5	
2		0	
3		7	
4		13	
5		16	
6		27	
7		30	
8		42	

Answer : (page 437)

Value < mid

Value = -5

Mid=(0+8)/2=4

Low=0

High=mid -1=4-1

-15	-5	0	7	13	16	27	30	42
0	1	2	3	4	5	6	7	8
low			high	mid				

Starting from the low which is -15.now compare the -5 form the next index which is at position 1.

-15	-5	0	7	13	16	27	30	42
0	1	2	3	4	5	6	7	8
low			high	mid				

-15	-5	0	7	13	16	27	30	42
0	1	2	3	4	5	6	7	8
low			high	mid				

Question No: 72 (Marks: 10)

Show the result of following sequence of instructions

Union(1,2)
Union(3,4)
Union(3,5)
Union(1,7)
Union(3,6)
Union(8,9)
Union(1,8)
Union(3,10)
Union(3,11)
Union(3,12)
Union(3,13)
Union(14,15)
Union(16,17)
Union(14,16)
Union(1,3)
Union(1,14)

When the unions are performed by height

Note: You have to show only Final tree, No need to show all steps.

Answer:-

Solve it yourself; take help from previous same question that I have solved in page 12 of this file.

Question No: 73 (Marks: 1)

If a Binary Tree has N internal nodes what are the no. of external nodes in it.

Answer:- (Page 304)

The No. of external nodes will be $N+1$

Question No: 74 (Marks: 1)

What is meant by Symmetry in equivalence relations?

Answer:- (Page 387)

Symmetry in equivalence relations mean for all elements x and y , $x R y$ if and only if $y R x$

Question No: 75 (Marks: 2)

How heap sort works to sort a set of data.

Answer:- Repeated

Question No: 76 (Marks: 2)

How we can apply Find operation on elements combined through Union operation.

Answer:-Repeated

Question No: 77 (Marks: 3)

How we can use concept of equivalence relations to generate a Maze.

Answer:-Repeated

Question No: 78 (Marks: 3)

Suppose we are sorting an array of eight integers using a some quadratic sorting algorithm. After four iterations of the algorithm's main loop, the array elements are ordered as shown here:

2 4 5 7 8 1 3 6

Which statement is correct? (Note: Our selectionsort picks largest items first.)

- A. The algorithm might be either selection sort or insertion sort.
- B. The algorithm might be selection sort, but it is not insertion sort.
- C. The algorithm is not selection sort, but it might be insertion sort.
- D. The algorithm is neither selection sort nor insertion sort.
- E. None of these.

Answer:-

C. The algorithm is not selection sort, but it might be insertion sort. (Correct)

Question No: 79 (Marks: 3)

How many leaf and non-leaf nodes are present in a complete binary tree if its depth is 7 ?

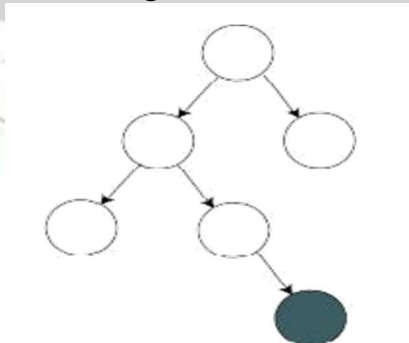
Answer:- repeat

Question No: 80 (Marks: 5)

If we insert a new element into an AVL tree of height 4, is one rotation sufficient to re-establish balance? Justify your answer.

Answer:-

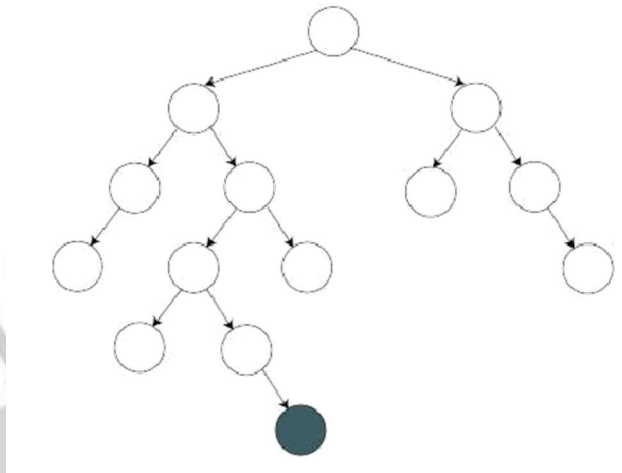
No, one rotation is not always sufficient to re-establish balance. For example, consider the insertion of the shaded node in the following AVL tree:



Though the original tree was balanced, more than one rotation is needed to restore balance following the insertion.

This can be seen by an exhaustive enumeration of the rotation possibilities.

The problem asks for a tree of height 4, so we can extend the above example into a larger tree:



Question No: 81 (Marks: 5)

Write down the C++ code from Selection Sort Algorithm.

Answer:- (Page 480)

```
void selectionSort(int *arr, int N)
{
    int posmin, count, tmp ;
    for (count=0;count<N;count++)
    {
        posmin = findIndexMin(arr, count, N) ;
        tmp=arr[posmin] ;
        arr[posmin]=arr[count] ;
        arr[count]=tmp ;
    }
}

int findIndexMin (int *arr, int start, int N)
{
    int posmin=start ;
    int index ;
    for(index=start; index < N; index++)
    if (arr[index]<arr[posmin])
        posmin=index ;
    return posmin ;
}
```

Question No.82

3 important characteristics of skip list.

Answer ; (page 442)

Search and update require linear time
Fast Searching of Sorted Chain
Provide alternative to BST (binary search trees) and related tree structures.
Balancing can be expensive.
Relatively recent data structure: Bill Pugh proposed it in 1990

Question No.83

Drawback of using array to store BST.

Answer:- (Page 21)

The drawback of using is the limitations that array being of fixed size can only store a fixed number of elements.

Therefore, no more elements can be stored after the size of the array is reached.

Question No.84

examples of hashing.

Answer:- repeat

Question No.85

c++ code to add node in doubly link list.

Answer ; (page 40)

```
class Node {  
public:  
int get() { return object; }; // returns the value of the element  
void set(int object) { this->object = object; }; // set the value of  
the element Node* getNext() { return nextNode; }; // get the  
address of the next node void setNext(Node* nextNode) // set  
the address of the next node { this->nextNode = nextNode; };  
Node* getPrev() { return prevNode; }; // get the address of the  
prev node void setPrev(Node* prevNode) // set the address of  
the prev node  
{ this->prevNode = prevNode;  
}; private:  
int object; // it stores the actual value of the  
element Node* nextNode; // this points to the  
next node Node* prevNode; // this points to  
the previous node };
```

Question No.86

What you conclude from the running time analysis of disjoint sets.

Answer (page 405)

We conclude that:

- ❖ union is clearly a constant time operation

- ❖ Running time of $\text{find}(i)$ is proportional to the height of the tree containing node i .
- ❖ This can be proportional to n in the worst case (but not always).
- ❖ Goal: Modify union to ensure that heights stay small

Question No.87

what is c++ template.

Answer:- (page 75)

A template is a function or class that is written with a generic data type. when a programmer uses this function or class the generic data type is replaced with the data type needed to be used in the template function or in the template class. We only give the data type of our choice while calling a template function or creating an object of the template class. The compiler automatically creates a version of that function or class with that specified data type.

Question No.88

explain and write the code for union and find operation of parent array in disjoint sets.

Answer:- (page 399)

Find (i)

This loop is used to find the parent of an element or the name of the set that contains that element.

```
// traverse to the root (-1)
for(j=i; parent[j] >= 0; j=parent[j])
    ;
return j;
```

Union (i, j)

Now let's see the code for the function of union. Here we pass two elements to the function union. The union finds the roots of i and j . If i and j are disjoint sets, it will merge them. Following is the code of this function.

```
root_i = find(i);
root_j = find(j);
if (root_i != root_j)
    parent[root_j] = root_i;
```

In the code, we see that at first it finds the root of tree in which i exists by the $\text{find}(i)$ method and similarly finds the root of the set containing j by $\text{find}(j)$. Then there is a check in if statement to see whether these sets (roots) are same or not. If these are not the same, it merges them in such a fashion that the root i is set as the parent of root j . Thus, in the array where the value of root j exists, the value of root i becomes there.

Question No: 89 (Marks: 2)

Convert the given infix form to postfix form.

$A+B/C-D^E-F$

Answer:- (Page 66)

$AB+C/D-EF -^$

Question No: 90 (Marks: 2)

How we can implement Table ADT using Linked List

Answer:- repeat

Question No: 91 (Marks: 2)

If we allow assignment to constants what will happen?

Answer:- repeat

Question No: 92 (Marks: 2)

How heap sort works to sort a set of data.

Answer:- repeat

Question No: 93 (Marks: 3)

Give your comment on the statement that heap uses least memory in array representation of binary trees. Justify your answer in either case.

Answer: - repeat

Question No: 94 (Marks: 3)

How we can use concept of equivalence relations to generate a Maze.

Answer: - repeat

Question No: 95 (Marks: 3)

"For smaller lists, linear insertion sort performs well, but for larger lists, quick sort is suitable to apply." Justify why?

Answer: - repeat

Question No: 96 (Marks: 5)

695041732

Show the first five merging steps for Merge sort on this array.

Answer:- repeat

Question No: 97 (Marks: 5)

What is Disjoint Sets? Explain with an example.

Answer:- (Page 388)

Disjoint sets - Two sets are disjoint if they have no elements in common.

Example:-Suppose there are many people around you. How can we separate those who are related to each other in this gathering?

We make groups of people who are related to each other. The people in one group will say that they are related to each other. Similarly we can have many groups. So every group will say that they are related to each other with family relationship and there is no group that is related to any other group. The people in the group will say that there is not a single person in the other group who is related to them.

Question No: 98 (Marks: 5)

Write the code of the percolateDown() function and also comment it.

Answer:- (page 369)

```
template <class eType>
void Heap<eType>::percolateDown( int hole )
{
    int child;
    eType tmp = array[ hole ];
    for( ; hole * 2 <= currentSize; hole = child )
    {
        child = hole * 2;
        if( child != currentSize && array[child+1] < array[ child ] )
            child++; // right child is smaller
        if( array[ child ] < tmp )
            array[ hole ] = array[ child ];
        else break;
    }
    array[ hole ] = tmp;
}
```

Question No: 99 (Marks: 5)

Here is an array with exactly 15 elements:

123456789101112131415.

Suppose that we are doing a binary search for an element. Indicate any elements that will be found by examining two or fewer numbers from the array.

Answer:- repeat

Question No:100

How we can implement Table ADT using Linked List (marks 2)

Answer: - repeat

Question No:101

which three basic operations are performed on AVL trees (mark 3)

Answer:- (page 457)

- ❖ Insert
- ❖ Delete
- ❖ search

Question No:102

write down the c++ code to implement insertion sort algorithm (5)

Answer: - repeat

Question No:103

Give the name of three Divide and Conquer algorithms. (marks 3)

Answer: - repeat

Question No:104

Here is an array of ten integers:

5389170264

Draw this array after the FIRST iteration of the large loop in an insertion sort (sorting from smallest to largest). This iteration has shifted at least one item in the array! (marks 3)

Answer: - repeat

Question No: 105

For smaller list linear insertion sort is perform well but for large list Quick sort suitable to apply/justify why?????????

Answer:-

Since for smaller lists number of comparisons and interchange (if needed) decreases radically in insertion sort than quick sort. For larger lists, opposite occurs in both sorting techniques.

Question No: 106

Here is a array of exactly 15 elements

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Suppose that we are doing a binary search for an elemnt .Indicate any element that will be found by examining two or fewer number from the array (5 mrks)

Answer:- repeat

Question No: 107

Write the applications of heap.

Answer:- (page 84)

The stack and heap are used in function calls. When we allocate memory dynamically in our programs, it is allocated from the heap. The use of priority queue with the help of heap is a major application. The priority queue is itself a major data structure, much-used in operating systems. Similarly priority queue data structure is used in network devices especially in routers. Heap data structure is also employed in sorting algorithms.

Question No: 108

Write down the c++ code for selection sort (5 marks)

Answer: - repeat

Question No: 109

Write the application of binary search tree

Answer: - (page 369)

The searching and sorting operations are very common in computer science with the help of binary search tree.

Question No: 110

Write the about disjoint set?

Answer: - repeat

Question No: 111

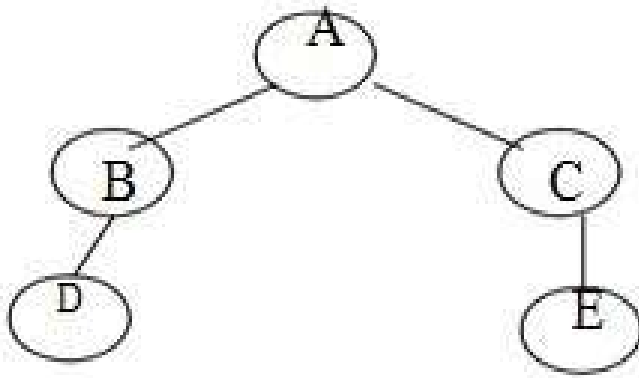
Heapify the elements of the following array (reading from left to right) into a Min Heap and show that Min Heap contents in the form of array as shown below,

original array	6	5	3	9	1	2	10	8	-

Answer: - repeat

Question No: 112

Represent the following binary tree in array



Answer: - repeat

Question No: 113

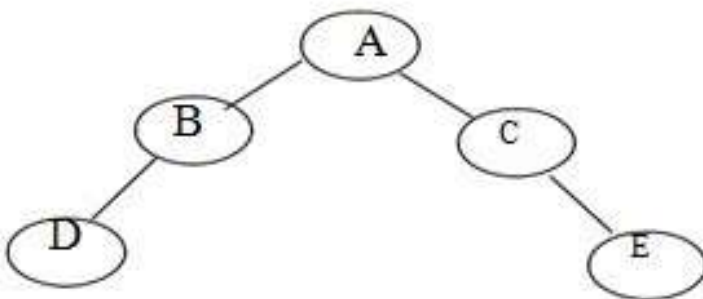
Write the any two operation of linked list that perform their function in one step. 2Marks

Answer: - (page 38)

- ❖ Add
- ❖ Remove
- ❖ Find

Question No: 114

Represent the following binary tree in array. 2Marks



Answer: - repeat

Question No: 115

Write different type of hashing. 2 Marks

Answer: -

Data structures that employ hashing methods to store keys (and information possibly associated to it) are denoted as **hashing types**.

Question No: 116

Give the table name that that we can on table abstract data type. 3mark

Question No: 117

Explain the process of Deletion in a Min-Heap 3 Marks

Answer: - repeat

Question No: 118

What is skip list? 3 Marks

Answer: - repeat

Question No: 119

What we can conclude from the running time array of disjoint set. 5 Marks

Answer: - repeat

Question No: 120

77, 33,44,11,88,22,66,55 sort the following array using insertion algorithm. 5 Marks

Answer: - See the same solution above

Question No: 121

What is disjoint set? 5 Marks

Answer: - repeat

Question No: 122

Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right

child nodes and Parent node of the
node D

(Marks: 5)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...

Answer: - repeat

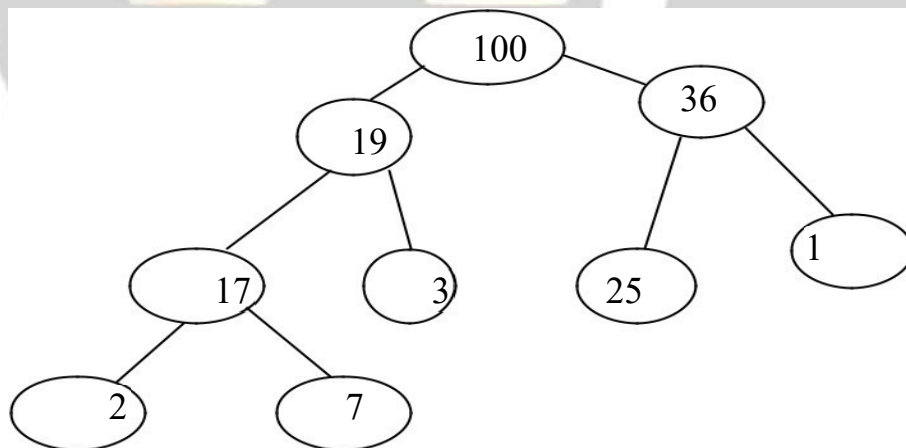
Question No.123

Give one example of Hashing.

Answer: - repeat

Question No.124

Convert this tree representation of a heap into the corresponding array representation.
(2)



Answer: - repeat

Question No.125

Here is an array of ten integers:

5 3 8 9 1 7 2 6 4

Draw this array after the FIRST iteration of the large loop in an insertion sort (Sorting from smallest to largest). This iteration has shifted at least one item in the array.

Answer:- repeat

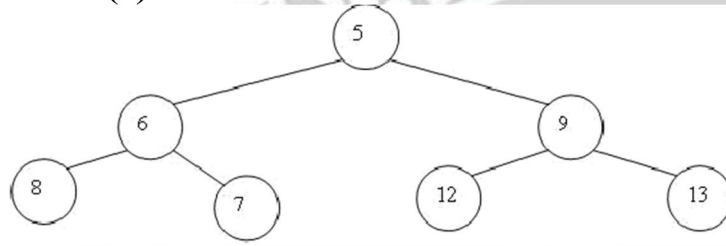
Question No.126

Which implementation of disjoint set is better with respect space and why? (3)

Answer: - repeat

Question No.127

Consider the following Min Heap apply operation delMin on it and show resultant Heap. (3)



Answer: - repeat

Question No.128

Which is Dynamic Equivalence Problem with an example? (5)

Answer: - (page 395)

We are using sets to store the elements. For this purpose, it is essential to remember which element belongs to which set. We know that the elements in a set are unique and a tree is used to represent a set. Each element in a tree has the same root, so the root can be used to name the set. The item (number) in the root will be unique as the set has unique values of items. We use this root as the name of set for our convenience. Otherwise, we can use any name of our choice. So the element in the root (the number) is used as the name of the set. The *find* operation will return this name. Due to the presence of many sets, there will be a collection of trees. In this collection, each tree will be representing one set. If there are ten elements, we will have ten sets initially. Thus there will be ten trees at the beginning. In general, we have N elements and initially there will be N trees. Each tree will have one element. Thus there will be N trees of one node. Now here comes a definition for this collection of trees, which states that a collection of trees is called a *forest*.

The trees used for the sets are not necessarily binary trees. That means it is necessary that each node should have a maximum of two children nodes. Here a n may have more than two children nodes.

To execute the union operation in two sets, we merge the two trees of these sets in such a manner that the root of one tree points to the root of other. So there will be one root, resulting in the merger of the trees.

In the find operation, when we call *find* (*x*), it helps us to know which set this *x* belongs to. Internally, we find this *x* in a tree in the *forest*. When this *x* is found in a tree the *find* returns the number at root node (the name of the set) of that tree.

Question No.129

Here is an array of ten integers

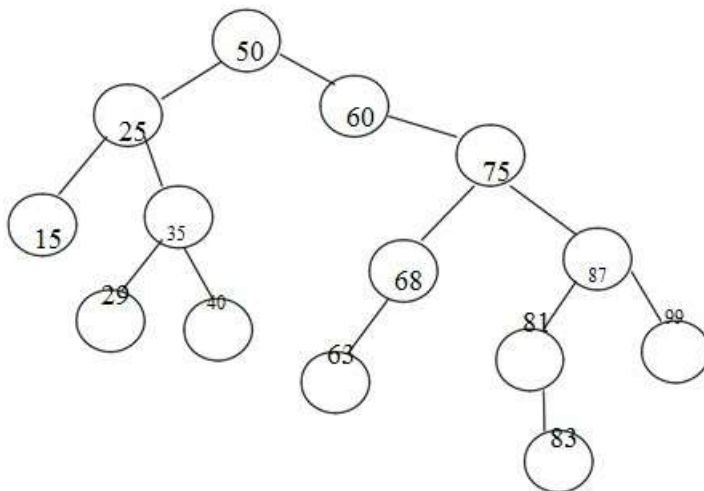
5 3 8 9 1 7 0 2 6 4

Sort the array by using selection sort algorithm and show content of array after each step. (5)

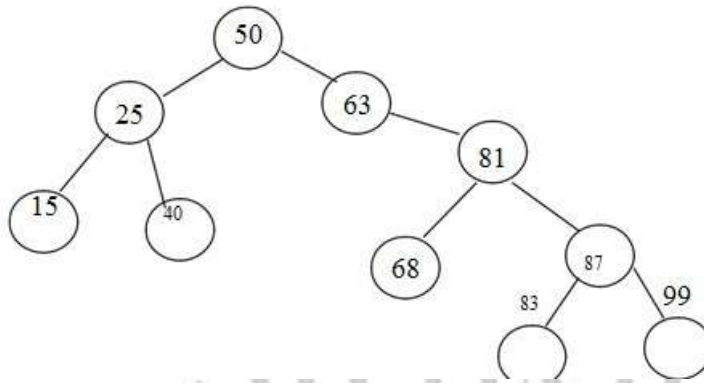
Answer: - repeat

Question No.130

Consider the following Binary tree and Delete the key 29, 35, 75 and 60 only. Show the resultant tree after deleting these keys. 5 marks



Answer:



Question No.131

Suppose we have the following representation for a complete Binary Search Tree, tell the Left and Right child nodes and Parent node of the Node **D**

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	...
0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1	1	1	1	1	2	...
										0	1	2	3	4	5	6	7	8	9	0	...

Answer: - repeat

Question No. 132

For a perfect binary tree of height 5, what will be the sum of heights of nodes?

2marks

Answer: - repeat

Question No.133

Properties of equivalence class.

Answer: - repeat

Question No.134

Two ways for the implementation of Queue data structure.

Answer: - (Page 89, 91)

1. Implementation of queue using linked list
2. Implementation of queue using Array.

Question No.135

Worst case of selection sort

Answer: -

The worst case occurs if the array is already sorted in descending order. Nonetheless, the time required by selection sort algorithm is not very sensitive to the original order of the array to be sorted.

Question No.136

“is a sibling of” set of all human beings is equivalence relation or not? Explain.

Answer: (Page 387)

Yes it is an equivalence relation

First rule is reflexive i.e. for all element $x \in S$, $x R x$. Suppose that x is Haris so $Haris R Haris$. This is true because everyone is related to each other. Second is Symmetry: for all elements x and y , $x R y$ if and only if $y R x$. Suppose that y is Saad. According to the rule, $Haris R Saad$ if and only if $Saad R Haris$. If two persons are related, the relationship is symmetric i.e. if I am cousin of someone so is he. Therefore if Haris is brother of Saad, then Saad is certainly the brother of Haris. The family relationship is symmetric. This is not the symmetric in terms of respect but in terms of relationship. The transitivity is: for all elements x , y and z . If $x R y$ and $y R z$, then $x R z$. Suppose x is Haris, y is Saad and z is Ahmed. If Haris —is related to Saad, Saad —is related to Ahmed. We can deduce that Haris —is related to Ahmed. This is also true in relationships. If you are cousin of someone, the cousin of that person is also related to you. He may not be your first cousin but is related to you.

Question No.137

Linked Memory

Linked memory is a memory in which the various cells of memory are not located continuously. In this process, each cell of the memory not only contains the value of the element but also the information where the next element of the list is residing in the memory. It is not necessary that the next element is at the next location in the memory. It may be anywhere in the memory. We have to keep a track of it.

Thus, in this way, the first element must explicitly have the information about the location of the second element. Similarly, the second element must know where the third element is located and the third should know the position of the fourth element and so on

Question No.138

3 5 9 8 1 7 0 2 6 4 , draw after the 1st iteration of large loop in a selection sort (smallest to largest)

Answer: - repeat

Question No. 139**5 marks**

Union commands 1, 2, 3, 4, 5
`union(4,2),union(3,1),union(5,4)`

Answer: - repeat

Question No.140

Code for removing an element from a linked list data structure if it is implementation by using linked memory methods.

Answer: Page 34

```
void remove() {  
    if( currentNode != NULL && currentNode !=  
        headNode) { lastCurrentNode-  
        >setNext(currentNode->getNext()); delete  
        currentNode;  
        currentNode = lastCurrentNode->getNext();  
        size--;  
    }  
};
```