CS301 PAAn BANA RAS WALA

2022 #Final-Term

ORANGE MONKEY TEAM

Important NOTE:-

Maybe toppers are successful in school or college carrier but backbenchers are successful in their life because toppers run according to books while backbenchers don't follow rules and regulation that is why backbenchers are successful. For life, there are no books and another reason why toppers can't succeed because they fear failure and don't try anything new. Backbenchers don't fear failure as they have faced the failure in their school or college carrier so that is the reason they try new things and at last gain success.

As Abdul Kalam nicely quoted, "The best brains of the nation may be found on the last benches of the classroom"

Translation into Urdu:

ہوسکتا ہے کہ ٹاپرز اسکول یا کالج کیریئر میں کامیاب ہوں لیکن بیک بینچر اپنی زندگی میں کامیاب ہوتے ہیں کیونکہ ٹاپر کتابوں کے مطابق چلتے ہیں جب کہ بیک بینچر قواعد و ضوابط پر عمل نہیں کرتے اسی وجہ سے بیک بینچر کامیاب ہوتے ہیں۔ زندگی کے لیے، کوئی کتابیں نہیں ہیں اور ایک اور وجہ ہے کہ ٹاپرز کامیاب نہیں ہو پاتے کیونکہ وہ ناکامی سے ڈرتے ہیں اور کچھ نیا کرنے کی کوشش نہیں کرتے۔ بیک بینچر ناکامی سے نہیں ڈرتے کیونکہ انہوں نے اپنے اسکول یا کالج کیریئر میں ناکامی کا سامنا کیا ہے یہی وجہ ہے کہ وہ نئی چیزیں آزماتے ہیں اور آخر کار کامیابی حاصل کرتے ہیں۔

جیسا کہ عبدالکلام نے اچھی طرح سے نقل کیا ہے، "قوم کے بہترین دماغ کلاس روم کے آخری بینچوں پر مل سکتے ہیں"۔

1. **When a complete binary tree represented by an array then if right child is at position 5 then left child will be at position _____.**

4….confirm

When a complete binary tree represented by an array then if right child is at position 5 then left child will be at position _____

a. 2
b. 3
c. 4

2. **Which of the following statement is not correct about binary trees, where binary tree traversal are carried out repeatedly?**

It is very cumbersome to modify the tree data structure as most of pointer fields are not NULL……confirm

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".

3. **When a complete binary tree represented by an array then for any array element at position I, the left child is at position _____.**

2i……confirm

- For any array element at position *i*, the left child is at 2*i*, +1) and the parent is at floor(*i*/2).

4. **When a complete binary tree represented by an array then for any array element at position I, the right child is at position _____.**

(2i+1)……confirm

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

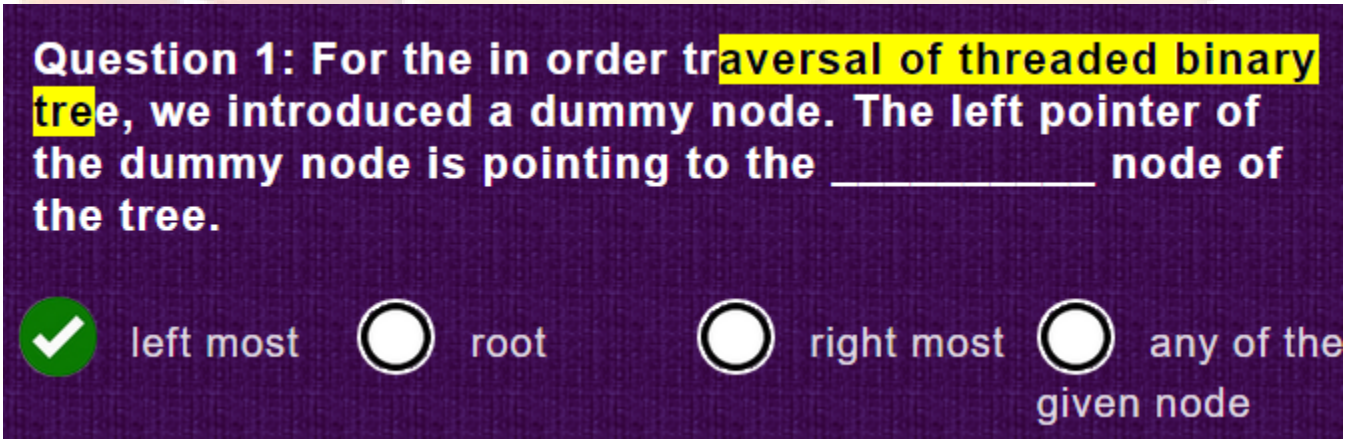
5. **When a complete binary tree represented by an array then for any array element at position I, the parent is at position _____.**

Floor(i/2)….confimr

For any array element at position *i*, +1) and the parent is at floor(*i*/2).

6. **For the inorder traversal of threaded binary tree, we introduced a dummy node. The left pointer of the dummy node is pointing to the ____ node of the tree.**

Left most….confirm



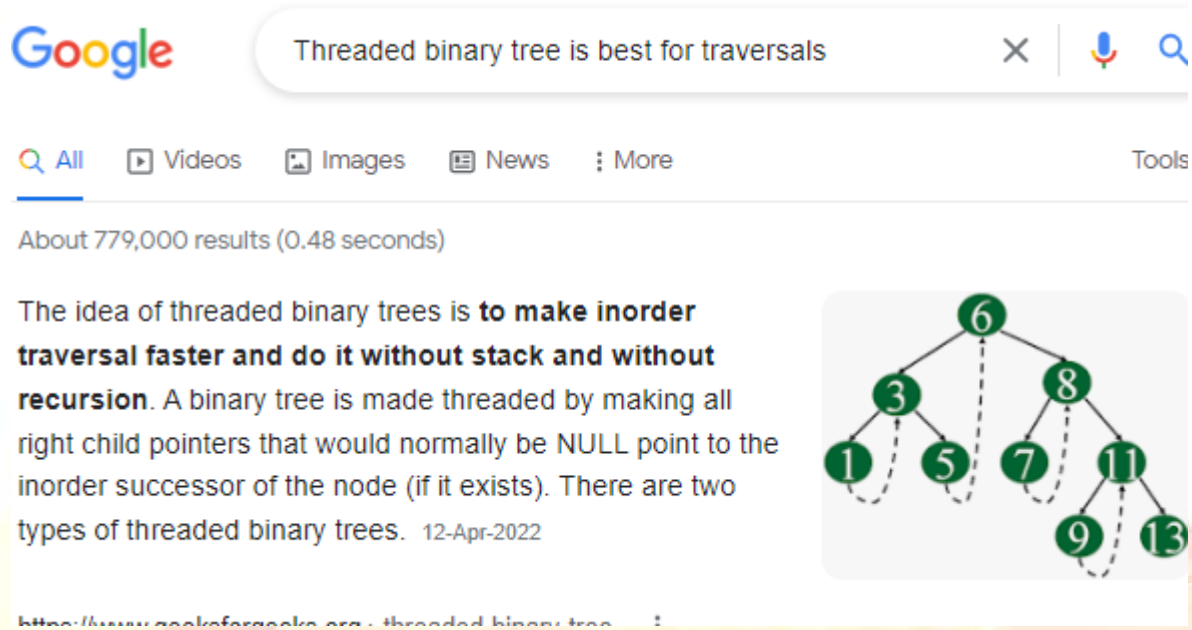
7. **Consider a max heap, represented by the following array 40,30,20,10,15,16,17,8,4**

Which of the following is the parent of node 17?

20

8. **Which of the following is best for traversal?**

Threaded binary Tree….confirm



9. **See the below code and fill the appropriate answer for ? sign**

```
/* This routine will traverse the binary search tree */
void fastInorder(TreeNode* p)
{ while((p=nexInorder(p)) != ?)
cout << p->getInfo(); }
```

dummy….confirm

```
/* This routine will traverse the binary search tree  */
void fastInorder(TreeNode* p)
{
        while((p=nexInorder(p)) != dummy)  cout << p->getInfo();
}
```

10. **Which of the following is a property of binary tree?**

A binary Tree with N internal nodes has 2N links, N+1 links to internal nodes and N-1 links to external nodes….confirm

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.

11. Traversing a binary tree can only be done using _____.

Both recursion and iteration …..confirm

Traversing a binary (Raise For Success) tree can only be done using _____

- Iteration
- Recursion
- Both recursion and iteration Correct

12. We implement the heap by _____.

Complete binary tree….confirm

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

13. Which of the following statement concerning heaps is Not true?

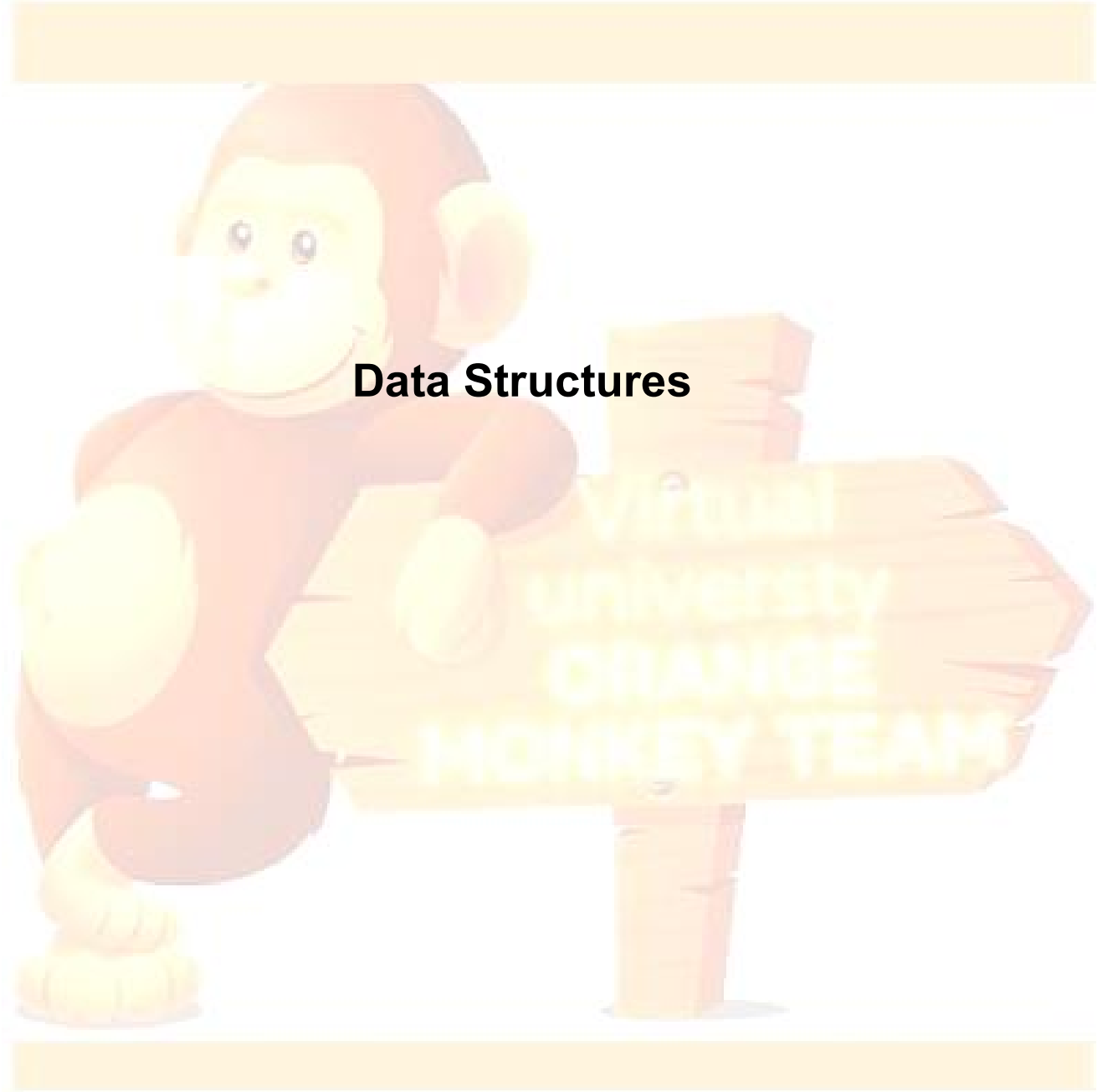
Traversing a heap in order provides access to the data in numeric or alphabetical order…..confrim

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.

Data Structures



ORANGE MONKEY TEAM

Table of Contents

Data Structures

Lecture No. 26

Reading Material

Data Structures and Algorithm Analysis in C++ Chapter. 4

4.4.2

Summary

- Hoffman 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 Hoffman 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*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.

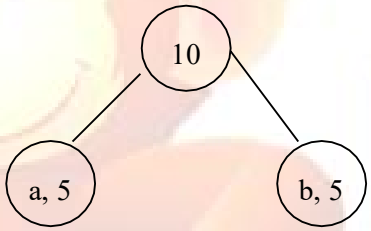


Fig 26.1:

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.

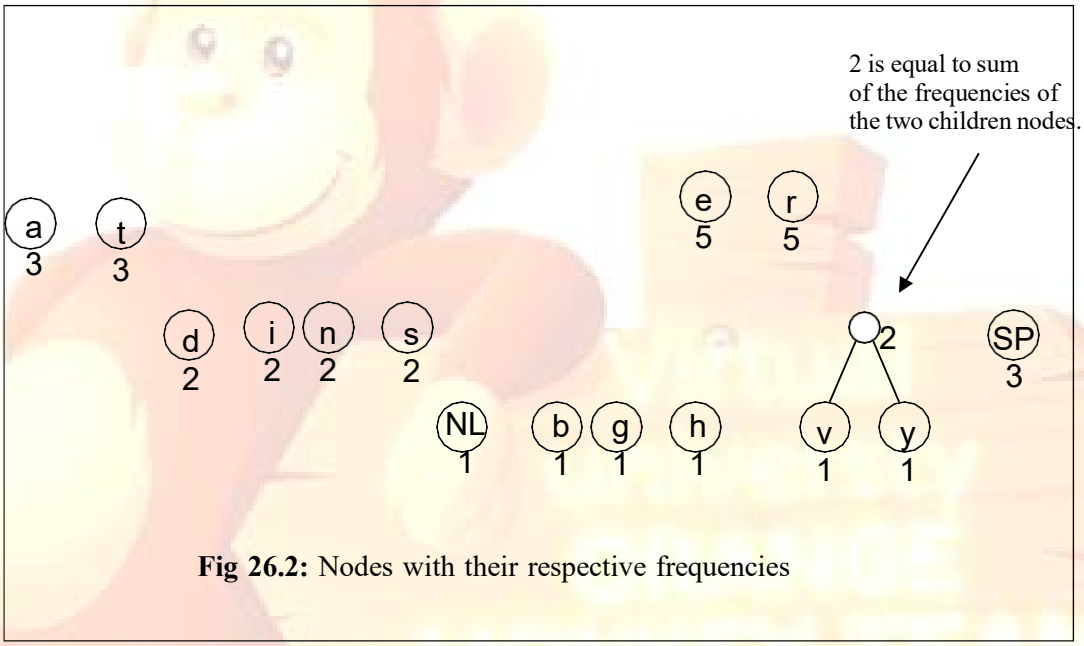


Fig 26.2: Nodes with their respective frequencies

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).

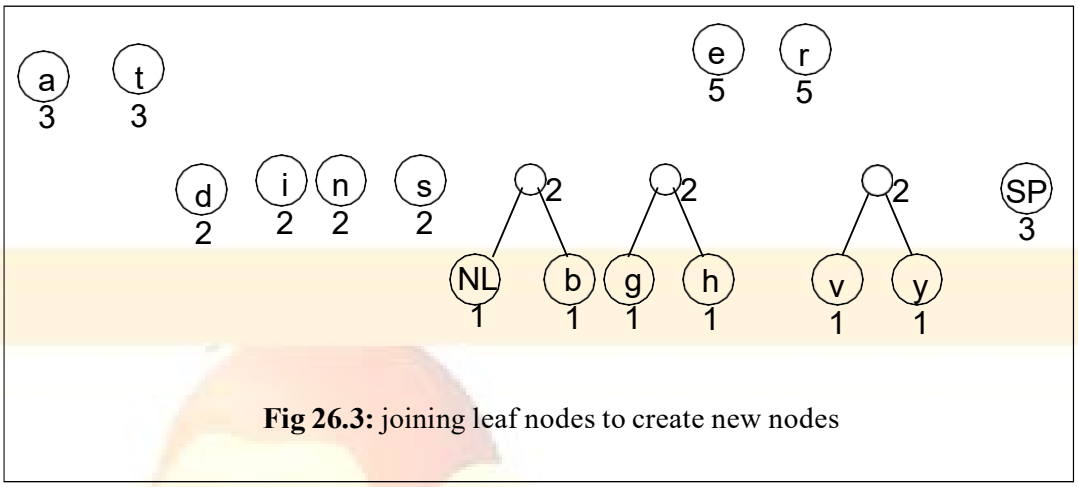


Fig 26.3: joining leaf nodes to create new nodes

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.

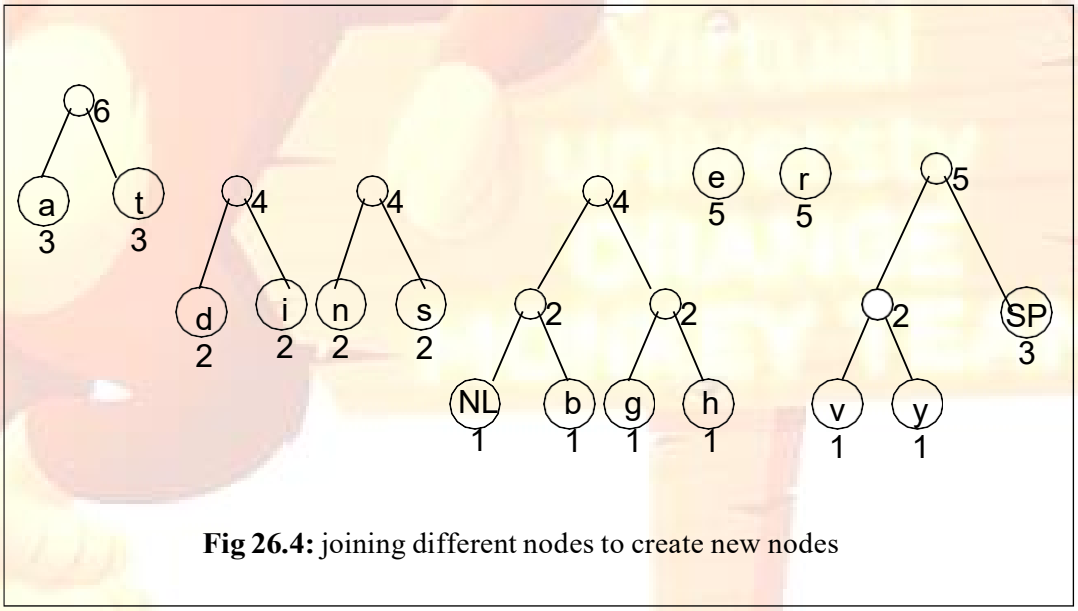


Fig 26.4: joining different nodes to create new nodes

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 5 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.

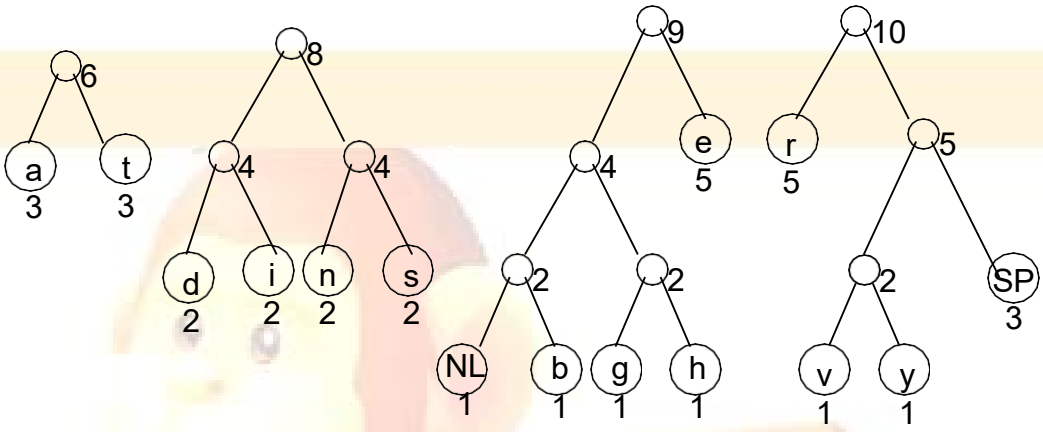


Fig 26.5: joining of nodes to build a tree

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.

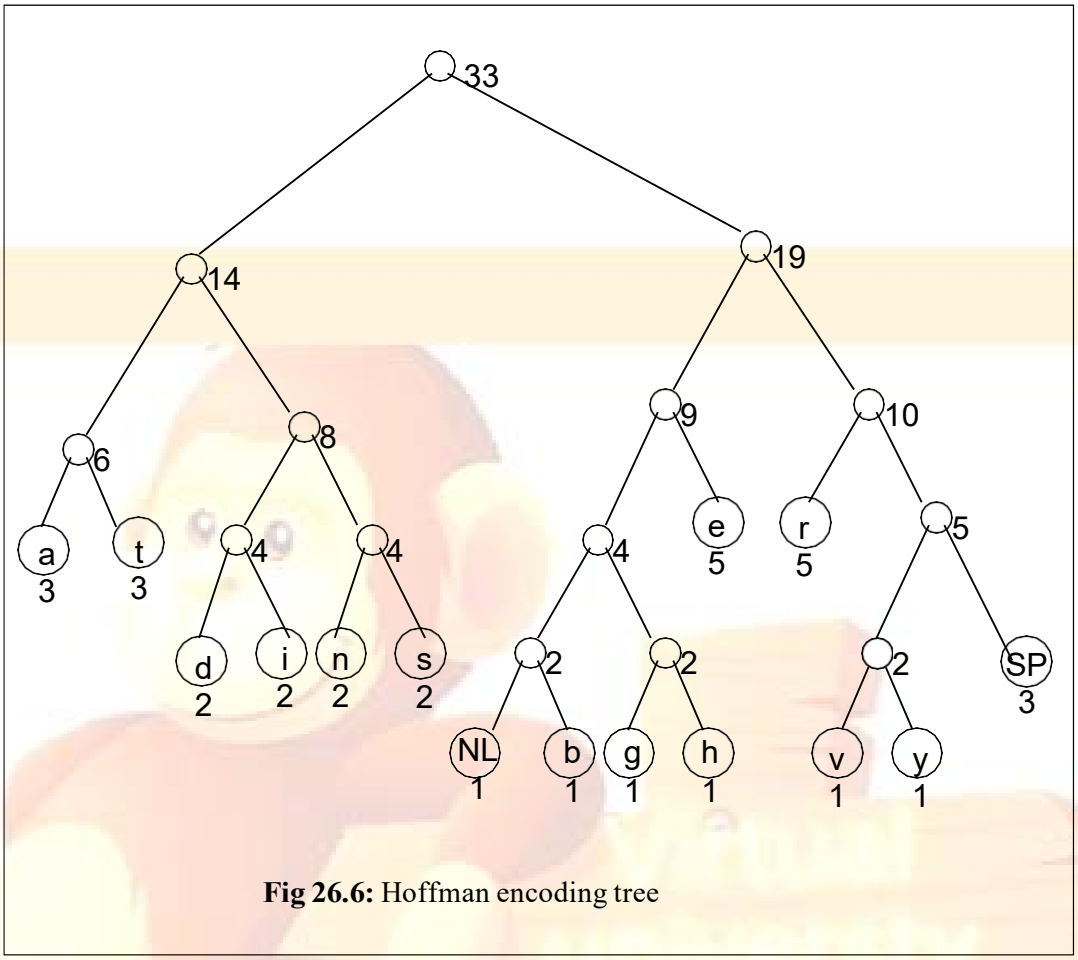


Fig 26.6: Hoffman encoding tree

Now we will perform other steps of Hoffman 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.

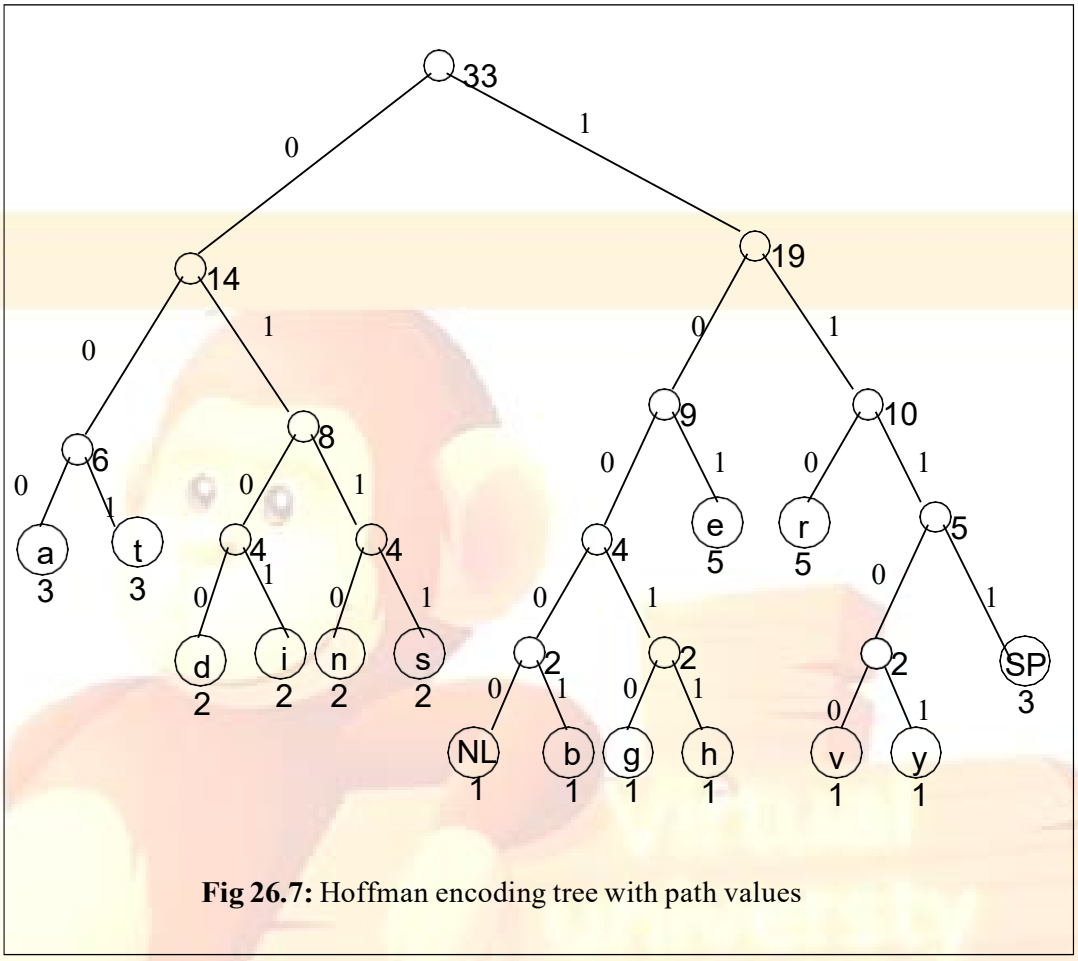


Fig 26.7: Hoffman encoding tree with path values

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
001110000111001011100111010101101001011110010010111101010000100101010011111000101010110000
11011101111100111010110101111000 0
```

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 Hoffman code. The encoded message is shown above.

Let’s compare this Hoffman 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 x 8 = 264. But in the Hoffman 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 Hoffman 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 Hoffman 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 form 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.

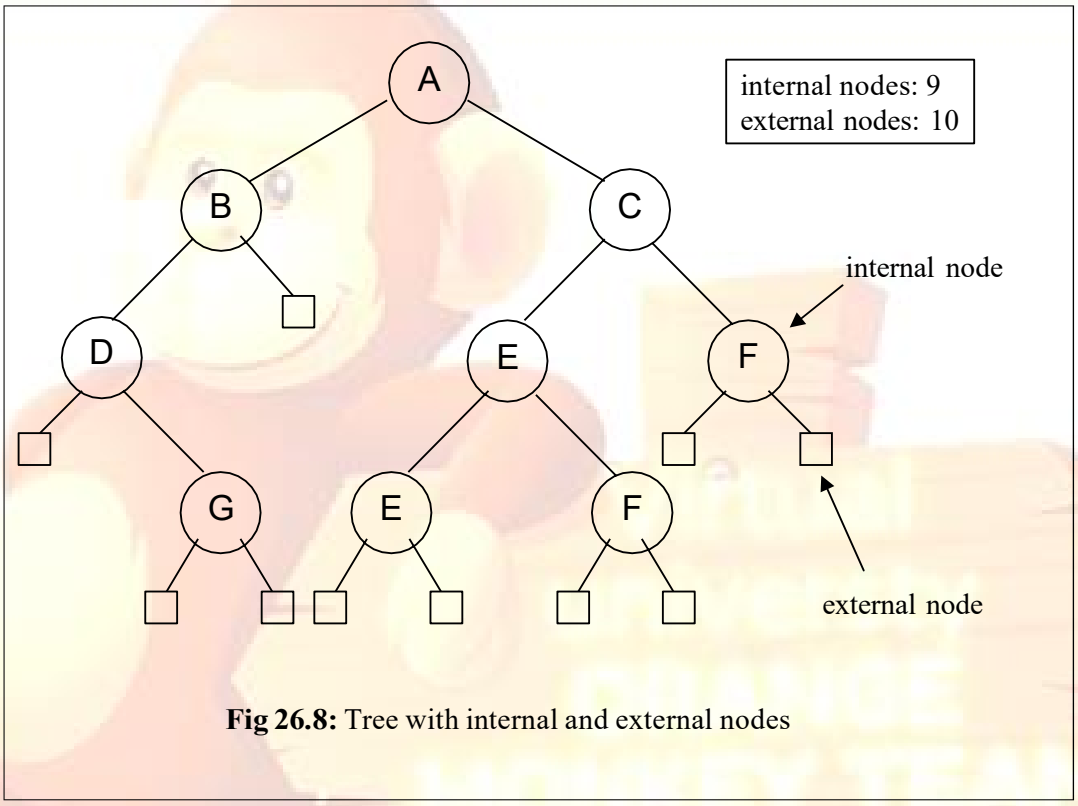


Fig 26.8: Tree with internal and external nodes

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 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.

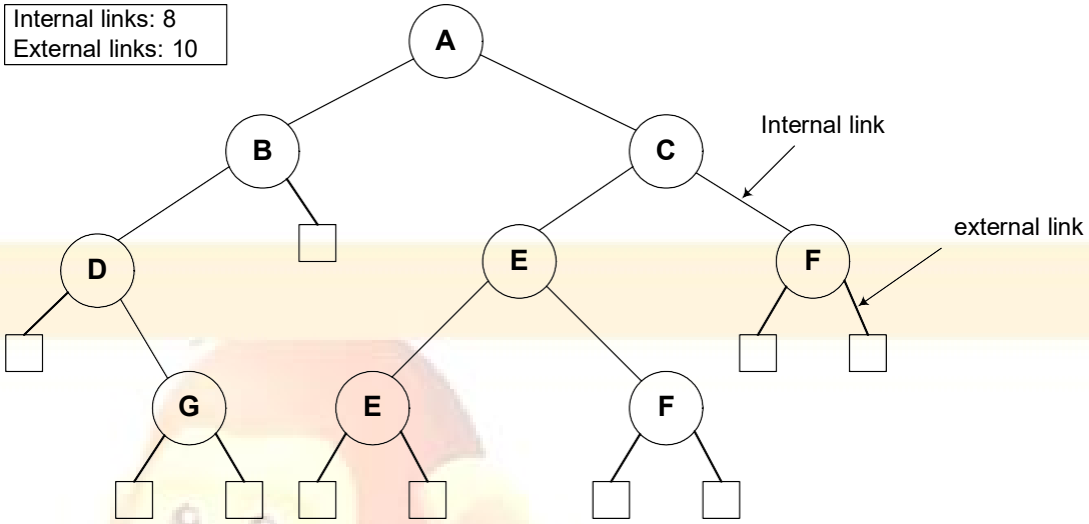


Fig 27.1

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 x 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.

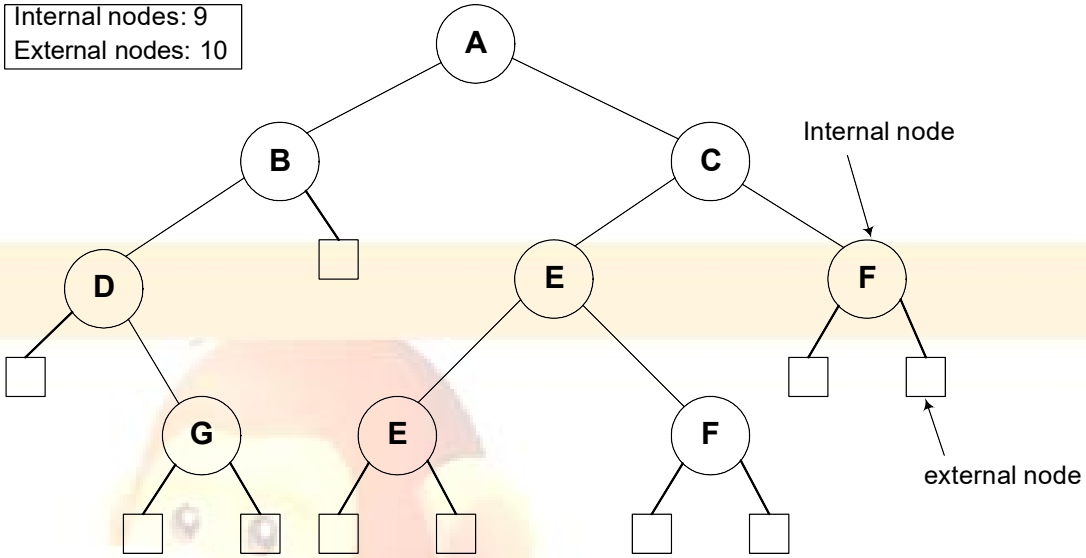


Fig 27.2

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

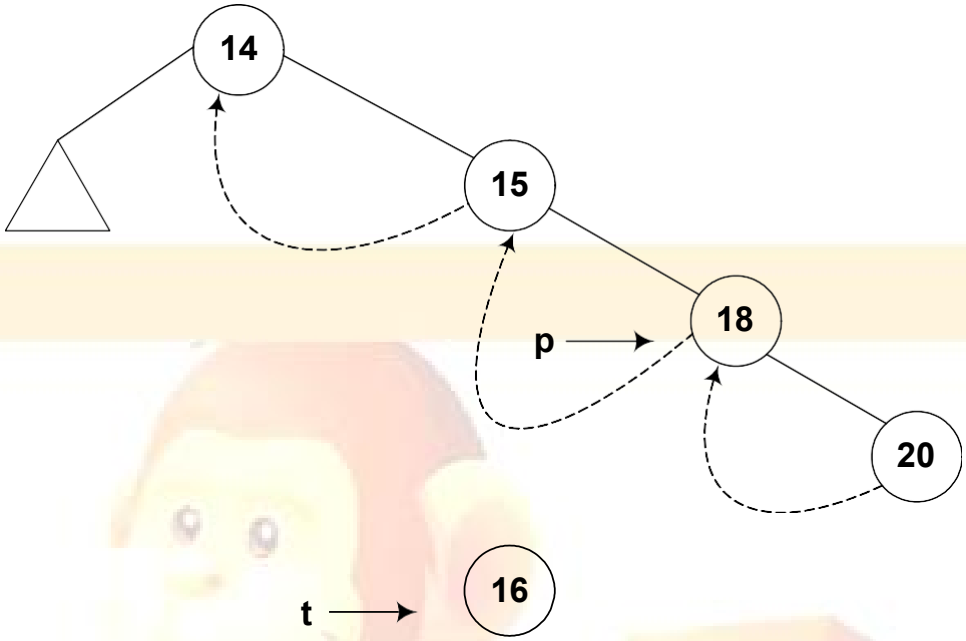


Fig 27.3

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.

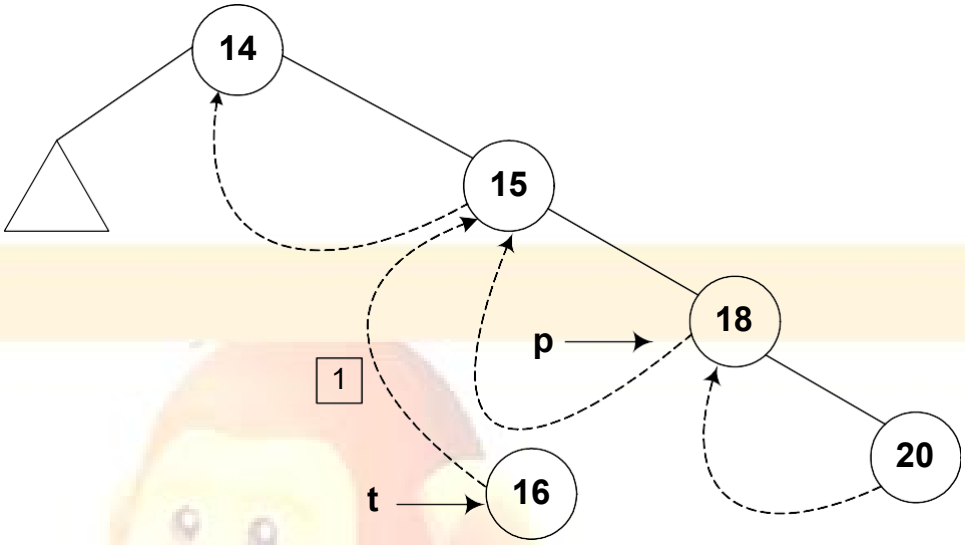


Fig 27.4

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->L = p->L; // copy the thread
2. t->LTH = thread;
3. t->R = p; // *p is successor of *t
4. t->RTH = thread;
5. p->L = t; // attach the new leaf
6. p->LTH = 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->L = p->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).

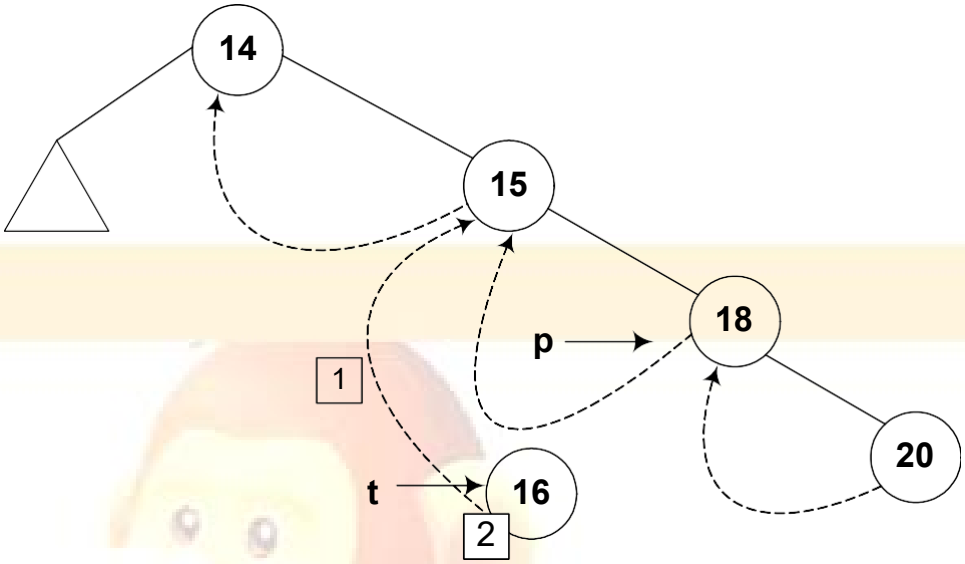


Fig 27.5

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

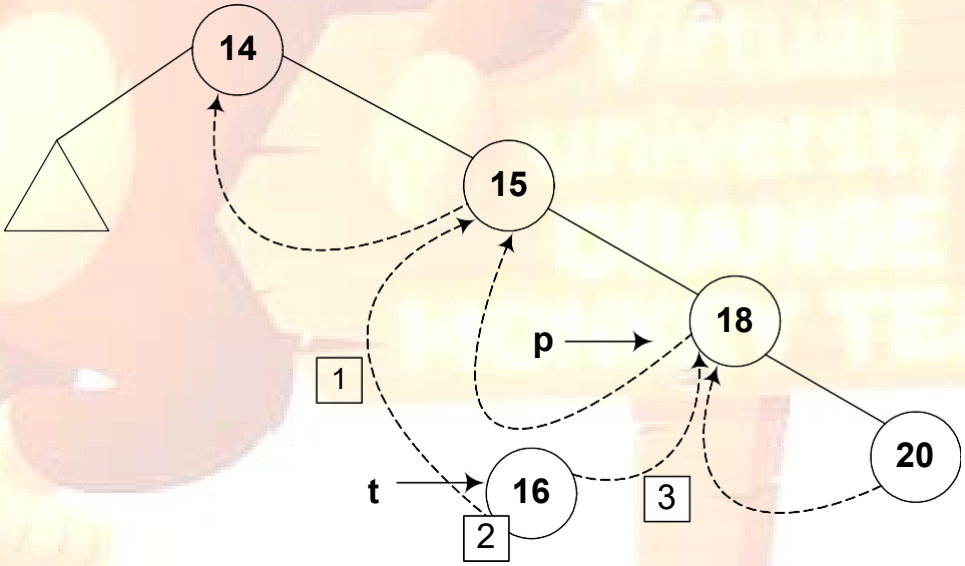


Fig 27.6

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

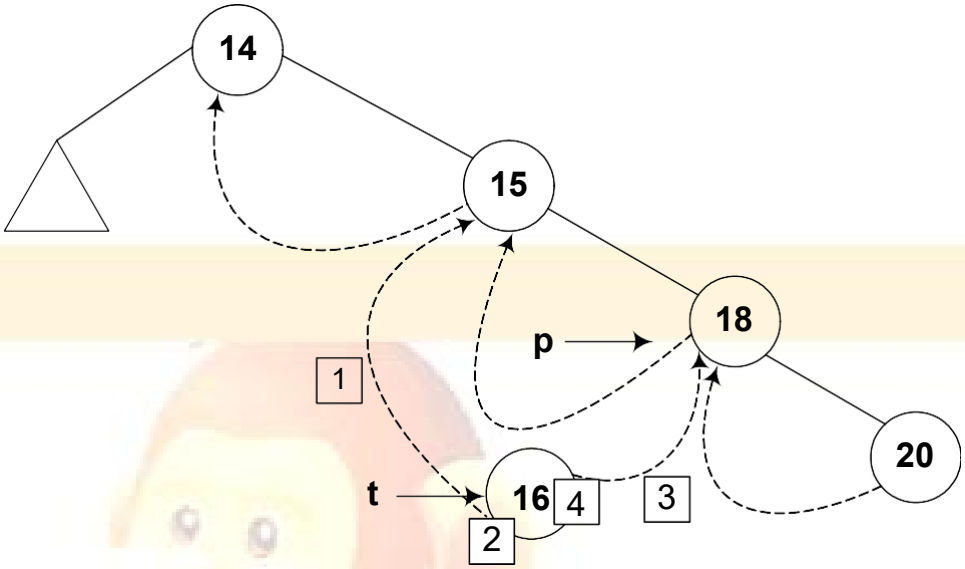


Fig 27.7

Next line, *t->RTH = thread* contains flag turning on code.

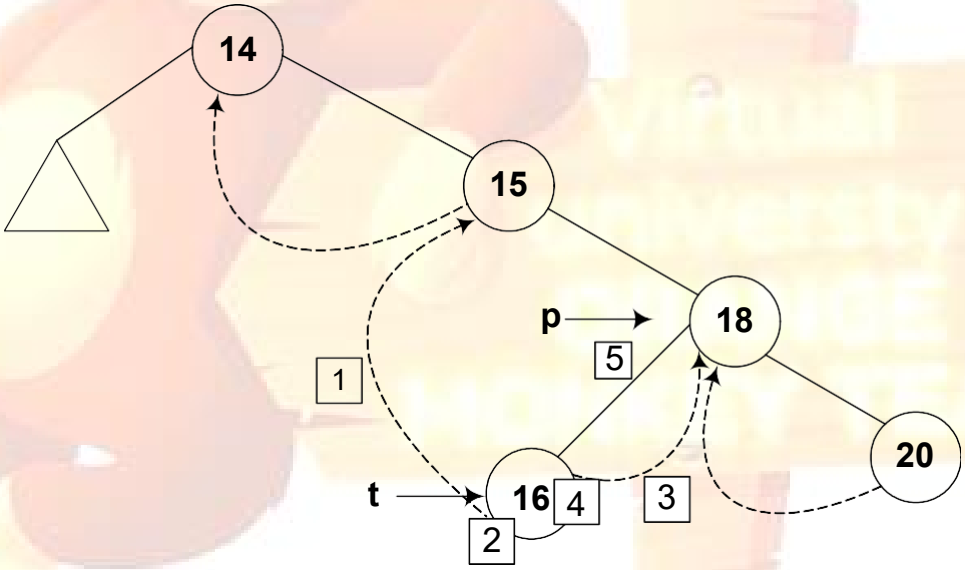


Fig 27.8

In the next line *p->L = t*, the node 16 is attached as the left child of the node 18.

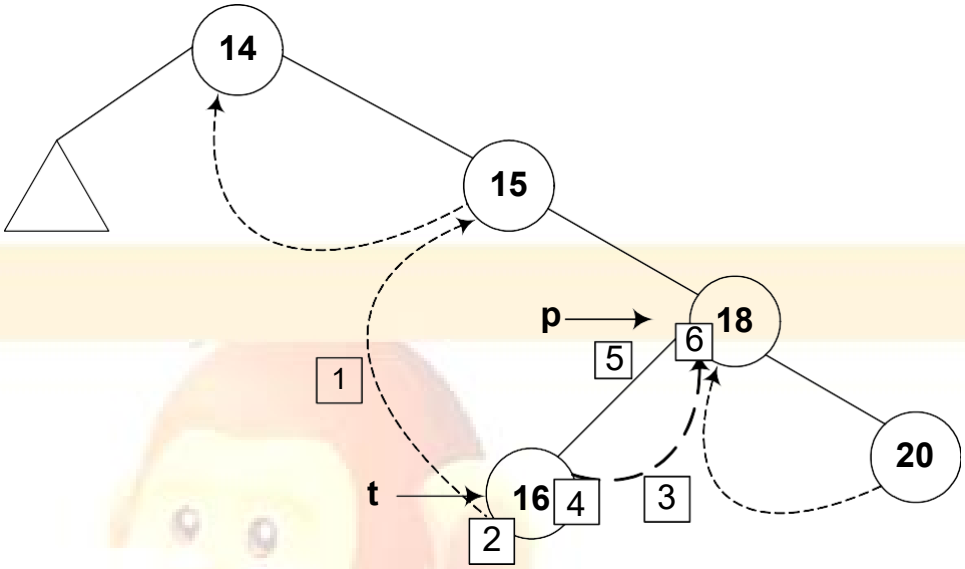


Fig 27.9

The flag is truned on in the last line, *p->LTH = child*.

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

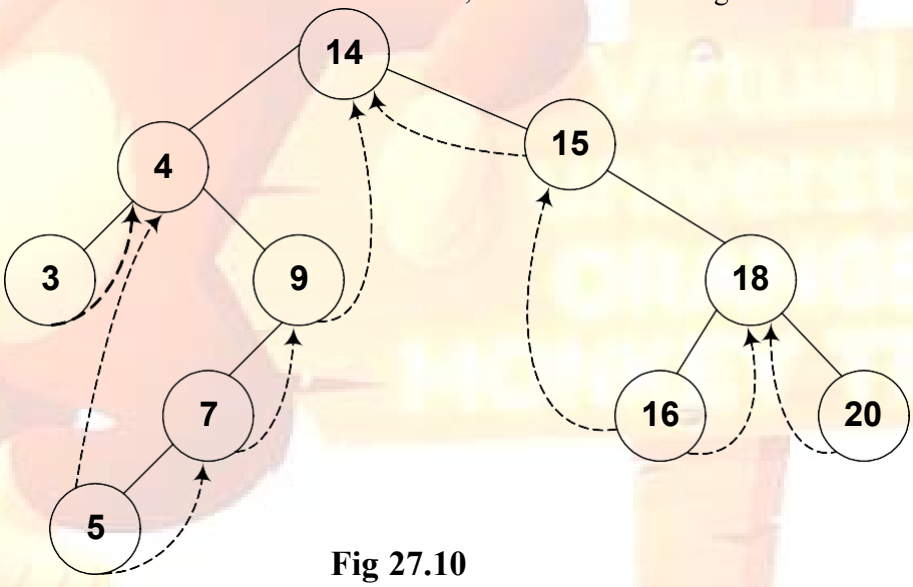


Fig 27.10

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?

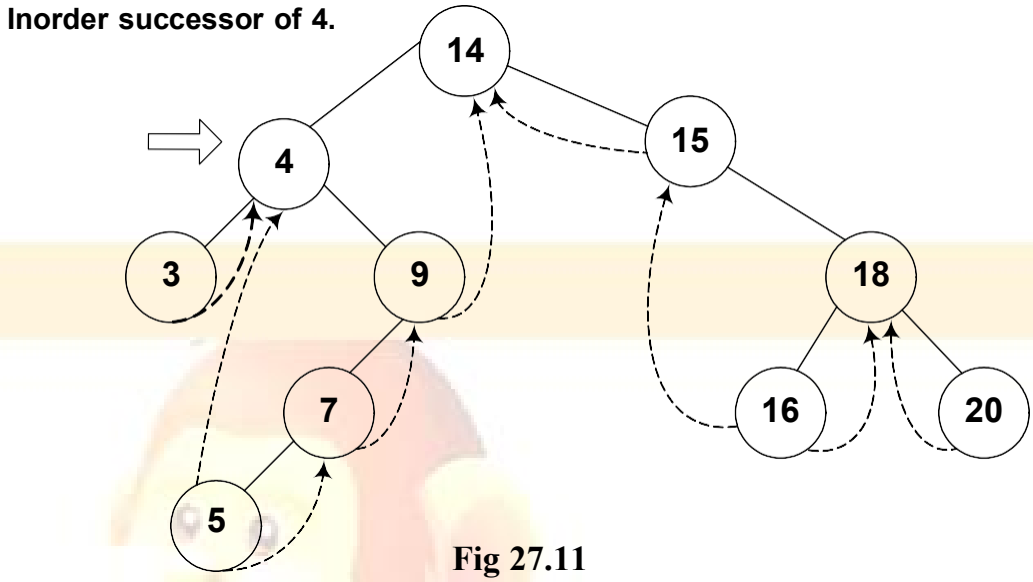


Fig 27.11

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.

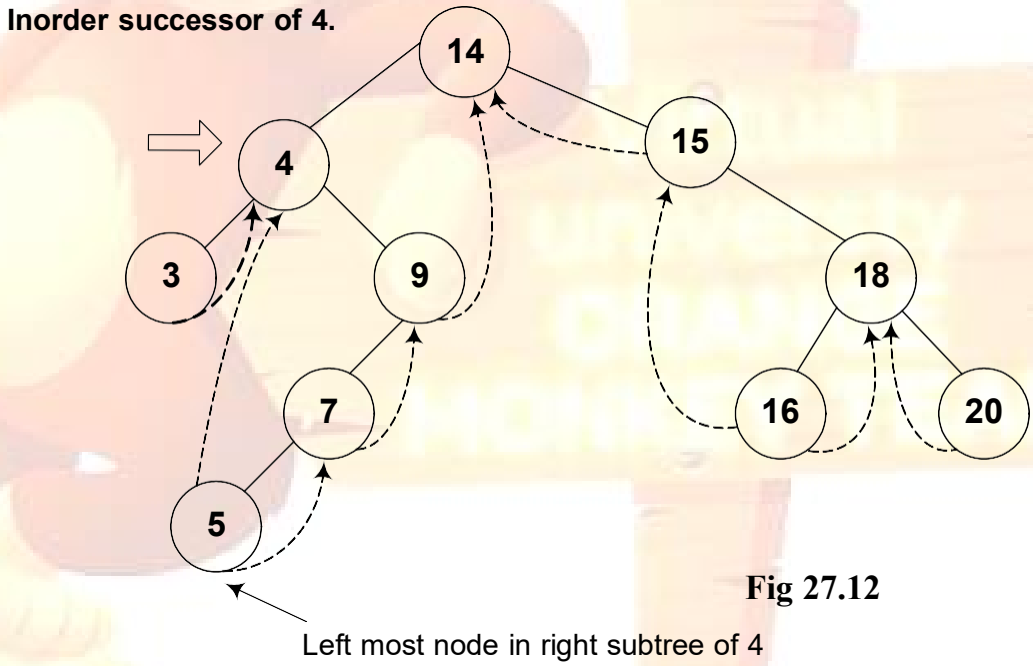


Fig 27.12

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.

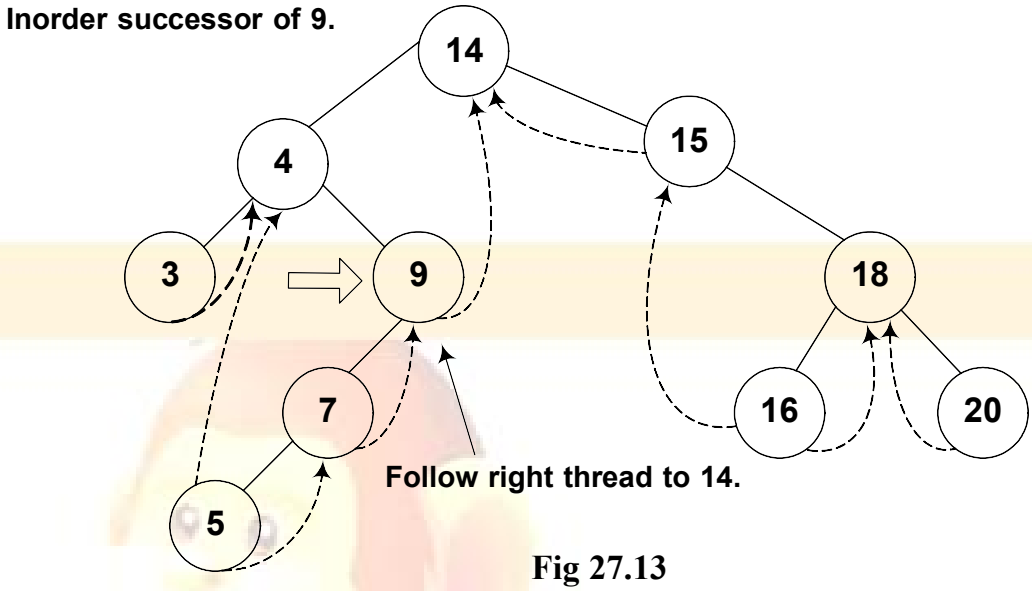


Fig 27.13

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.*

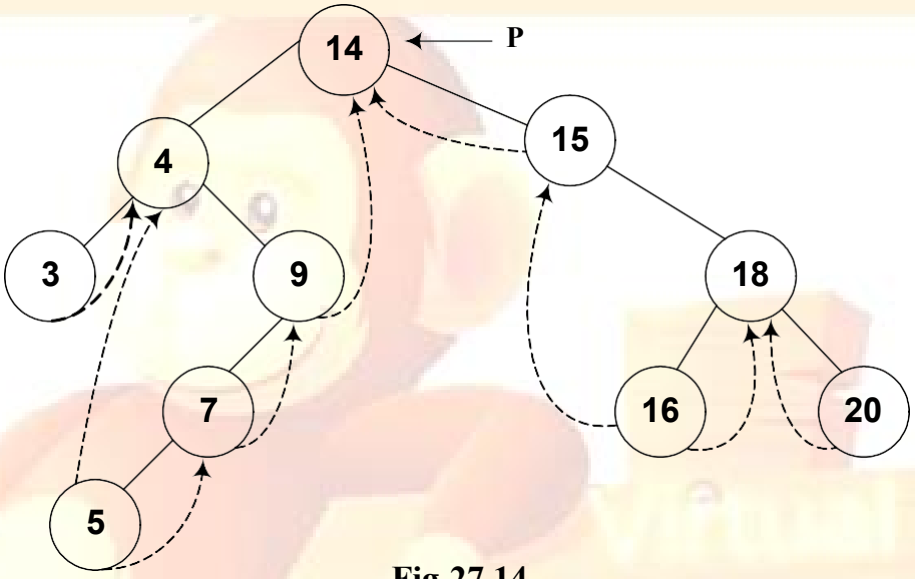


Fig 27.14

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 *nexInorder* 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.

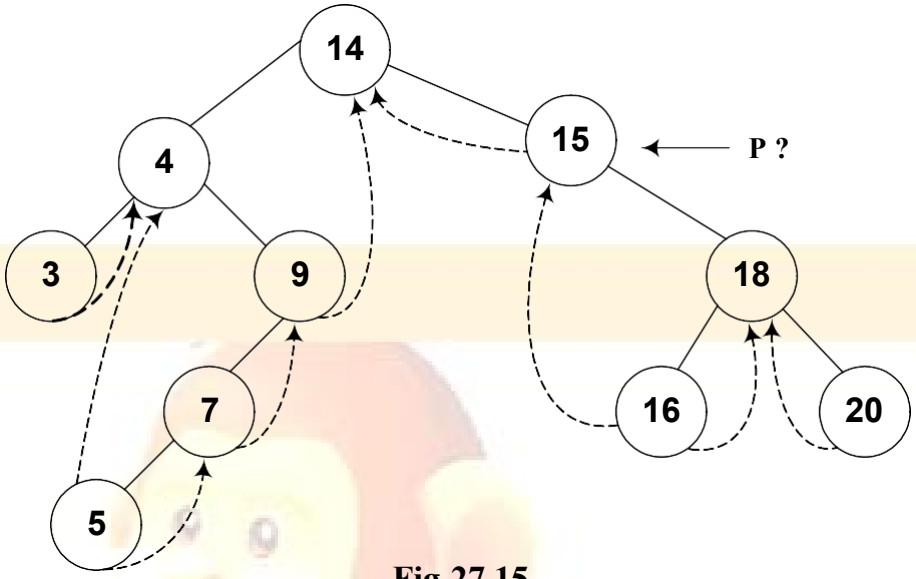


Fig 27.15

– *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

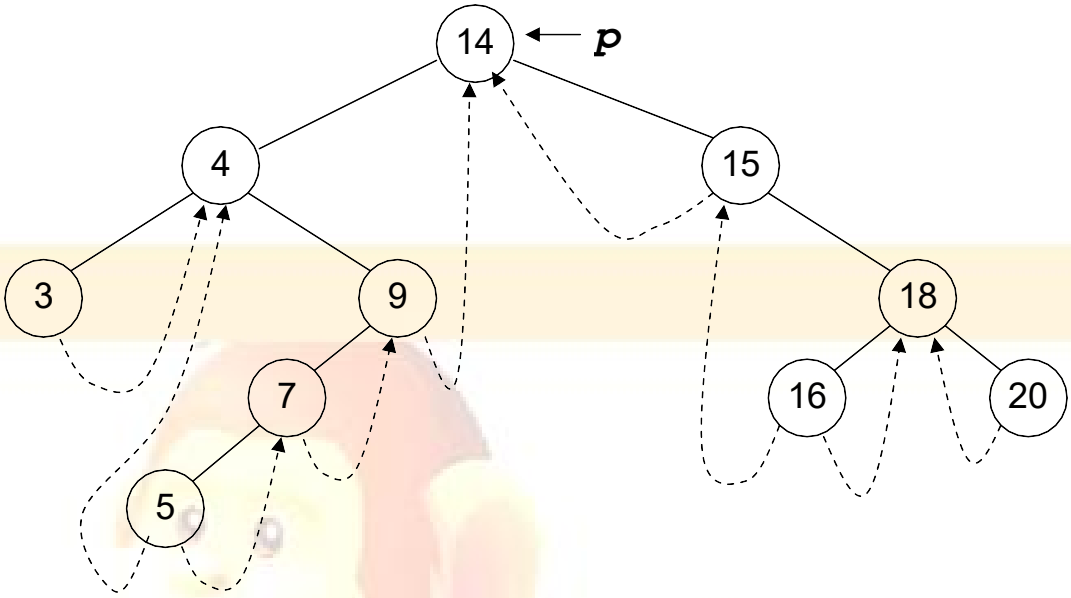
- Inorder traversal in threaded trees
- Complete Binary Tree

Inorder traversal in threaded trees

Discussion on the inroder 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;
  }
}
```

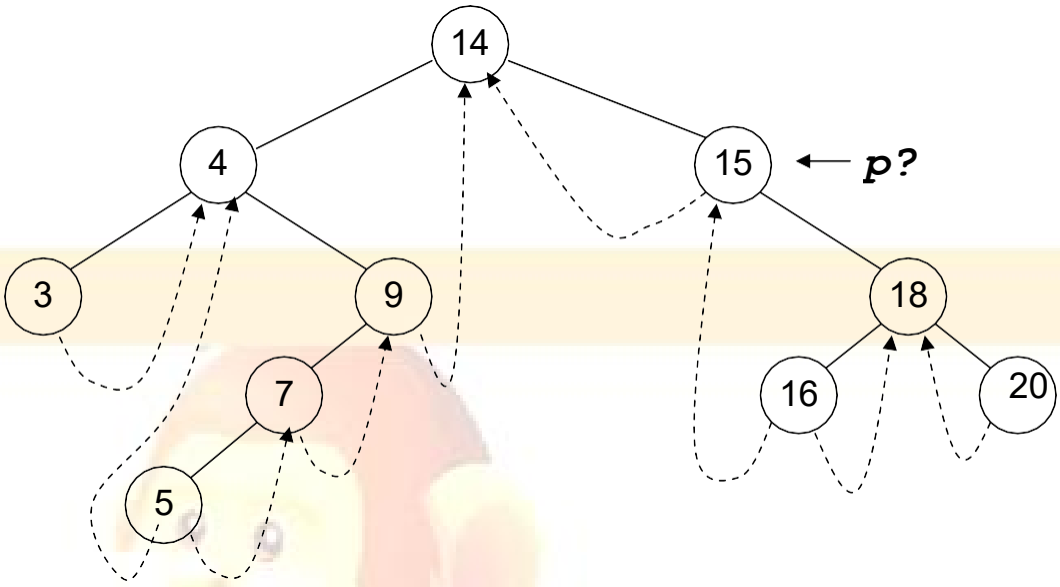
When we apply this routine on the sample tree, it does not work properly because the pointer that points to the node goes in the wrong direction. How can we fix this problem? Let's review the threaded binary tree again:



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

Our *nextInoder* 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 loops 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.

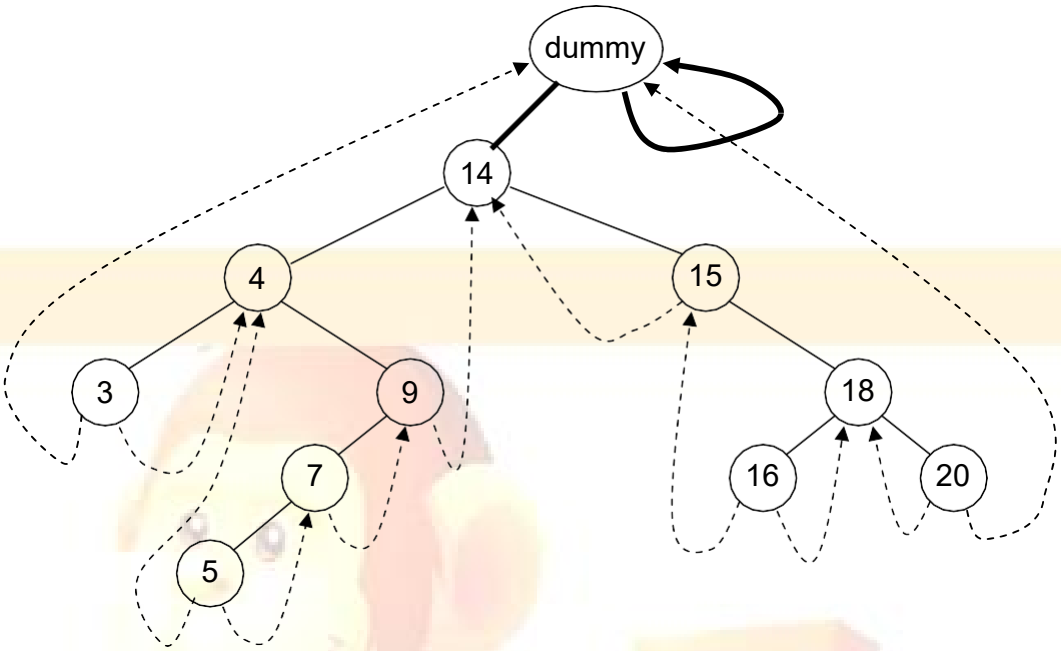
We will pass the root of the tree to the *nextInorder* routine. The pointer *p* is pointing to the node 14 i.e. the root node. As the right pointer of the node 14 is not a thread, so the pointer *p* will move to the node 15 as shown below:



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.

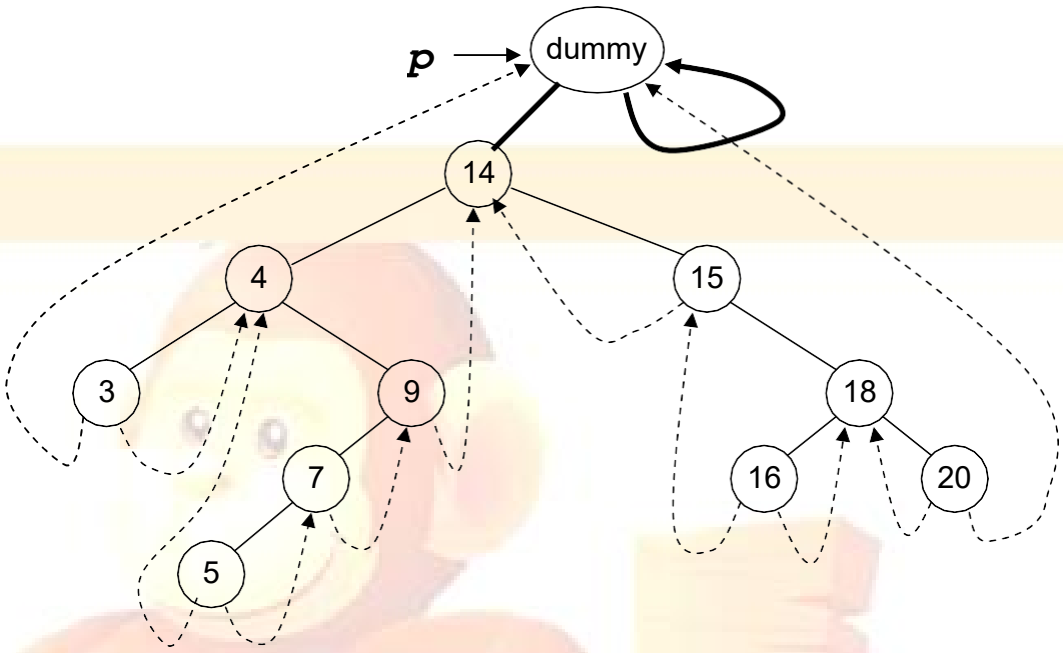
```
/* This routine will traverse the binary search tree  */

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);
```

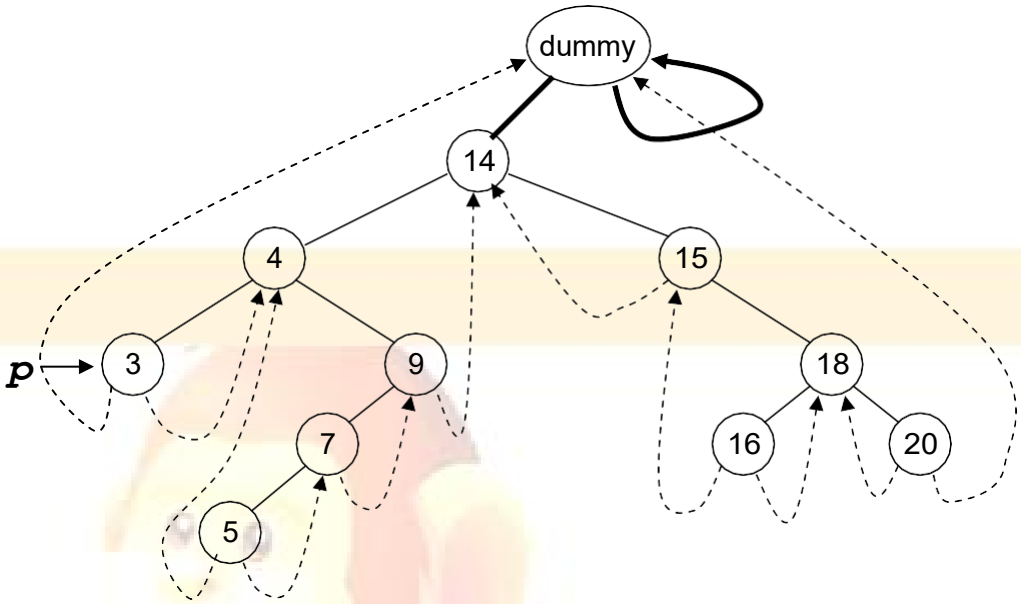
We are not passing it the root of the tree but the *dummy* node. Now we will get the correct values and see in the diagrams below that *p* is now moving in the right direction. Let's try to understand this with the help of diagrams.

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

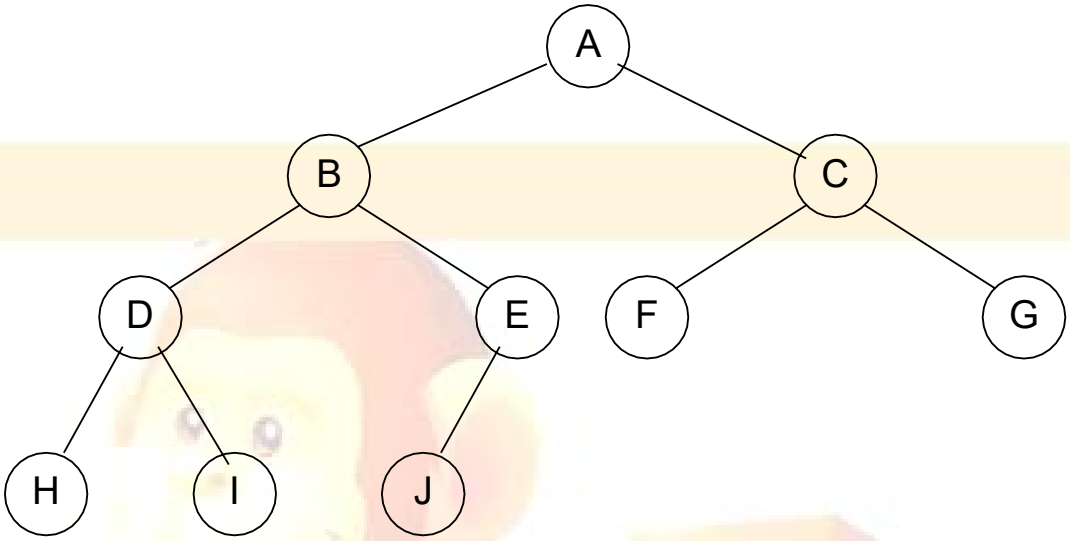
We have earlier discussed the properties of the binary trees besides talking about the internal and external nodes' theorem. Now we will discuss another property of binary trees that is related to its storage before dilating upon the complete binary tree and the heap abstract data type.

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:



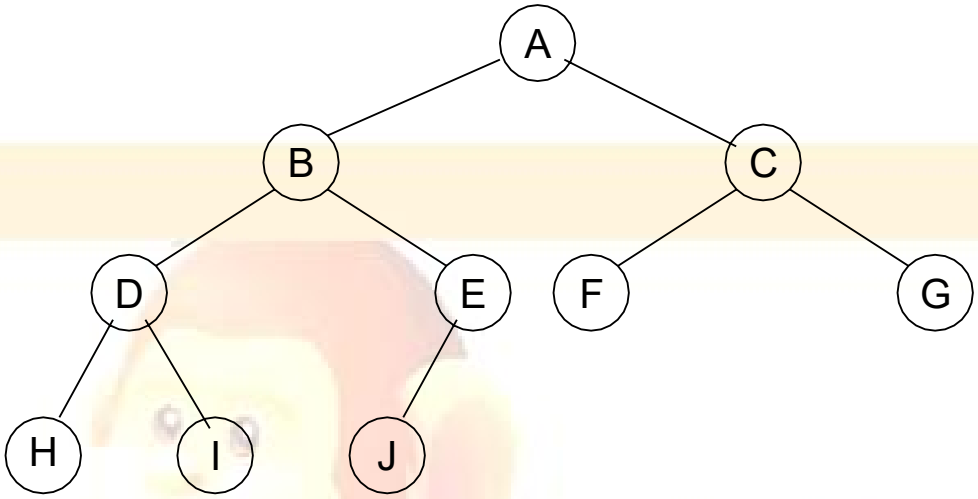
In the above tree, we have nodes as A, B, C, D, E, F, G, H, I, J. The node D has two children at the lowest level whereas node E has only left child at the lowest level that is J. The right child of the node E is missing. Similarly node F and G also lack child nodes. This is a complete binary tree according 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 $\log_2 N$ 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.

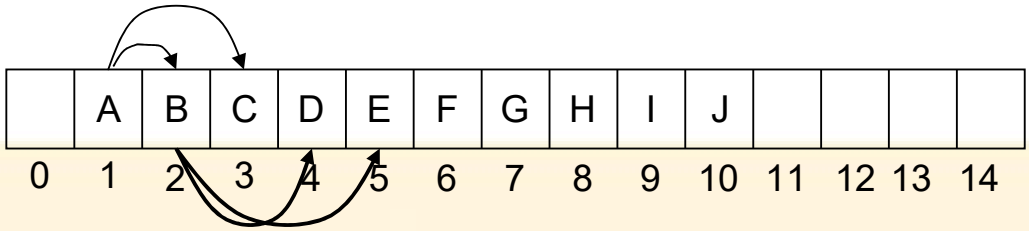
The root node of this tree is *A* and the left and right children of *A* are *B* and *C*. Now look at the array. While storing elements in the array, we follow a rule given below:

- For any array element at position *i*, the left child is at 2*i*, the right child is at (2*i* +1) and the parent is at 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.

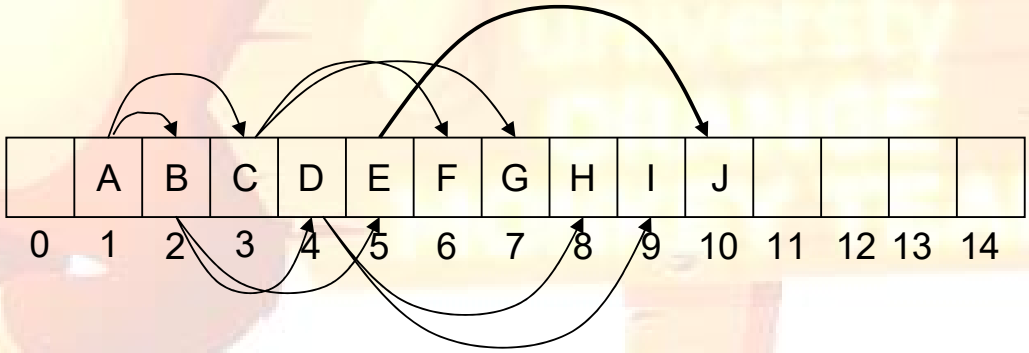
Lets see this formula on the above array. We have *A* at the first position and it has two children *B* and *C*. According to the formula the *B* will be at the *2i* i.e. 2nd position and C will be at *2i+1* i.e. 3rd position. Take the 2nd element i.e. *B*, it has two children *D* and *E*. The position of *B* is 2 i.e. the value of *i* is 2. Its left child *D* will be at positon *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:

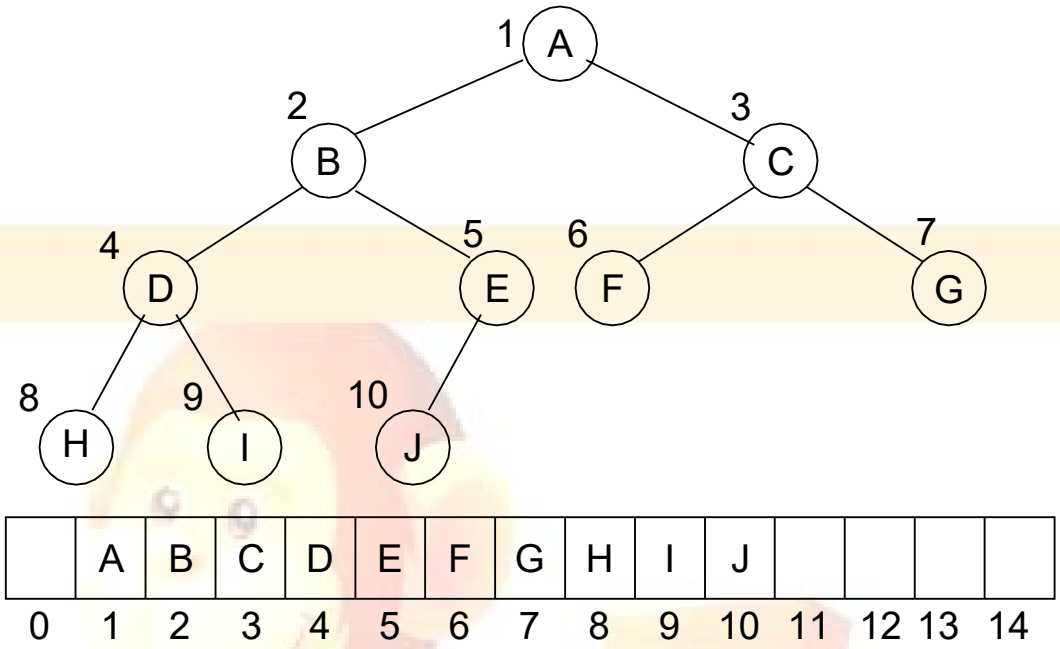


If we want to keep the tree's data in the array, the children of *B* should be at the position 4 and 5. This is true. We can apply this formula on the remaining nodes also. Now you have understood how to store tree's data in an array. In one respect, we are using pointers here. These are not C++ pointers. 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 *ith* 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 *ith* 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 *ith* position, its parent will be at *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 *floor(10/2)* i.e. 5 which is true. As the node *I* is at position 9, its parent should be at *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 *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.

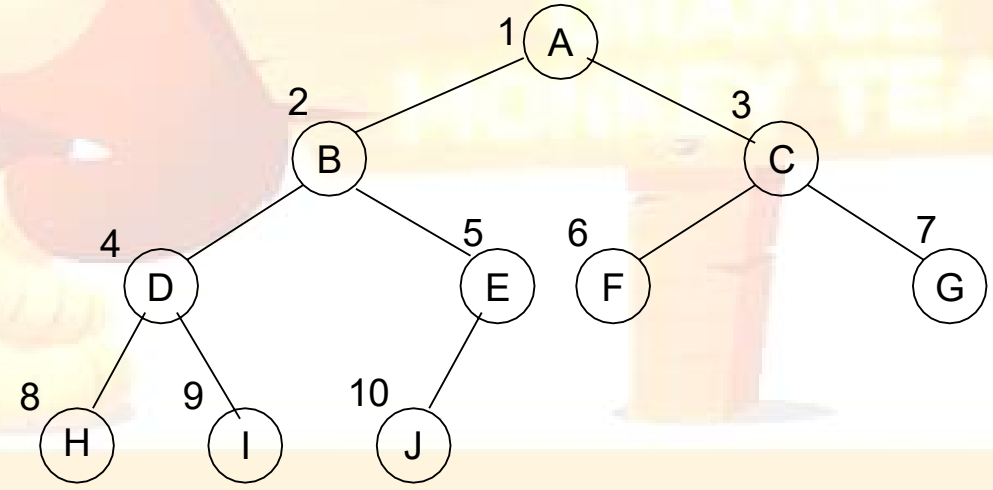
From the above discussion we note three things. 1) We have a complete binary tree, which stores some information. It may or may not be a binary search tree. This tree can be stored in an array. We use *2i* and *2i+1* indexing scheme to put the nodes in the array. Now we can apply the algorithms of tree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



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

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.

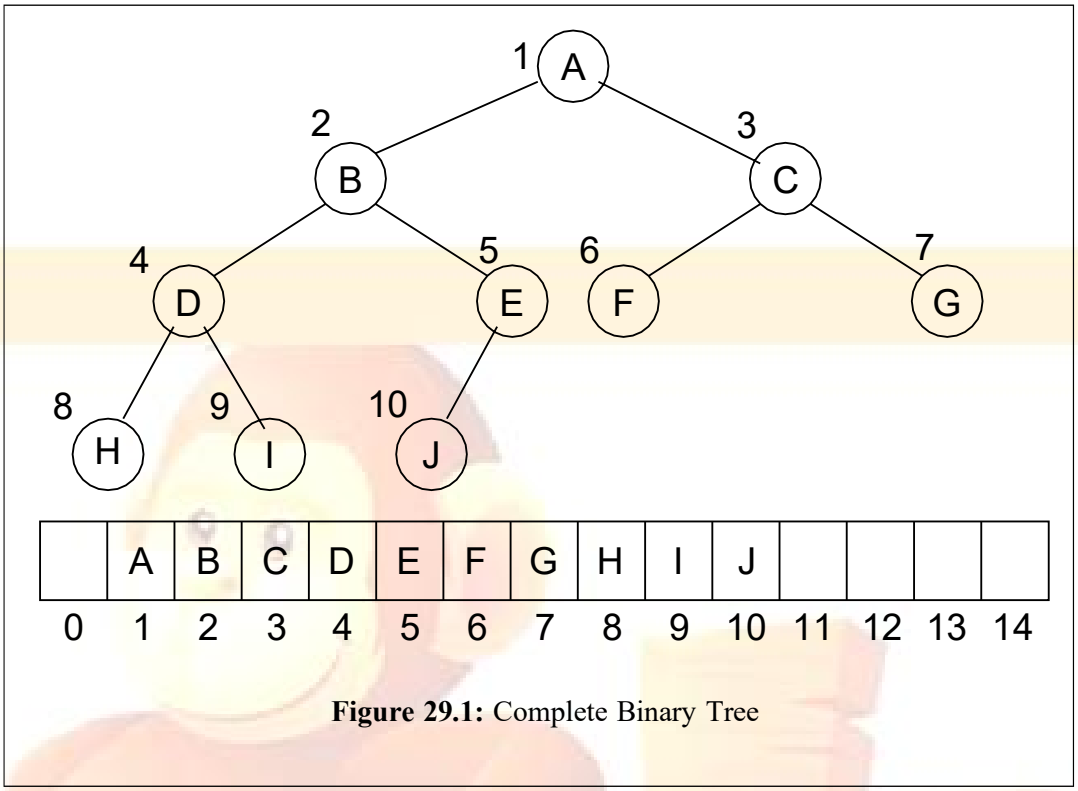


Figure 29.1: Complete Binary Tree

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).

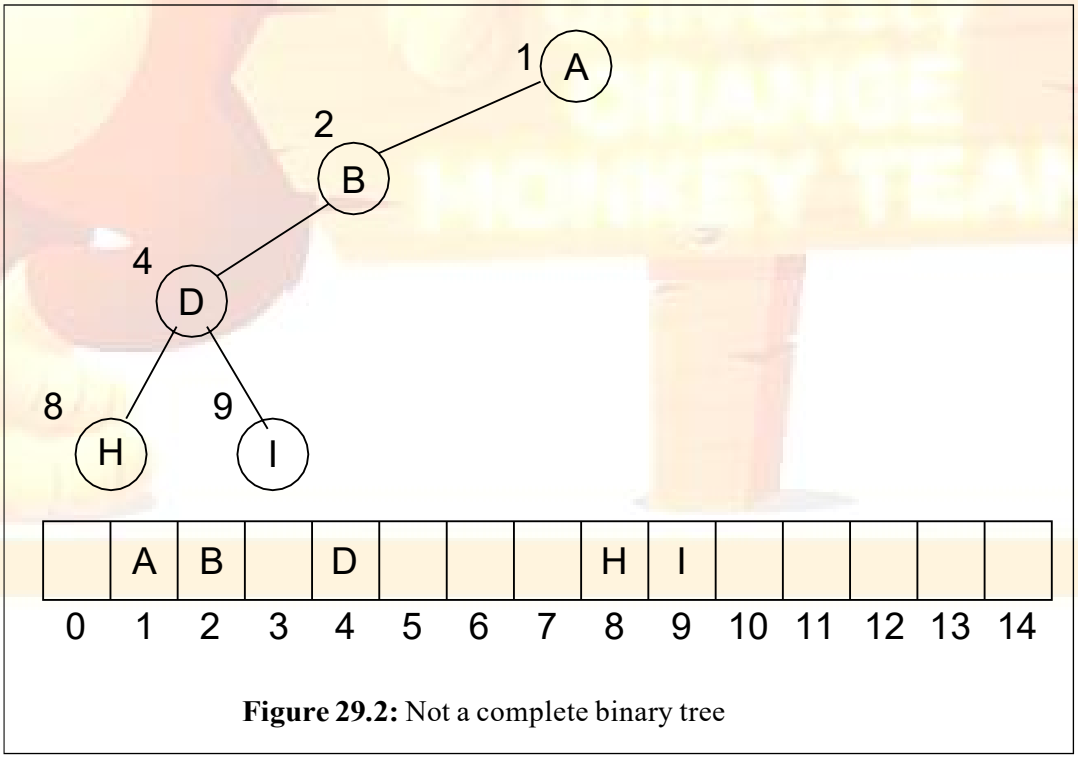


Figure 29.2: Not a complete binary tree

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.

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.

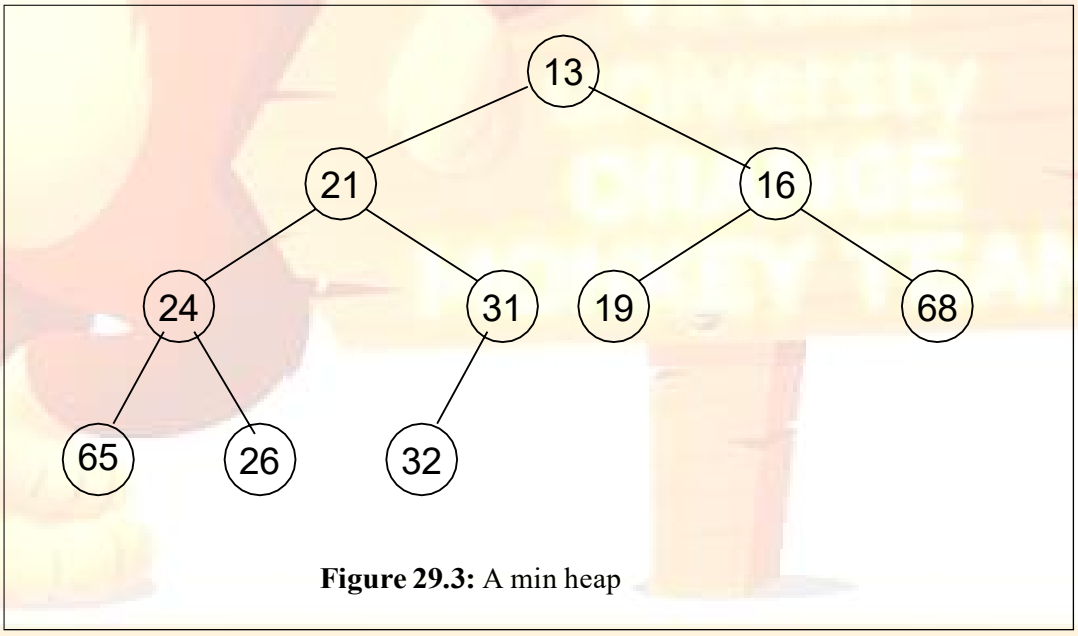


Figure 29.3: A min heap

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.

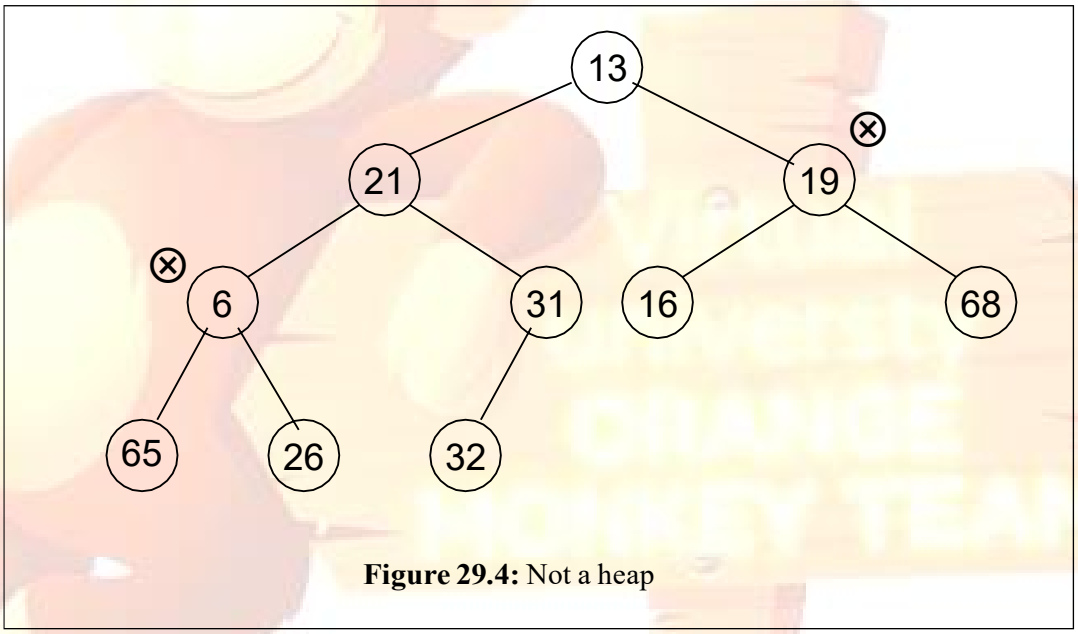


Figure 29.4: Not a heap

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.

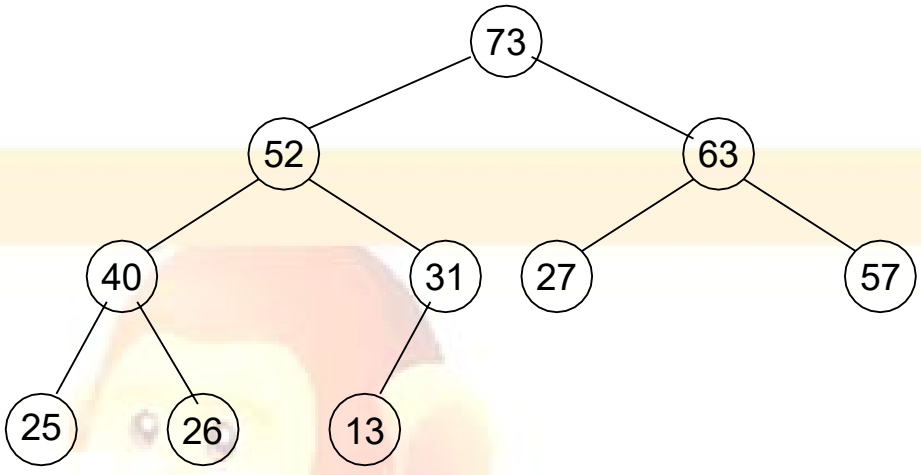


Figure 29.5: A max heap

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.

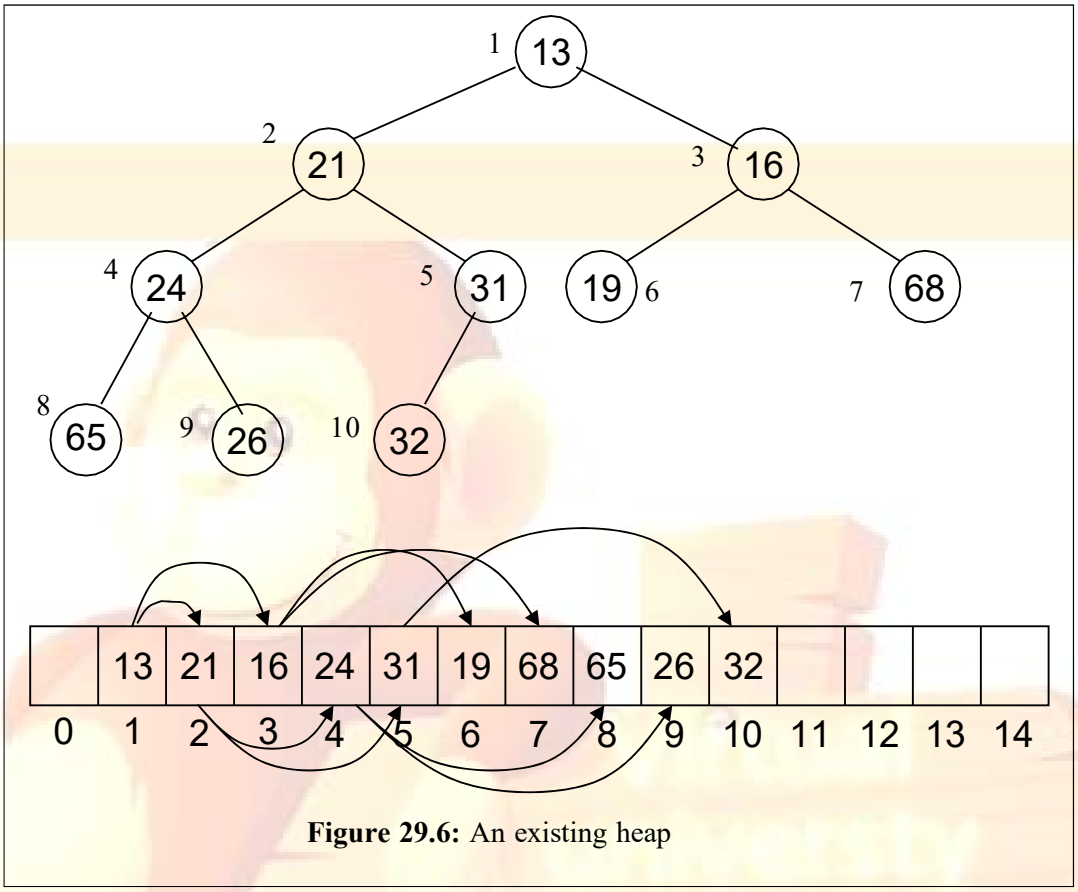


Figure 29.6: An existing heap

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.

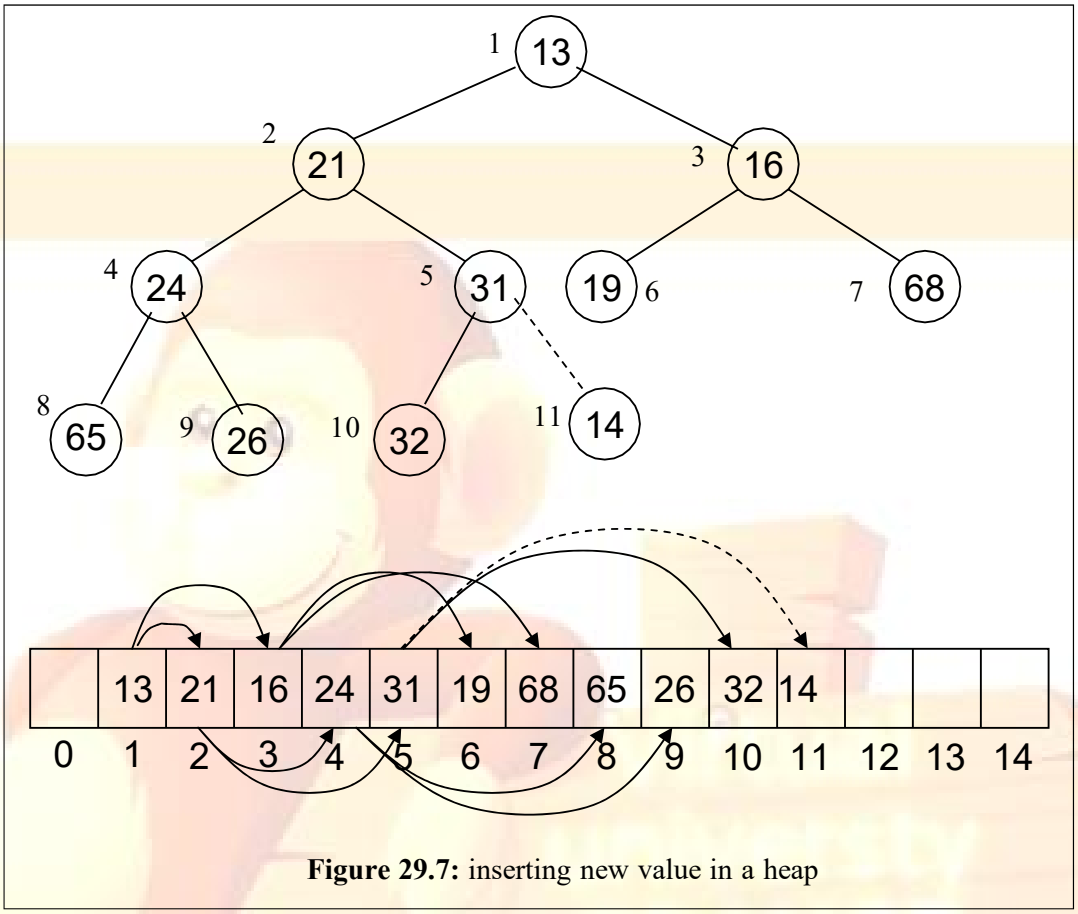


Figure 29.7: inserting new value in a heap

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.

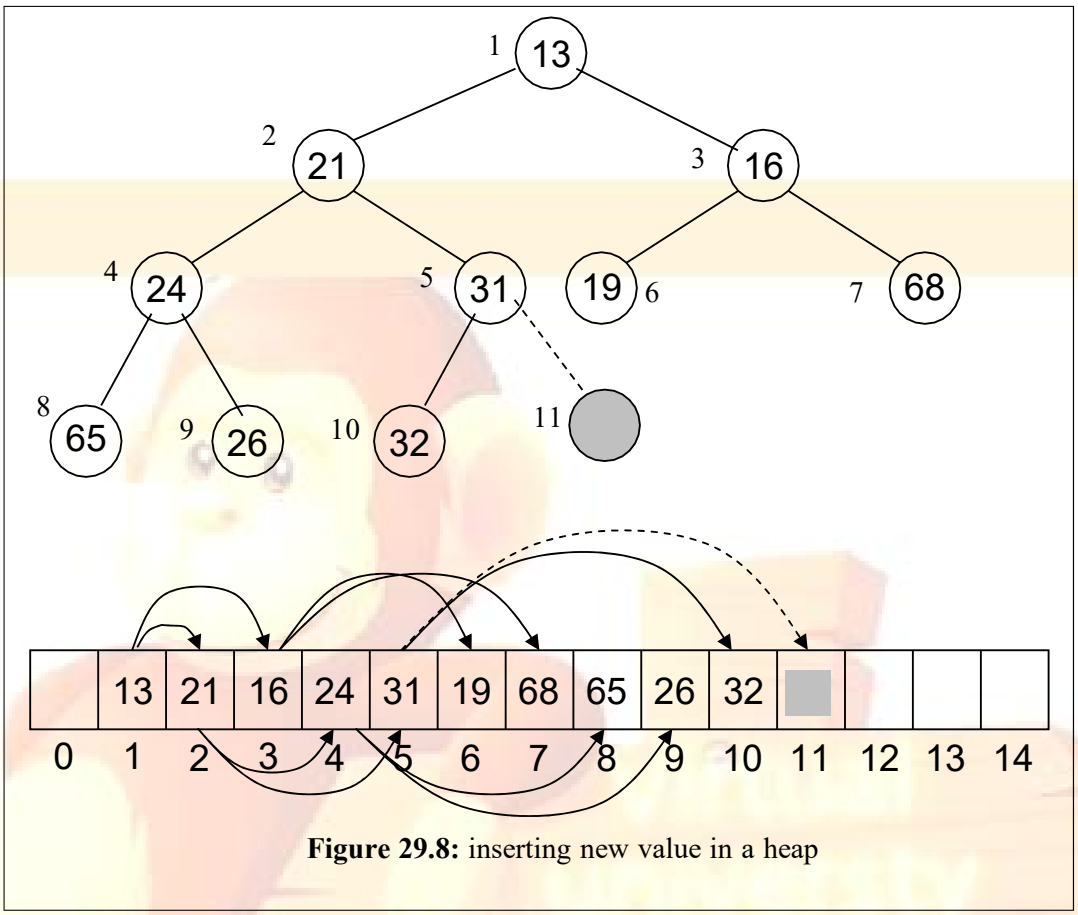
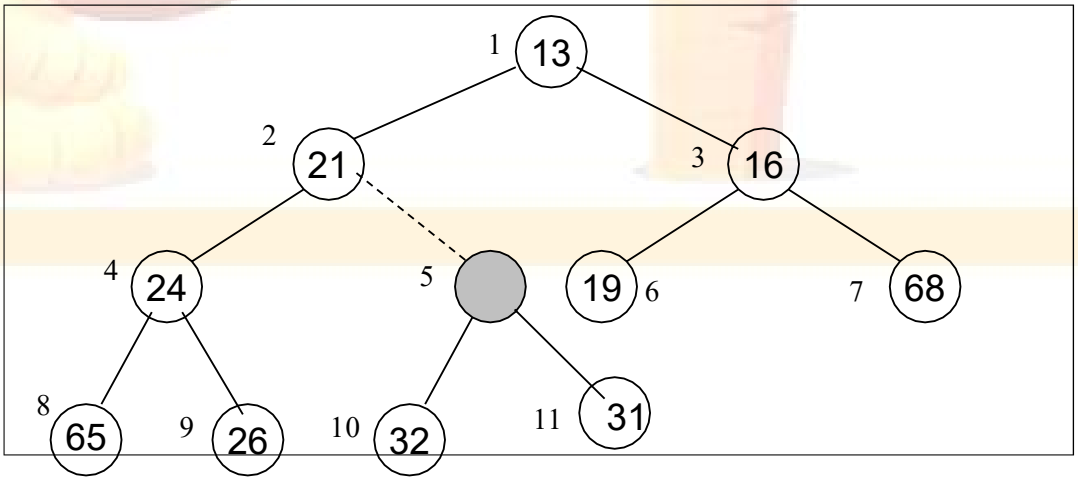


Figure 29.8: inserting new value in a heap

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.



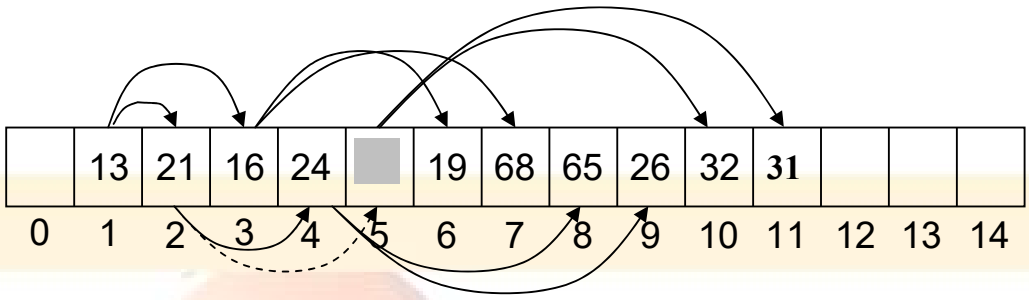


Figure 29.9: inserting new value in a heap

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.

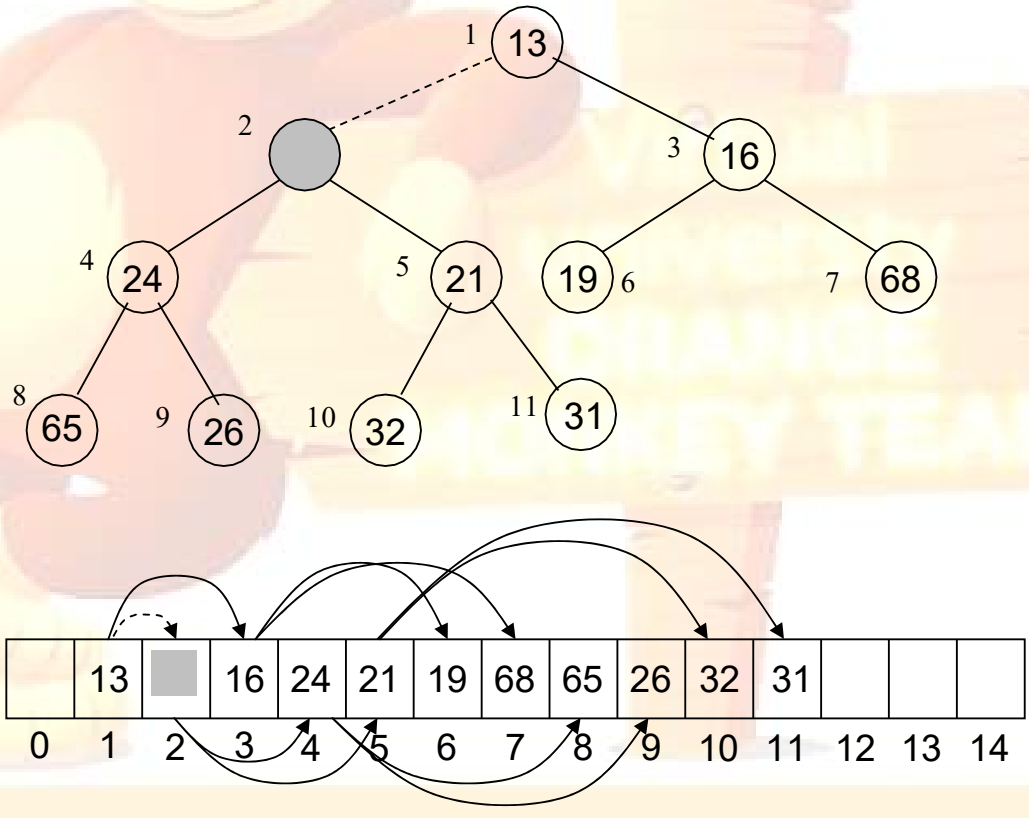


Figure 29.10: inserting new value in a heap

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.

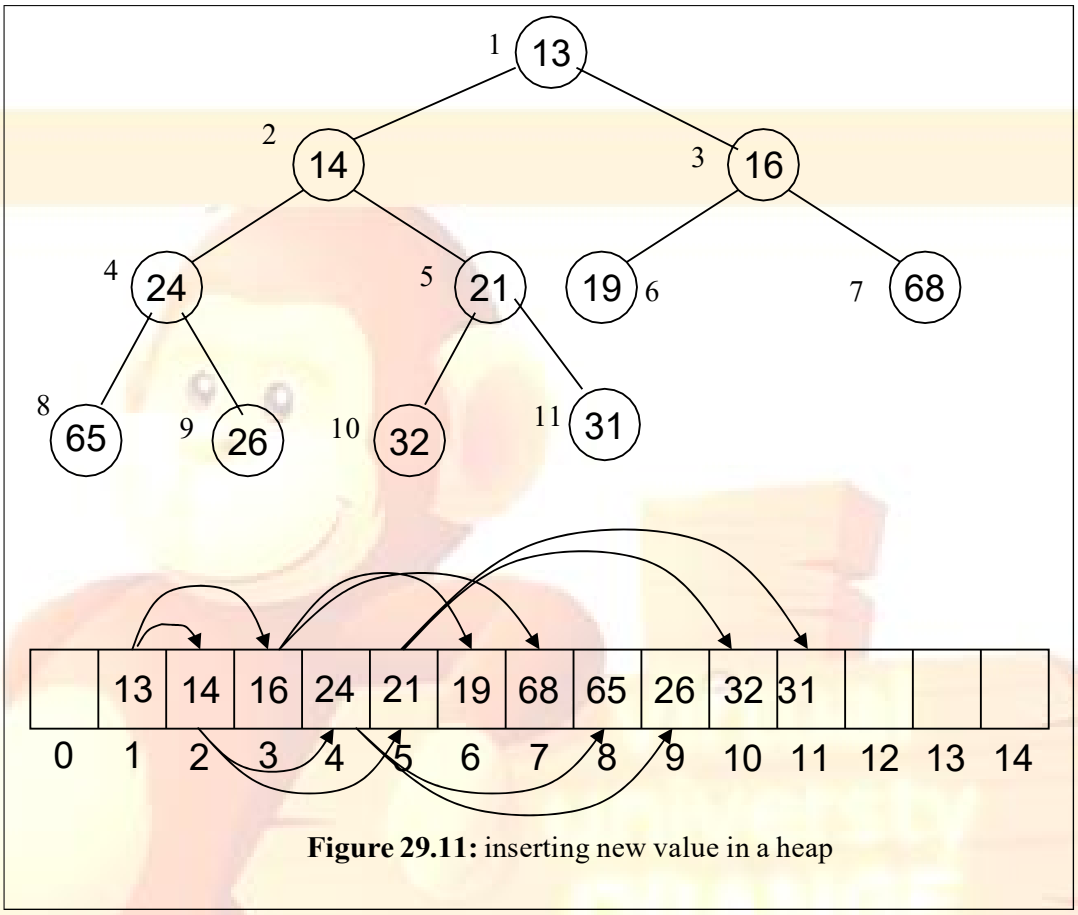


Figure 29.11: inserting new value in a heap

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 *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.

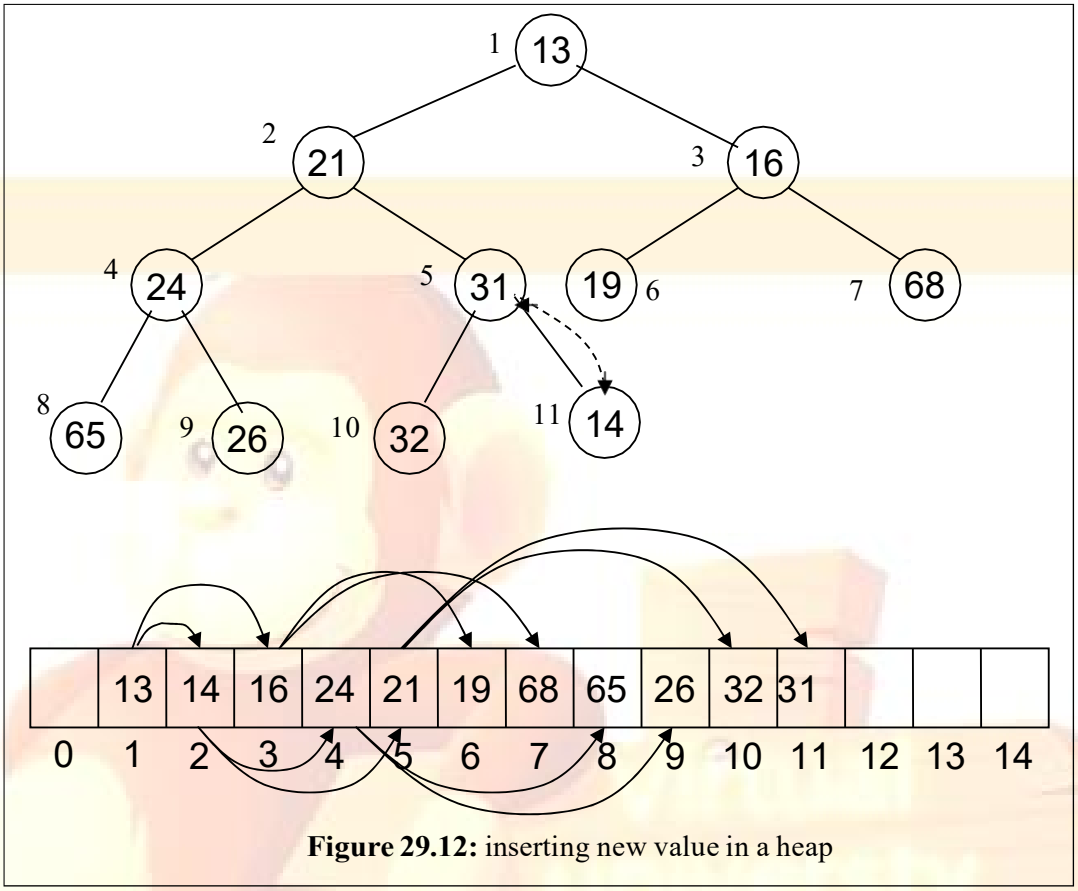
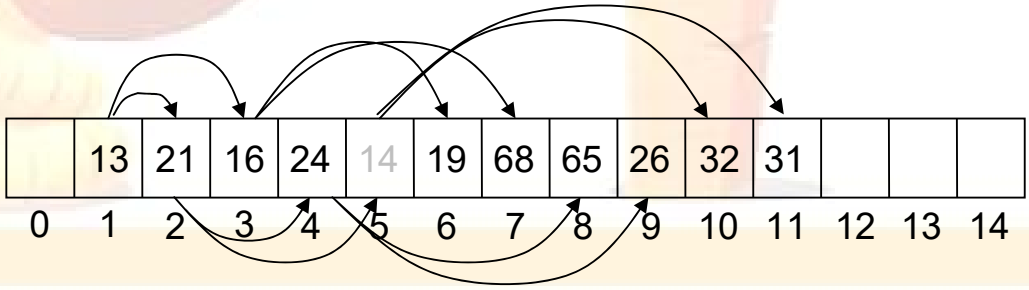


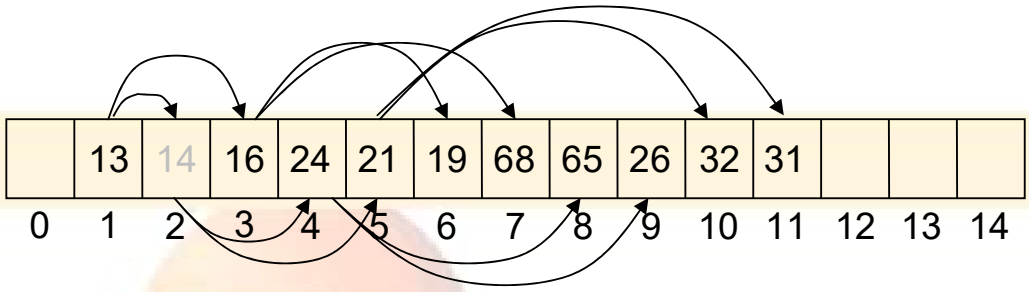
Figure 29.12: inserting new value in a heap

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 31due 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 *floor (i / 2)*. The parent node of 14 will be at position *floor (5/2)* that is position 2. We can see that the node at position 2 is 21. Thus 21 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 21 and 21 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.

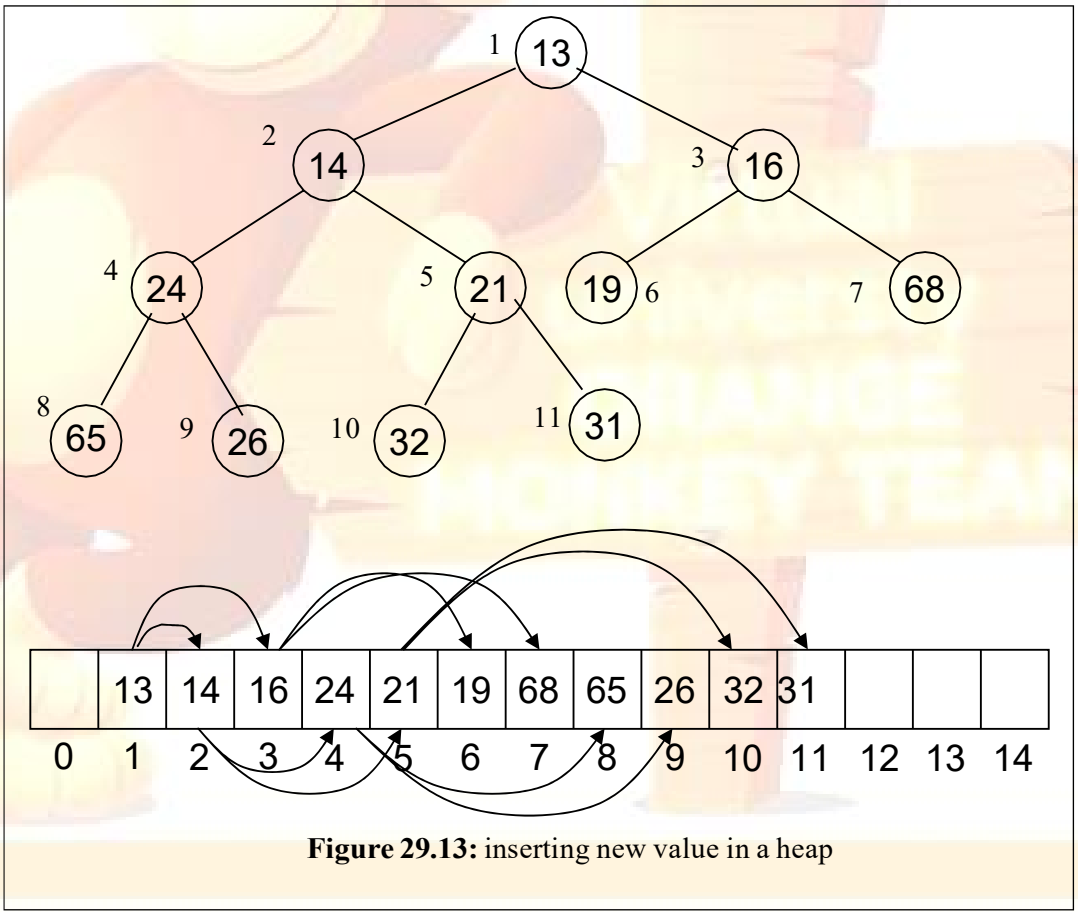


Figure 29.13: inserting new value in a heap

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 x 2 that is 12. This is shown in the following figure.

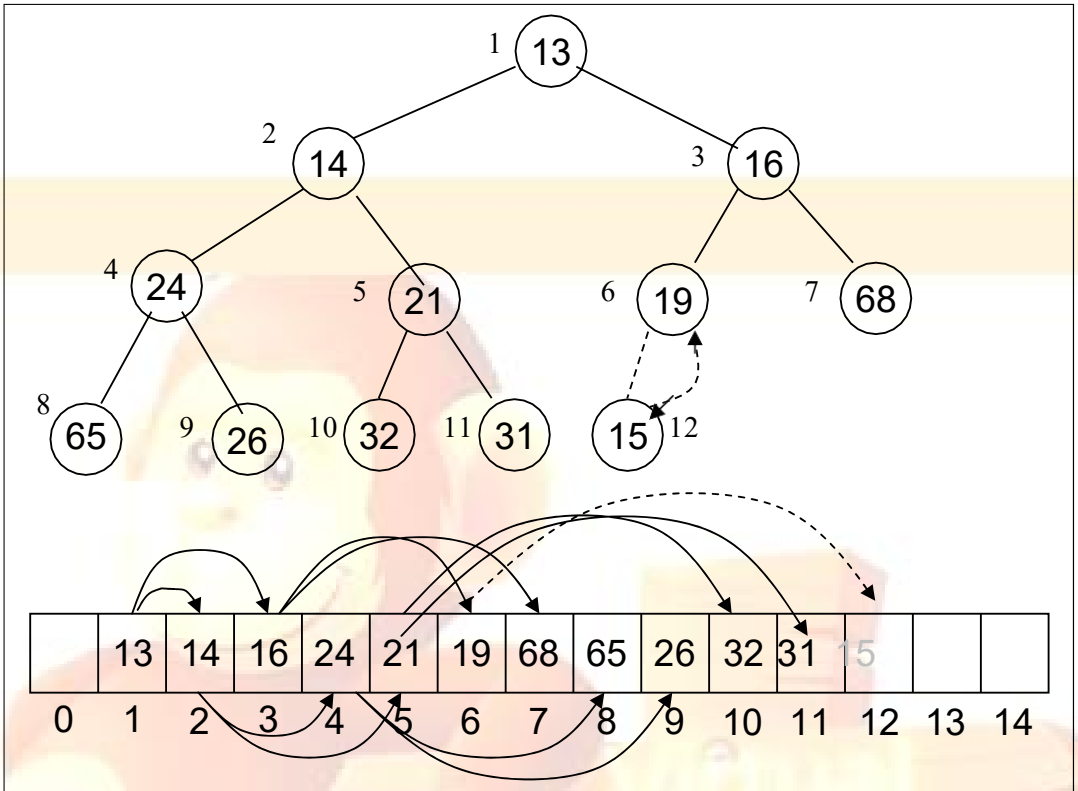
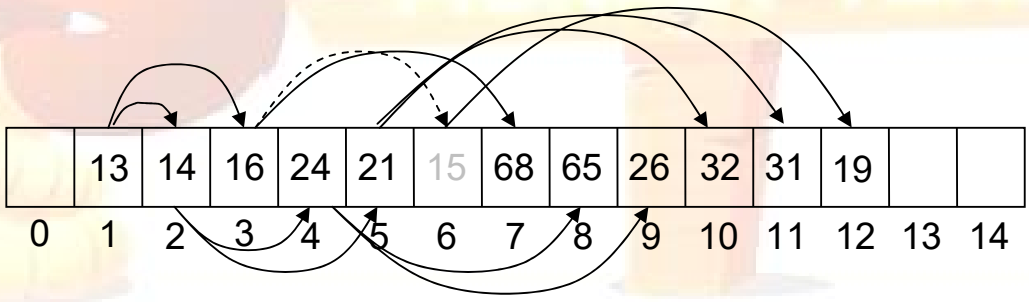


Figure 29.14: inserting new value in a heap

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.

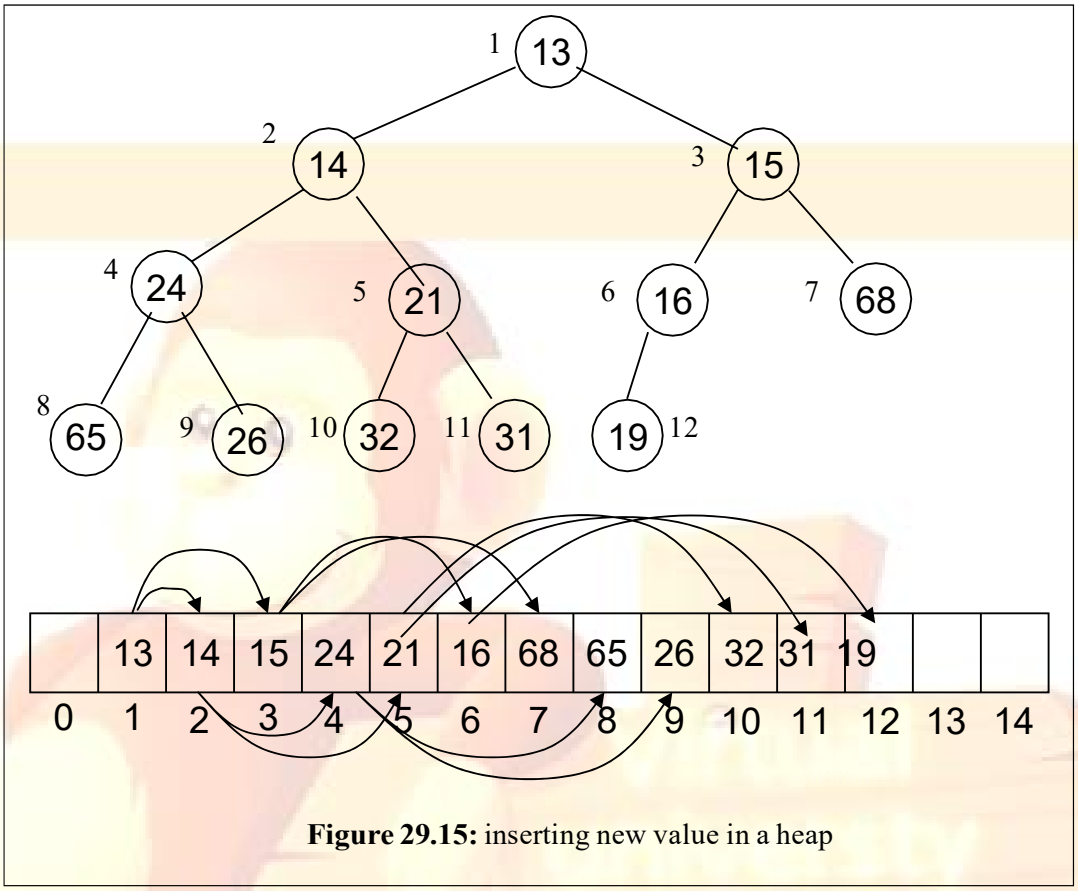


Figure 29.15: inserting new value in a heap

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.

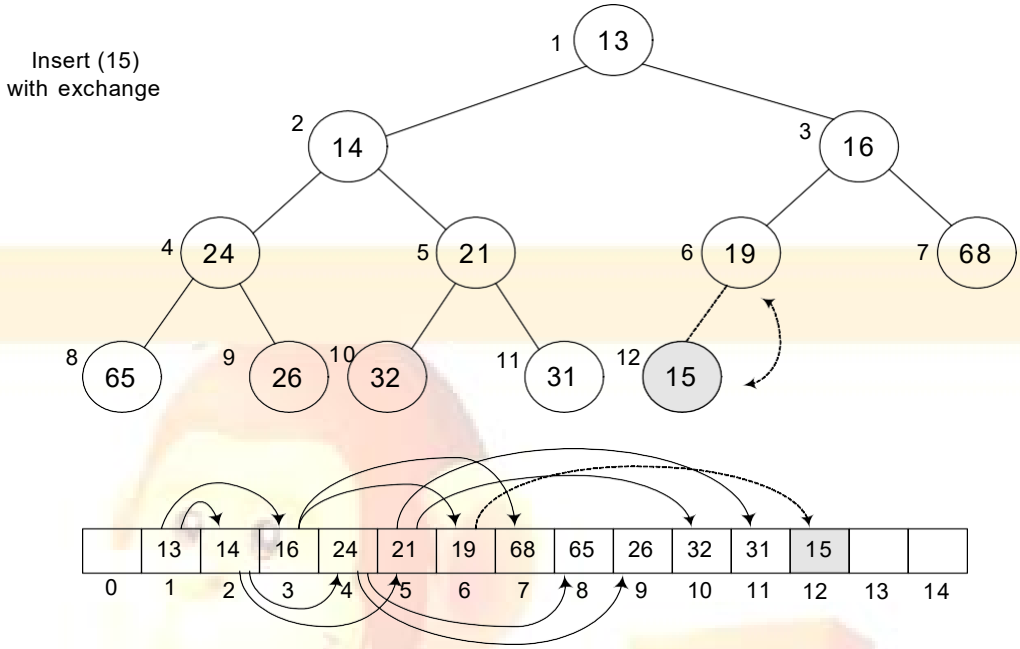


Fig 30.1

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 = 12th position node or the value at 12th 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.

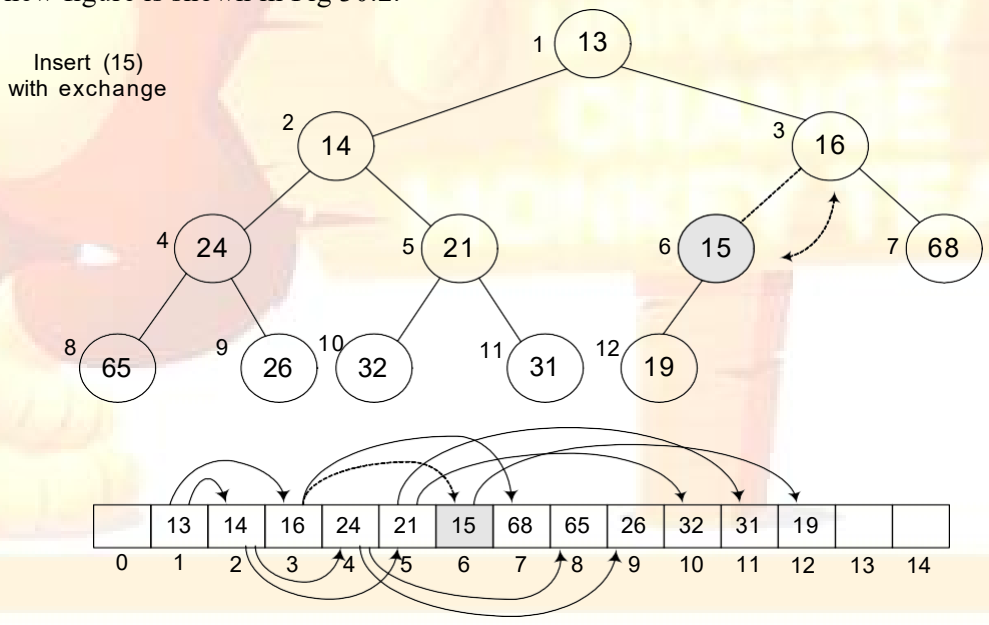


Fig 30.2

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.

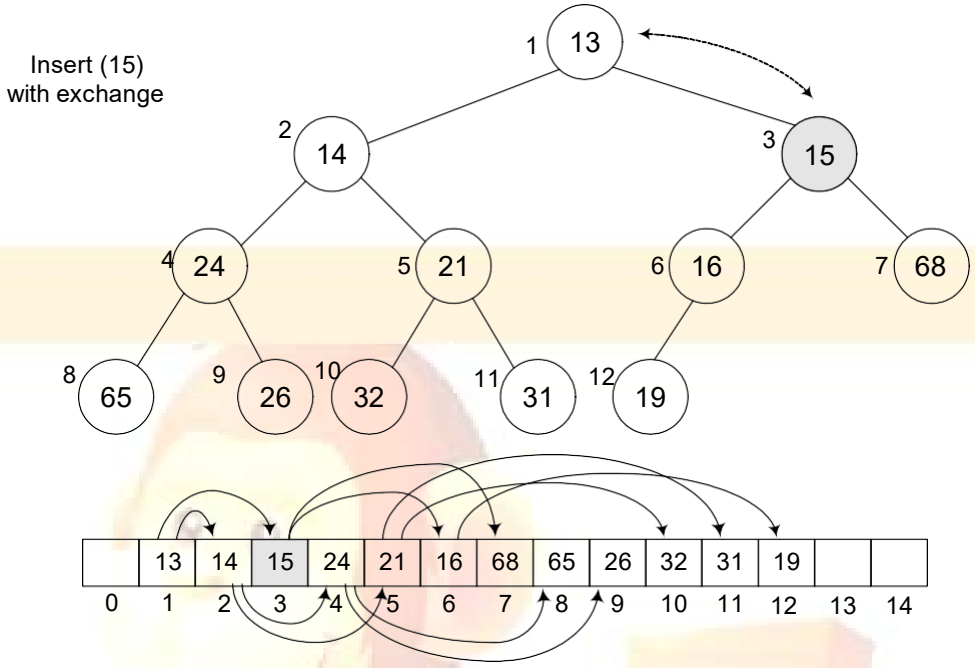


Fig 30.3

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.

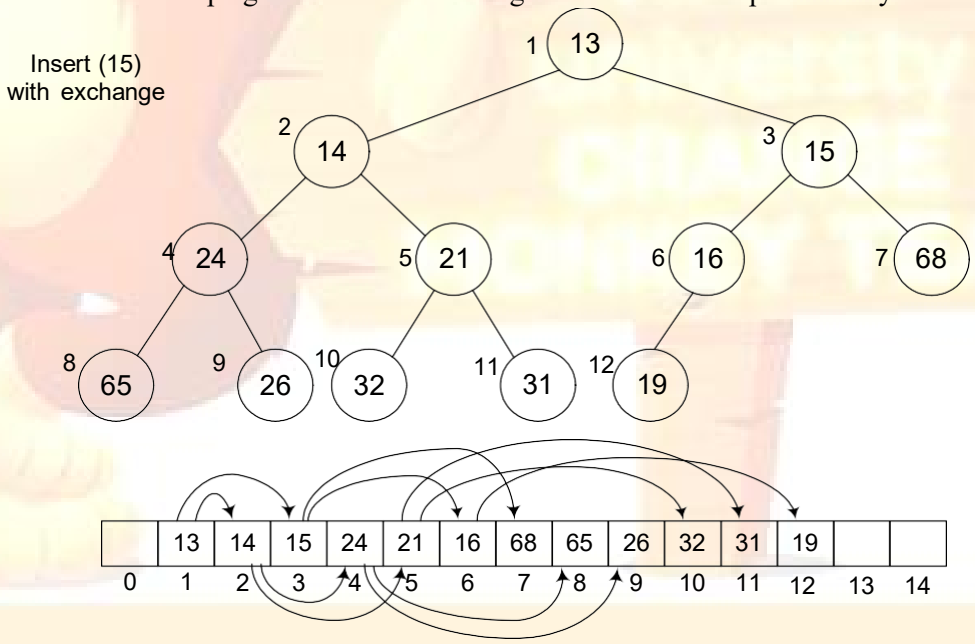


Fig 30.4

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.

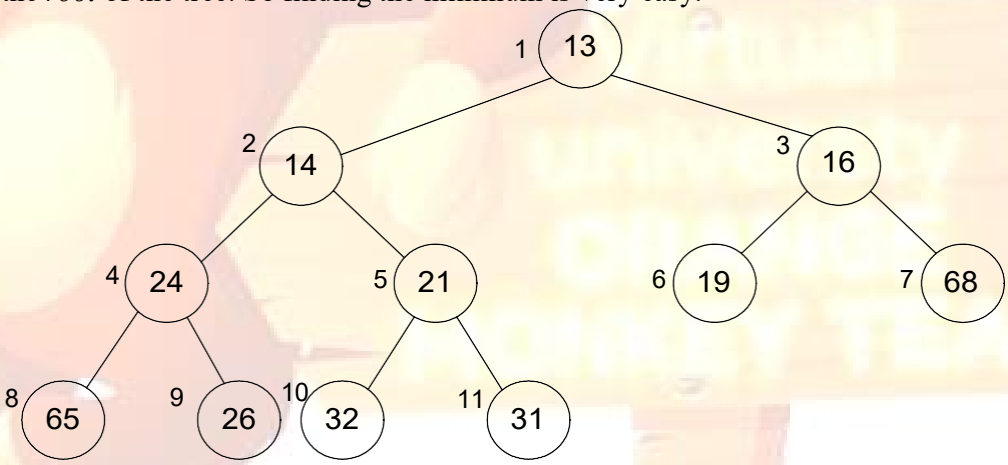


Fig 30.5

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.*

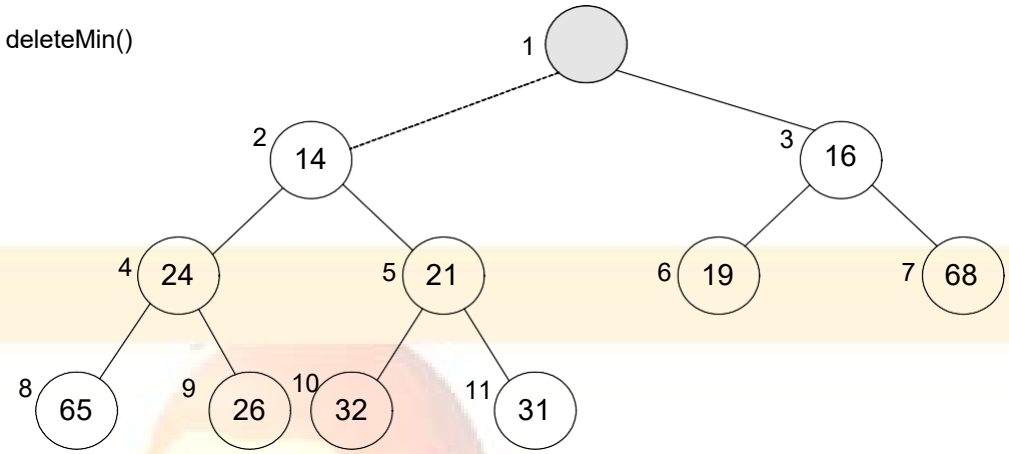


Fig 30.6

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.

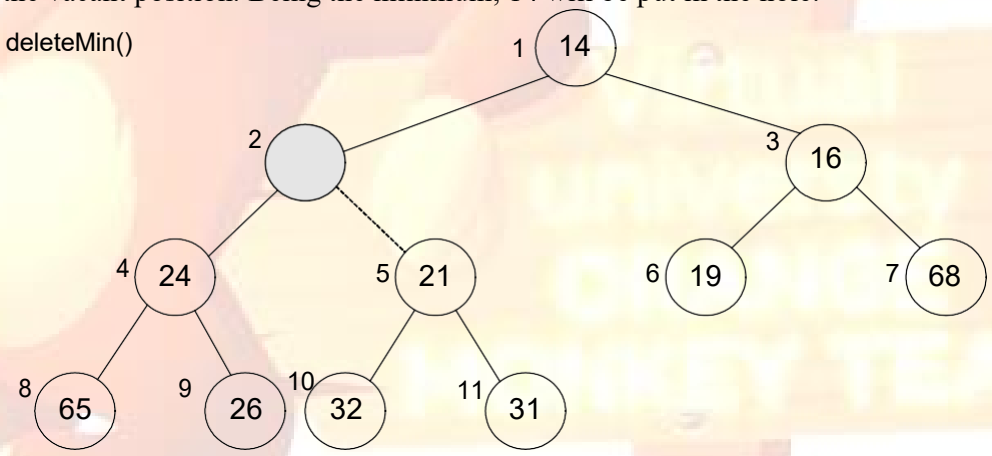


Fig 30.7

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.)

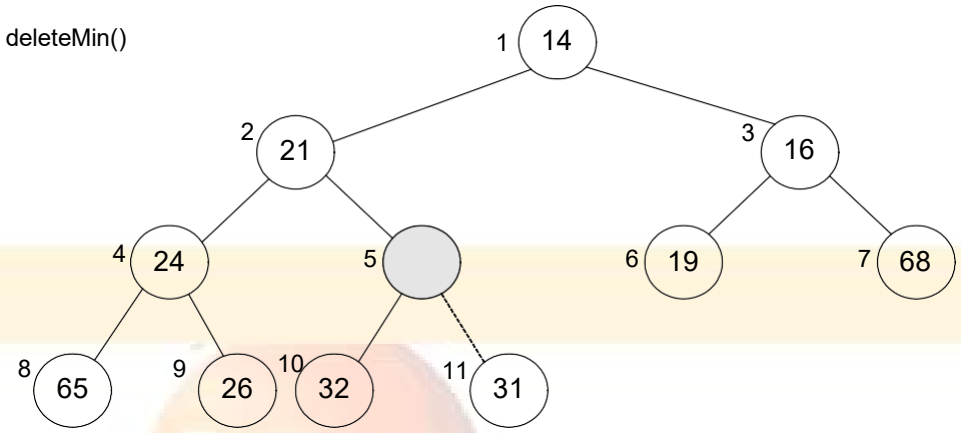


Fig 30.8

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.

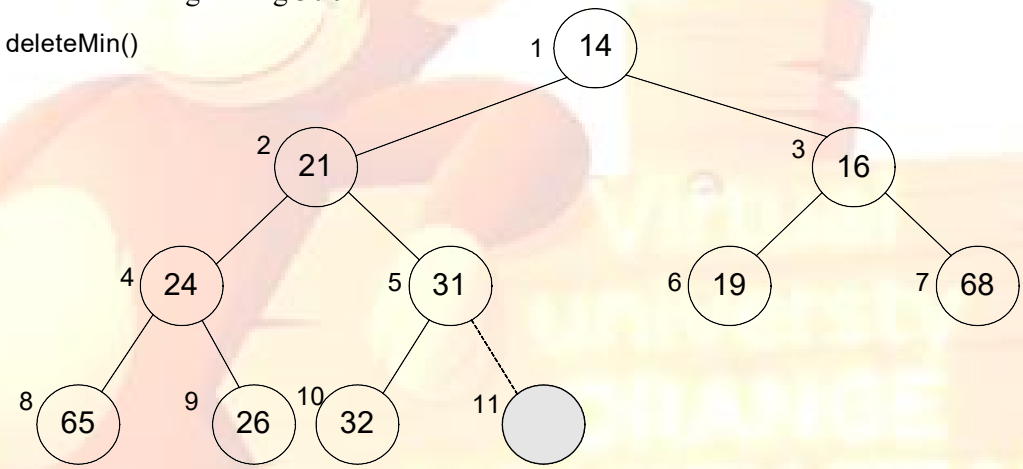


Fig 30.9

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

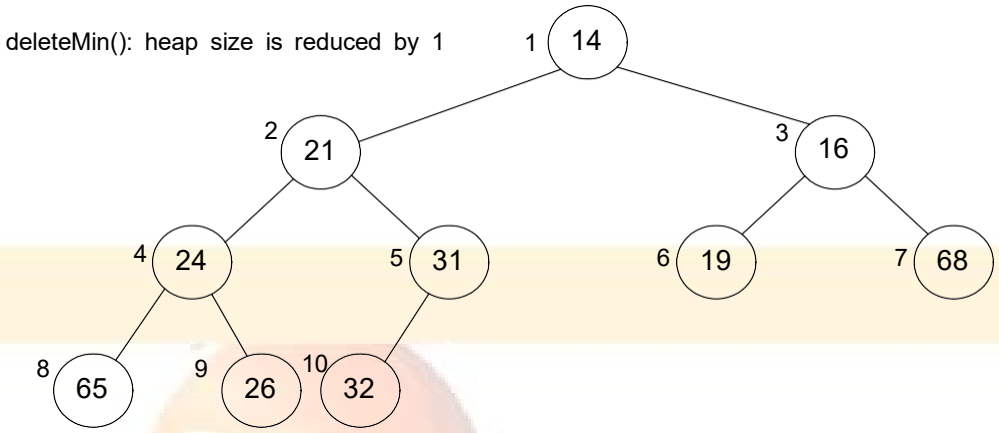


Fig 30.10

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 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 l 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_2N (if new key percolates all the way up to the root).*
- *The worst time for building a heap of N keys could be $Nlog_2N$.*
- *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 built a heap that will be 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.

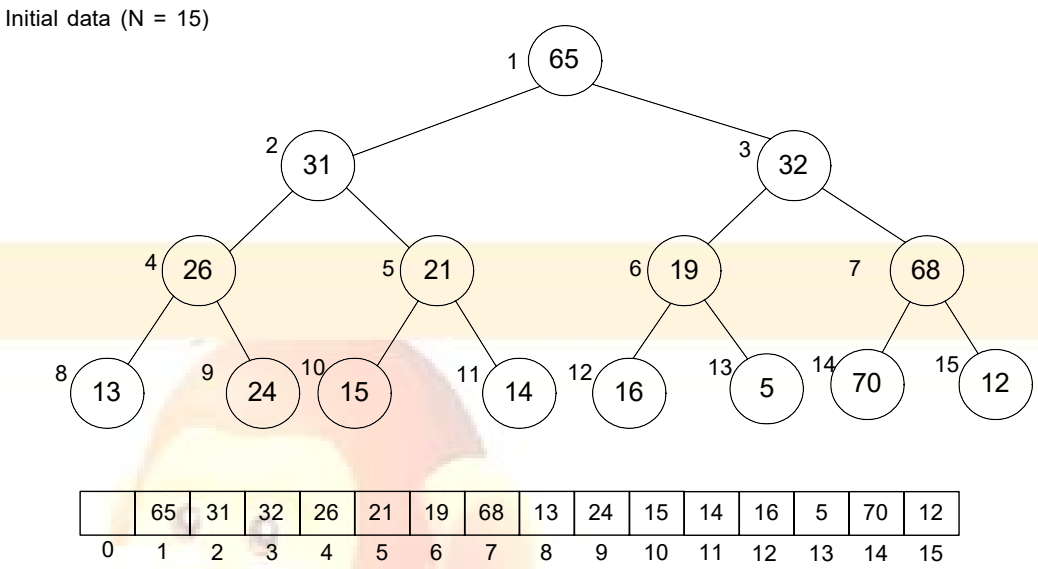


Fig 30.12

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 pe*rcolateDown*(*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.

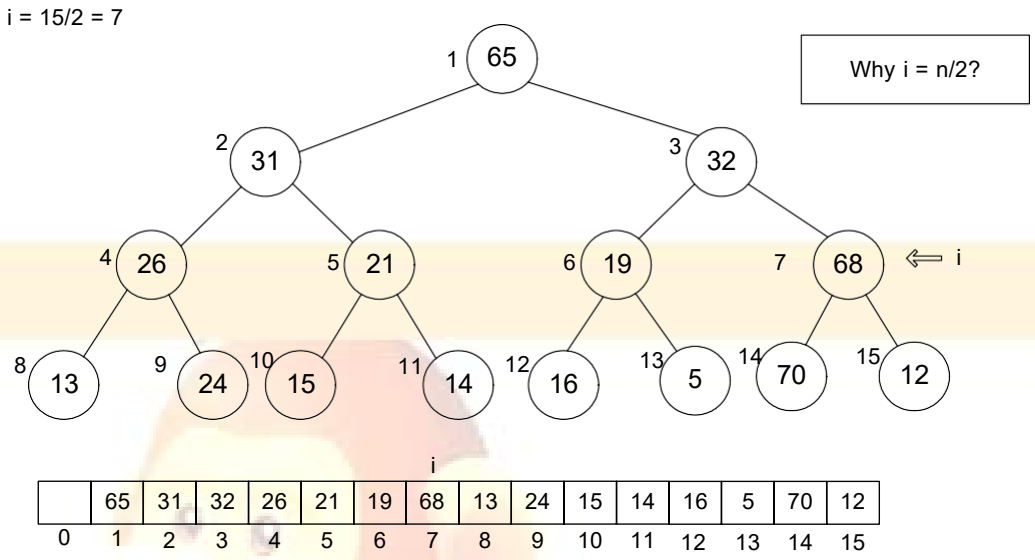


Fig 30.13

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.

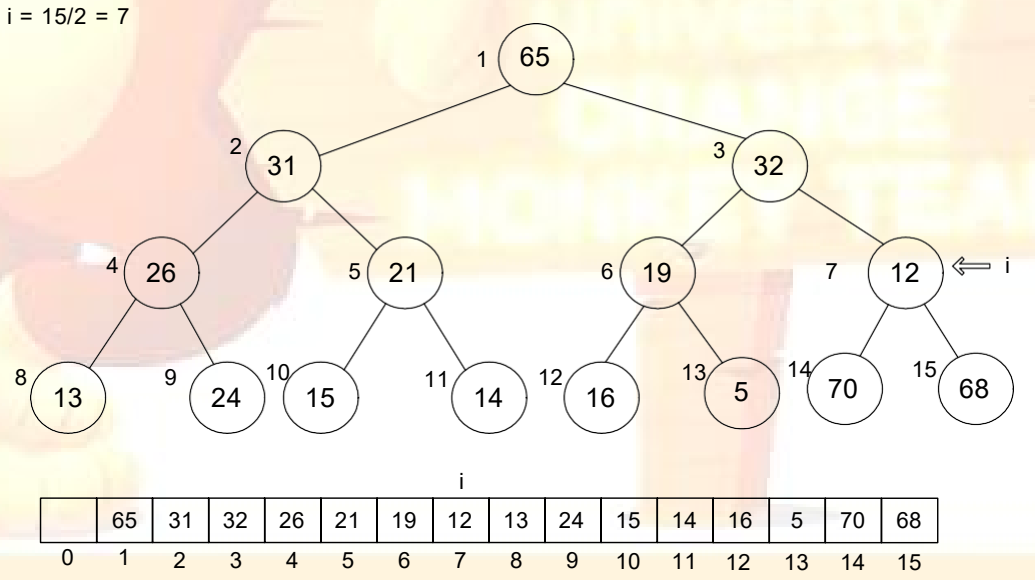


Fig 30.14

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.

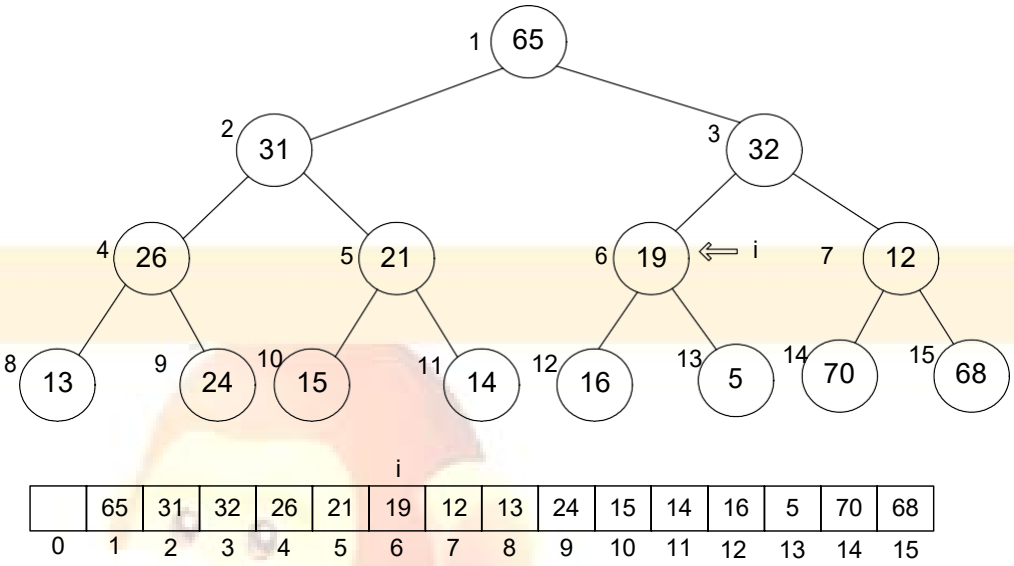


Fig 30.15

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.

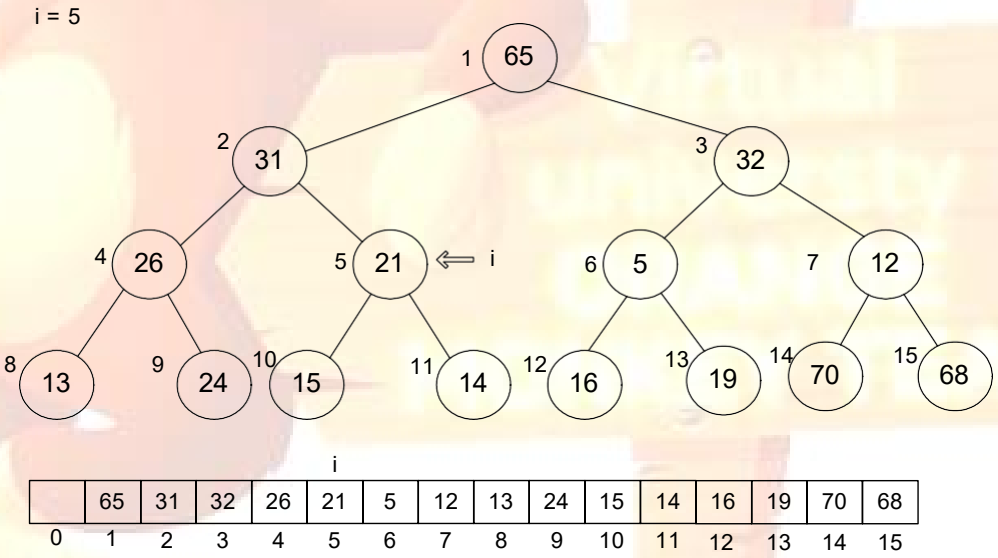


Fig 30.16

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

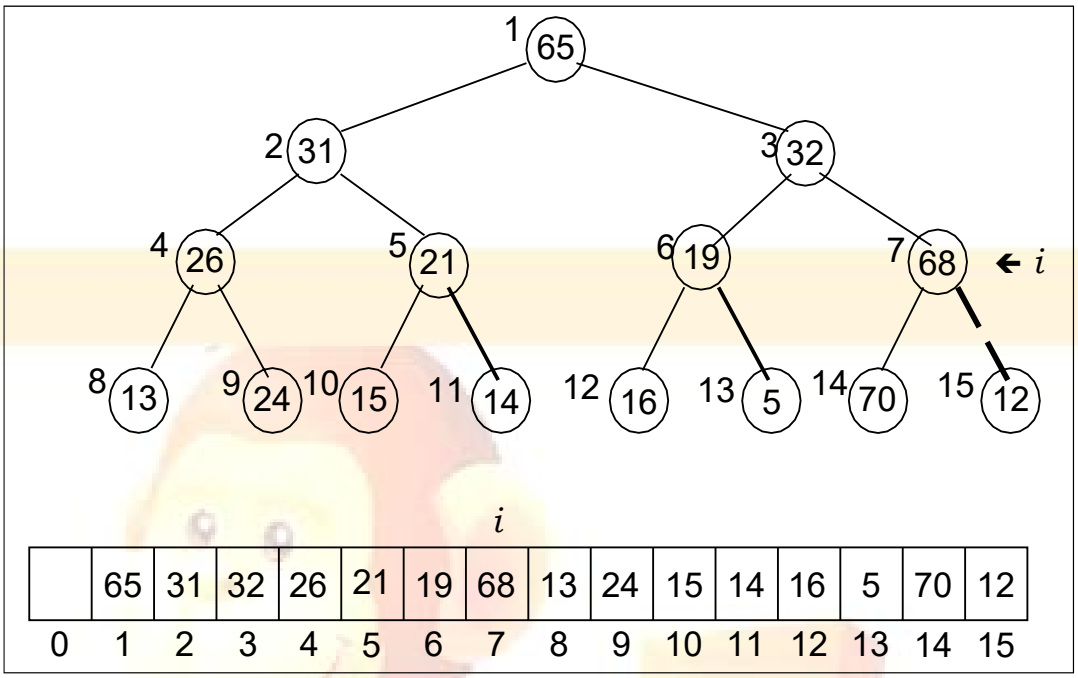
In the previous lecture, we discussed about the *BuildHeap* method of heap abstract data structure. In this handout, you are going to know why a programmer adopts this method. Suppose we have some data that can be numbers or characters or in some other form and want to build a *min-Heap* or *max-Heap* 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 *NlogN* algorithm i.e. the time required for this will be proportional to *NlogN*. 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 *NlogN*.

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.

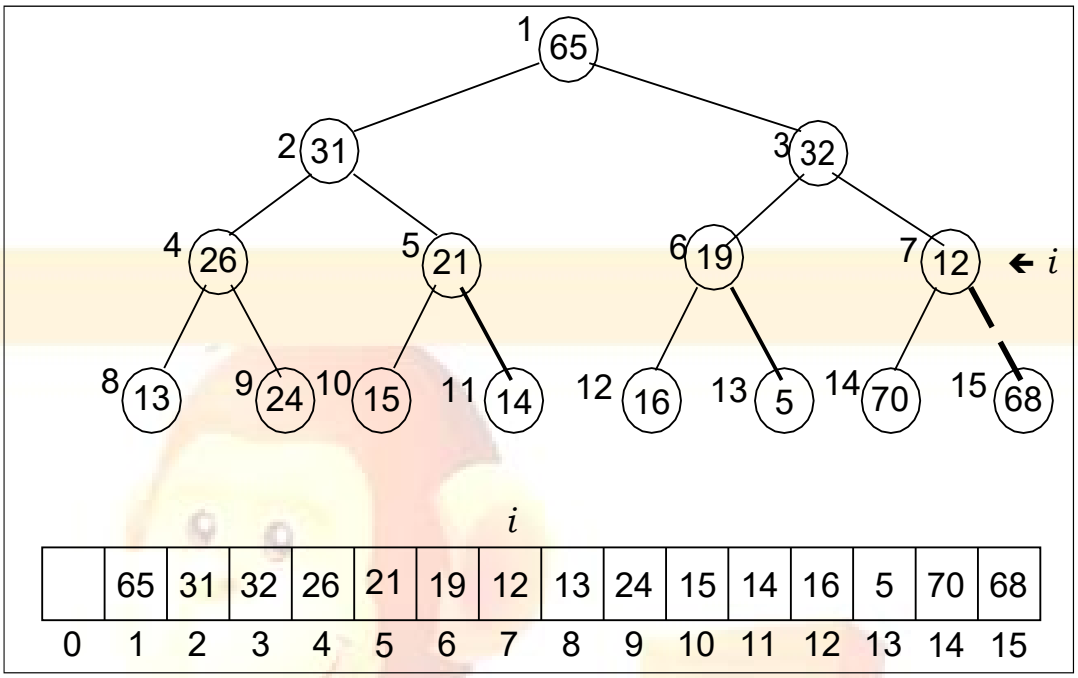
```
for ( i = N/2; i > 0; i-- )
        percolateDown(i);
```

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:

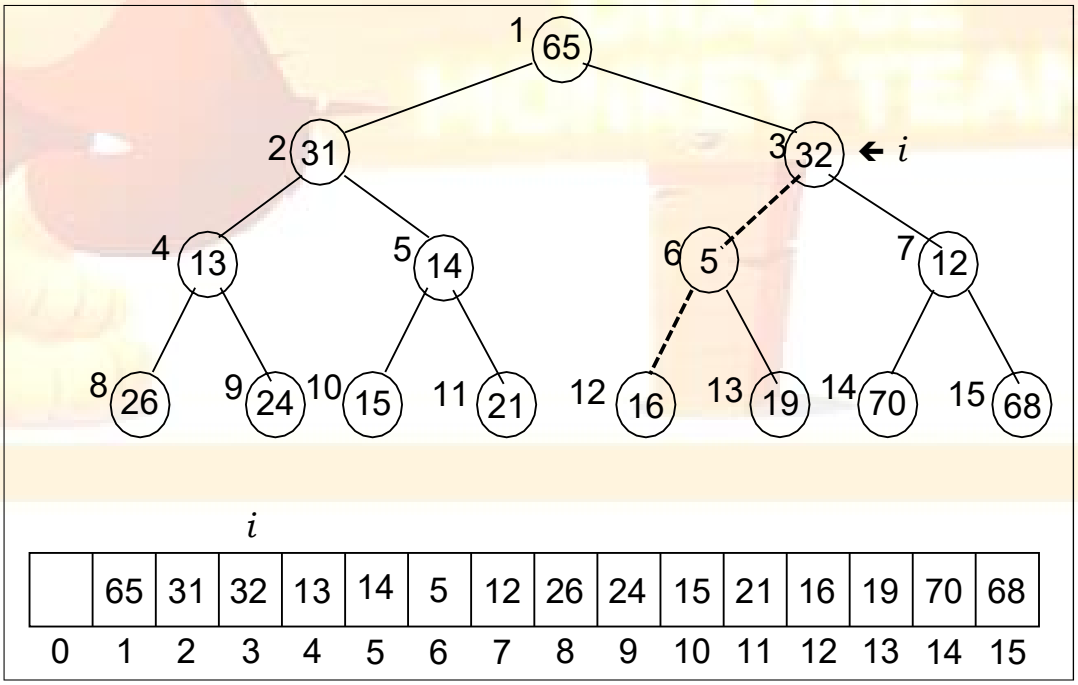


In the above diagram, there is an array containing the elements as 65, 31, 32, 26, 21, 19, 68, 13, 24, 15, 14, 16, 5, 70, 12. We have all the data in this array. The zeroth element of the array is empty. We kept it empty intentionally to apply the *2i*, *2i+1* scheme. We will take this array as a complete 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.



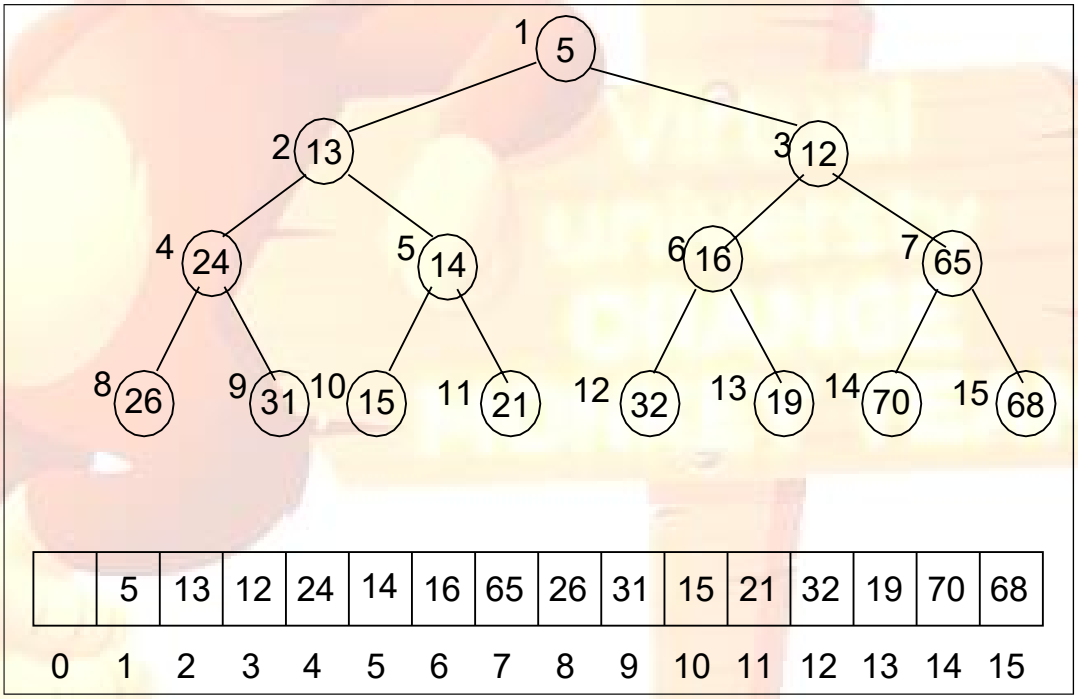
Look at this subtree with 12 as the root node. Its left and right children are smaller than root. This subtree is a minimum heap. Now in the loop, the value of *i* is decreased by 1, coming to the mark of 6. The element at 6th position is 19 that has been passed to *percolateDown* method. It will convert 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 l 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 *NlogN* 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, ə)* and then performing *deleteMin()*. First of all, we will decrease the value of the node by ə 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 ə. 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 ə 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

- perculateDown 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-- ];
```

perculateDown Method

Then we call the *perculateDown* method. This is the same method earlier employed in the build heap process. We passed it the node, say i, form where it starts its function to restore the heap order. Let's look at this *perculateDown* 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 <= 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 *hol*e. 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 *perculateDown* 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 *perculateDown* 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 = 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 perculateDown 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 ans 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.

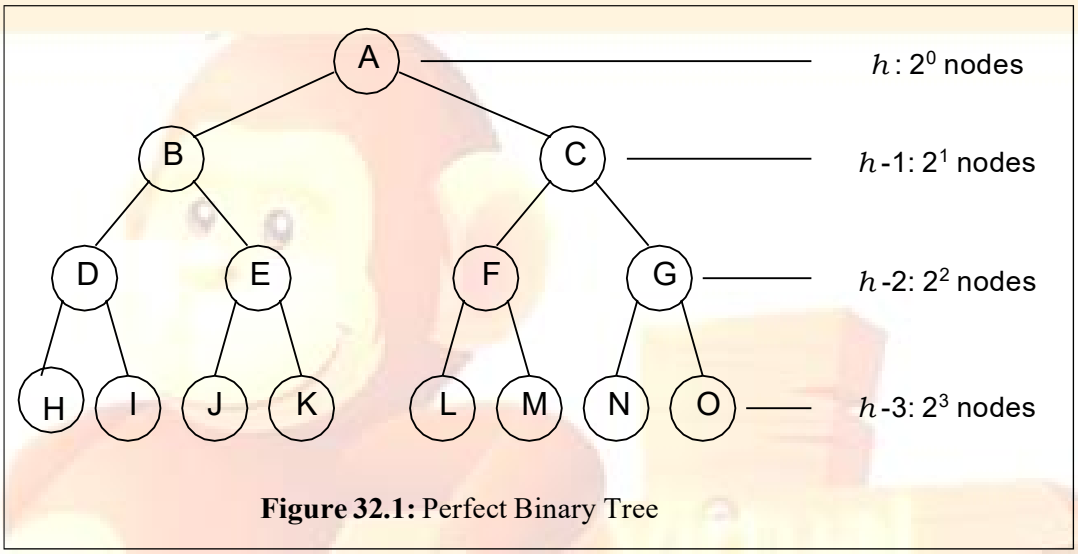


Figure 32.1: Perfect Binary Tree

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

$$S = \sum 2^i (h - i), \text{ for } i = 0 \text{ to } h - 1$$
$$S = h + 2(h-1) + 4(h-2) + 8(h-3) + \ldots + 2^{h-1} \quad (1)$$

Now multiplying by 2 the both sides of equation

$$2S = 2h + 4(h-1) + 8(h-2) + 16(h-3) + \ldots + 2^h \quad (2)$$

Now subtracting the equation 2 from equation 1

$$-S = h - 2 - 4 - 8 - 16 - \ldots - 2^{h-1} - 2^h$$
$$S = -h + 2 + 4 + 8 + 16 + \ldots + 2^{h-1} + 2^h$$
$$S = (2^{h+1} - 1) - (h+1)$$

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.

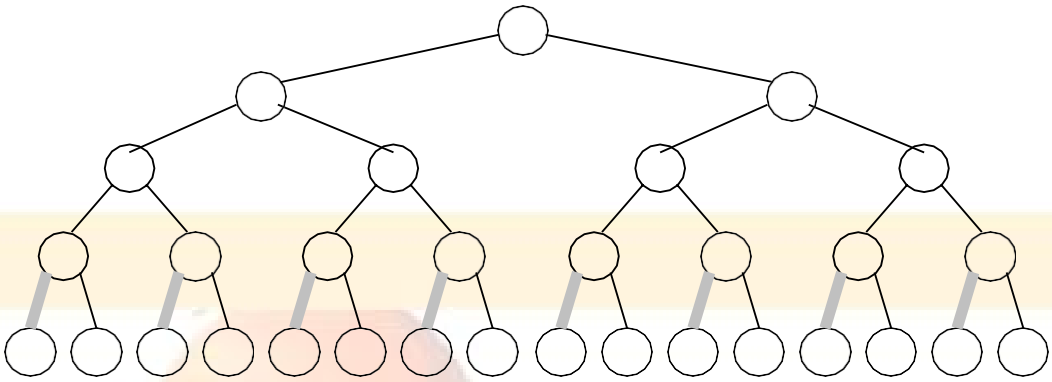


Figure 32.2: Marking the left edges for height 1 nodes

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.

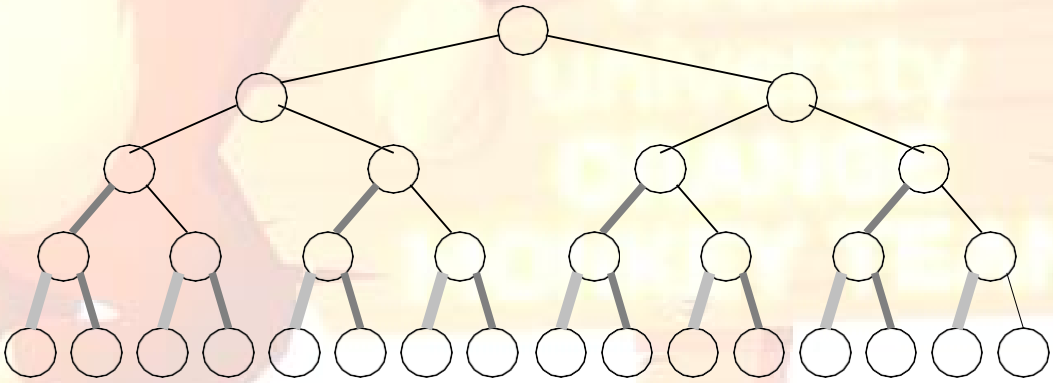


Figure 32.3: Marking the first left edge and the subsequent right edge for height 2 nodes

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.

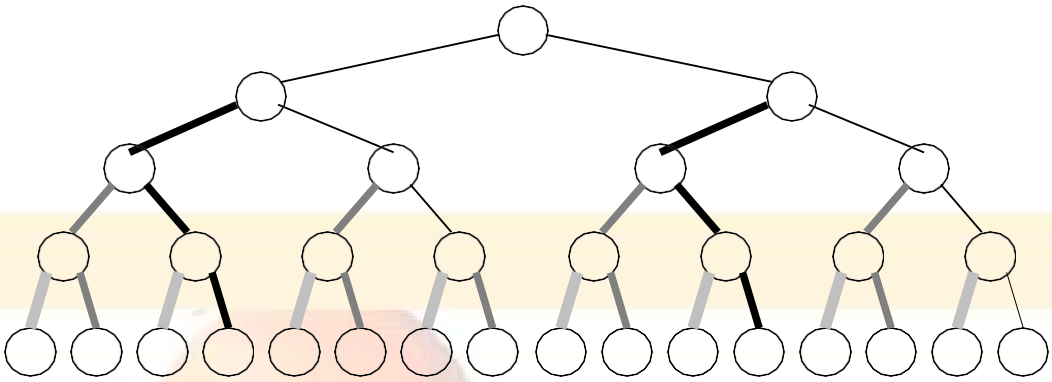


Figure 32.4: Marking the first left edge and the subsequent two right edges for height 3 nodes

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.

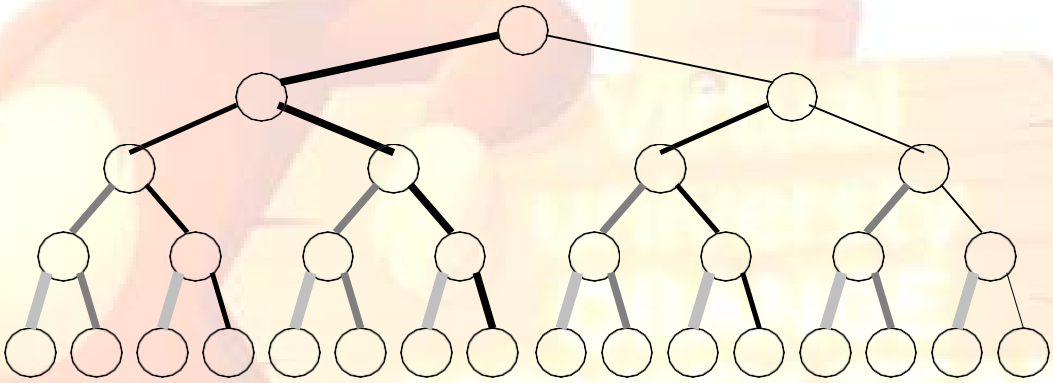


Figure 32.5: Marking the first left edge and the subsequent three right edges for height 4 nodes

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.

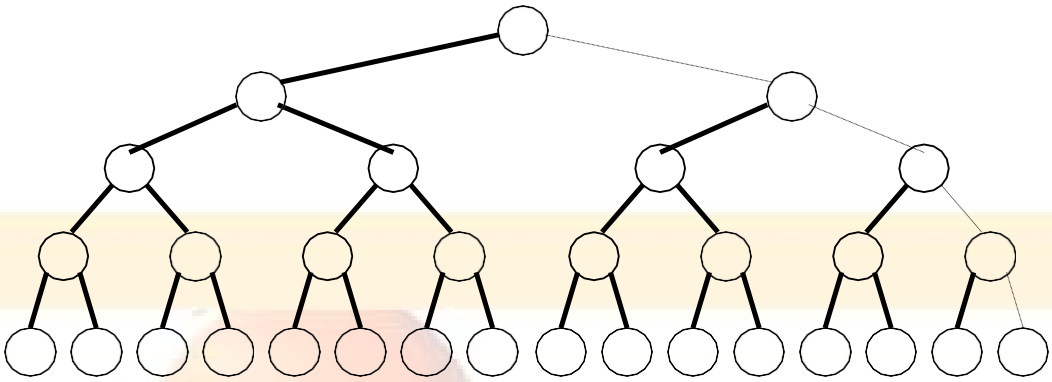


Figure 32.6: Marked and unmarked edges

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 (marled 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 $Nlog_2N$. 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 is*Full()* 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 10^{th} 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 Kth smallest element. *BuildHeap* works in linear time to make a min or a max-heap.

– The interesting case is $k = \lceil N/2 \rceil$ as it is also is 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 $N/2^{th}$ 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 100^{th} 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' 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 elliptics- 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 propertiesReflexivity: for all element x ∈ 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.

AL-JUNAID INSTITUTE GROUP

CS301-DATA STRUCTURE
FINAL TERM MCQS
Prepared by: JUNAID MALIK

AL-JUNAID INSTITUTE GROUP

ALL answers are verified if found any mistake then Correct ACCORDINGLY

1. If unions are done by weight (size), the depth of any element is never greaterthen
   a. $\log_2 n$
   b. $n \log_2 n$
   c. log 3n
   d. log n*n
2. Which of the following is NOT true regarding the skip list
   a. Each list Si contains the special keys + infinity and – Infinity
   b. List SO contains the keys of S in non-decreasing orderc. List Sh contains only then n special keys
   d. Each list is a subsequence of the previous one
3. Which of the following is NOT an open addressing technique to resolvecollisions
   a. Quadratic probing
   b. Double hashing
   c. Cubic probing
   d. Linear probing
4. Which of the following possible operations are performed on Table ADT?
   a. Insert, Remove
   b. Find, Remove
   c. Insert, Find
   d. Insert, Find, Remove
5. Binary search can be categorized into which of the following?
   a. Greedy algorithm
   b. Dynamic programmingc. Divide and conquer
   d. Brute force technique
6. Suppose there is an image of 7*7 now we will have matrix of_____rows and ___
   _____columns
   .a. 7.7
   b. 49.49
   c. 100.100
   d. 8.8
7. Which of the following is NOT true regarding the maze generation?
   a. Randomly remove walls until the entrance and exit cells are in the sameset

b. Removing a wall is the same as doing a union operation
c. Do not remove a randomly chosen wall if the cell it separates are alreadyin the same set
d. None of the given

8. Consider a hash table of size seven, with starting index zero, and a hash function (3x + 4)mod7. Assuming the hash table is initially empty. Which of thefollowing is the contents of the 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.
   a. 8, _, _, _, _, _, 10
   b. 1, _, _, _, _, _, 3
   c. 1, 10, 8, _, _, _, 3
   d. 1, 8, 10, _, _, _, 3
9. Suppose a hash function returns 6 for the given value. At which index of arraythe value will be saved?
   a. 0
   b. 5
   c. 6
   d. 7
10. If a hash function returns 4 for more than one value, this condition is called _____.
   a. Mergin
   gb.
   Collision
   c. Mapping
   d. Clustering
11. During the union by size method, all the array element initialized to -1 shows;
   a. Tree is complete
   b. Initial condition of tree
   c. Every tree has two elements each.
   d. None of the

given

12. What is a skip list?
   a. A linked list with size value in nodes
   b. A linked list that allows slower search within an ordered sequencec. A linked list that allows faster search within an ordered sequence
   d. A tree which is in the form of linked list
12. Which of the following is true regarding the maze generation?
   a. Randomly remove walls until the entrance and exit cells are in the sameset

b. Removing a wall is the same as doing a union operation
c. Do not remove a randomly chosen wall if the cell it separates are alreadyin the same set
d. All of the given

13. What is the best definition of a collision in a hash table?
a. Two entries are identical except for their keys
b. Two entries with different data have the exact same key
c. Two entries with different keys have the same exact has value.
d. Two entries with the exact same key have different hash values.

14. Suppose there is an image segmented into pixels. Each pixel has _____ neighbor(s).
a. 0
b. 4
c. 8
d. 16

15 The array in binary search in sub divided_____.
a. Once
b. Twice
c. N time
d. Until a sublist is no more divisible

16. What is the time complexity of binary search with iteration?
a. O(nlogn)
b. O(n2)
c. O(n)
d. O(logn) https://www.geeksforgeeks.org/binary-search/

17. Which of the following is NOT an implementation of Table ADT?
a. Sorted Sequential Array
b. Stack
c. Linked List
d. Skip List

18. In 1990, Bill Pugh proposed an enhancement on linked lists and the new datastructure was termed as
a. Linked list
b. B-Tree
c. Skip list
d. Spelling checker

19. During union by size method, which data structure is used to improve thebalancing of tree?

a. Array
b. Stack
c. Linked list
d. Tree

20. What is the depth of any tree if the union operation is performed by height?
a. O(N)
b. O(N log N)
c. O(log N)
d. O(M log N)

21. Which of the following is NOT true regarding the maze generation?
a. Randomly remove walls until the entrance and exit cells are in the sameset
b. Removing a wall is the same as doing a union operation
c. Remove a randomly chosen wall if the cell it separates are already in thesame set
d. Do not remove a randomly chose wall if the cells it separate are alreadyin the same set

22. Which of the following statement is NOT correct regarding Table ADT
a. In the table , the type of information in columns may be differentb. A table consist of several columns known as entities
c. The row of table called a record
d. A major use of table is in databases where we build and use tables forkeeping information

23. The union operation is based on size or weight but the reducing the in betweenlinks or path comparison from___to the_____is done by find method
a. Root, node b. Nodes, root
c. Root , root
d. Node, node

24. Which is the hash function used in linear probing?
a. hi(x) = hash(x) mode table size
b. hi(x) = (hash(x) +f(i^2)) mode table sizec. hi(x) = (hash(x) +f(i)) mode table size
d. hi(x) = X mod 17

25. A table consist of several columns , known asa. Fields
b. Entity
c. Tuple
d. record

26. A hash function returns a _____val
ue
a. Integer
b. Double
c. Float
d. char

27. 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

28. Sorting procedure normally takes ____times
a. NLogN
b. 2N
c. N*N*N
d. N

29. The expression if(!heap-> is empty ())
Checks
a. Heap is empty
b. Heap is full
c. Heap is not empty

30. 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

31. 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

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. Cannot be determined

33. 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

34. 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

35. Which property of equivalence relation is satisfied if we say:
Ahmad is cousin of Ali and Ali is also cousin of Ahmed
a. Reflexivity
b. Symmetry
c. Transitivity
d. All of the given

36. Which one of the following is NOT the property of equivalence relation?
a. Reflexive
b. Symmetric
c. Transitive
d. Associative

37. 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

38. The total number of nodes on 10^{th} level of perfect binary tree are
a. 256
b. 512
c. 1024
d. Can't be determined

39. 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

40. The percolate Down procedure will move the smaller value ____ and bigger value ____.
a. Left, right
b. Right, left
c. Down, up
d. Up, down

41. 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)

42. Which of the following method is helpful in creating the heap at once?

a. Insert
b. Add
c. Update
d. percolateDown

43. .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

44. .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

45.
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 Expressionif (! heap->isFull()) Check

q. Heap is empty
r. Heap is full
s. Heap is not empty
t. Heap is not full

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 5^{th} 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

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-**

► 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 x 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
► $Nlog_2N$
► **log_2N (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(nlogn) 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 tems 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)**

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

► 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 (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

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 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.

- ► 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(nlogn) 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)**

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 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

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;
}
```

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 i 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

► **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.

► 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 (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

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.**

99 The arguments passed to a function should match in number, type and order with the parameters in th 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;
}
```

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
- ► **preculateDown (Page 370)**

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

► 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.

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 log2 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

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

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?

- ab+c*d-
- **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]

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)
- preculateDown(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
- **<= 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 x 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

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 , preculateDown 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 in not required at all.
- None of the given options.

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

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___.

- N
- N^2
- $Nlog_2N$
- **log_2N (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 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: 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.

► 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

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

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)
{

```
    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

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

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 x 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
- $Nlog_2N$
- **log_2N (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(nlogn) sorts
- Interchange sort (not sure)
- Average time is quadratic
- **None of the given options. (Page 488)**

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
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
- ► **<= 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 (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

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.

► **I**nitiall**y** 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 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 o

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

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
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 abov

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
- ► **<= 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(nlogn) 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.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► 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.

FINALTERM 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
► **<= 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

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
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 te 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
► $Nlog_2N$
► **log_2N (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?

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► **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

FINALTERM 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
- **► preculateDown (Page 358)**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

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 x 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
- ► $Nlog_2N$
- ► **log_2N (page 440)**

Question No: 14 (Marks: 1) - Please choose one
Consider te 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

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

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 prolong to the *root* node.

► **Yes (Page 267)**
► No

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

FINALTERM 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)**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

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 log2 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 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.

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 (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: 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**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► 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

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

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)
- ► preculateDown(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:

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► 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 x 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 te 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 , preculateDown 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]

FINALTERM 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

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

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
- ► $Log_2 (n) - 1$
- ► $2^n - 1$

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

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

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► **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.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

- 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.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari