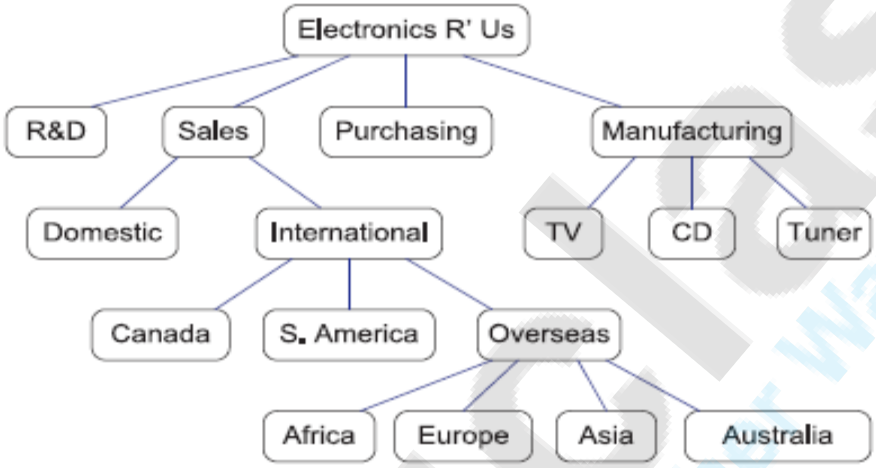
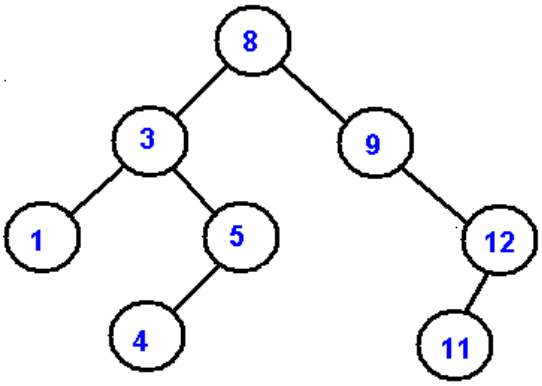
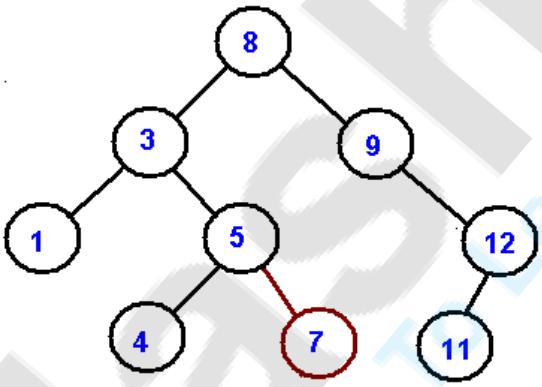
Q.1	Explain General Tree: Insertion and deletion.
Ans.	**General Tree:** A Tree in which each node having either 0 or more child nodes is called **general tree**. So we can say that a Binary Tree is a specialized case of General tree. General Tree is used to implement File System A tree is an abstract data type that stores elements hierarchically. With the exception of the top element, each element in a tree has a parent element and zero or more children elements. A tree is usually visualized by placing elements inside ovals or rectangles, and by drawing the connections between parents and children with straight lines.  Following are the important terms with respect to tree. Path − Path refers to the sequence of nodes along the edges of a tree. Root − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node. Parent − Any node except the root node has one edge upward to a node called parent. Child − The node below a given node connected by its edge downward is called its child node. Leaf − The node which does not have any child node is called the leaf node. Subtree − Subtree represents the descendants of a node. Visiting − Visiting refers to checking the value of a node when control is on the node. Traversing − Traversing means passing through nodes in a specific order. Levels − Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on. keys − Key represents a value of a node based on which a search operation is to be carried out for a node.

Insertion

The insertion procedure is quite similar to searching. We start at the root and recursively go down the tree searching for a location in a BST to insert a new node. If the element to be inserted is already in the tree, we are done (we do not insert duplicates). The new node will always replace a NULL reference.



before insertion



after insertion

Deletion

Deletion is somewhat more tricky than insertion. There are several cases to consider. A node to be deleted (let us call it as toDelete)

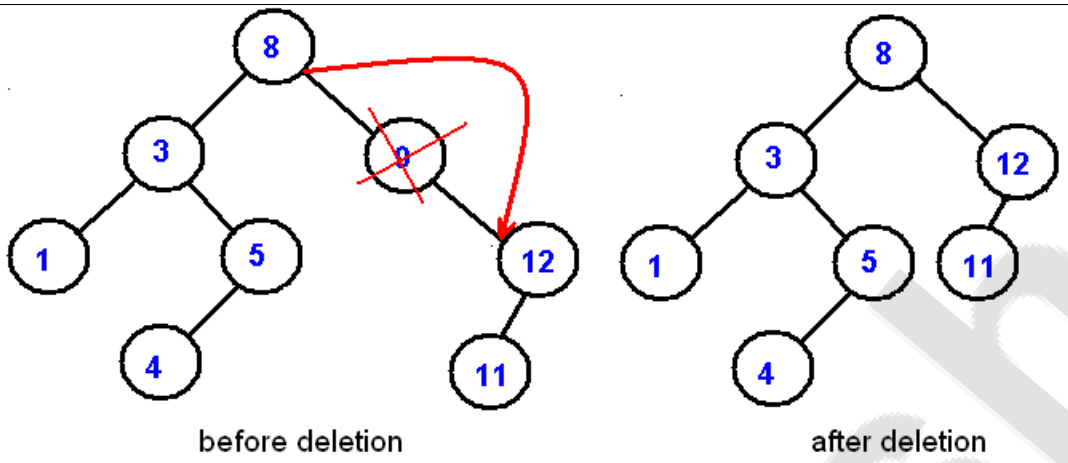
is not in a tree;

is a leaf;

has only one child;

has two children.

If toDelete is not in the tree, there is nothing to delete. If toDelete node has only one child the procedure of deletion is identical to deleting a node from a linked list - we just bypass that node being deleted



Q2. Binary tree : insertion and deletion

Ans.

Binary tree:

A binary tree is an ordered tree in which every node has at most two children.

1. Every node has at most two children.
2. Each child node is labeled as being either a left child or a right child.
3. A left child precedes a right child in the ordering of children of a node.

The subtree rooted at a left or right child of an internal node is called the node's left subtree or right subtree, respectively. A binary tree is proper if each node has either zero or two children. Some people also refer to such trees as being full binary trees. Thus, in a proper binary tree, every internal node has exactly two children. A binary tree that is not proper is improper

Algorithm 10.2: A binary search tree T and an ITEM of information is given. $P(N)$ denotes the parent of a node N, and $S(N)$ denotes the inorder successor of N. The algorithm deletes ITEM from T.

Step 1. Use Algorithm 10.1 to find the location of the node N which contains ITEM and keep track of the location of the parent node $P(N)$. (If ITEM is not in T, then STOP and Exit.)

Step 2. Determine the number of children of N. There are three cases:

(a) **N has no children.** N is deleted from T by simply replacing the location of N in the parent node $P(N)$ by the NULL pointer.

(b) **N has exactly one child M.** N is deleted from T by replacing the location of N in the parent node $P(N)$ by the location of M. (This replaces N by M.)

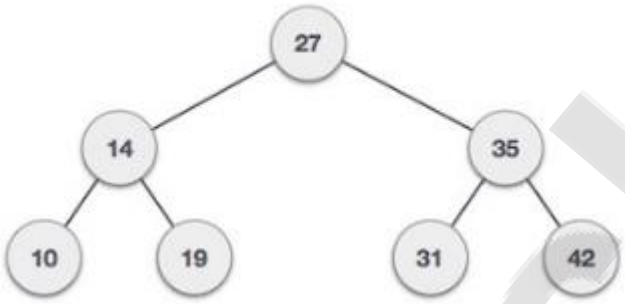
(c) **N has two children.**

(i) Find the inorder successor $S(N)$ of N. (Then $S(N)$ has no left child.)

(ii) Delete $S(N)$ from T using (a) or (b).

(iii) Replace N by $S(N)$ in T.

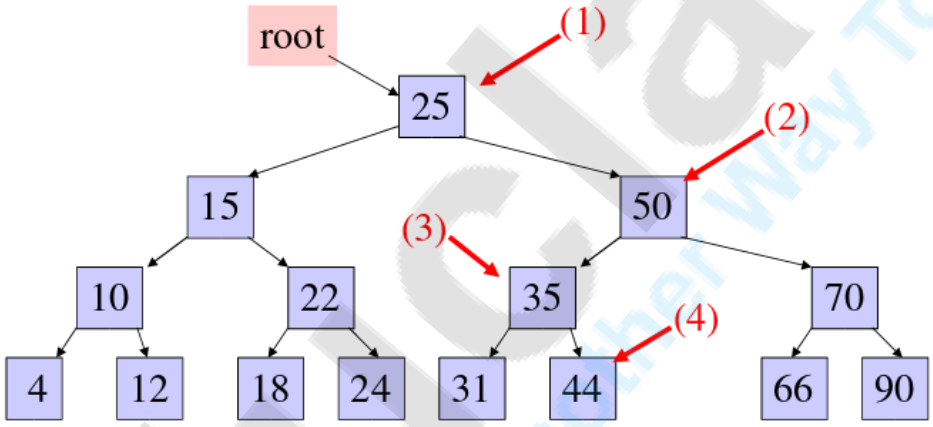
Step 3. Exit.

Q.3	Binary Search tree
Ans.	**Binary Search Tree:** A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties − The left sub-tree of a node has a key less than or equal to its parent node's key. The right sub-tree of a node has a key greater than to its parent node's key. Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as – left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)  **Basic Operations** Following are the basic operations of a tree − 1. Search − Searches an element in a tree. 2. Insert − Inserts an element in a tree. 3. Pre-order Traversal − Traverses a tree in a pre-order manner. 4. In-order Traversal − Traverses a tree in an in-order manner. 5. Post-order Traversal − Traverses a tree in a post-order manner. **Search Operation** Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

Example: search for 45 in the tree

(key fields are show in node rather than in separate obj ref to by data field):

1. start at the root, 45 is greater than 25, search in right subtree
2. 45 is less than 50, search in 50's left subtree
3. 45 is greater than 35, search in 35's right subtree
4. 45 is greater than 44, but 44 has no right subtree so 45 is not in the BST



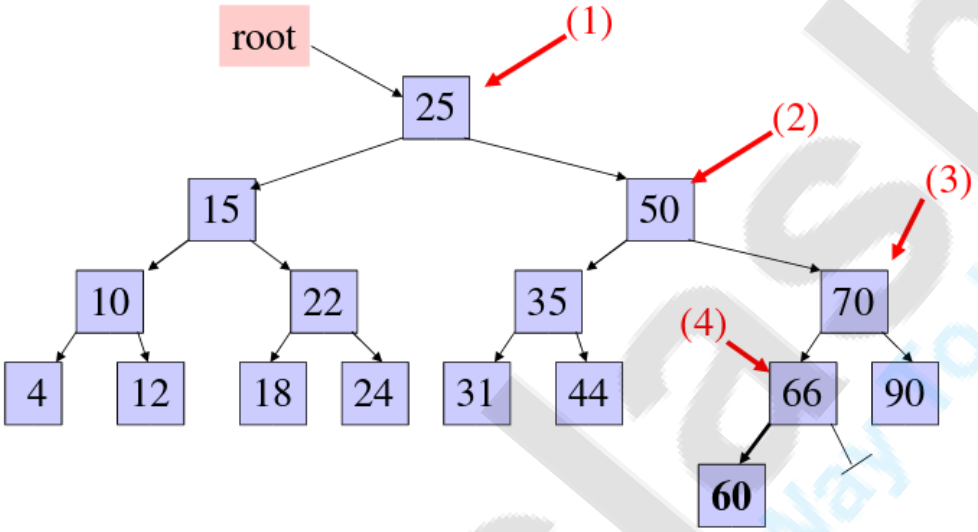
Insert Operation

Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

<table>
<tr><td></td><td>

Example: insert 60 in the tree:

1. start at the root, 60 is greater than 25, search in right subtree
2. 60 is greater than 50, search in 50's right subtree
3. 60 is less than 70, search in 70's left subtree
4. 60 is less than 66, add 60 as 66's left child



</td></tr>
<tr><td>Q.5</td><td>Traversal of binary tree</td></tr>
<tr><td>Ans.</td><td>

Traversal of search :

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −

1. In-order Traversal
2. Pre-order Traversal
3. Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

1. In-order Traversal

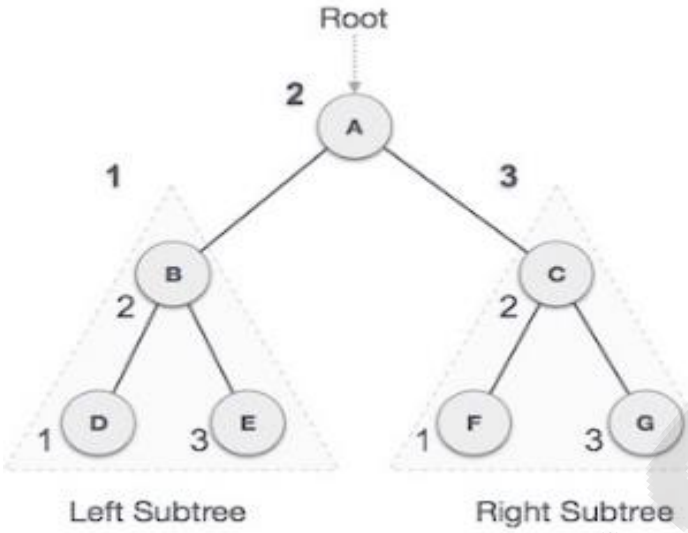
In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.

Algorithm:

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.

</td></tr>
</table>



We start from A, and following in-order traversal, we move to its left subtree B. B is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be −

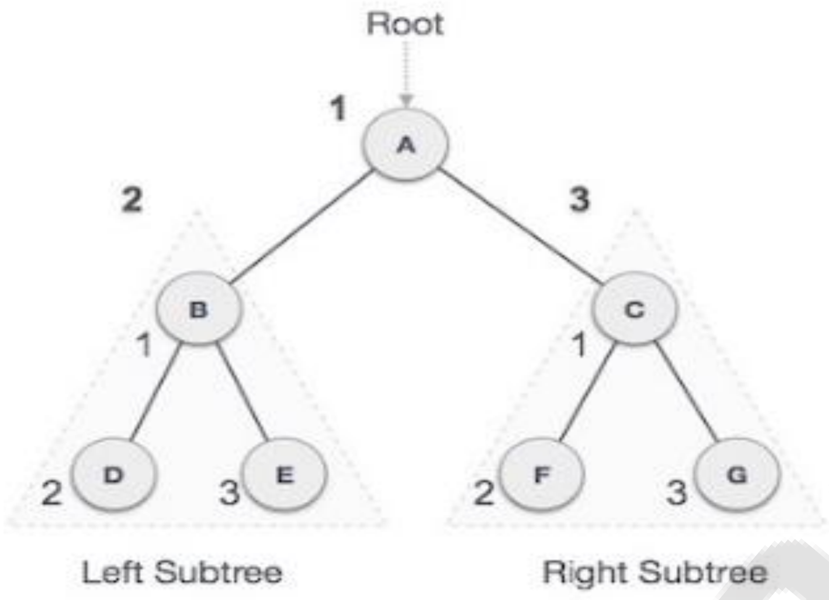
D → B → E → A → F → C → G

2. Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Algorithm :

Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.



We start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B. B is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

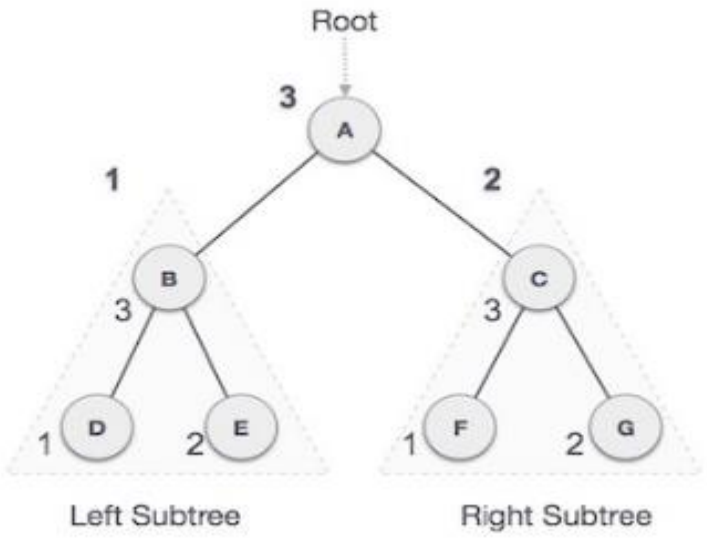
A → B → D → E → C → F → G

3. Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node

Algorithm:

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.



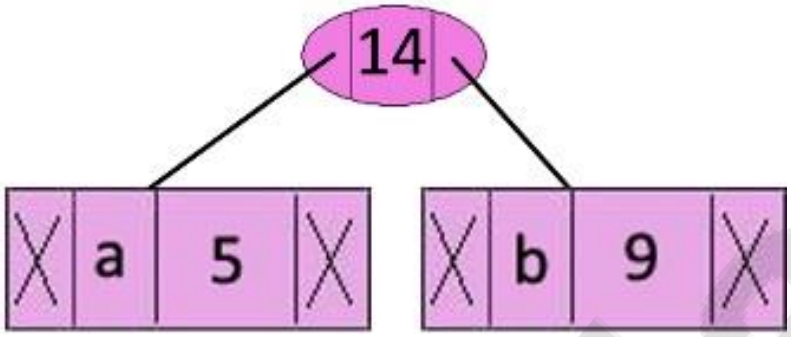
We start from A, and following pre-order traversal, we first visit the left subtree B. B is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

D → E → B → F → G → C → A

Q. 6	Huffman tree
Ans	Input is array of unique characters along with their frequency of occurrences and output is Huffman Tree. 1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root) 2. Extract two nodes with the minimum frequency from the min heap. 3. Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap. 4.Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete Let us understand the algorithm with an example: character Frequency a 5 b 9 c 12 d 13

e	16
f	45

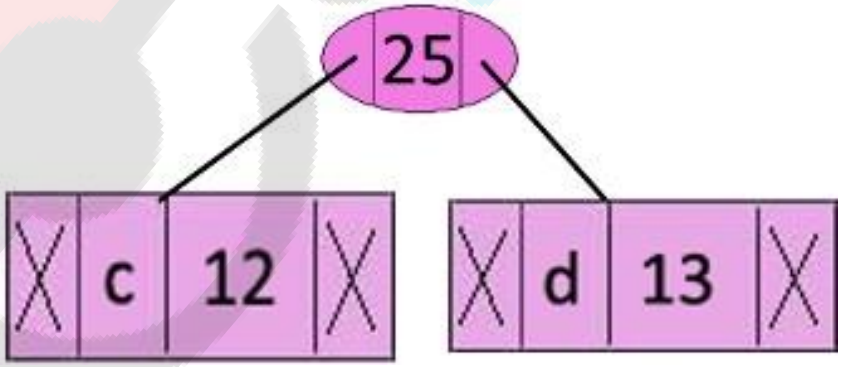
Step 1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.
Step 2 Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5 + 9 = 14



Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

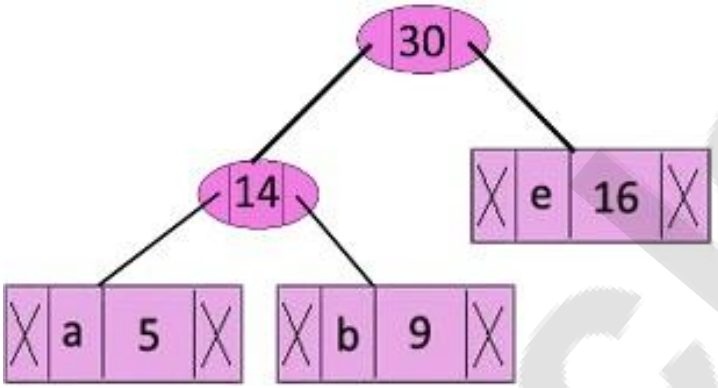
Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25



Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes.

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

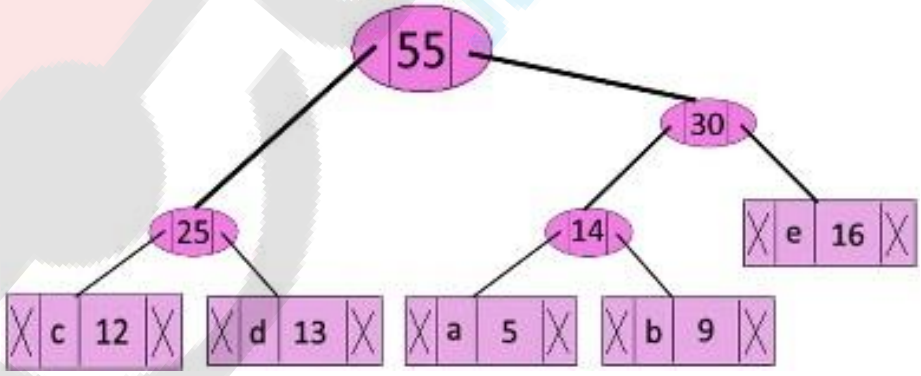
. Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30



Now min heap contains 3 nodes.

character	Frequency
Internal Node	25
Internal Node	30
f	45

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55

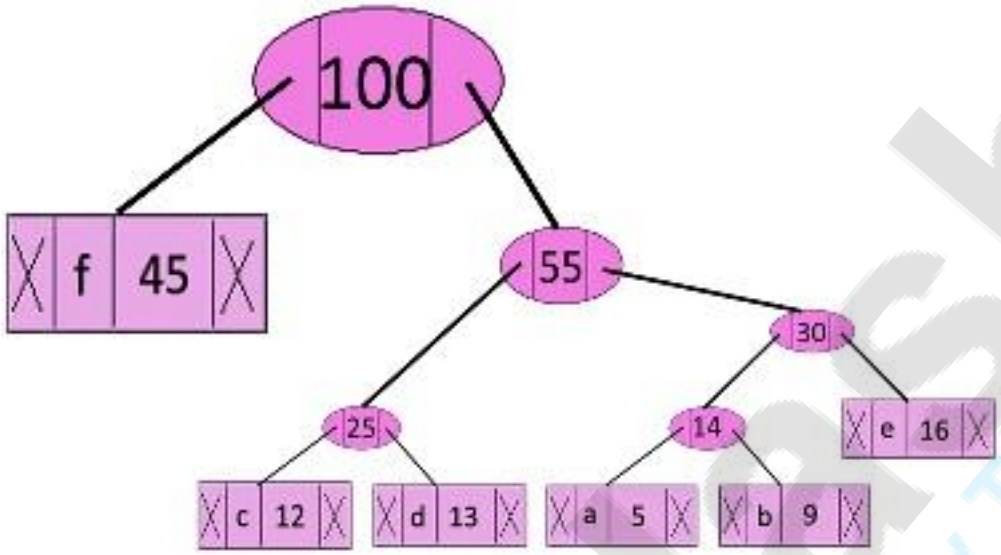


Now min heap contains 2 nodes.

character	Frequency
f	45

Internal Node 55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100



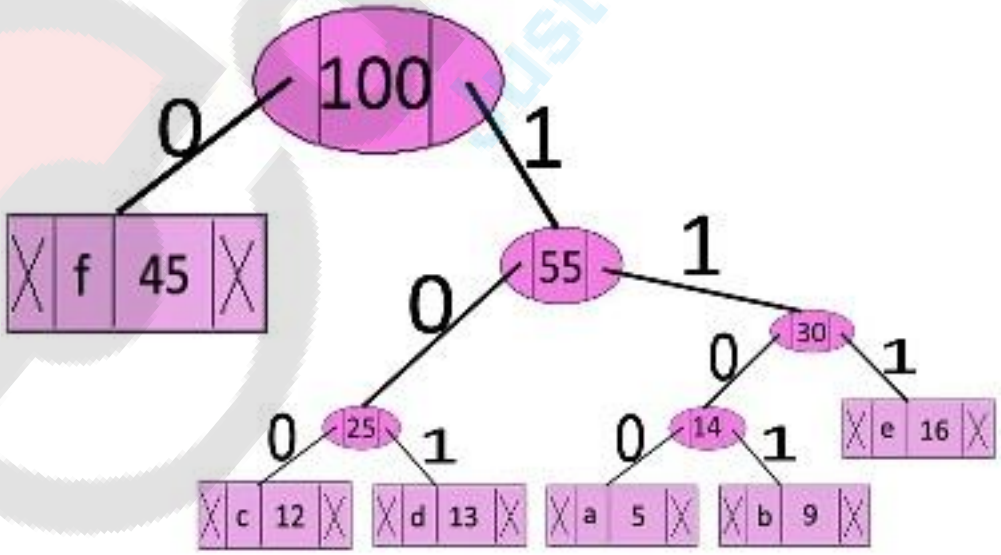
Now min heap contains only one node.

character Frequency

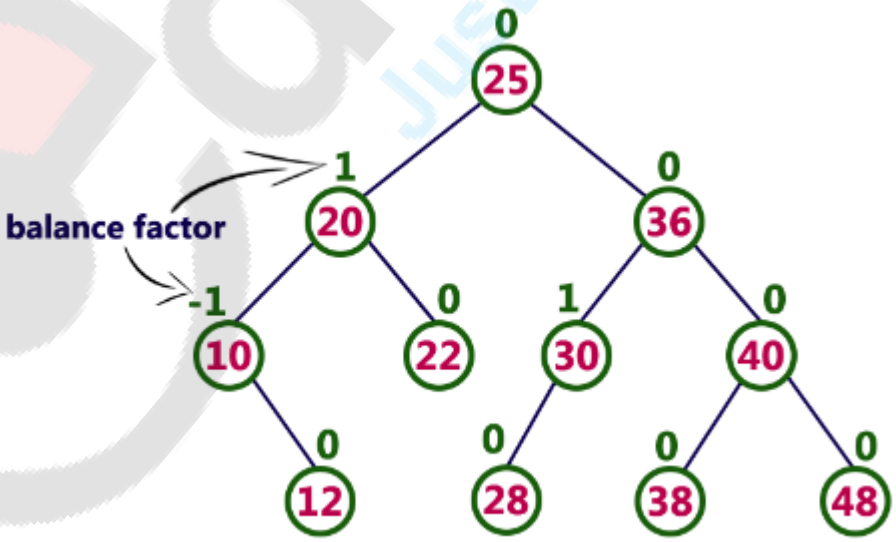
Internal Node 100

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:
Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



The codes are as follows:

	character code-word f 0 c 100 d 101 a 1100 b 1101 e 11
Q.7	AVL tree and its rotation
Ans.	AVL TREE: AVL tree is a self balanced binary search tree. That means, an AVL tree is also a binary search tree but it is a balanced tree. A binary tree is said to be balanced, if the difference between the hieghts of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if for every node, height of its children differ by at most one. In an AVL tree, every node maintains a extra information known as balance factor. The AVL tree was introduced in the year of 1962 by G.M. Adelson-Velsky and E.M. Landis. **An AVL tree is defined as follows...** **An AVL tree is a balanced binary search tree. In an AVL tree, balance factor of every node is either -1, 0 or +1.** **Balance factor = heightOfLeftSubtree – heightOfRightSubtree**  The above tree is a binary search tree and every node is satisfying balance factor

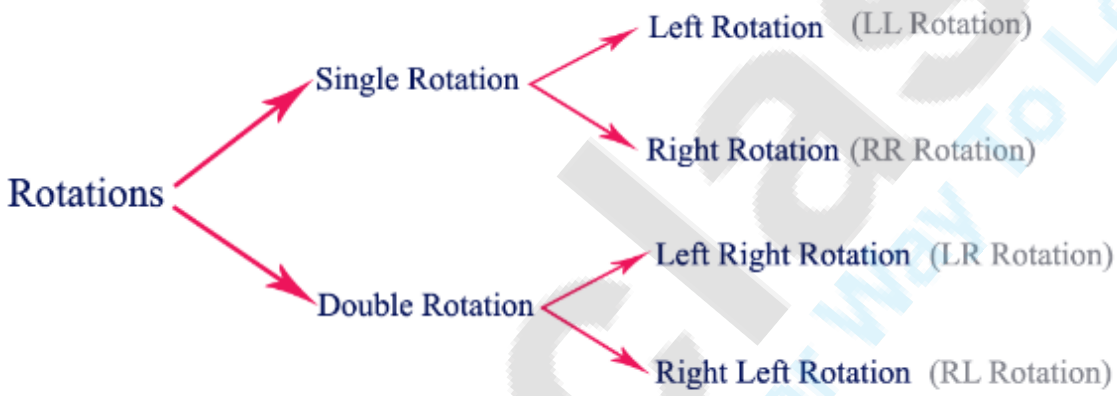
condition. So this tree is said to be an AVL tree.

AVL Tree Rotations

Rotation is the process of moving the nodes to either left or right to make tree balanced.

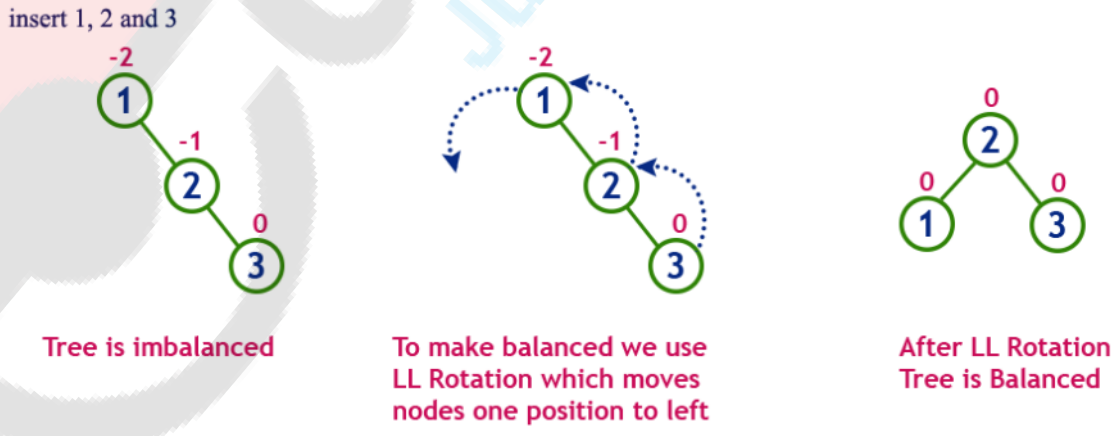
In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use rotation operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.

There are four rotations and they are classified into two types.



Single Left Rotation (LL Rotation):

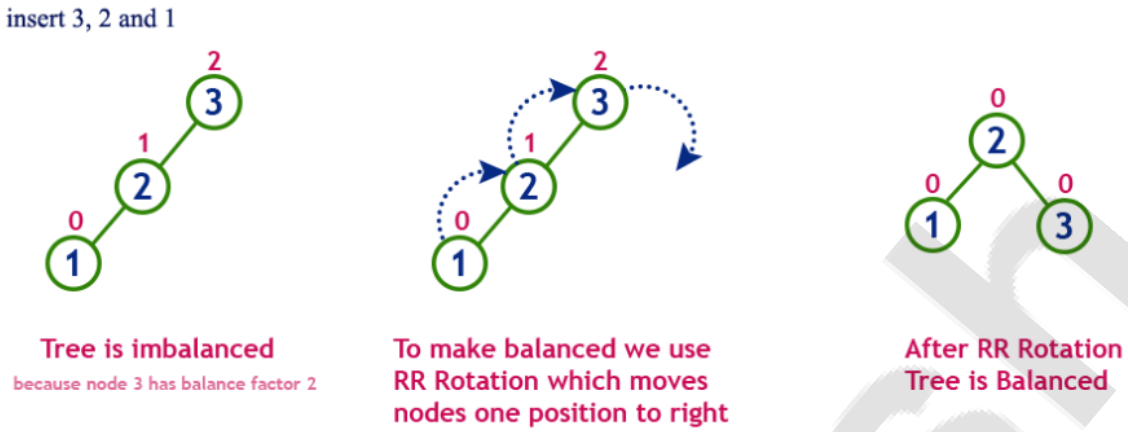
In LL Rotation every node moves one position to left from the current position. To understand LL Rotation, let us consider following insertion operations into an AVL Tree...



Single Right Rotation (RR Rotation):

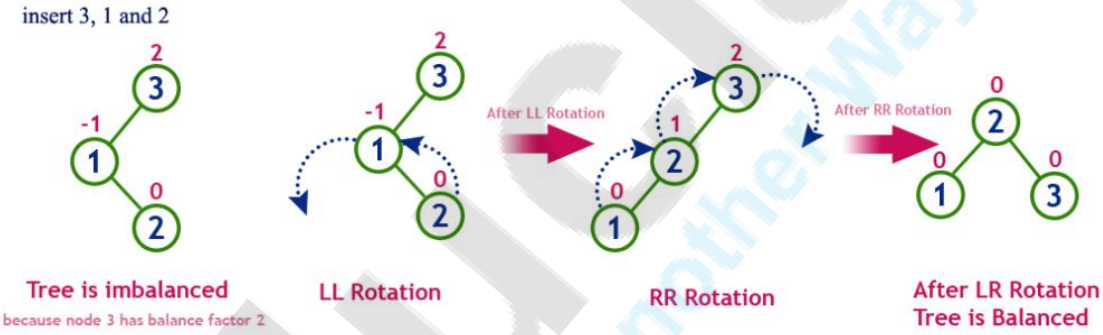
In RR Rotation every node moves one position to right from the current position. To understand RR Rotation, let us consider following insertion operations into an AVL

Tree...



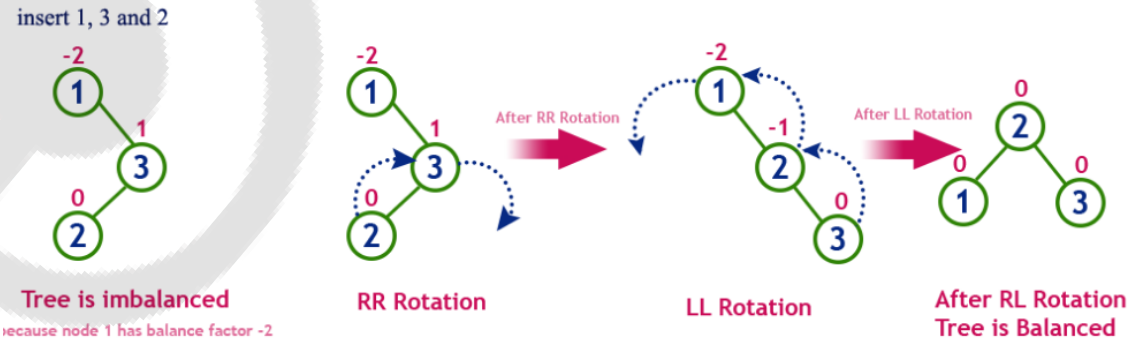
Left Right Rotation (LR Rotation):

The LR Rotation is combination of single left rotation followed by single right rotation. In LR Roration, first every node moves one position to left then one position to right from the current position. To understand LR Rotation, let us consider following insertion operations into an AVL Tree...



Right Left Rotation (RL Rotation):

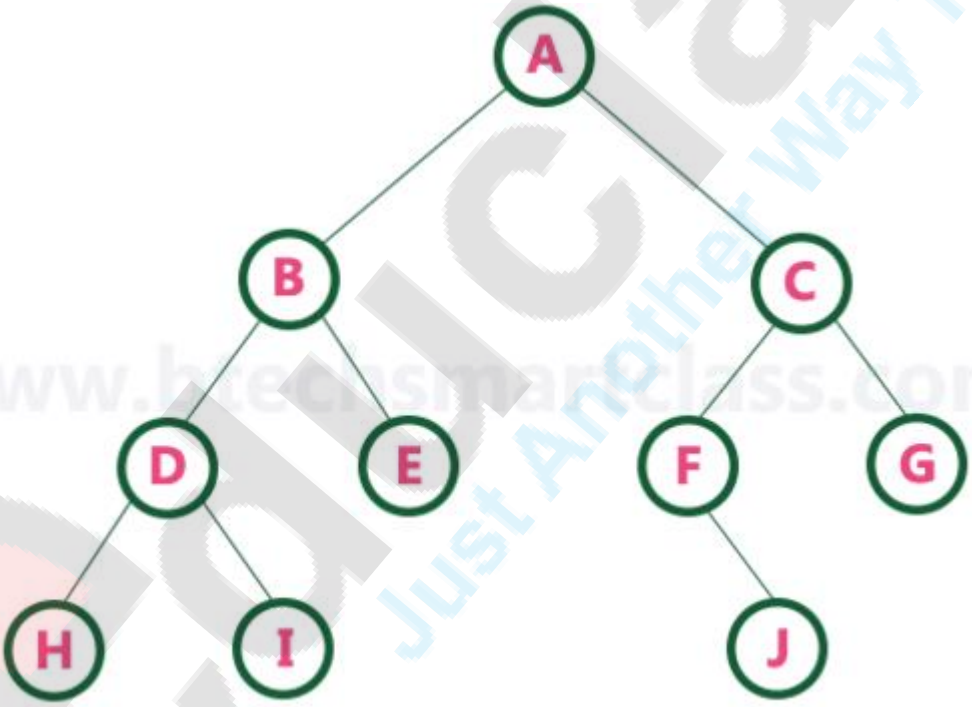
The RL Rotation is combination of single right rotation followed by single left rotation. In RL Roration, first every node moves one position to right then one position to left from the current position. To understand RL Rotation, let us consider following insertion operations into an AVL Tree...



Q. 8	Threaded Binary tree
Ans.	**Threaded Binary Tree:**

Threaded Binary Tree is also a binary tree in which all left child pointers that are NULL (in Linked list representation) points to its in-order predecessor, and all right child pointers that are NULL (in Linked list representation) points to its in-order successor.

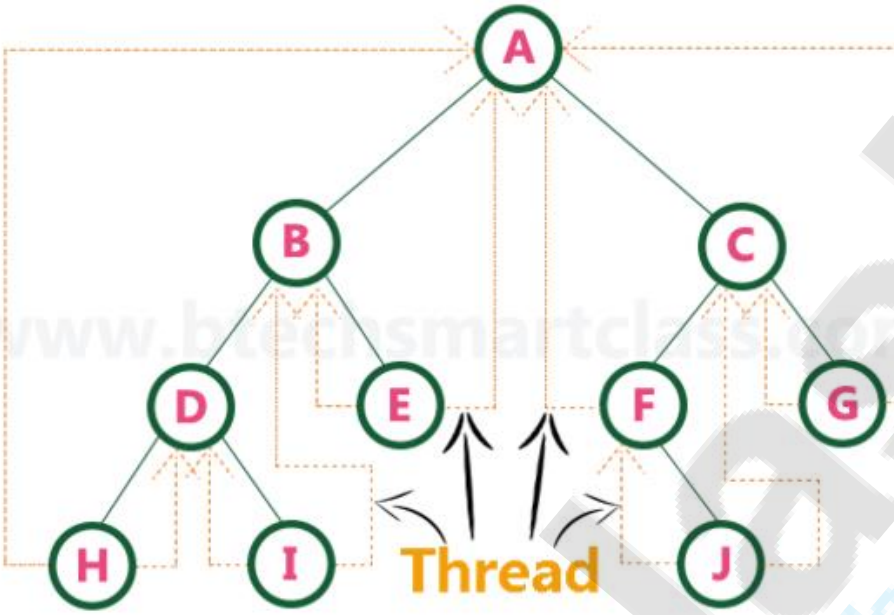
A. J. Perlis and C. Thornton have proposed new binary tree called "Threaded Binary Tree", which make use of NULL pointer to improve its traversal processes. In threaded binary tree, NULL pointers are replaced by references to other nodes in the tree, called threads.

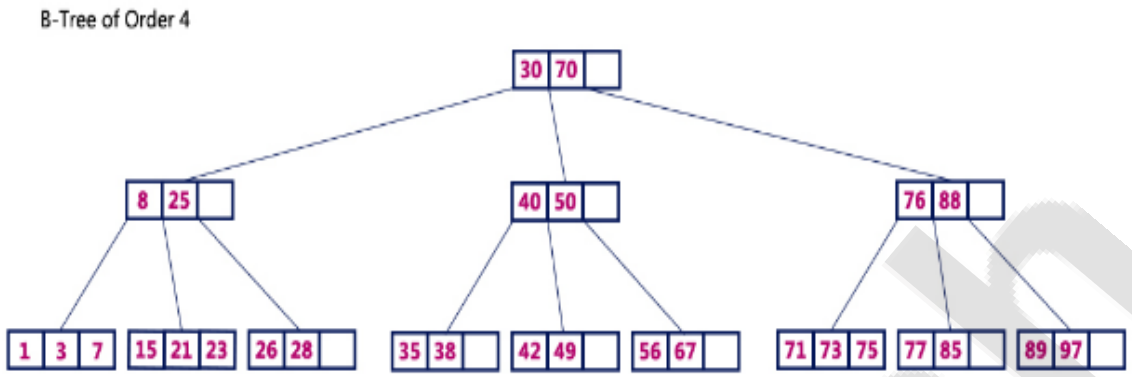


To convert above binary tree into threaded binary tree, first find the in-order traversal of that tree...

H - D - I - B - E - A - F - J - C - G

When we represent above binary tree using linked list representation, nodes H, I, E, F, J and G left child pointers are NULL. This NULL is replaced by address of its in-order predecessor, respectively (I to D, E to B, F to A, J to F and G to C), but here the node H

	does not have its in-order predecessor, so it points to the root node A. And nodes H, I, E, J and G right child pointers are NULL. This NULL ponters are replaced by address of its in-order successor, respectively (H to D, I to B, E to A, and J to C), but here the node G does not have its in-order successor, so it points to the root node A.  In above figure threadeds are indicated with dotted links.
Q. 9	Define B tree with example.
Ans.	**B tree:** B-Tree is a self-balanced search tree with multiple keys in every node and more than two children for every node. In a binary search tree, AVL Tree, Red-Black tree etc., every node can have only one value (key) and maximum of two children but there is another type of search tree called B-Tree in which a node can store more than one value (key) and it can have more than two children. B-Tree was developed in the year of 1972 by Bayer and McCreight with the name Height Balanced m-way Search Tree. Later it was named as B-Tree. **B-Tree of Order m has the following properties...** Property #1 - All the leaf nodes must be at same level. Property #2 - All nodes except root must have at least [m/2]-1 keys and maximum of m-1 keys. Property #3 - All non leaf nodes except root (i.e. all internal nodes) must have at least m/2 children. Property #4 - If the root node is a non leaf node, then it must have at least 2 children. Property #5 - A non leaf node with n-1 keys must have n number of children. Property #6 - All the key values within a node must be in Ascending Order. For example, B-Tree of Order 4 contains maximum 3 key values in a node and maximum 4 children for a node.



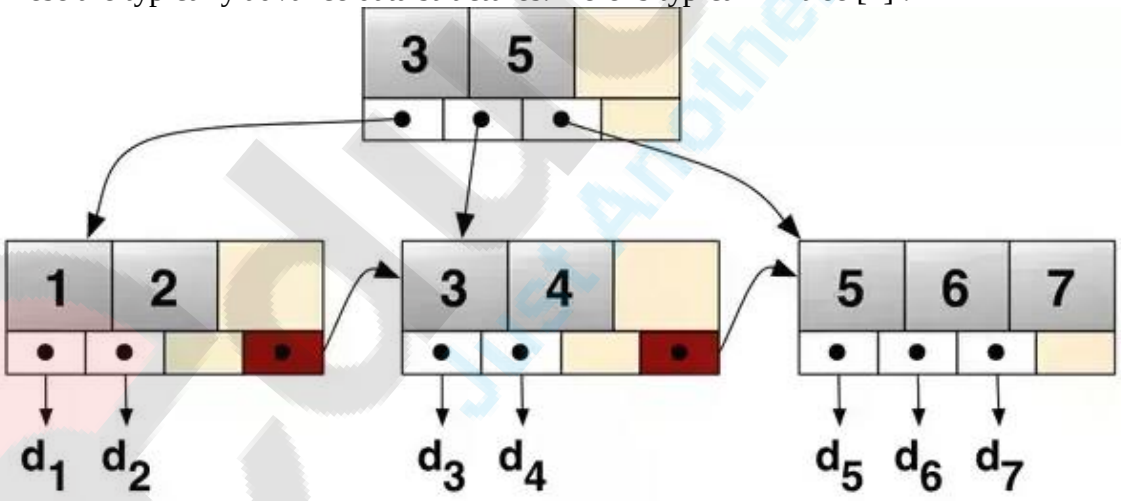
Q. 10 B* (B star) tree

Ans.

B* tree:

B* trees (B, B*, B+) trees are data structures that most commonly used in databases and file systems. In databases these data structures are used for indexing purpose.

The most significant difference between a normal binary tree and a B* tree is that :
Multiple data points in a single node instead of one
Each node has a sorted list of data points as well as the pointers to the next node
It's balanced always using elegant recursion
It minimises number of disk reads
These are typically advance data structures. Here is typical B+ tree [1] :



Q.11 Heap

Ans.

Heap:

Heap data structure is a specialized binary tree based data structure. Heap is a binary tree with special characteristics. In a heap data structure, nodes are arranged based on thier value. A heap data structure, some time called as Binary Heap.

There are two types of heap data structures and they are as follows...

1. Max Heap
2. Min Heap

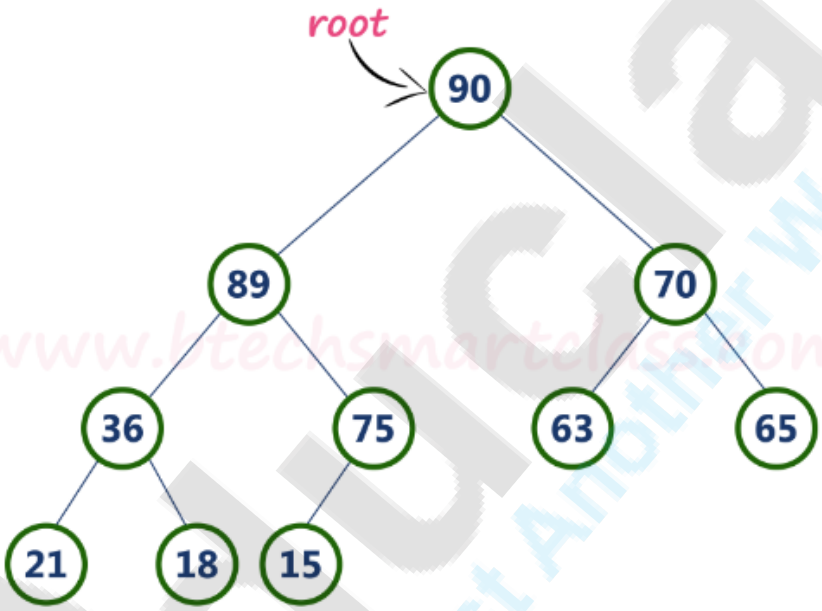
Every heap data structure has the following properties...

Property #1 (Ordering): Nodes must be arranged in a order according to values based on Max heap or Min heap.

Property #2 (Structural): All levels in a heap must full, except last level and nodes must be filled from left to right strictly.

Max Heap:

Max heap data structure is a specialized full binary tree data structure except last leaf node can be alone. In a max heap nodes are arranged based on node value.



Above tree is satisfying both Ordering property and Structural property according to the Max Heap data structure.

Operations on Max Heap

The following operations are performed on a Max heap data structure...

1. Finding Maximum
2. Insertion
3. Deletion

Finding Maximum Value Operation in Max Heap

Finding the node which has maximum value in a max heap is very simple. In max heap, the root node has the maximum value than all other nodes in the max heap. So, directly we can display root node value as maximum value in max heap.

Insertion Operation in Max Heap

Insertion Operation in max heap is performed as follows...

Step 1: Insert the newNode as last leaf from left to right.
Step 2: Compare newNode value with its Parent node.
Step 3: If newNode value is greater than its parent, then swap both of them.
Step 4: Repeat step 2 and step 3 until newNode value is less than its parent nede (or) newNode reached to root.

Deletion Operation in Max Heap

In a max heap, deleting last node is very simple as it is not disturbing max heap properties.

Deleting root node from a max heap is title difficult as it disturbing the max heap properties. We use the following steps to delete root node from a max heap...

Step 1: Swap the root node with last node in max heap
Step 2: Delete last node.
Step 3: Now, compare root value with its left child value.
Step 4: If root value is smaller than its left child, then compare left child with its right sibling. Else goto Step 6
Step 5: If left child value is larger than its right sibling, then swap root with left child. otherwise swap root with its right child.
Step 6: If root value is larger than its left child, then compare root value with its right child value.
Step 7: If root value is smaller than its right child, then swap root with rith child. otherwise stop the process.
Step 8: Repeat the same until root node is fixed at its exact position.

Q.12 M way (multi way) tree

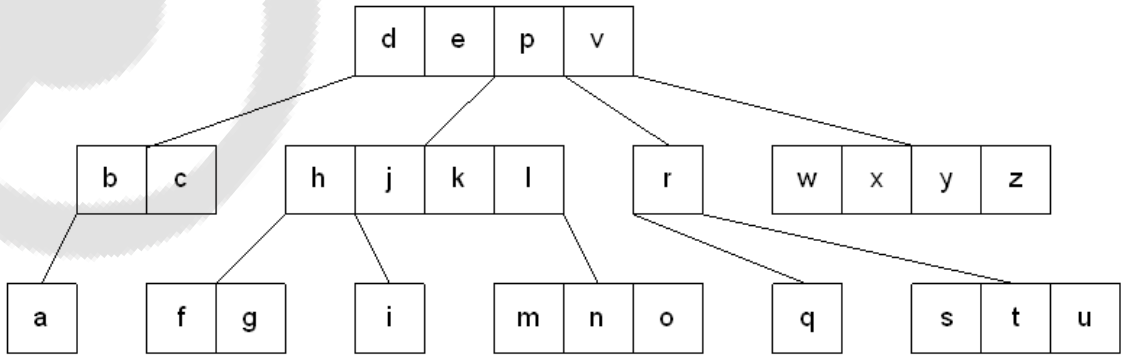
Ans.

Multiway Tree:

A multiway tree is a tree that can have more than two children. A multiway tree of order m (or an m-way tree) is one in which a tree can have m children.

As with the other trees that have been studied, the nodes in an m-way tree will be made up of key fields, in this case m-1 key fields, and pointers to children.

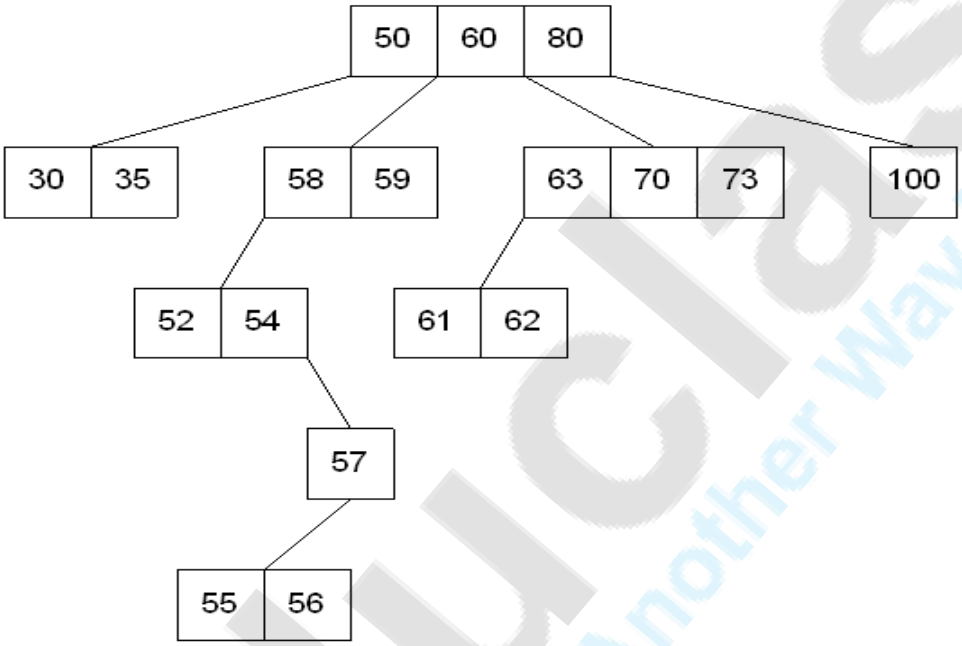
multiway tree of order 5



To make the processing of m-way trees easier some type of order will be imposed on the keys within each node, resulting in a multiway search tree of order m (or an m-way search tree). By definition an m-way search tree is a m-way tree in which:

Each node has m children and m-1 key fields
The keys in each node are in ascending order.
The keys in the first i children are smaller than the ith key
The keys in the last m-i children are larger than the ith key

4-way search tree



M-way search trees give the same advantages to m-way trees that binary search trees gave to binary trees - they provide fast information retrieval and update. However, they also have the same problems that binary search trees had - they can become unbalanced, which means that the construction of the tree becomes of vital importance.