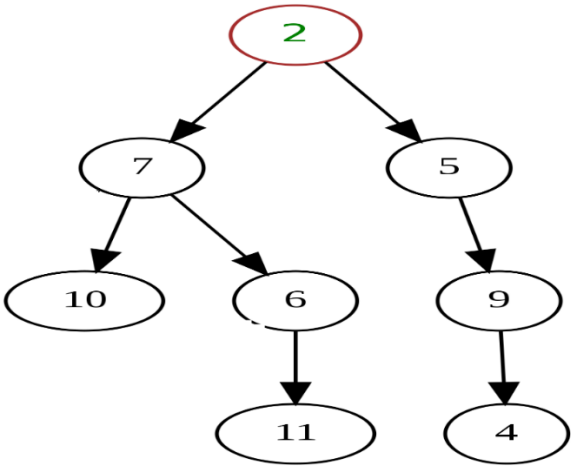
Unit IV

Trees,Binary Trees

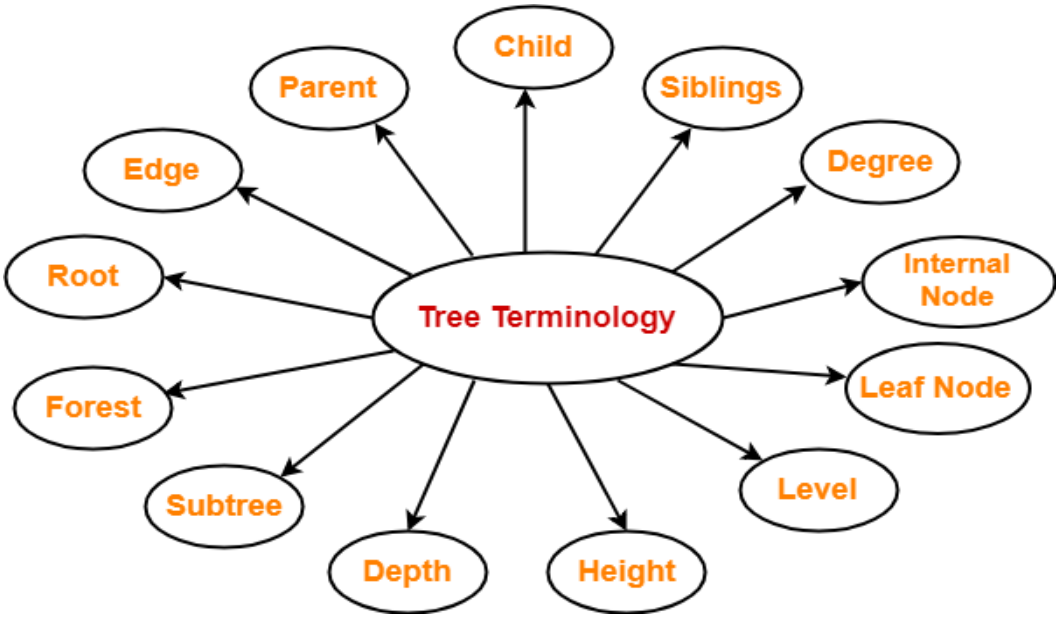
Tree:

A Tree is a non Linear hierarchial data structure that consists of nodes connected by edges.

Note: A Tree will never a closed circuit/Loop



Terminology



Node: A node is an element that contains the data value and edge to its child nodes.

Edge: It is the link between any two nodes.

Root: The first node from where the tree begins is called as a root node, In any tree, there will be only one root node.

Parent:

The node which has a branch from it, to any other node is called as a parent node, a parent node can have any number of child nodes.

Ex: 2, 7, 5, 6, 9

Child-

The node which is a descendant(derived) of some other node is called as a child node.

All the nodes except root node are child nodes.

Siblings-

Nodes which belong to the same parent are called as siblings.

Degree-

Degree of a node is the total number of children of that node.

Degree of a tree is the highest degree of a node among all the nodes in the tree.

- Degree of node 7 = 3
- Degree of node 5 = 1

Internal Node-

The node which has at least one child is called as an internal node.

Internal nodes are also called as non-terminal nodes.

Every non-leaf node is an internal node.

ex: 7,6,9

Leaf Node-

The node which does not have any child is called as a leaf node.

Leaf nodes are also called as external nodes or terminal nodes.

Level-

In a tree, each step from top to bottom is called as level of a tree.

The level count starts with 0 and increments by 1 at each level.

In above example nodes at each level are,

level 0 : 2

level 1: 7 5

level 2: 10 6 9

level 3: 11 4

Height-

Total number of edges that exist on the longest path from any leaf node to a particular node is called as height of that node.

Height of a tree is the height of root node.

Height of all leaf nodes = 0

- Height of node 2 = 3
- Height of node 5 = 2

Depth-

Total number of edges from root node to a particular node is called as depth of that node. Depth of a tree is the total number of edges from root node to a leaf node in the longest path.

Depth of the root node will be 0

Depth(2) =0

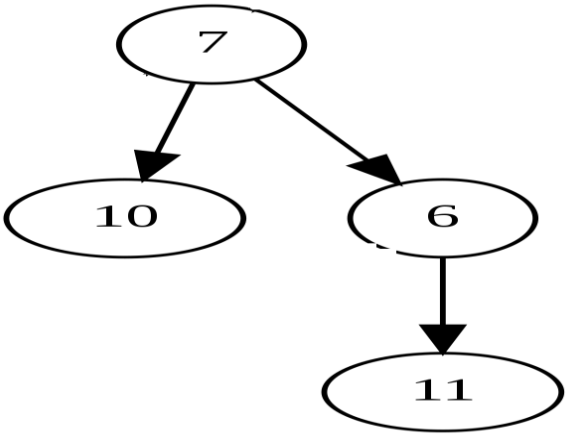
Depth(5) = 1

Depth(11) = 3

Subtree-

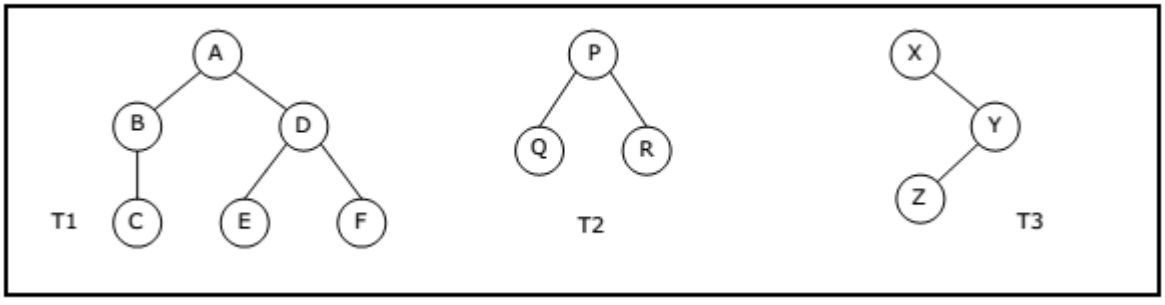
In a tree, each child from a node forms a subtree recursively. Every child node forms a subtree on its parent node.

example :



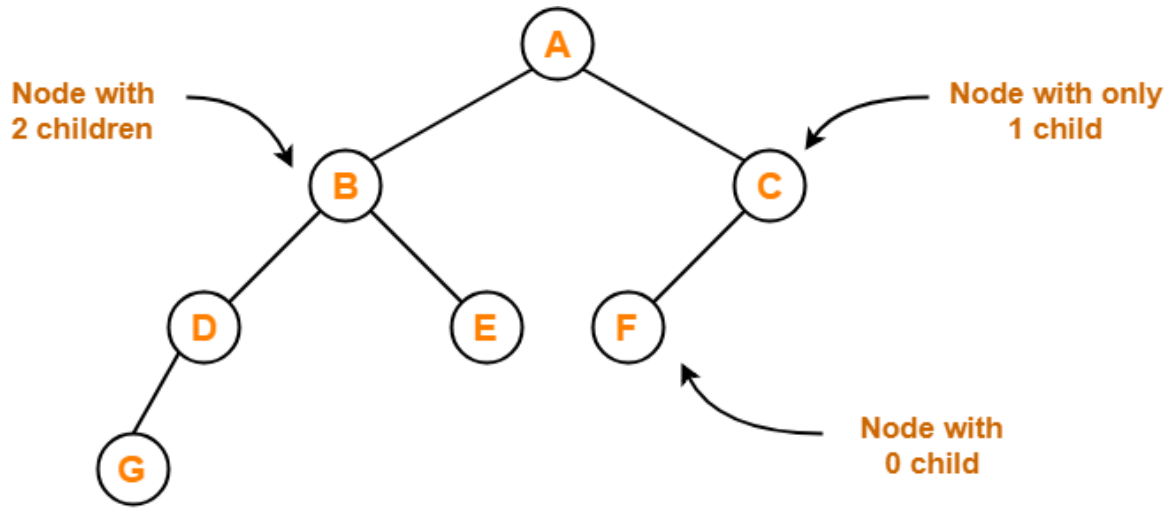
Forest:

A forest is a set of disjoint trees.



A Forest F

Binary Tree: Binary tree is a special tree data structure in which each node can have at most 2 children. Thus in a binary tree, Each node has either 0 child or 1 child or 2 children.



Binary Tree Example

Properties:

1. **Minimum number of nodes in a binary tree of height H = H + 1**
2. **Maximum number of nodes in a binary tree of height H = $2^{H+1} - 1$**
3. **Total Number of leaf nodes in a Binary Tree = Total Number of nodes with 2 children + 1**
4. **Maximum number of nodes at any level 'L' in a binary tree = 2^{L}**

Binary Tree representation – A Binary Tree can be represednted either using an array, or using linked nodes.

Array representation:

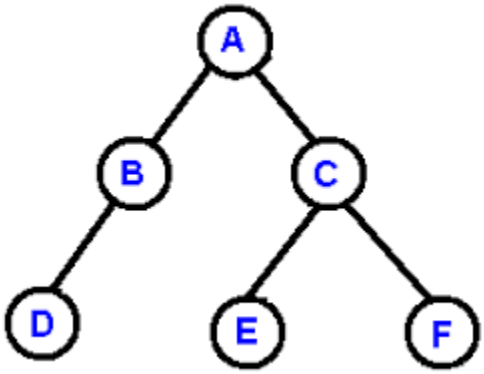
In Array representation, to implement a binary tree:

- The root of the tree will be in index 1 of the array (nothing is at index 0).
    - then onwards define the position of every other node in the tree recursively as:
- If parent node is at index i,

- The position of its left child will be at `2*i`.
- The position of its right child will be at `2n + 1`.

Note: viceversa, if the child node(either left or right) position is at `n` parent node position will be at floor(`n/2`)

Example :



Root is A, index is 1,

B is left child and C is right child of A

index(B) = 2*(index of A) = 2

index(C) = 2*(index of A) +1 = 3

index(D) = 2*(index of B) = 4

index(E) = 2*(index of C) = 6

index(F) = 2*(index of C) +1 = 7

Node	**------**	**A**	**B**	**C**	**D**	**-----**	**E**	**F**
index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Note: Node B doesn't have a right child node, so index 5 is left unused in array,

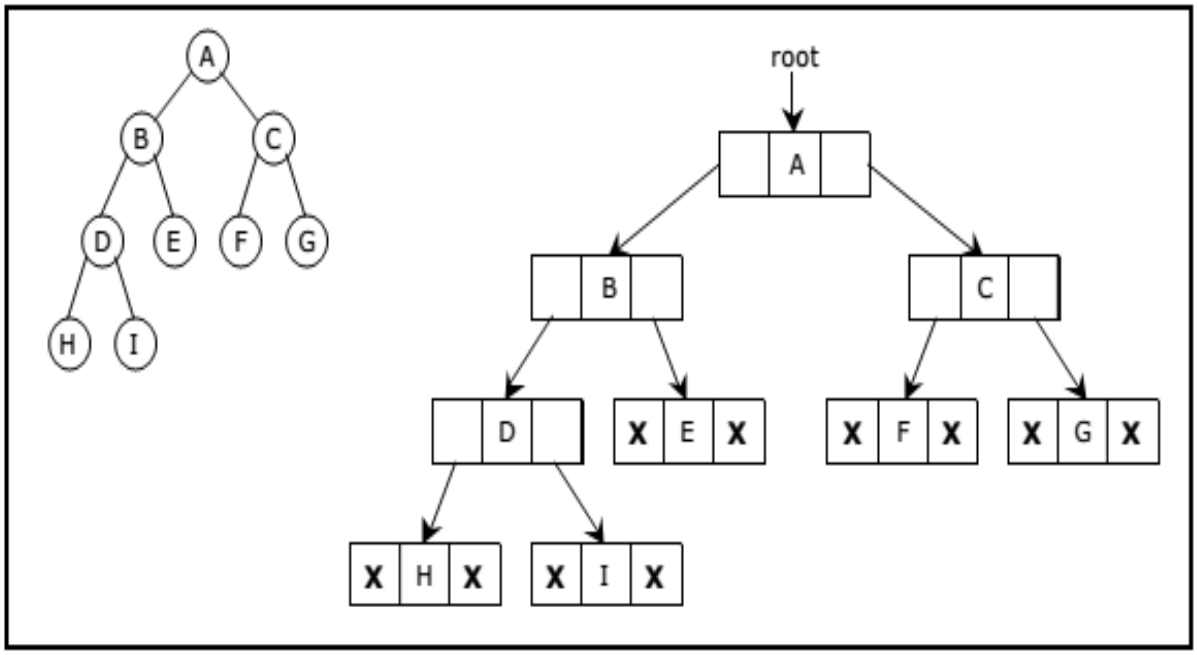
Linked list Representation:

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.

Left Child Address	Data	Right Child Address

```
struct TreeNode
{
        struct TreeNode *leftChild;
        int data;
        struct TreeNode *rightChild;
}
```

Example:



Tree Traversals:

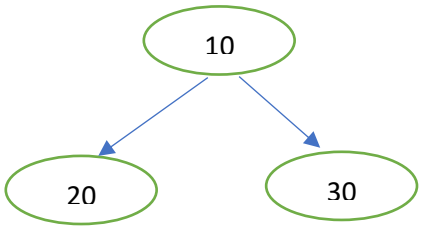
Traversing a tree means visiting every node in the tree. Generally in a Linear data structure like array, elements will be traversed sequentially.

Index	0	1	2	3	4	5	6	7
Value	12	14	16	22	34	56	44	45

Traversing array elements sequentially : 12, 14, 16, 22, 34, 56, 44, 45

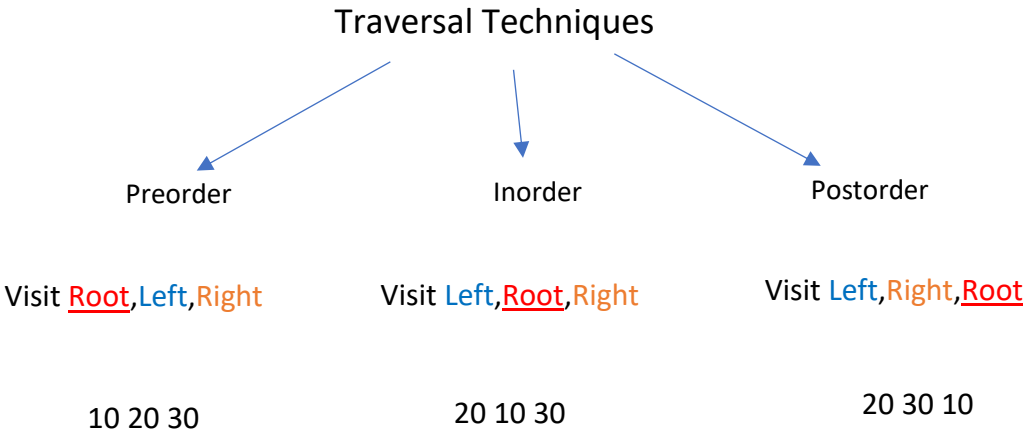
where as in a Hierarchial data structure like trees, we follow a traversal technique to visit the nodes of a tree.

Example :



In the above Tree, 10 is the root element, 20 and 30 are left and right child nodes of 10.

if we are going to traverse(visit/display) the tree, which order to follow ?



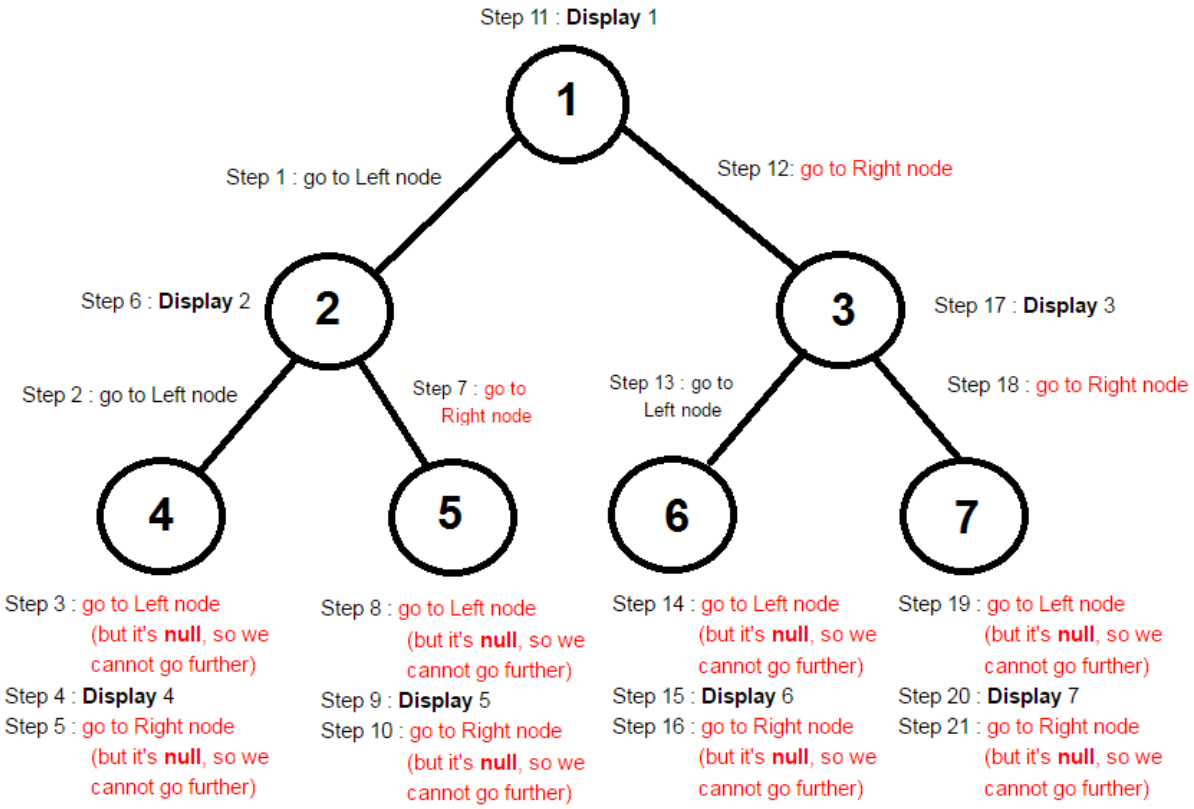
There are three ways in which we can traverse a tree –

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

We need to apply the same traversal technique at each and every node of sub tree also.

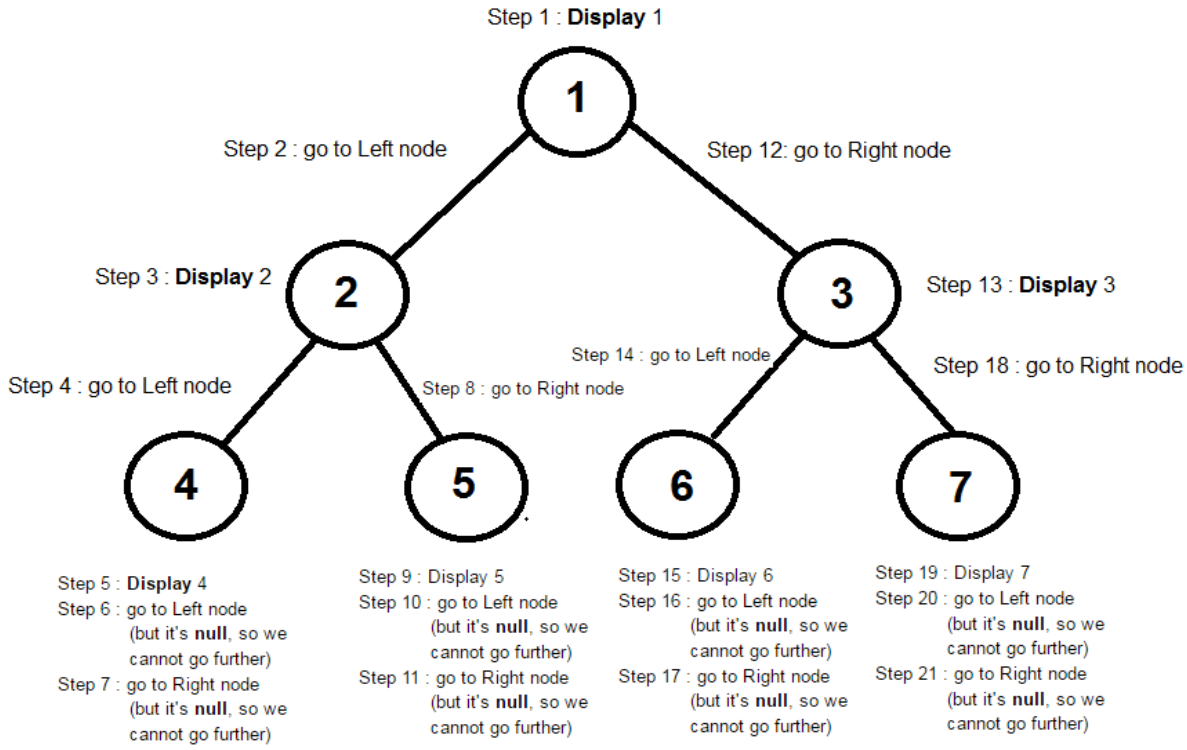
Inorder traversal

First, visit all the nodes in the left subtree
next the root node
then visit all the nodes in the right subtree



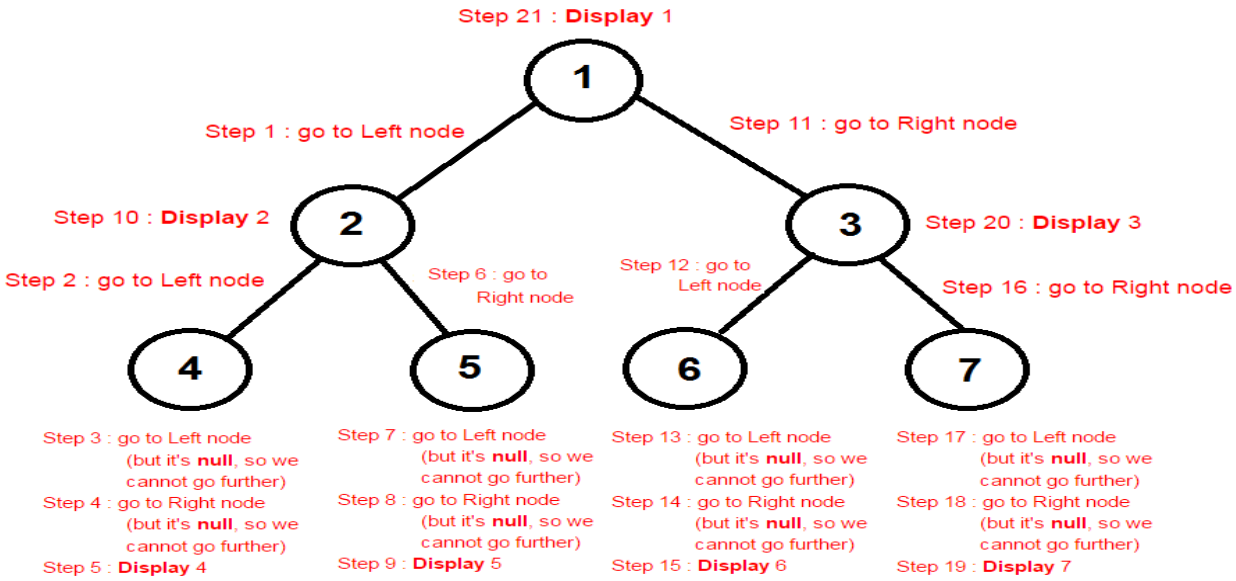
Preorder traversal

Visit root node
Visit all the nodes in the left subtree
Visit all the nodes in the right subtree



Postorder traversal

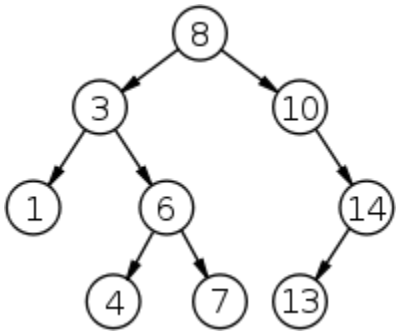
Visit all the nodes in the left subtree
Visit all the nodes in the right subtree
Visit the root node



Binary Search Tree:

Binary Search Tree, is a binary tree data structure which has the following properties:

- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub-tree each must also be a binary search tree.



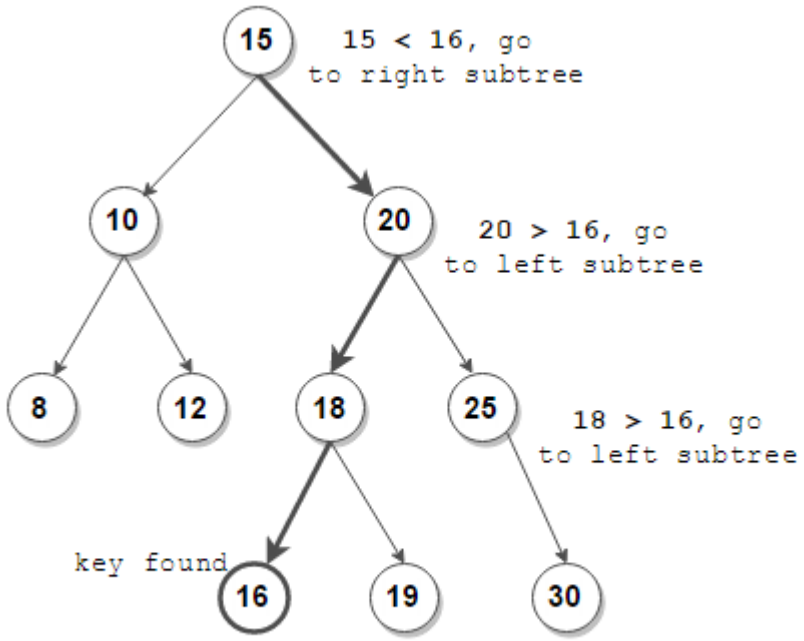
The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every node to search a given key.

Following are the basic operations of a tree −

- **Search** − Searches an element in a tree.
- **Insert** − Inserts an element in a tree.
- **Delete** – Delete an element from tree.
- **Pre-order Traversal** − Traverses a tree in a pre-order manner.
- **In-order Traversal** − Traverses a tree in an in-order manner.
- **Post-order Traversal** − Traverses a tree in a post-order manner.

Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is equal to root value, we return root. If key is greater than root's key, we search on right sub-tree of root node. Otherwise we search in left sub-tree.



Search (root, 16)

Algorithm :

If root == NULL
return NULL;
If number == root->data
return root->data;
If number < root->data
return search(root->left)
If number > root->data
return search(root->right)

Insertion

Inserting a value in the correct position is similar to searching because we try to maintain the rule that the left subtree is lesser than root and the right subtree is larger than root.

We keep going to either right subtree or left subtree depending on the value and when we reach a node whose left or right subtree is null, we put the new node there.

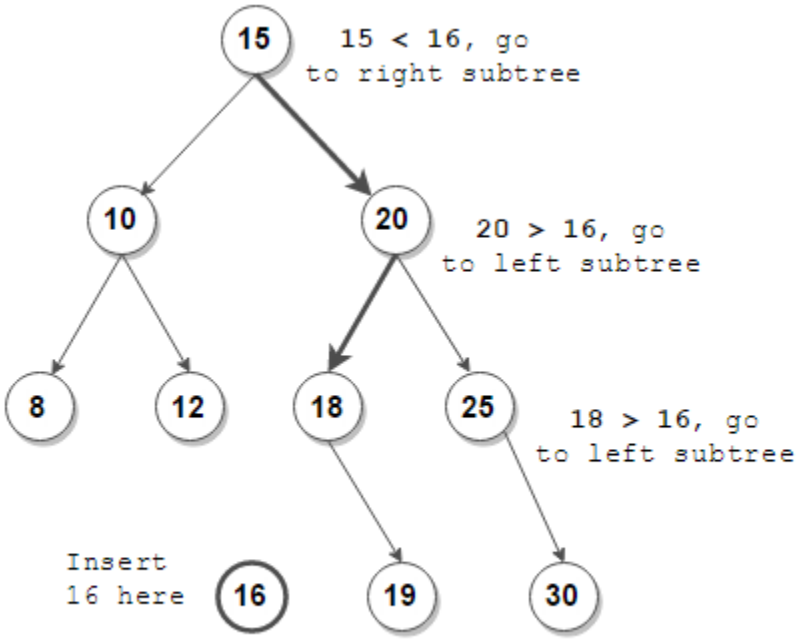
Algorithm:

```
If node == NULL
    return createNode(data)

if (data < node->data)
    node->left  = insert(node->left, data);

else if (data > node->data)
    node->right = insert(node->right, data);

return node;
```



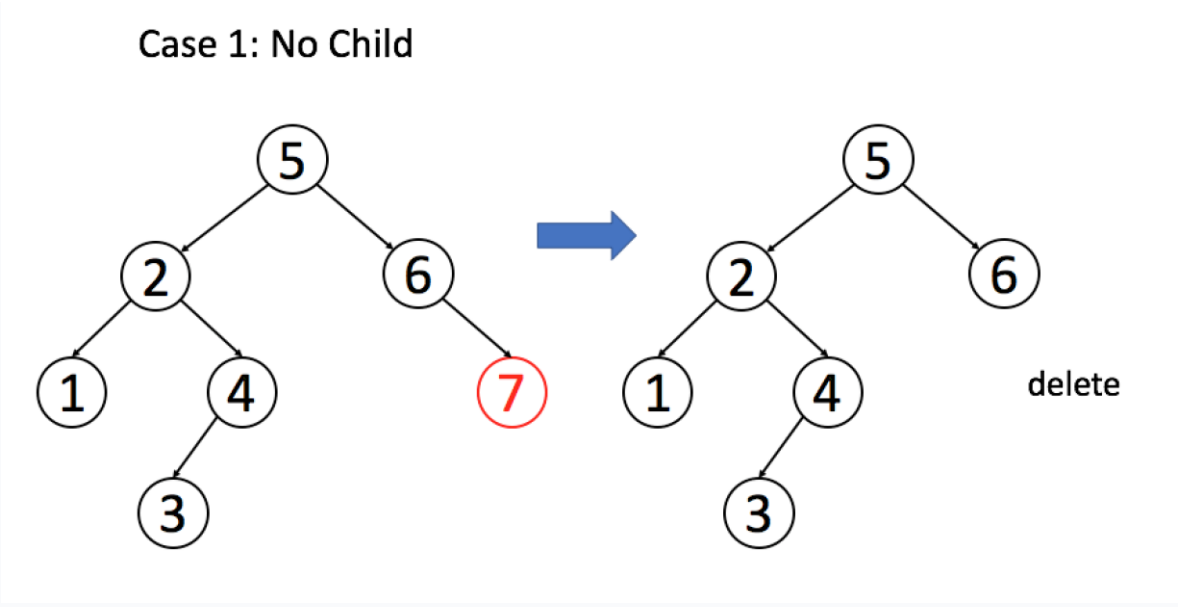
Insert (root, 16)

Deletion

There are three cases for deleting a node from a binary search tree.

Case I

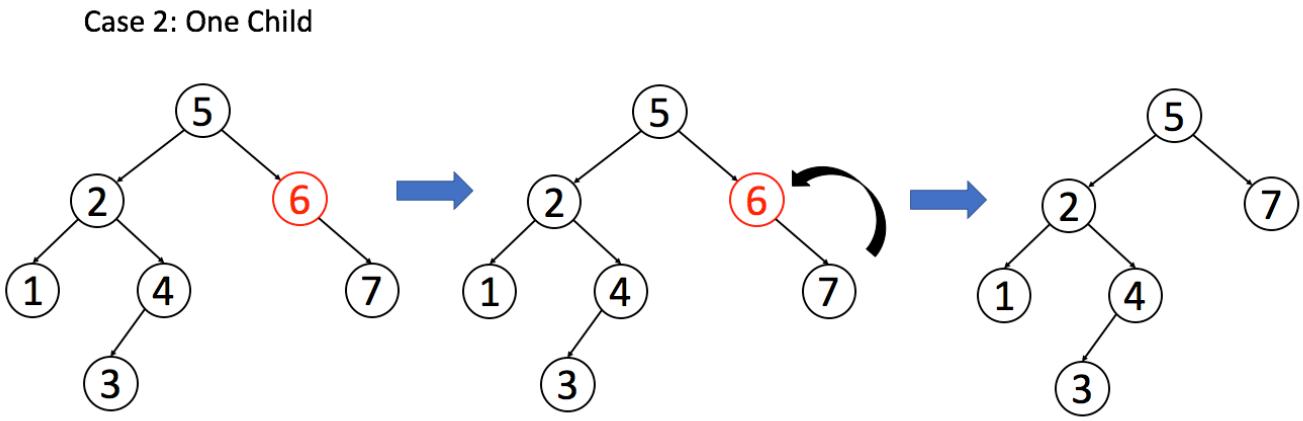
In the first case, the node to be deleted is the leaf node. In such a case, simply delete the node from the tree.



Case II

In the second case, the node to be deleted lies has a single child node. In such a case follow the steps below:

1. Replace that node with its child node.
2. Remove the child node from its original position



Case III

In the third case, the node to be deleted has two children. In such a case follow the steps below:

1. Get the inorder successor of that node.
2. Replace the node with the inorder successor.
3. Remove the inorder successor from its original position.