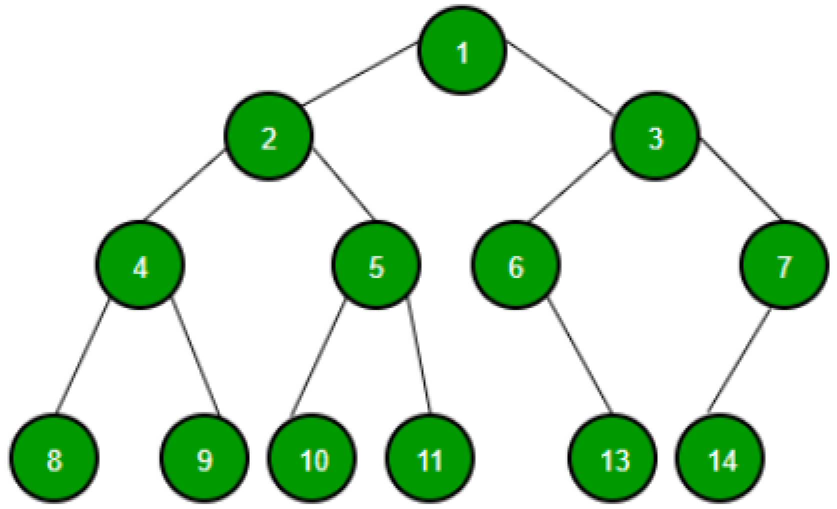
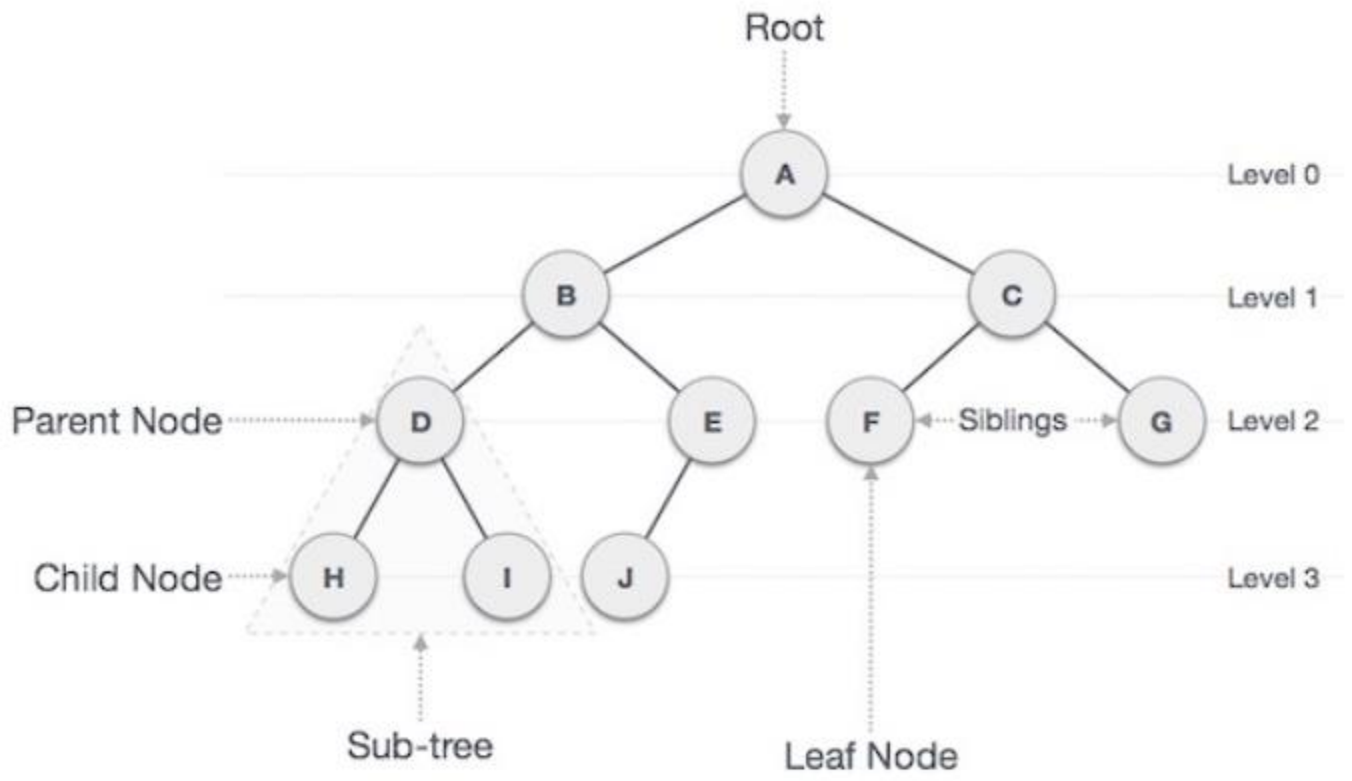
Binary Tree

- A tree whose elements have at most 2 children is called a binary tree.
  - Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



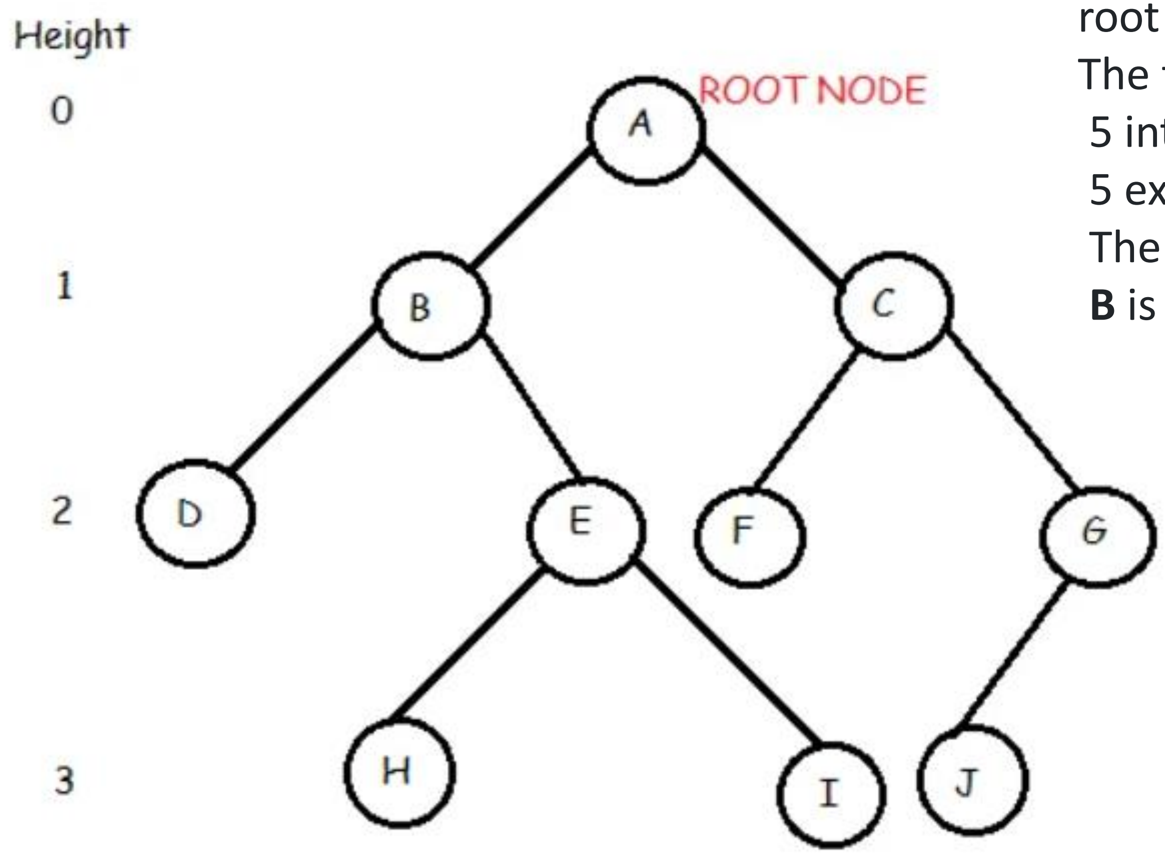

Root
A
Level 0
B
C
Level 1
Parent Node
D
E
F
Siblings
G
Level 2
Child Node
H
I
J
Level 3
Sub-tree
Leaf Node

Important Terms

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.

- **Leaf/External node** -Node with no children.
- **Internal node**: Node with atleast one children.
- **Depth of a node**: Number of edges from root to the node.
- **Height of a node**: Number of edges from the node to the deepest leaf. Height of the tree is the height of the root.



root node is **A**.
The tree has 10 nodes
5 internal nodes, i.e, **A,B,C,E,G**
5 external nodes, i.e, **D,F,H,I,J**.
The height of the tree is 3.
B is the parent of **D** and **E** while **D** and **E** are children of **B**.

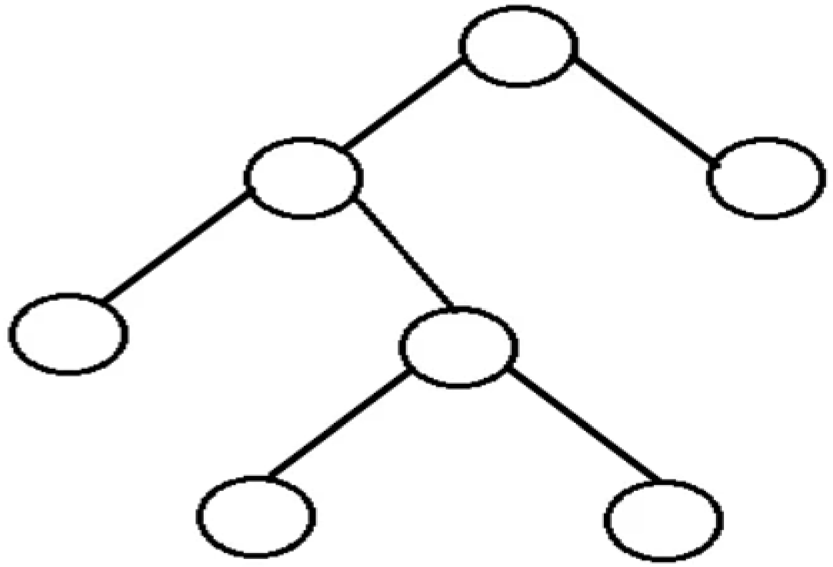
Advantages of Trees

Trees are so useful and frequently used, because they have some very serious advantages:

- Trees reflect structural relationships in the data.
- Trees are used to represent hierarchies.
- Trees provide an efficient insertion and searching.
- Trees are very flexible data, allowing to move subtrees around with minumum effort.

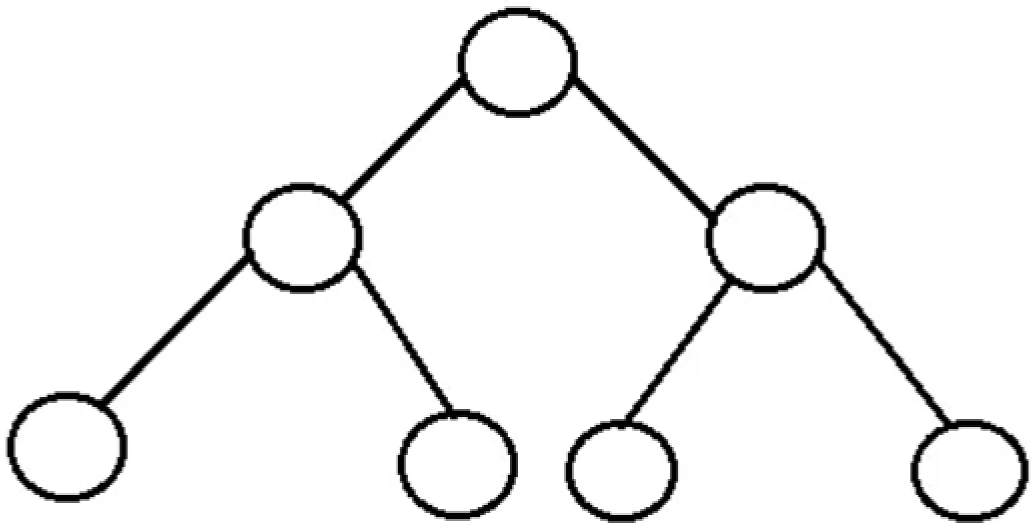
Types of Binary Trees (Based on Structure)

- **Rooted binary tree:** It has a root node and every node has atmost two children.
- **Full binary tree:** It is a tree in which every node in the tree has either 0 or 2 children.



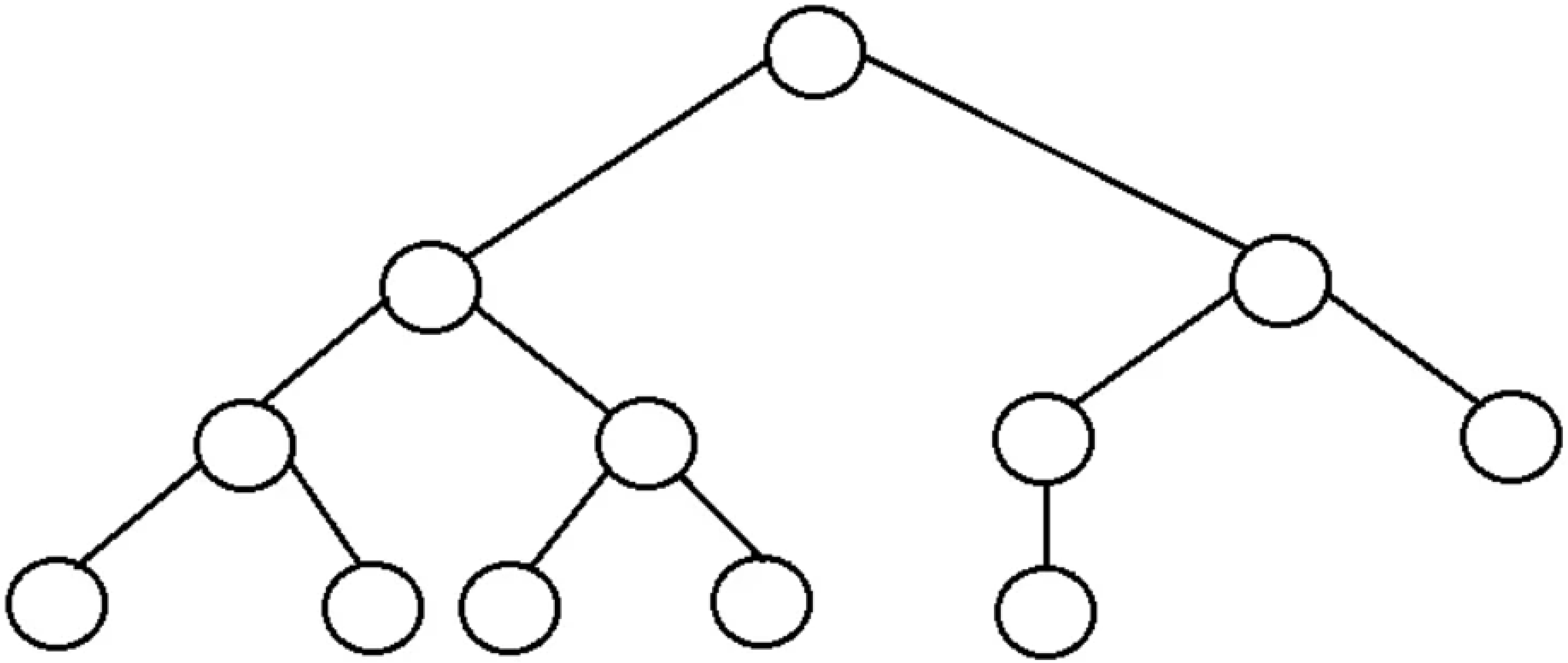
- ❑ The number of nodes, n, in a full binary tree is atleast n = 2h – 1, and atmost n = 2^{h+1} – 1, where h is the height of the tree.
- ❑ The number of leaf nodes l, in a full binary tree is number, L of internal nodes + 1, i.e, l = L+1.

Perfect binary tree: It is a binary tree in which all interior nodes have two children and all leaves have the same depth or same level.



- A perfect binary tree with *l* leaves has $n = 2l-1$ nodes.
- In perfect full binary tree, $l = 2h$ and $n = 2h+1 - 1$ where, *n* is number of nodes, *h* is height of tree and *l* is number of leaf nodes

Complete binary tree: It is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

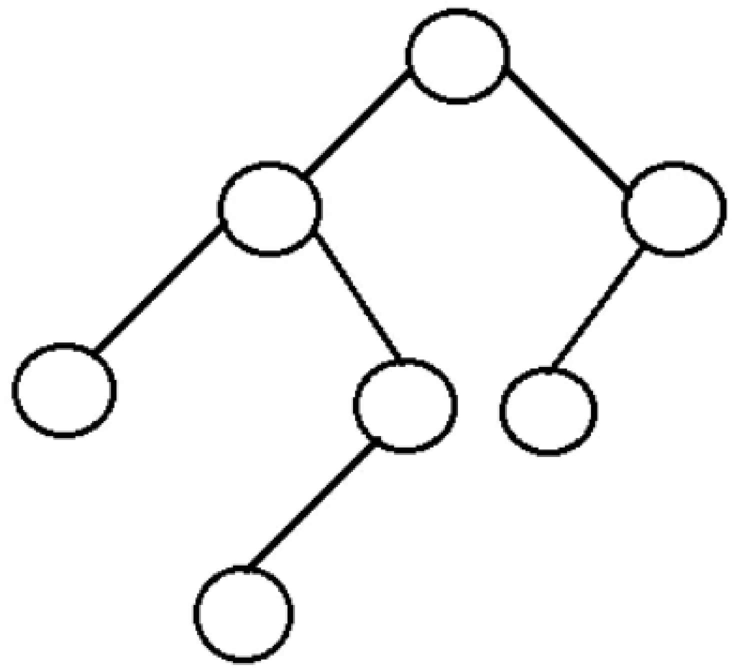


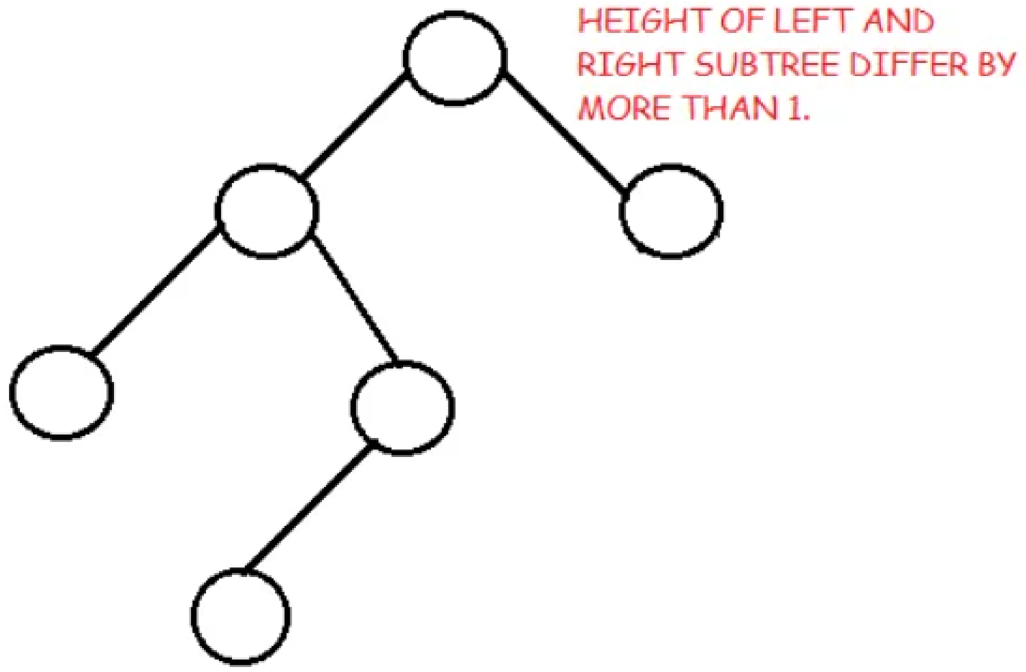
- The number of internal nodes in a complete binary tree of n nodes is *floor(n/2)*.

Balanced binary tree: A binary tree is height balanced if it satisfies the following constraints:

1. The left and right subtrees' heights differ by at most one, AND
2. The left subtree is balanced, AND
3. The right subtree is balanced

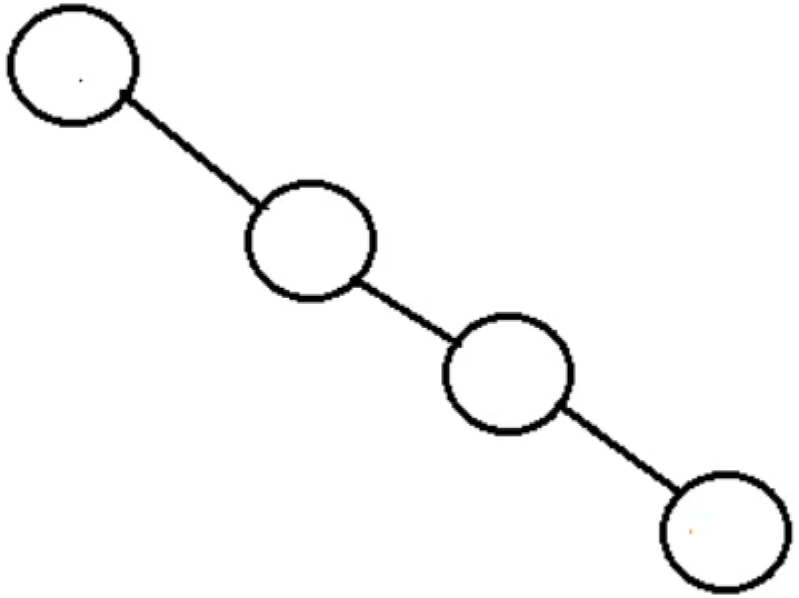
An empty tree is height balanced.


HEIGHT BALANCED BINARY TREE



NOT A HEIGHT BALANCED BINARY TREE

Degenarate tree: It is a tree is where each parent node has only one child node. It behaves like a linked list.



Tree Traversal

- 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.

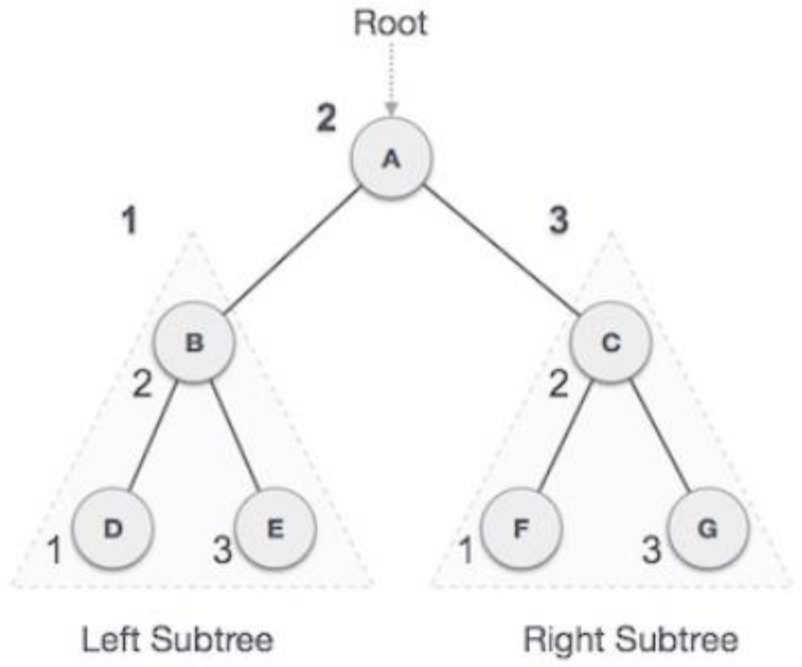
i.e. we cannot randomly access a node in a tree.

- we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.
- There are three ways which we use to traverse a tree −
  - In-order Traversal
  - Pre-order Traversal
  - Post-order Traversal

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.

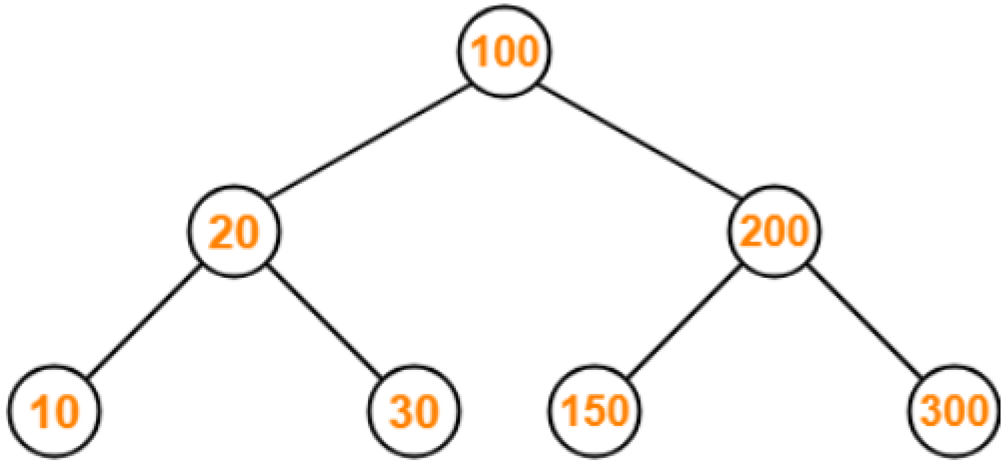


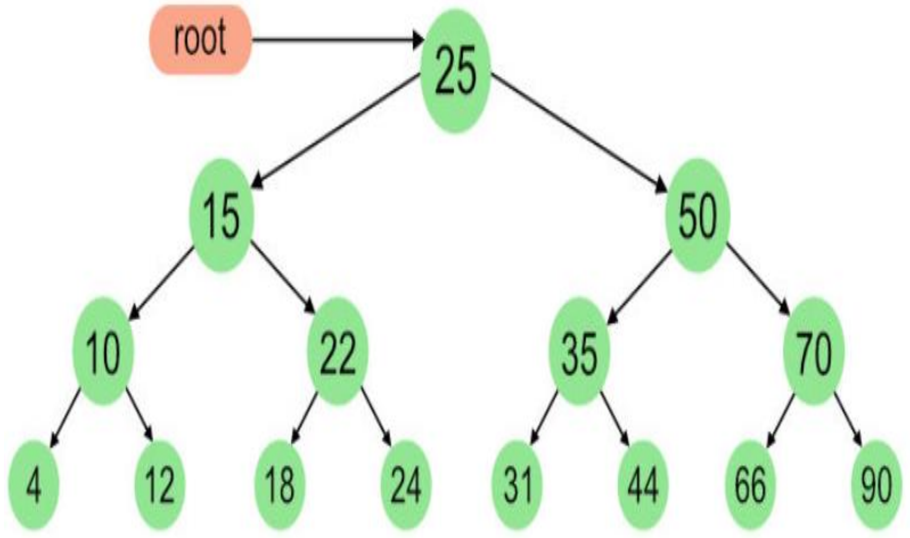
Algorithm

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

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 −

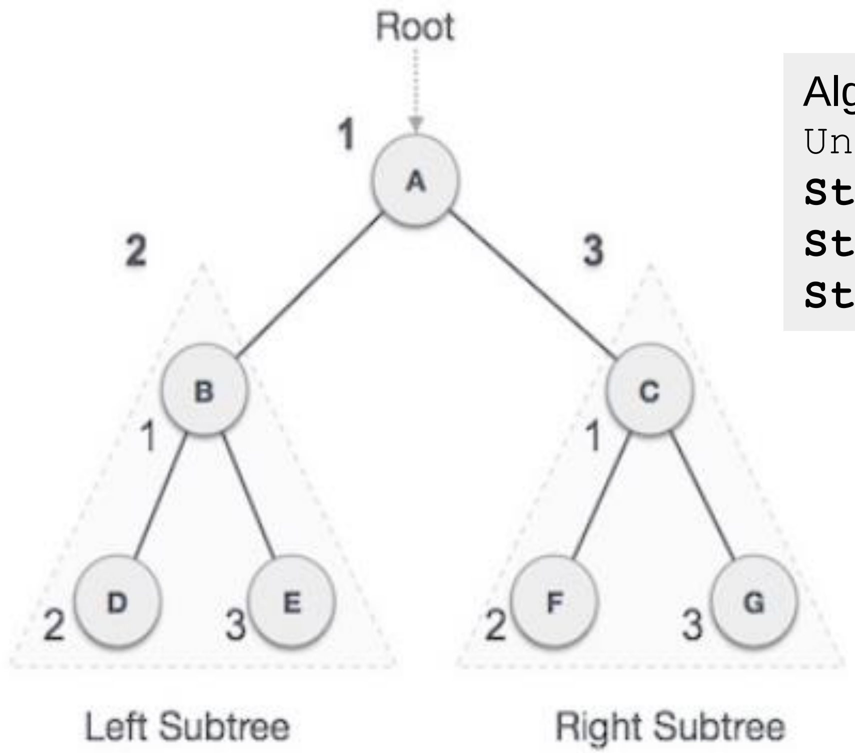
D → B → E → A → F → C → G


100
20
200
10
30
150
300


root
25
15
50
10
22
35
70
4
12
18
24
31
44
66
90

Pre-order Traversal

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



Algorithm

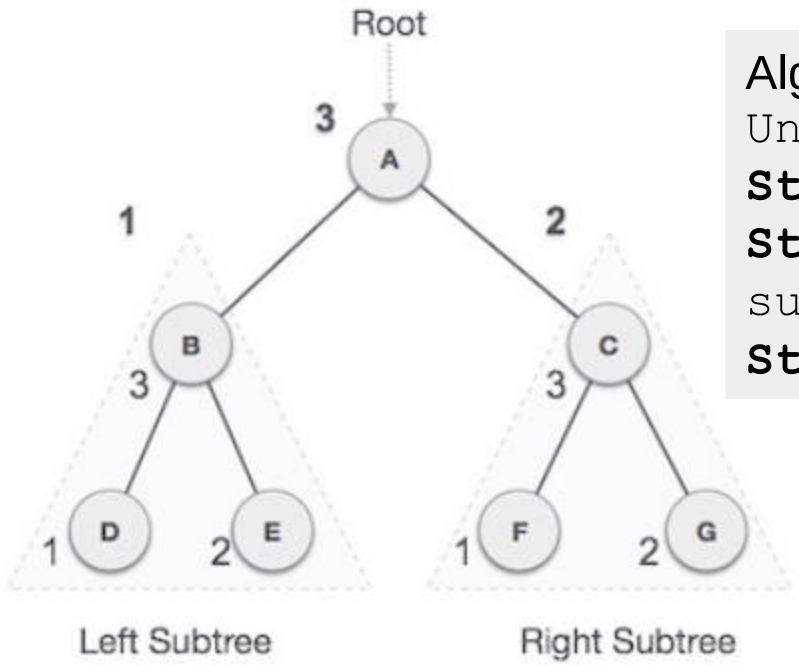
```
Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.
```

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 −

$$A \to B \to D \to E \to C \to F \to G$$

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.

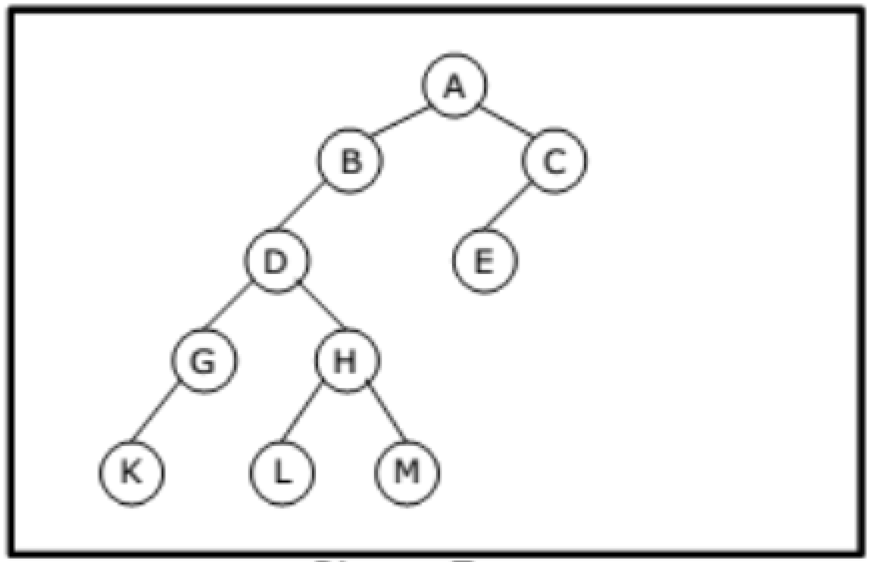


Algorithm

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

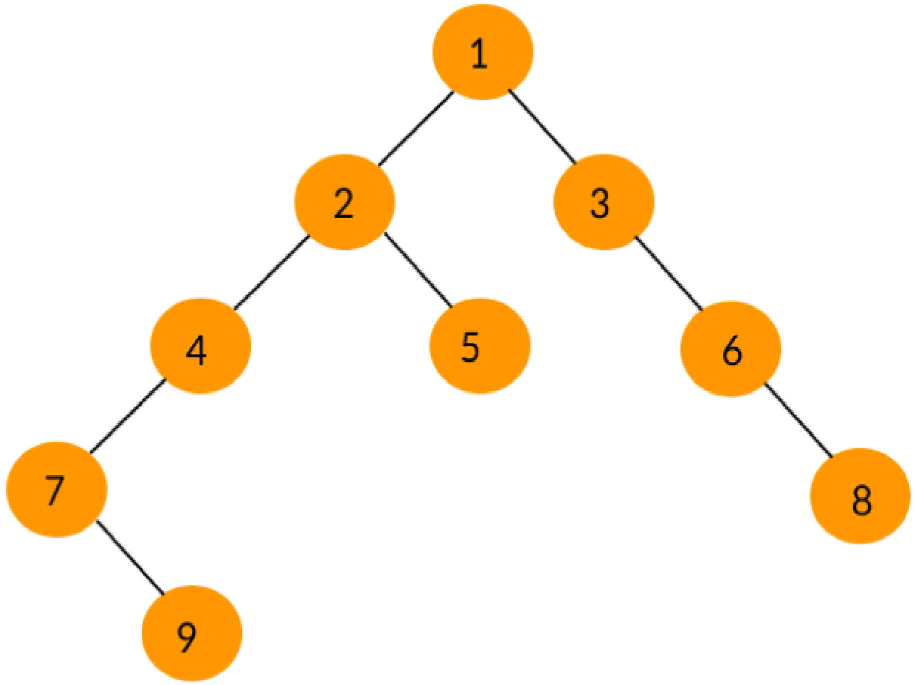
We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

$$D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$$



Binary Tree

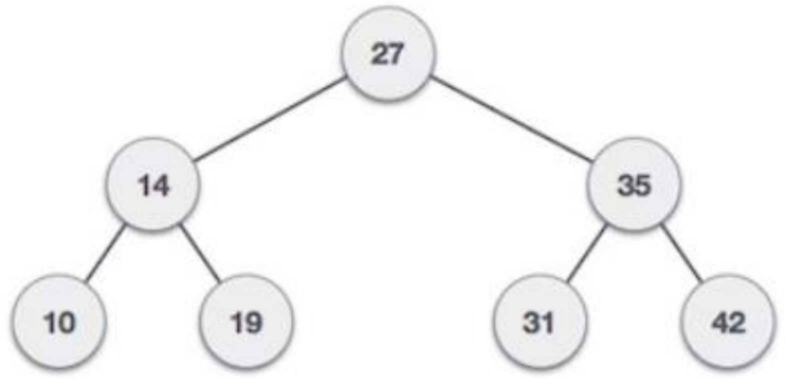
Consider the given binary tree,



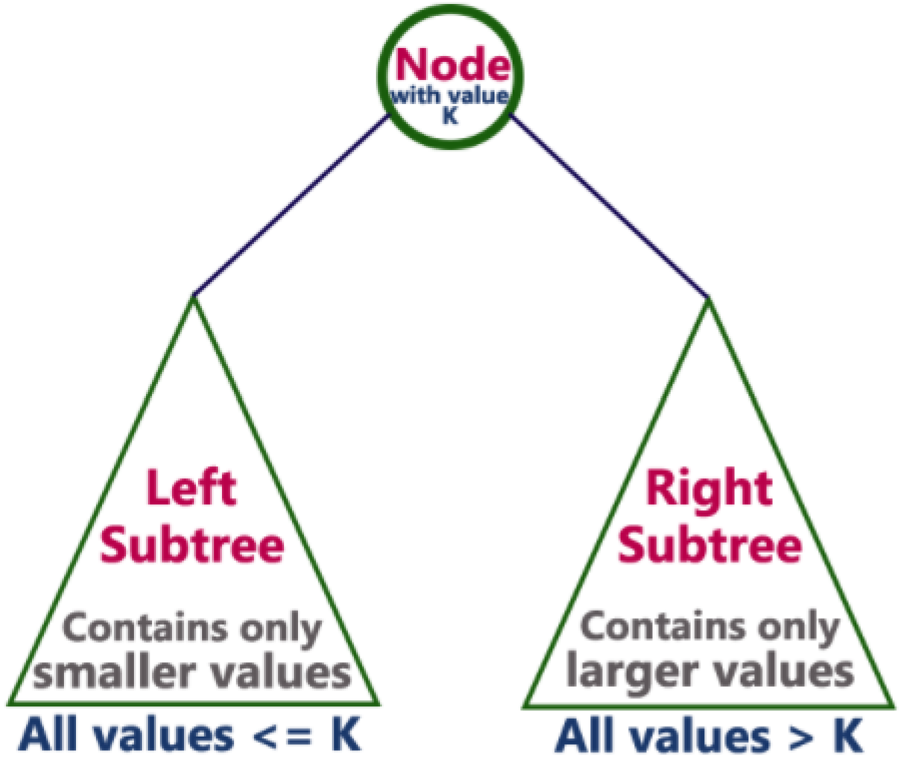
Binary search tree

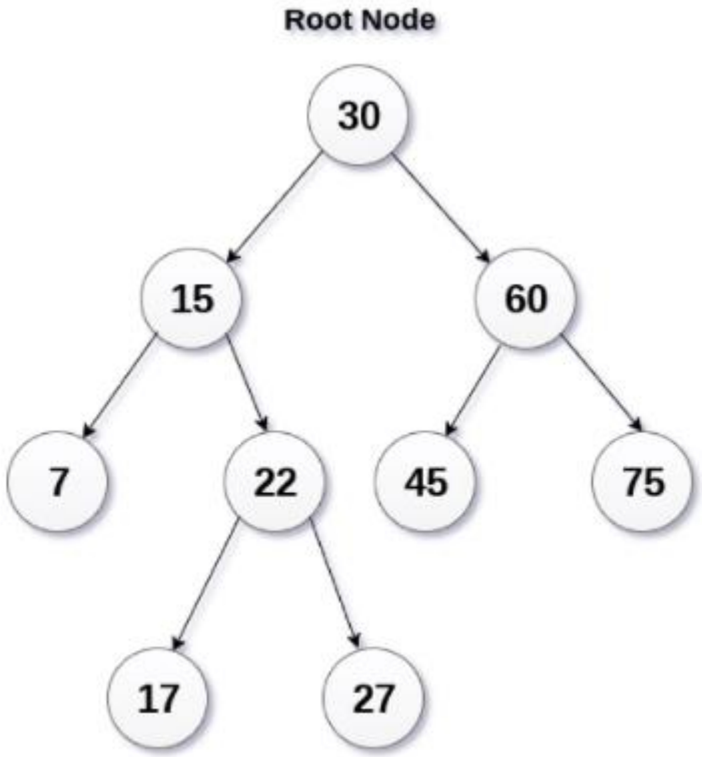
A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties −

- The value of the key of the left sub-tree is less than the value of its parent (root) node's key.
- The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.

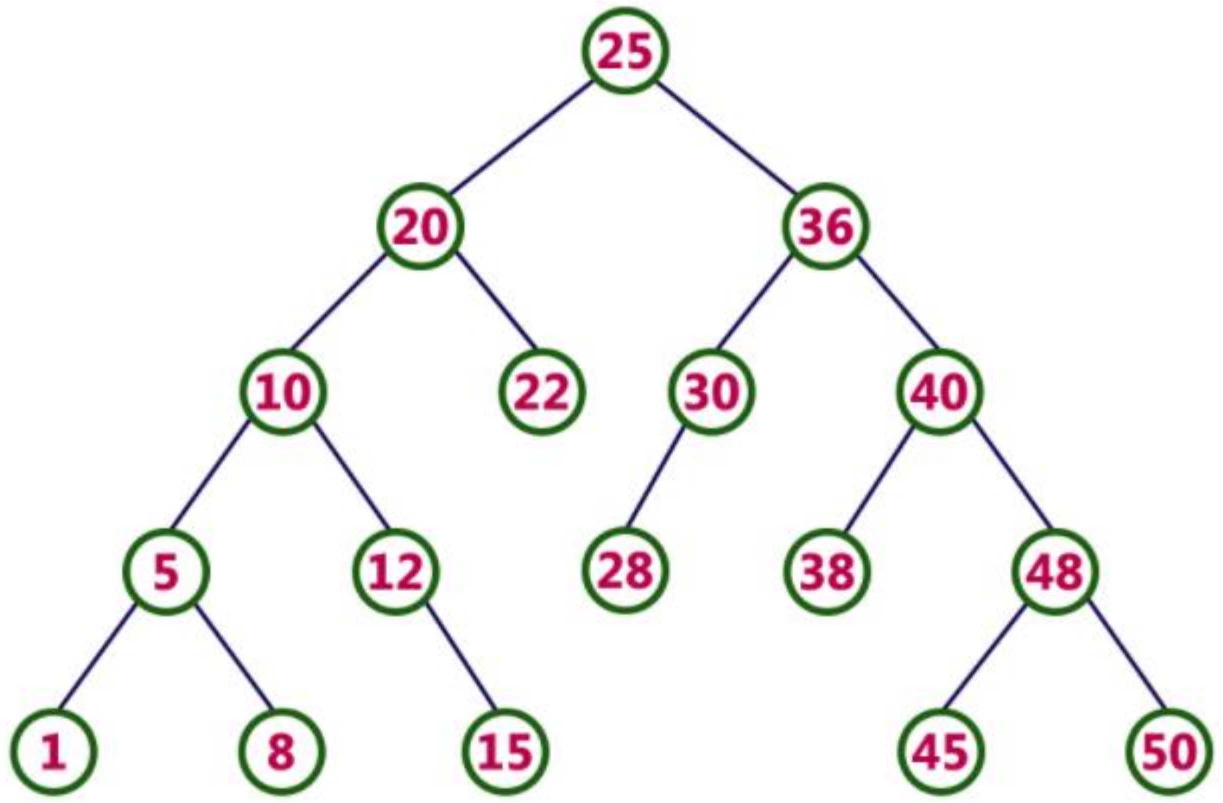


In a binary search tree, all the nodes in the left subtree of any node contains smaller values and all the nodes in the right subtree of any node contains larger values as shown in the following figure...





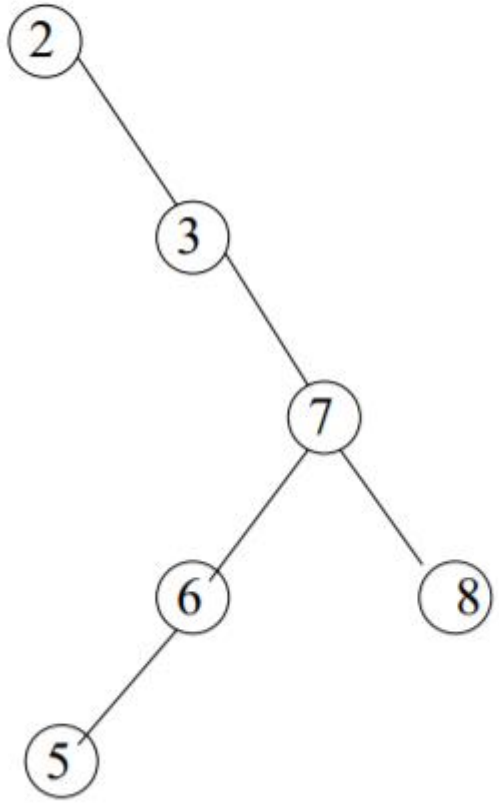
Binary Search Tree

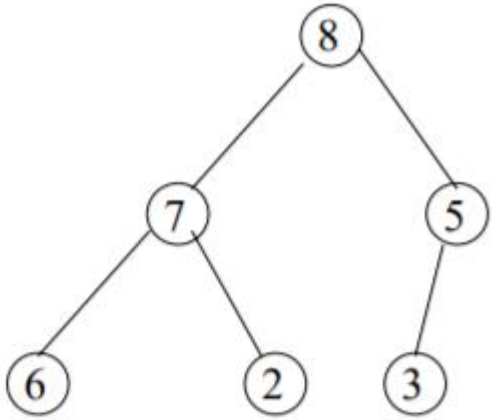


In this tree, left subtree of every node contains nodes with smaller values and right subtree of every node contains larger values.

Every binary search tree is a binary tree but every binary tree need not to be binary search tree.

Identify the binary search tree

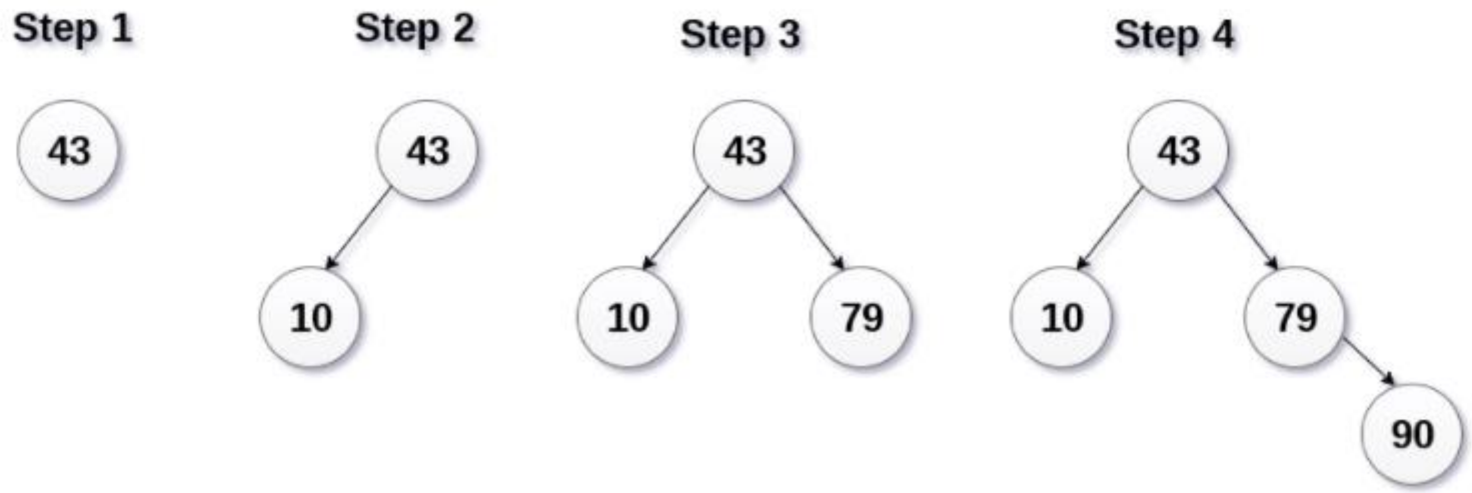


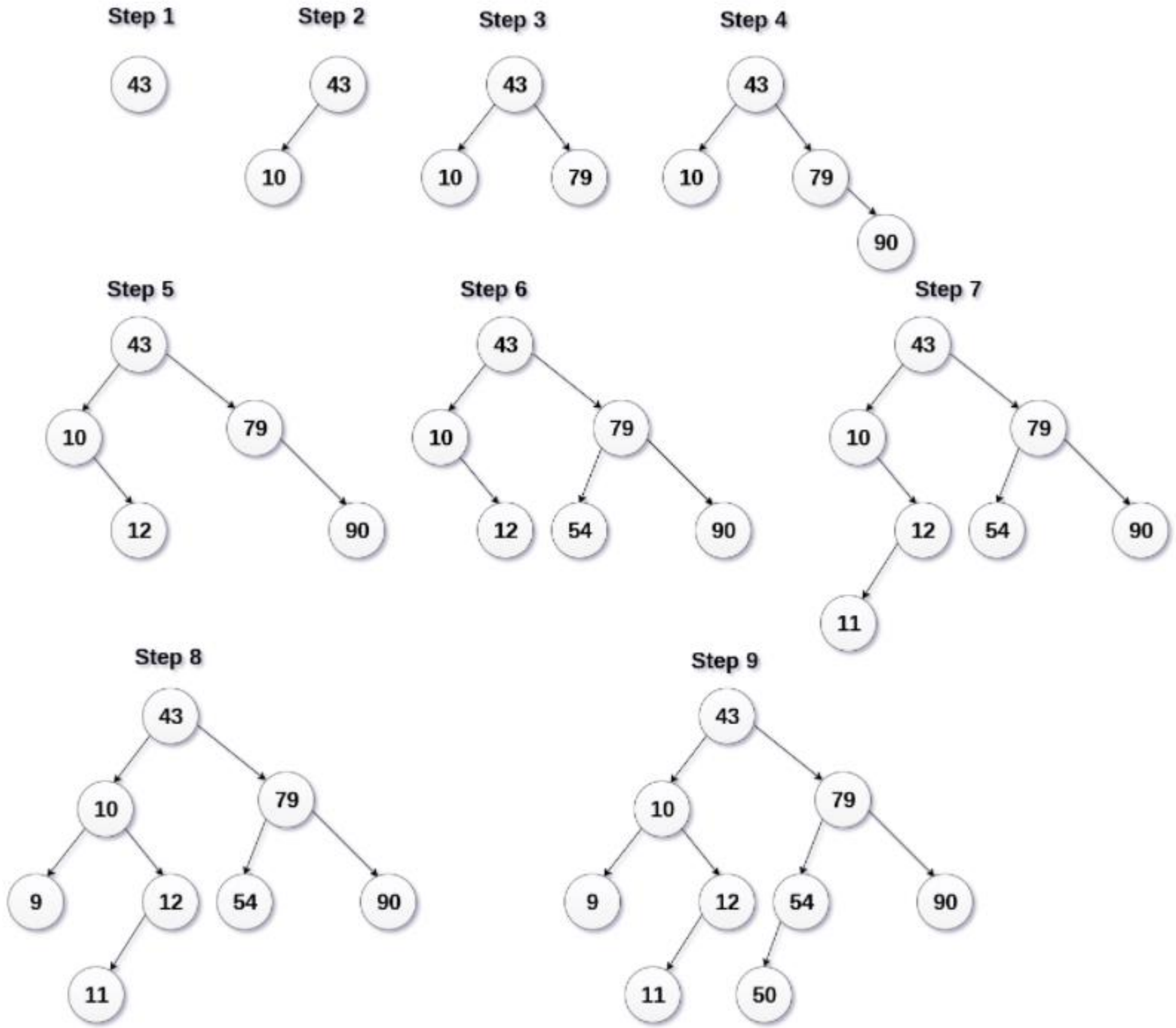


Create the binary search tree using the following data elements.

43, 10, 79, 90, 12, 54, 11, 9, 50

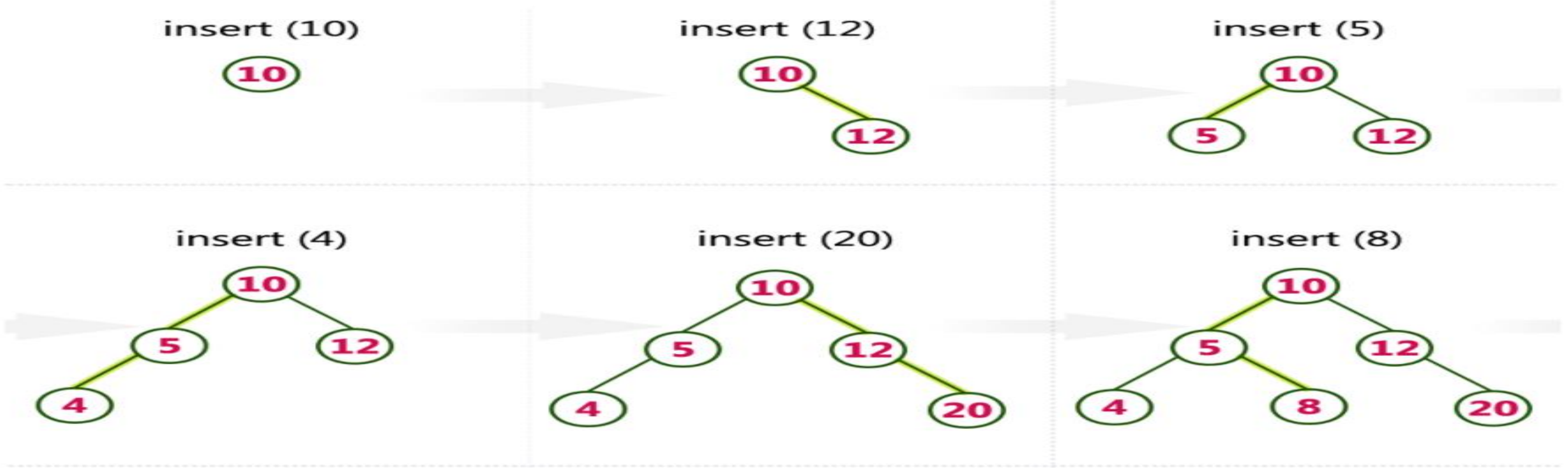
- ❑ Insert 43 into the tree as the root of the tree.
- ❑ Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
- ❑ Otherwise, insert it as the root of the right of the right sub-tree.

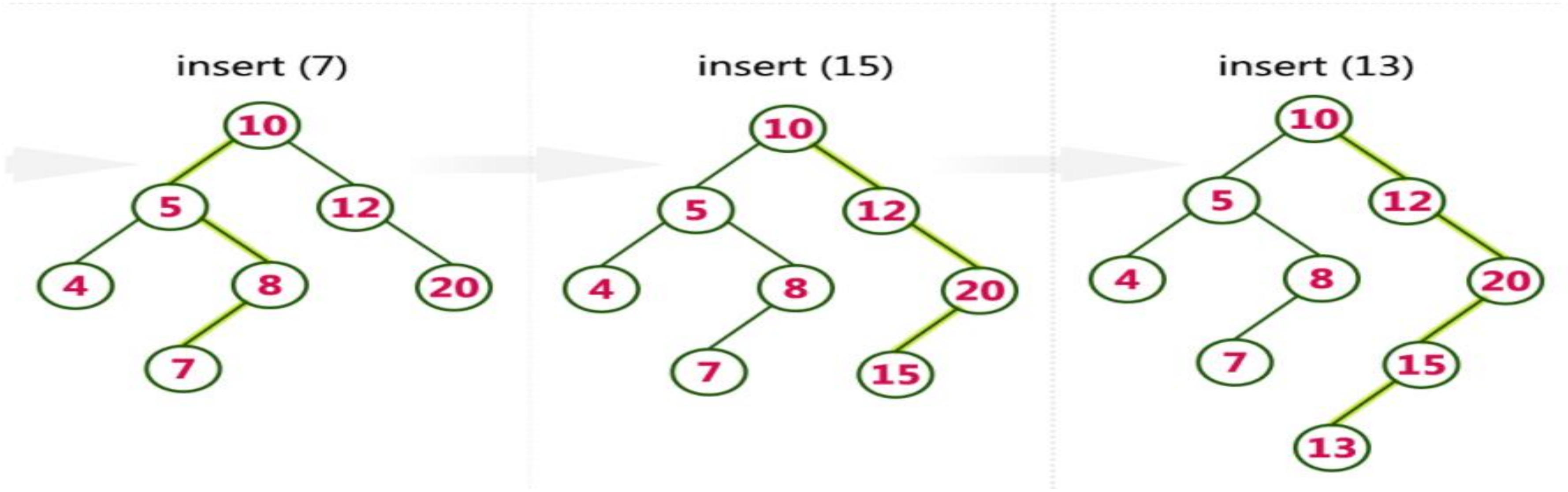



Step 1
43
Step 2
43
10
Step 3
43
10
79
Step 4
43
10
79
90
Step 5
43
10
79
12
90
Step 6
43
10
79
12
54
90
Step 7
43
10
79
12
54
90
11
Step 8
43
10
79
9
12
54
90
11
Step 9
43
10
79
9
12
54
90
11
50

Construct a Binary Search Tree by inserting the following sequence of numbers...

10,12,5,4,20,8,7,15 and 13


insert (10)
insert (12)
insert (5)
insert (4)
insert (20)
insert (8)


insert (7)
insert (15)
insert (13)

Search Operation in BST

In a binary search tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed as follows...

Step 1 - Read the search element from the user.

Step 2 - Compare the search element with the value of root node in the tree.

Step 3 - If both are matched, then display "Given node is found!!!" and terminate the function

Step 4 - If both are not matched, then check whether search element is smaller or larger than that node value.

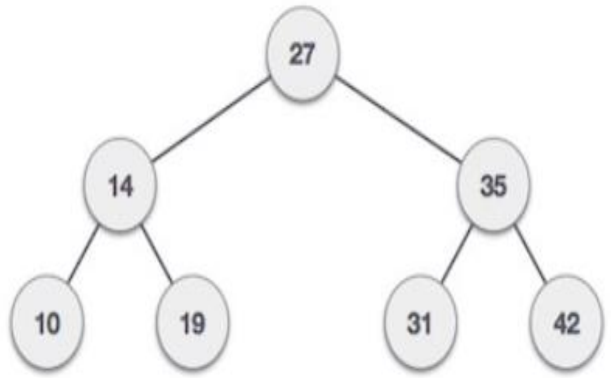
Step 5 - If search element is smaller, then continue the search process in left subtree.

Step 6- If search element is larger, then continue the search process in right subtree.

Step 7 - Repeat the same until we find the exact element or until the search element is compared with the leaf node

Step 8 - If we reach to the node having the value equal to the search value then display "Element is found" and terminate the function.

Step 9 - If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.


27
14
35
10
19
31
42