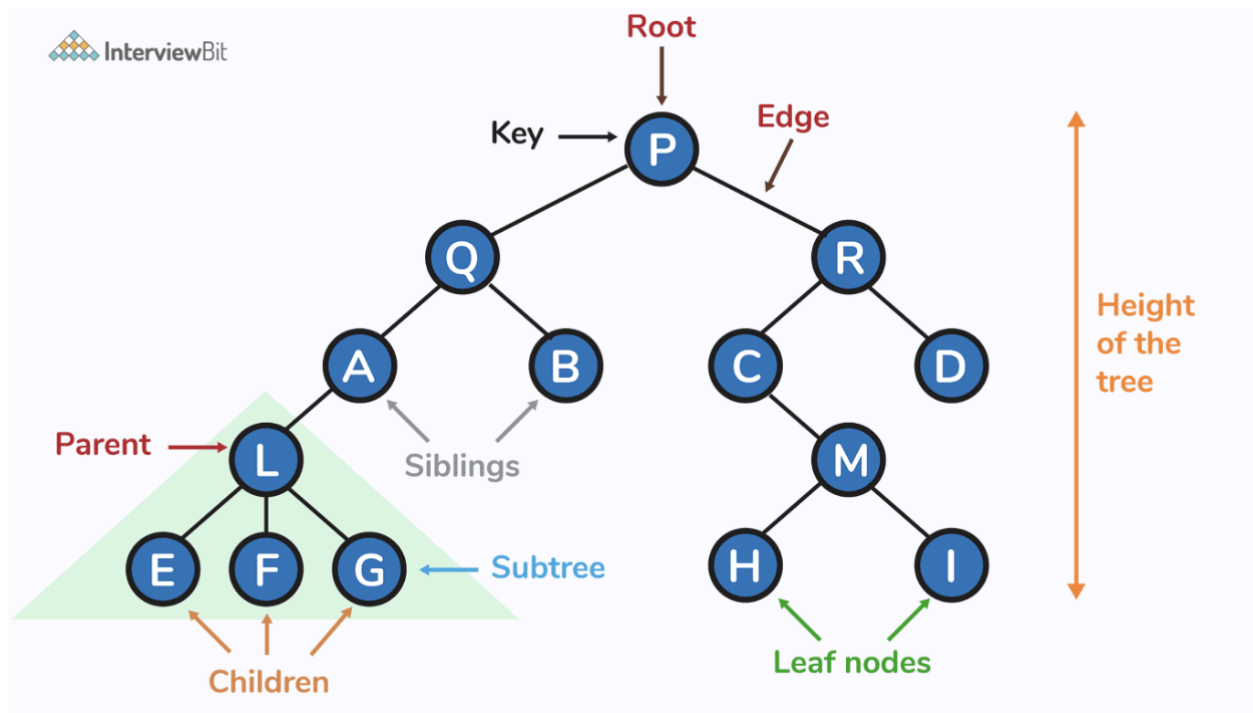


# Tree

A tree is a widely used data structure in computer science. It consists of nodes connected by edges, forming a hierarchical structure. Each node can have zero or more child nodes, except for the root node which has no parent.

Some of the applications of trees are:

- Filesystems — files inside folders that are in turn inside other folders.
- Comments on social media – comments, replies to comments, replies to replies etc form a tree representation.
- Family trees — grandparents, parents, children, grandchildren etc represent the family hierarchy.



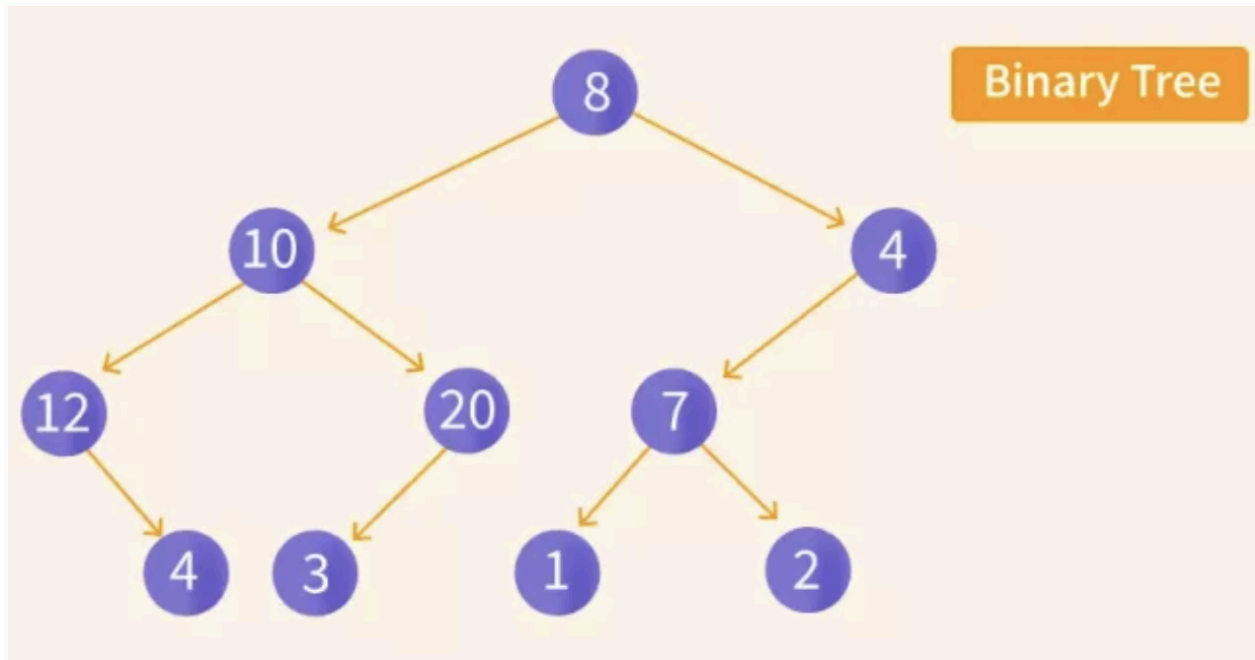
The most commonly used tree data structure are —

- Binary Tree
- Binary Search Tree

Lets try to see both these tree data structures in a more detailed manner.

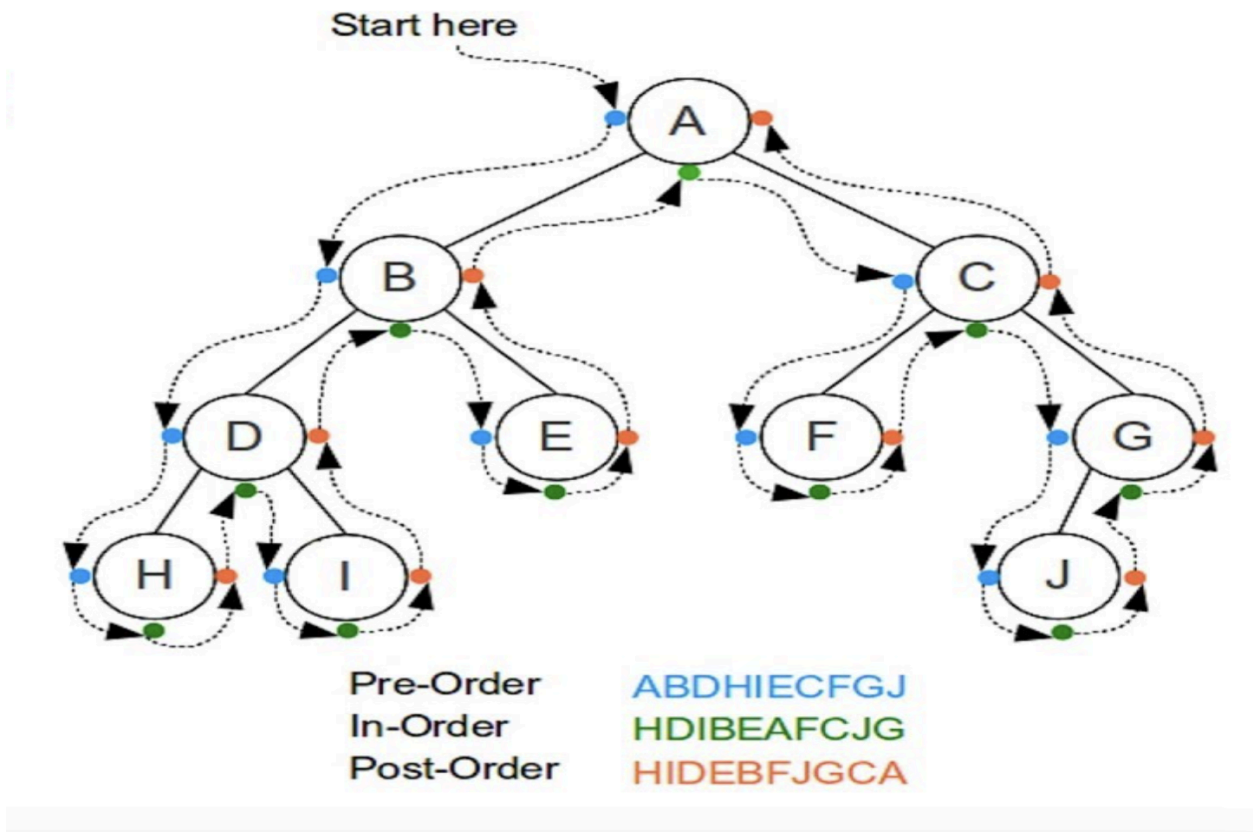
## Binary Tree

A binary tree is a tree data structure whose **all nodes have either zero, one, or at most two children nodes**. These two children are generally referred to as left and right children respectively.



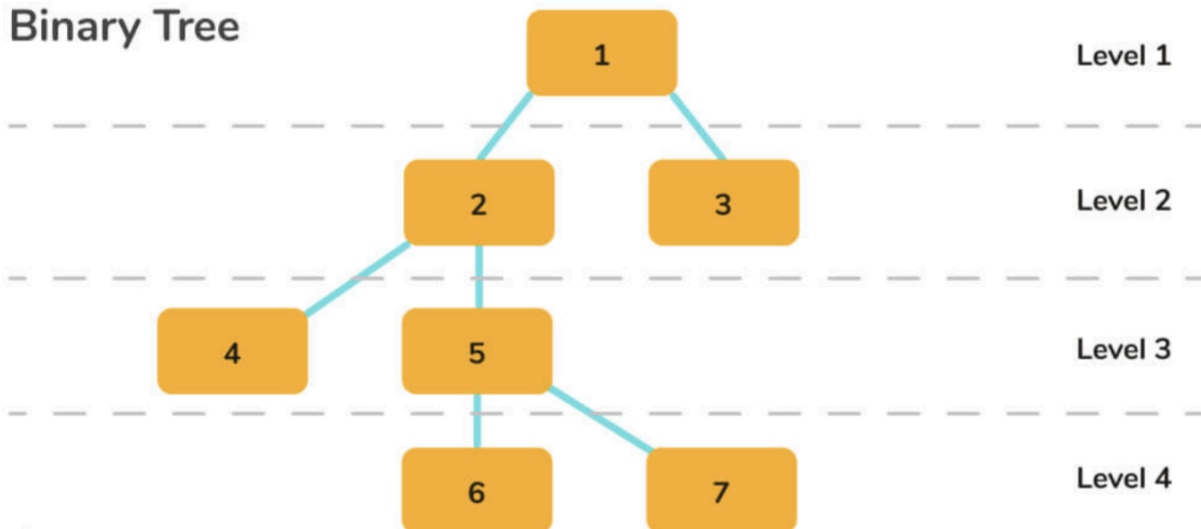
### Traversal in Binary Tree

- Preorder Traversal
- Inorder Traversal
- Postorder Traversal
- Level order Traversal



## Level Order Traversal

### Binary Tree



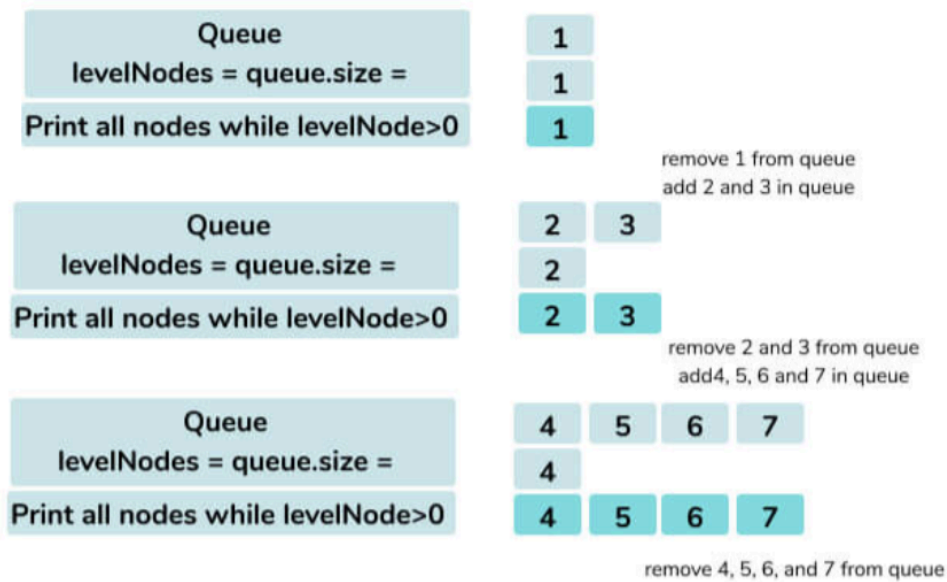
### Output

1  
2 3

4 5

6 7

### Approach :-



### Code:-

```
void printLevelOrder(root)
{
    Queue<Node> queue = new LinkedList<Node>();
    queue.add(root);
    while (!queue.isEmpty()) {
        Node tempNode = queue.poll();
        System.out.print(tempNode.data + " ");
        if (tempNode.left != null) {
            queue.add(tempNode.left);
        }
        if (tempNode.right != null) {
            queue.add(tempNode.right);
        }
    }
}
```

Practice Questions :- [Reverse Level Order](#)

Left View of Binary tree (You can find this question in **Advanced**

**Dsa : Trees 2: Views & Types** )

[Right View of Binary tree](#)

## Search in Binary Tree

The idea is to use tree traversals to traverse the tree and while traversing check if the current node matches with the given node.

- Print YES if any node matches with the given node and return back.
- If the tree is completely traversed and none of the nodes matches then print NO.

```
boolean ifNodeExists( Node root, int key)
{
    if (root == null)
        return false;
    if (root.data == key)
        return true;
    if( ifNodeExists(root.left, key) || ifNodeExists(root.right, key) )
        return true;
    return false;
}
```

## Src node to root node Path

**Problem statement-** Given a binary tree with distinct nodes. Store the path from the given “x” node to the root node in an arraylist.

**Approach :-**

A recursive function that traverses the different paths in the binary tree to find the required “x” node.

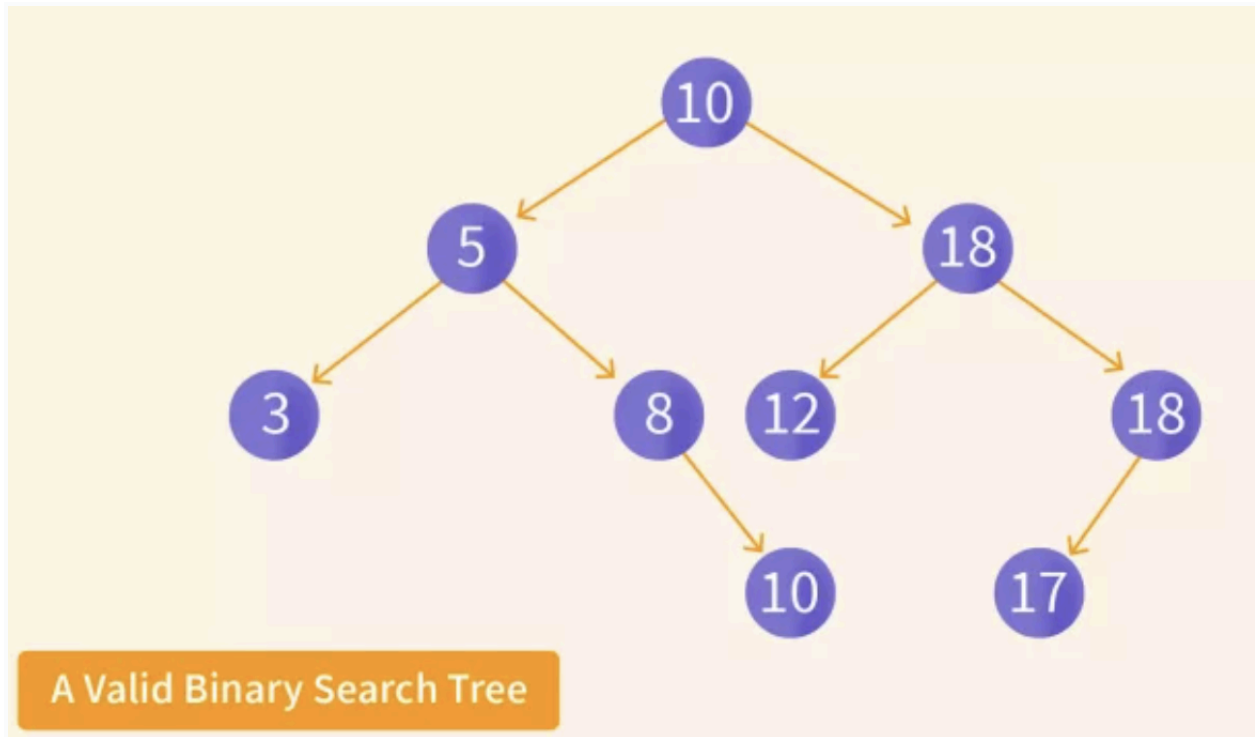
- If a node x is present then return true and accumulate the path nodes in some arraylist.
- If no path found, return empty arraylist

```
ArrayList<Node> path = new ArrayList<>();
boolean ifNodeExists( Node root, int key) {
    if (root == null)
        return false;
    if (root.data == key){
        path.add(root);
        return true;
    }
    if( ifNodeExists(root.left, key) || ifNodeExists(root.right, key)) {
        path.add(root);
        return true;
    }
    return false;
}
```

Practice Questions - [Symmetric binary tree](#)

## Binary Search Tree

A binary search tree (BST) is a sorted binary tree, where we can easily search for any key using the binary search algorithm. To sort the BST, it has to have the following properties: The node's left subtree contains only a key that's smaller than the node's key.



Let's talk about the special properties of the Binary Search Tree.

- The value of all the nodes in the left subtree of the root node is less than the value of the root node.
- The value of all the nodes in the right subtree is greater than the value of the root node.
- Properties 1 and 2 are true for all the nodes of the tree.

## Insertion in BST

Let's say the element we're trying to insert is K. The algorithm for insert operation in Binary tree is as follows :-

- if node == null, create a new node with the value of the key field equal to K. We return this newly created node directly from here.
- if  $K \leq \text{node.key}$ , it means K must be inserted in the left subtree of the current node. We repeat(recur) the process from step 1 for the left subtree.
- else  $K > \text{node.key}$ , which means K must be inserted in the right subtree of the current node. We repeat(recur) the process from step 1 for the right subtree.
- Return the current node.

## Search in BST

- if(node==null) k is not present in the tree.
- if(k<node.data), it means k must be present in the left subtree of the current node. We **recur** the process from step 1 for the left subtree.
- if(k > node.data), it means k must be present in the right subtree of the current node. We **recur** the process from step 1 for the right subtree.
- if( k == node.data ) we have found k so return true.

## LCA in BST

**Problem Statement** - Given a BST , find the lowest common ancestor of two given nodes in the Binary Search Tree.

The lowest common ancestor is defined between two nodes p & q as the lowest node in a tree that has both p & q as its descendants.

### Approach-

- To find LCA, start from the root and compare it with both the given values.
  - If both values are smaller than current node's value, we move to left subtree
  - If both values are greater than current node's value, we move to right subtree
  - If neither condition is met, it means the current node is the LCA since one value is smaller and the other is greater.
  - We continue this process recursively until we find the LCA or reach a null position.

```
Node lowestCommonAncestor(Node root, Node p, Node q) {  
    If (root.val > p.val && root.val > q.val)  
        return lowestCommonAncestor(root.left,p,q);  
    If (root.val < p.val && root.val < q.val)  
        return lowestCommonAncestor(root.right,p,q);  
    return root;  
}
```

Revision video -

[https://drive.google.com/drive/folders/1I2l65U4ZuAxMsaGTkT0Lplbbnyzc8Y2B?usp=share\\_link](https://drive.google.com/drive/folders/1I2l65U4ZuAxMsaGTkT0Lplbbnyzc8Y2B?usp=share_link)



