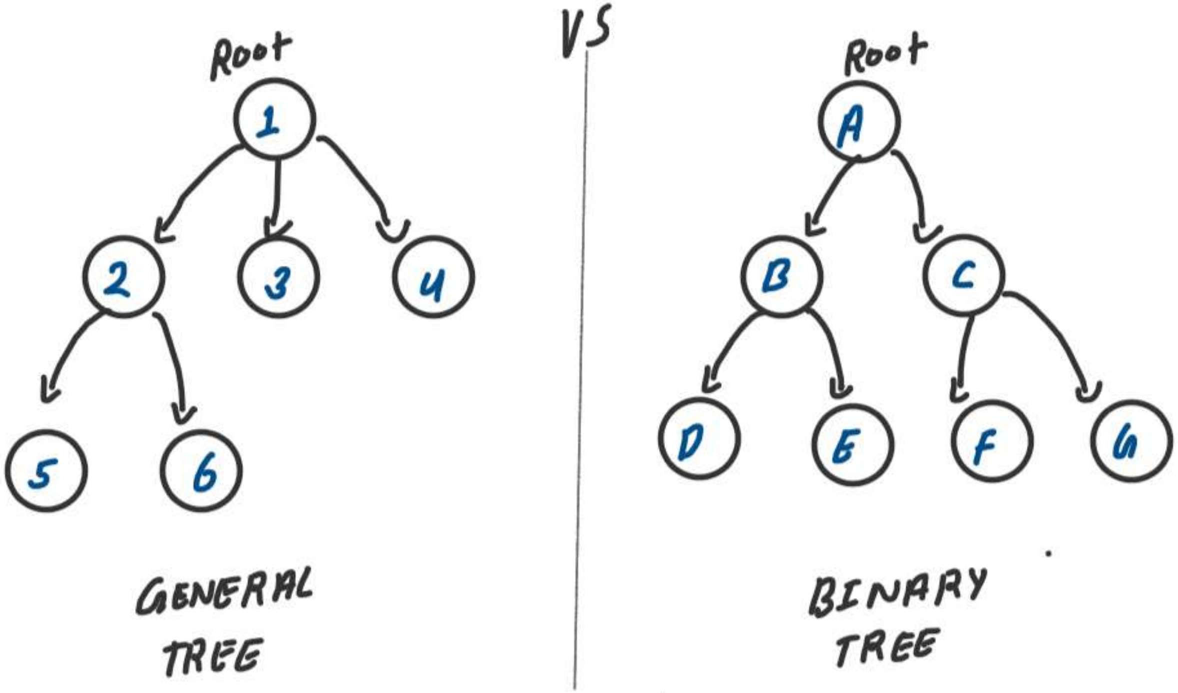
30/11/2023

BINARY TREE
CLASS - 1

📂 1. What is Binary Tree?

What is Binary Tree?

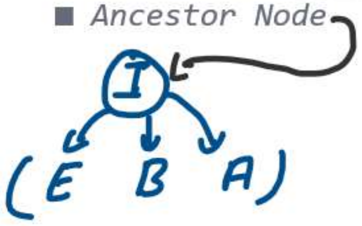
- *Binary tree is non linear data structure*
- *Binary tree is combination of nodes*
- *Binary tree have at most 2 node*



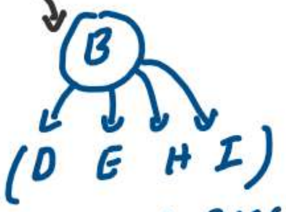
2. Terms of Binary Tree

Terms of binary tree

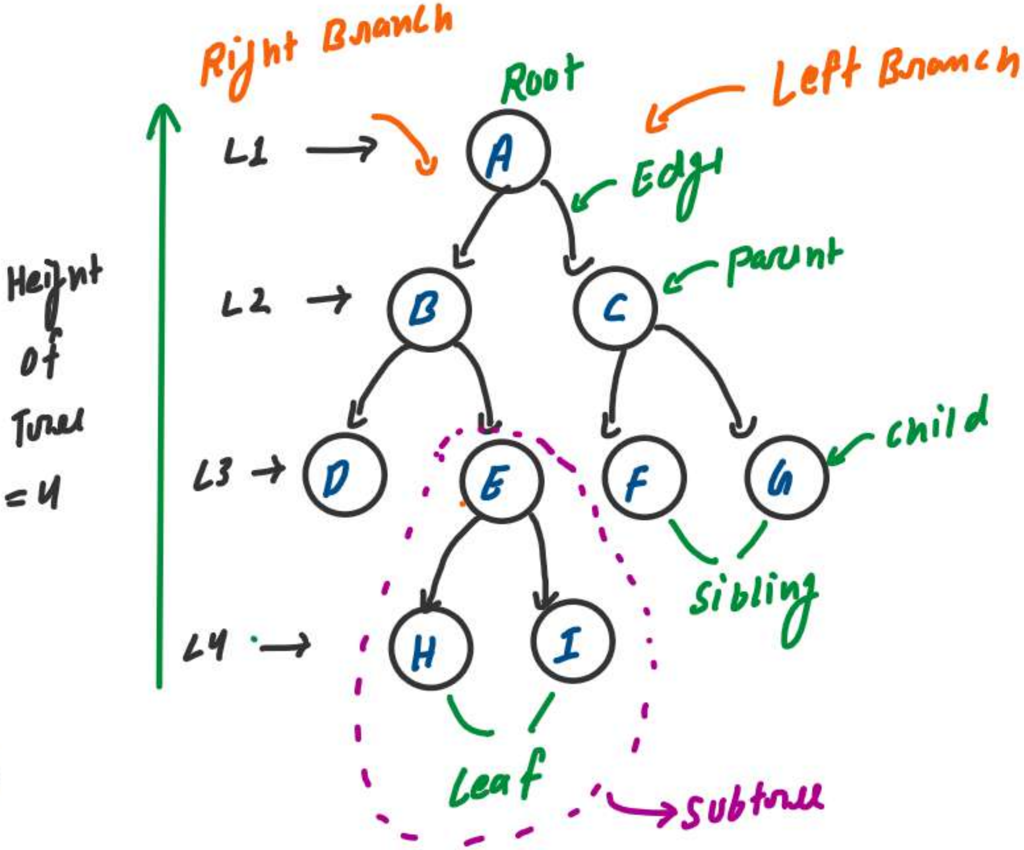
- Root Node (Starting Node) A
- Parent Node C
- Child Node F G
- Leaf Node D H I F G
- Level 4
- Edge 8
- Neighbour E, F
- Sibling (BC) (DE) (F G) (H I)
- Subtree (E H I)
- Height of tree 4
- Descendant
- Ancestor Node



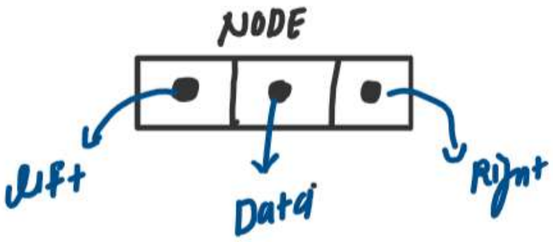
All are Ancistar for I



All are Discendant for B

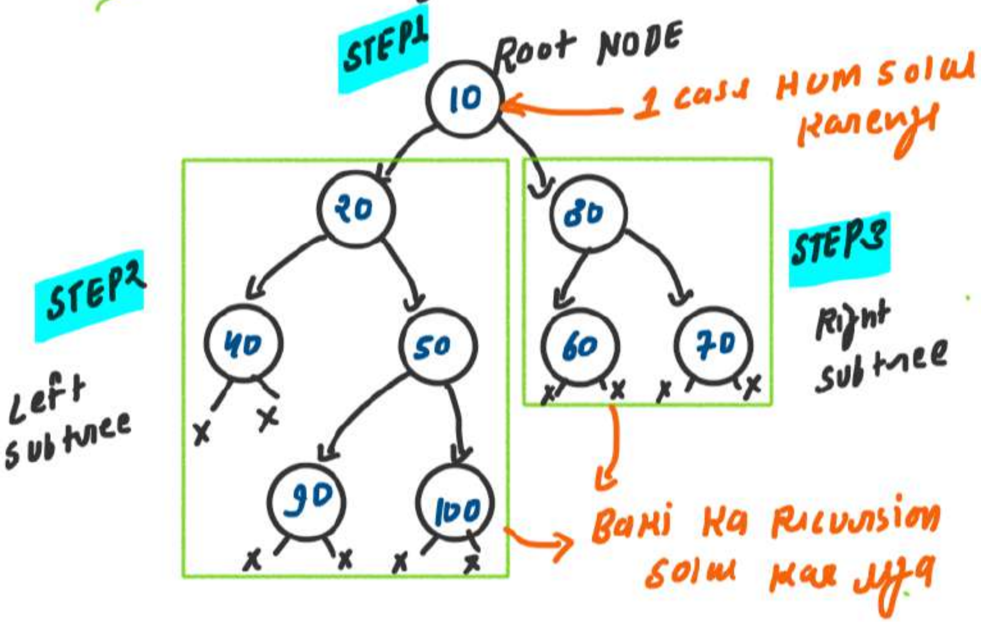


3. Implementation of Binary Tree



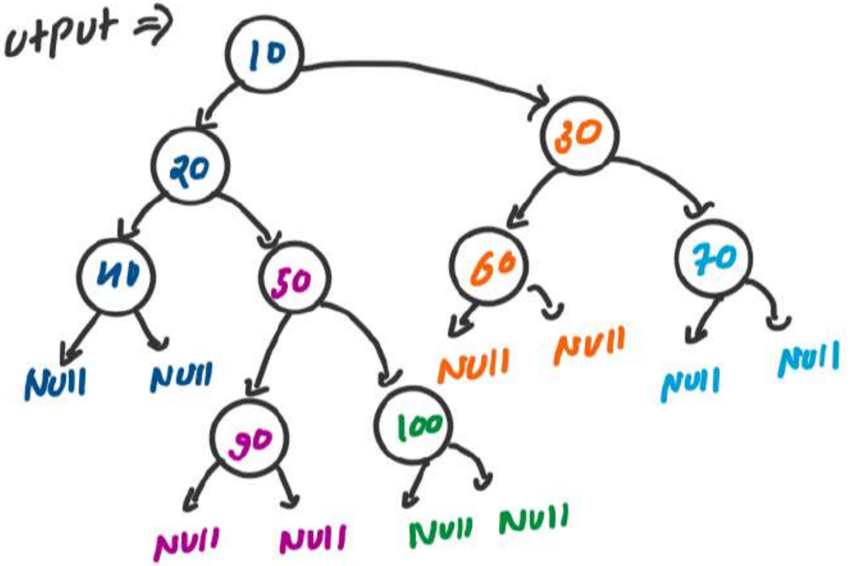
```
class Node {

    int data;
    Node* left;
    Node* right;

}
```

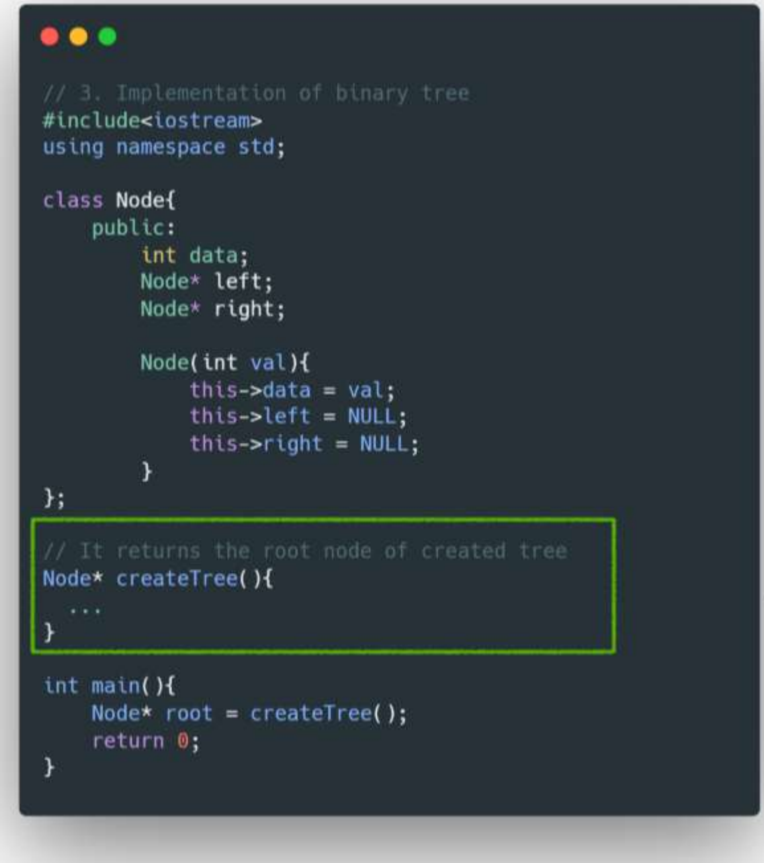
Implement Binary Tree



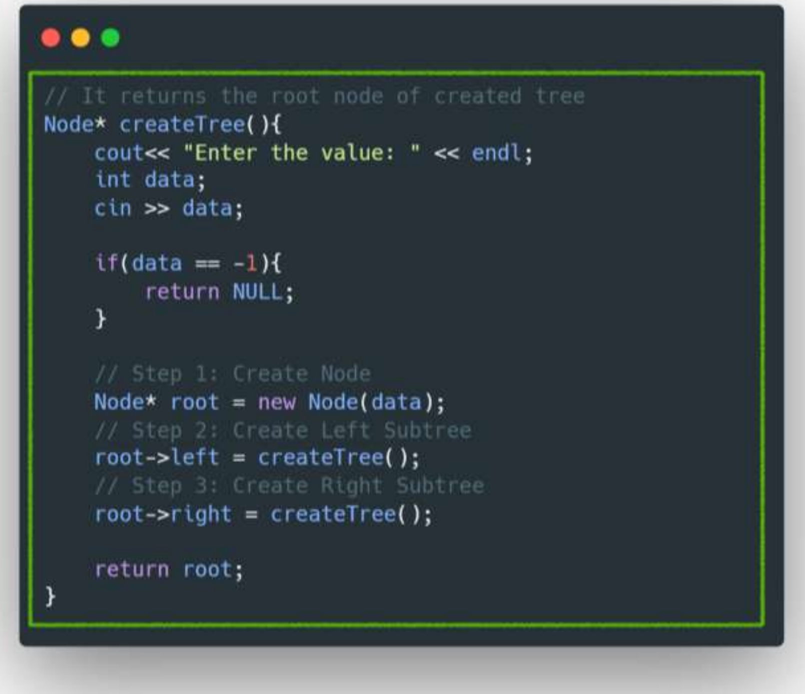
Input ⇒ 10 20 40 -1 -1 50 90 -1 -1 100 -1 -1 30 60 -1 -1 70 -1 -1

Output ⇒





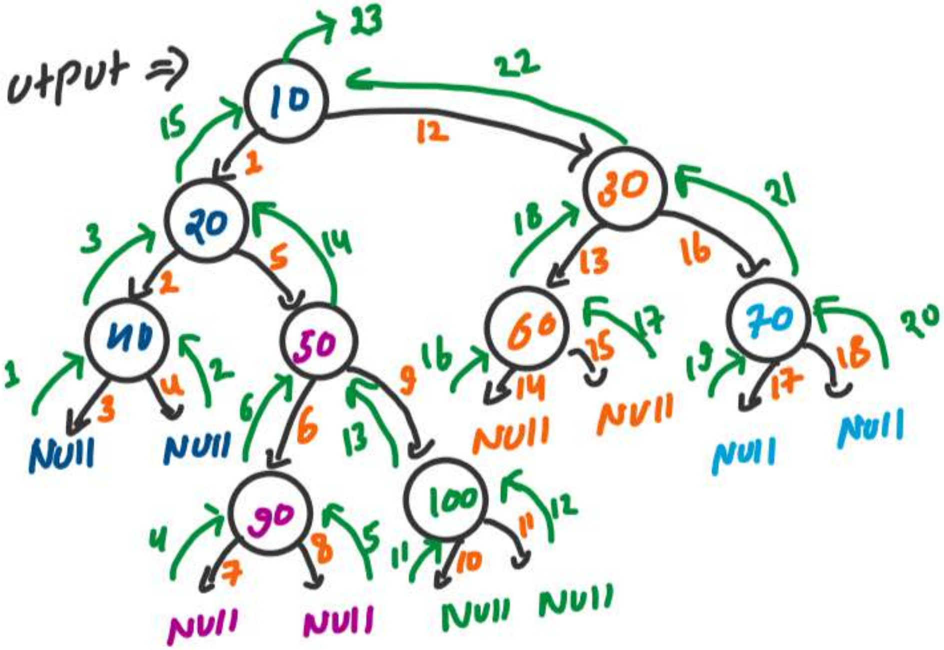
```
// 3. Implementation of binary tree
#include<iostream>
using namespace std;

class Node{
    public:
        int data;
        Node* left;
        Node* right;

        Node(int val){
            this->data = val;
            this->left = NULL;
            this->right = NULL;
        }
};
// It returns the root node of created tree
Node* createTree(){
  ...
}

int main(){
    Node* root = createTree();
    return 0;
}
```



```
// It returns the root node of created tree
Node* createTree(){
    cout<< "Enter the value: " << endl;
    int data;
    cin >> data;

    if(data == -1){
        return NULL;
    }

    // Step 1: Create Node
    Node* root = new Node(data);
    // Step 2: Create Left Subtree
    root->left = createTree();
    // Step 3: Create Right Subtree
    root->right = createTree();

    return root;
}
```

Input ⇒ 10 20 40 -1 -1 50 90 -1 -1
100 -1 -1 30 60 -1 -1 70 -1 -1

Output ⇒

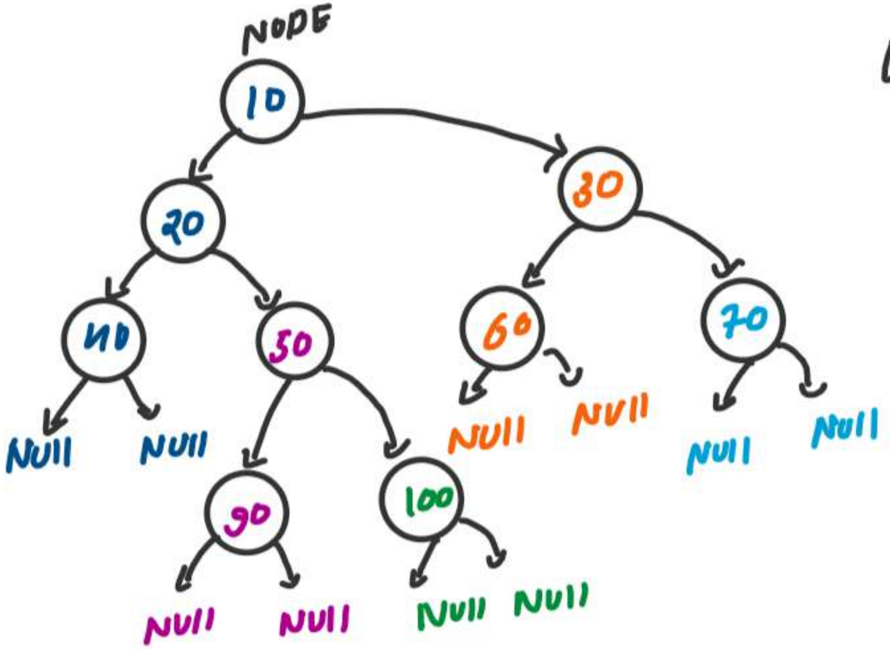


T.C. ⇒ O(N)

S.C. ⇒ O(N)

where N is total number of nodes in the B.T.

📂 4. Three Binary Tree Traversals



Where

N - Current Node (NODE Print Karna Hai)

L - Left Node

R - Right Node

I. Pre-order traversal (NLR)

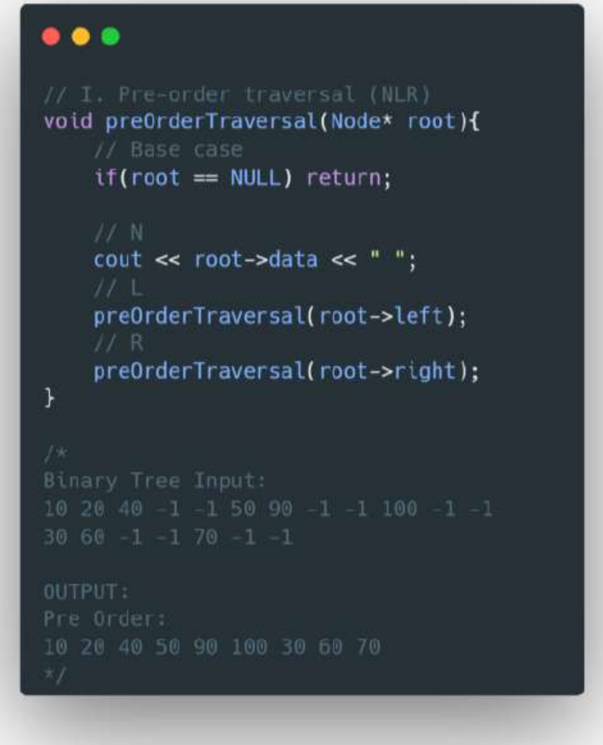
10 20 40 50 90 100 30 60 70

II. In order traversal (LNR)

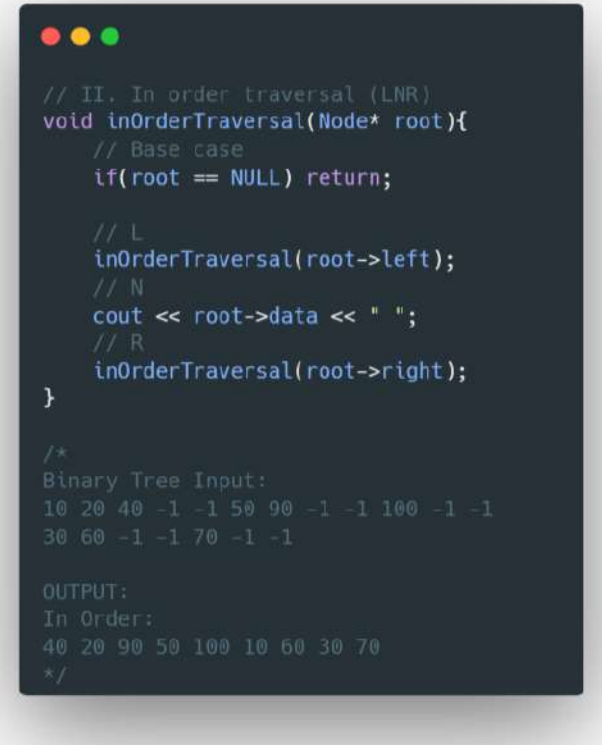
40 20 90 50 100 10 60 30 70

III. Post order traversal (LRN)

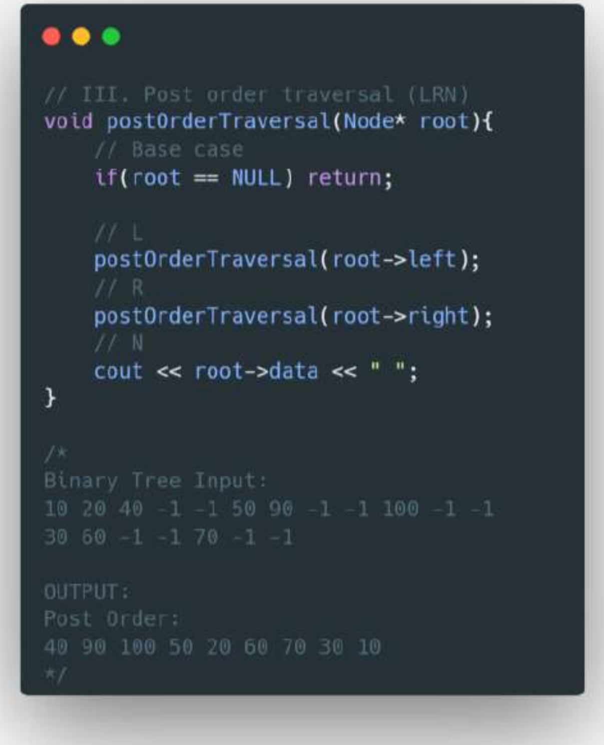
40 90 100 50 20 60 70 30 10



```
// I. Pre-order traversal (NLR)
void preOrderTraversal(Node* root){
    // Base case
    if(root == NULL) return;

    // N
    cout << root->data << " ";
    // L
    preOrderTraversal(root->left);
    // R
    preOrderTraversal(root->right);
}

/*
Binary Tree Input:
10 20 40 -1 -1 50 90 -1 -1 100 -1 -1
30 60 -1 -1 70 -1 -1

OUTPUT:
Pre Order:
10 20 40 50 90 100 30 60 70
*/
```

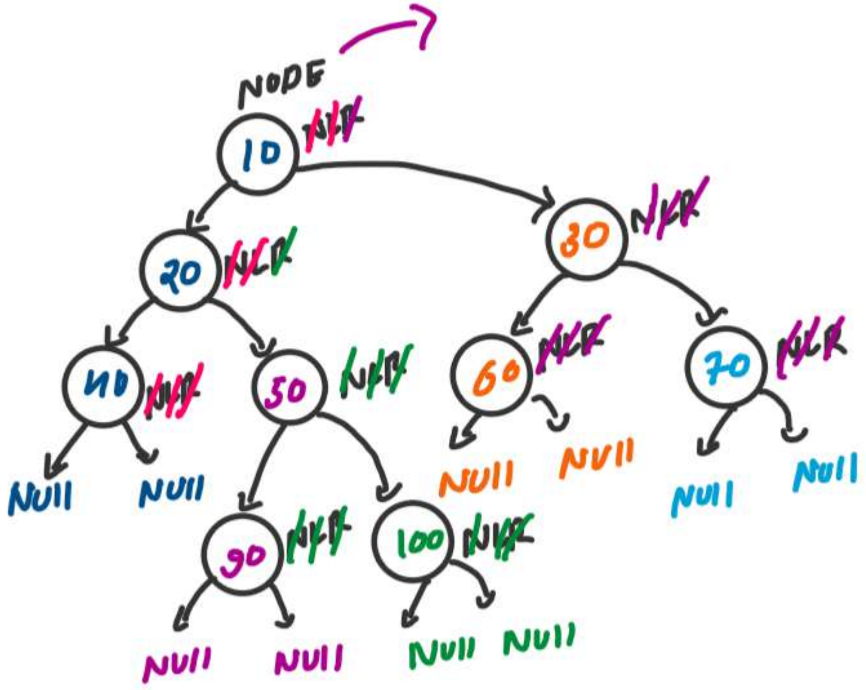


```
// II. In order traversal (LNR)
void inOrderTraversal(Node* root){
    // Base case
    if(root == NULL) return;

    // L
    inOrderTraversal(root->left);
    // N
    cout << root->data << " ";
    // R
    inOrderTraversal(root->right);
}

/*
Binary Tree Input:
10 20 40 -1 -1 50 90 -1 -1 100 -1 -1
30 60 -1 -1 70 -1 -1

OUTPUT:
In Order:
40 20 90 50 100 10 60 30 70
*/
```



```
// III. Post order traversal (LRN)
void postOrderTraversal(Node* root){
    // Base case
    if(root == NULL) return;

    // L
    postOrderTraversal(root->left);
    // R
    postOrderTraversal(root->right);
    // N
    cout << root->data << " ";
}

/*
Binary Tree Input:
10 20 40 -1 -1 50 90 -1 -1 100 -1 -1
30 60 -1 -1 70 -1 -1

OUTPUT:
Post Order:
40 90 100 50 20 60 70 30 10
*/
```

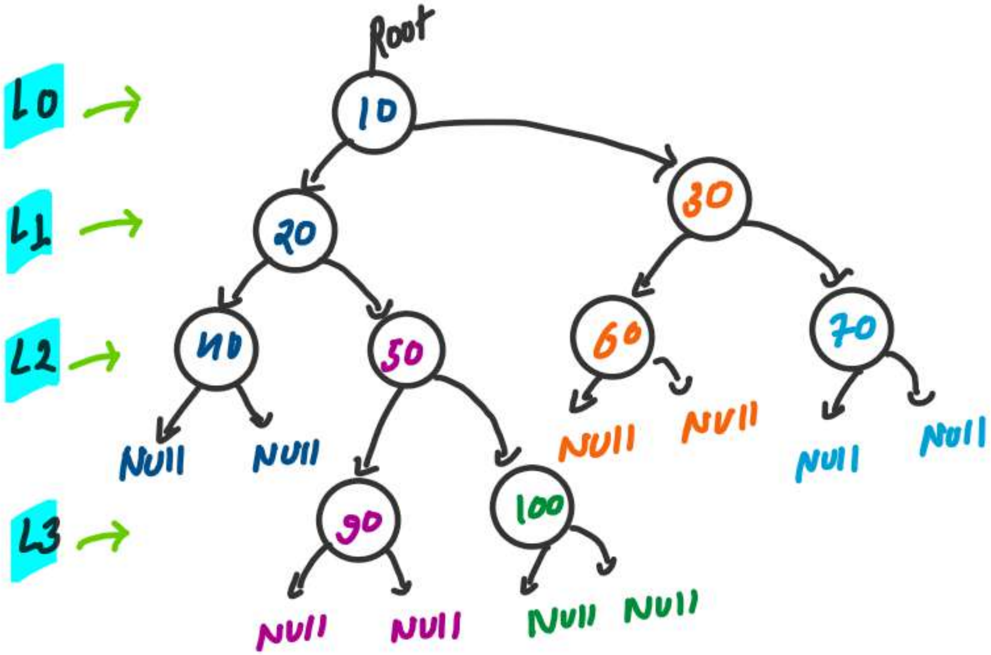
DRY RUN FOR PRE-ORDER (NLR)



PRE ORDER

10 20 40 50 90 100 30 60 70

📂 5. Level Order Traversal of Binary Tree in a Line

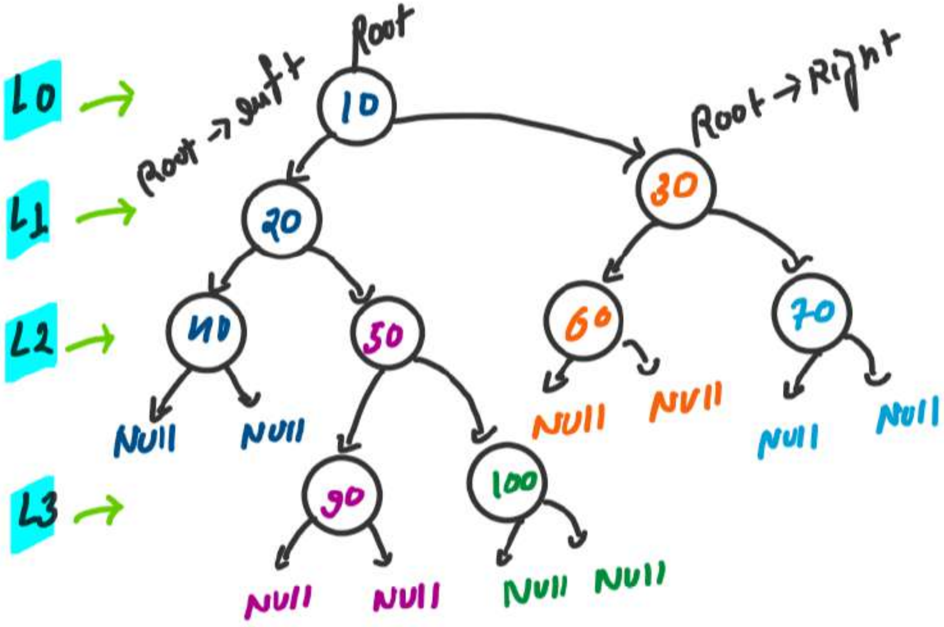


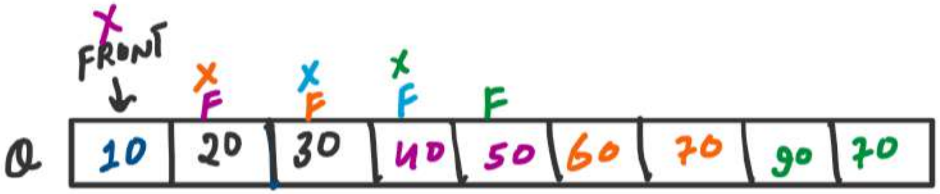
Output

$\underbrace{10}_{L0}\quad \underbrace{20\ \ 30}_{L1},\quad \underbrace{40\ \ 50\ \ 60\ \ 70}_{L2},\quad \underbrace{90\ \ 100}_{L3}$

↳ solve this problem using **Queue**

Logic Building





Q	10	20	30	40	50	60	70	90	70

```
Queue<Node*> Q;
Q.push(Root);
```

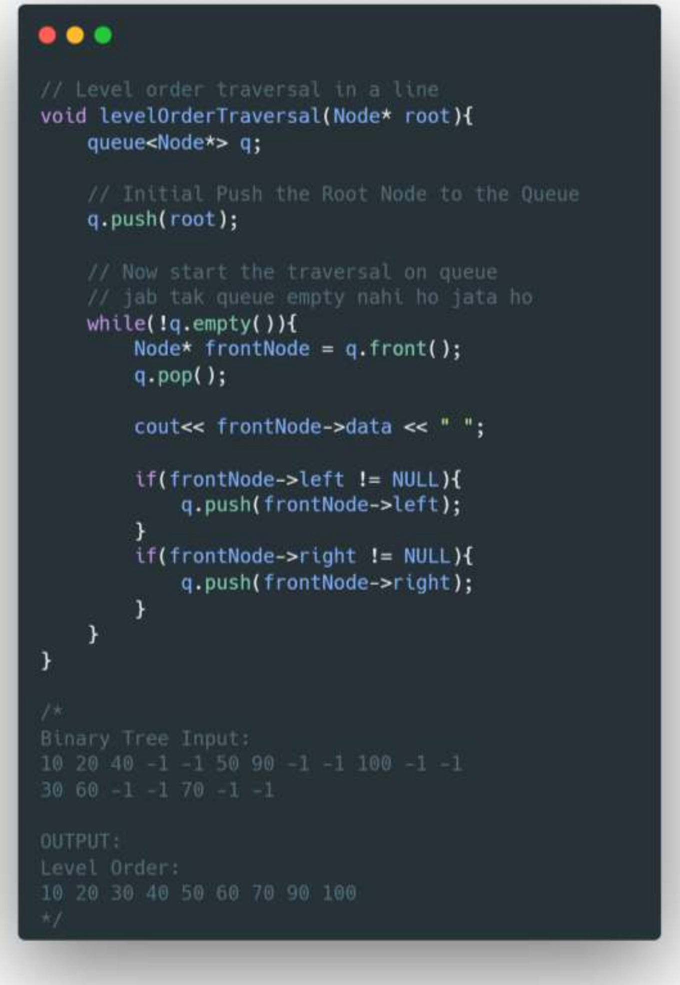
} Initialy push the Root Node in the queue

Now start the traversal on the Queue

(A) Fetch the front from Queue and pop front

(B) push front→left to Queue

(C) push front→Right to Queue

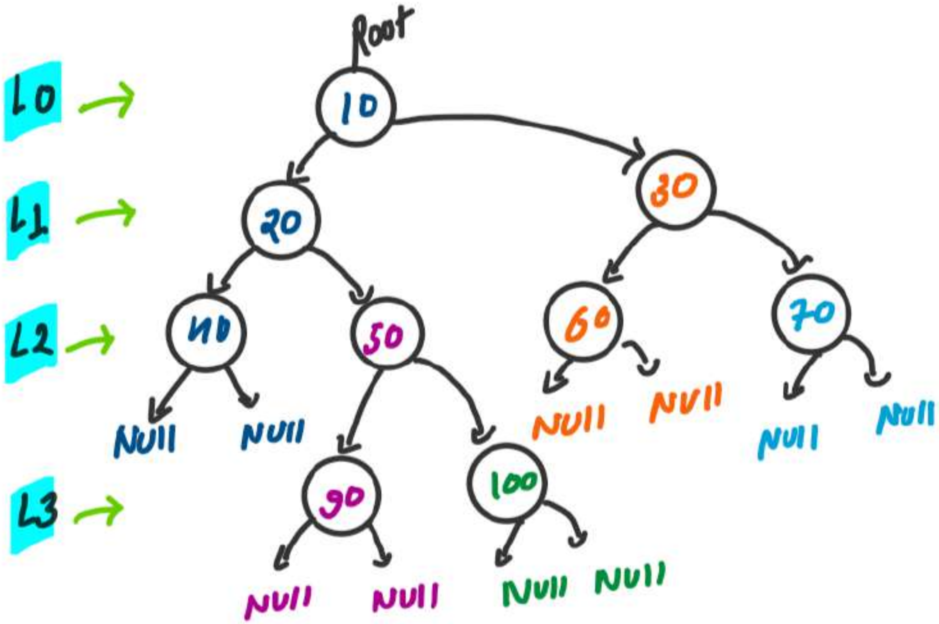


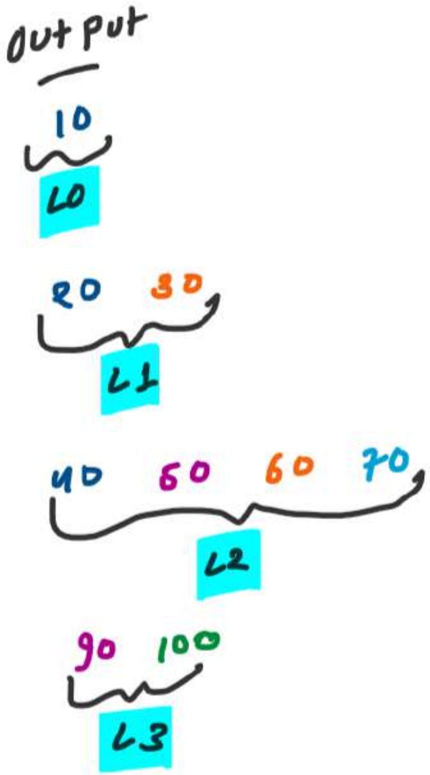
```cpp
// Level order traversal in a line
void levelOrderTraversal(Node* root){
    queue<Node*> q;

    // Initial Push the Root Node to the Queue
    q.push(root);

    // Now start the traversal on queue
    // jab tak queue empty nahi ho jata ho
    while(!q.empty()){
        Node* frontNode = q.front();
        q.pop();

        cout<< frontNode->data << " ";

        if(frontNode->left != NULL){
            q.push(frontNode->left);
        }
        if(frontNode->right != NULL){
            q.push(frontNode->right);
        }
    }
}

/*
Binary Tree Input:
10 20 40 -1 -1 50 90 -1 -1 100 -1 -1
30 60 -1 -1 70 -1 -1

OUTPUT:
Level Order:
10 20 30 40 50 60 70 90 100
*/
```

Time Complexity: *O(N),*
where N is total number of nodes in binary tree

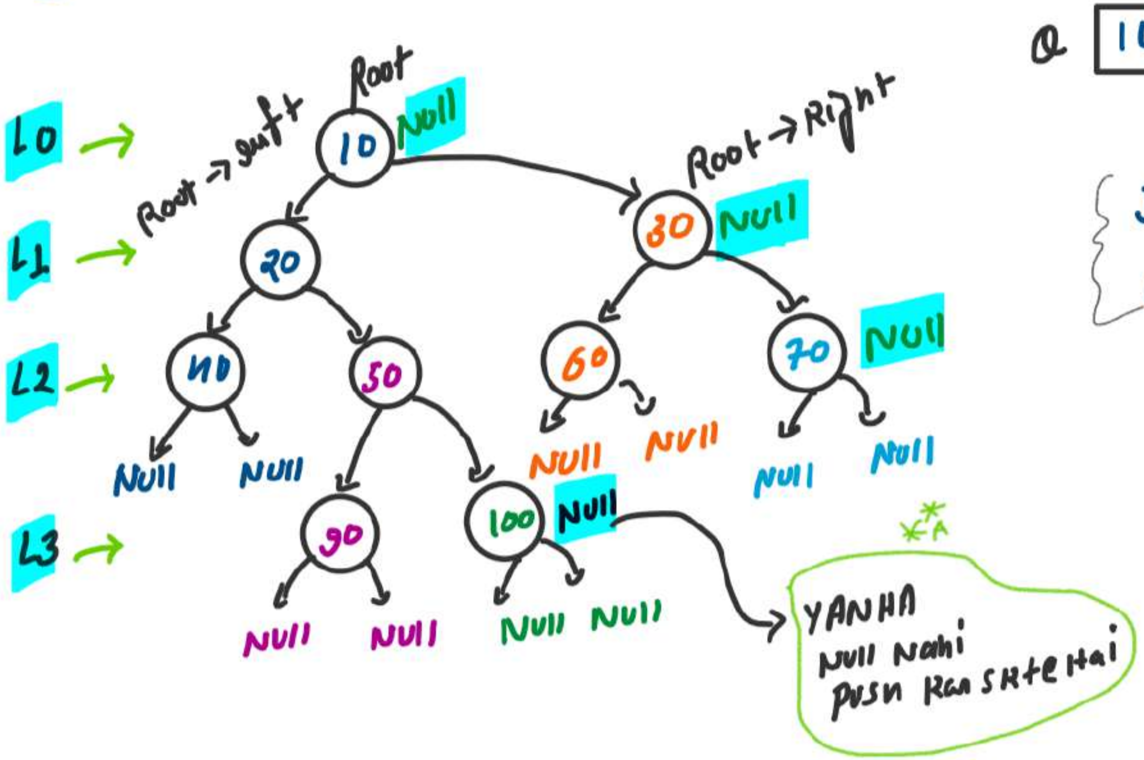
Space Complexity: *O(L),*
where L is maximum number of nodes in the level of binary tree

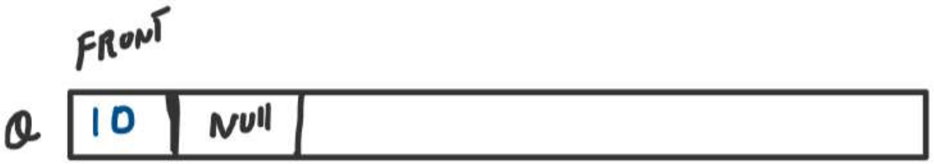
6. Level Order Traversal of Binary Tree in level wise





Logic Building





{ Jab 1 Level complete Hoga to
Usi Null se mark Kardo }

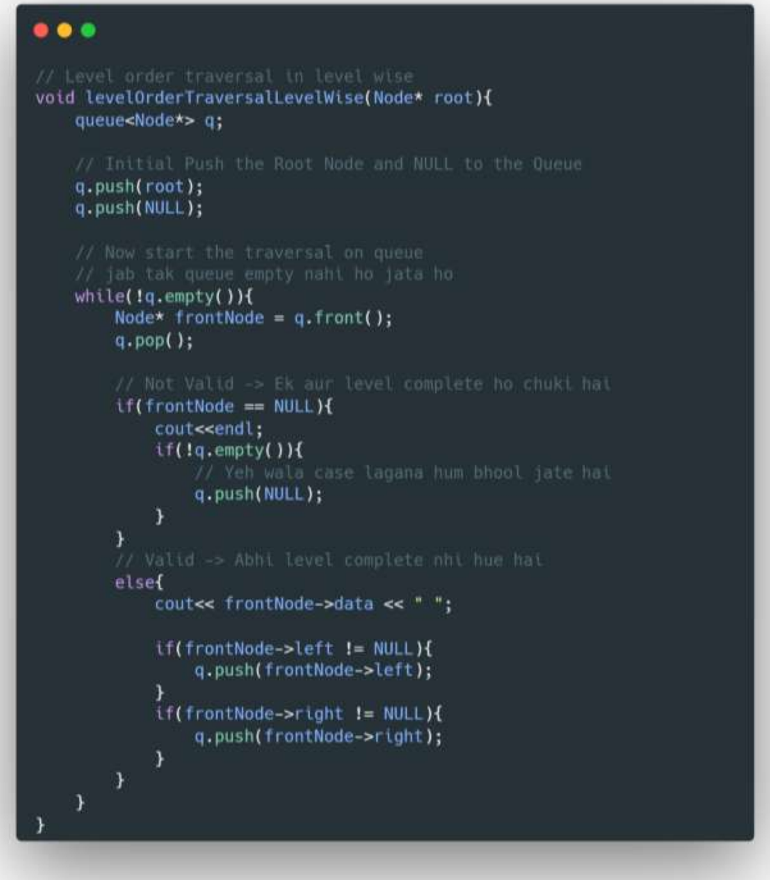
Initialy

```
Q.push(Root);
Q.push(Null);
```

Traversal

```
Q.pop()
front [NULL]
  ↳ cout<< endl
    Q.push(NULL)
```

```
Q.pop()
Front [NULL]
  ↳ Q.push(front→left)
  ↳ Q.push(front→Right)
```



```cpp
// Level order traversal in level wise
void levelOrderTraversalLevelWise(Node* root){
    queue<Node*> q;

    // Initial Push the Root Node and NULL to the Queue
    q.push(root);
    q.push(NULL);

    // Now start the traversal on queue
    // jab tak queue empty nahi ho jata ho
    while(!q.empty()){
        Node* frontNode = q.front();
        q.pop();

        // Not Valid -> Ek aur level complete ho chuki hai
        if(frontNode == NULL){
            cout<<endl;
            if(!q.empty()){
                // Yeh wala case lagana hum bhool jate hai
                q.push(NULL);
            }
        }
        // Valid -> Abhi level complete nhi hue hai
        else{
            cout<< frontNode->data << " ";

            if(frontNode->left != NULL){
                q.push(frontNode->left);
            }
            if(frontNode->right != NULL){
                q.push(frontNode->right);
            }
        }
    }
}
```

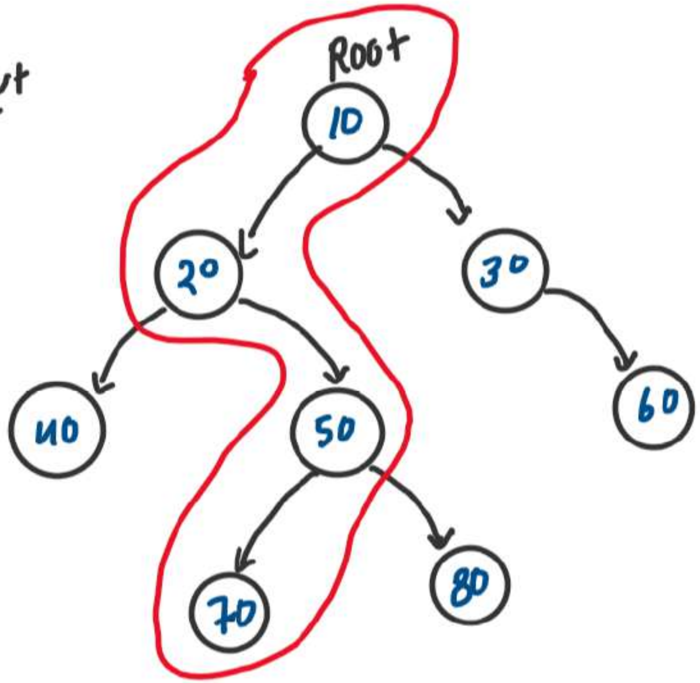
Time Complexity: *O(N),*
where N is total number of nodes in binary tree

Space Complexity: *O(L),*
where L is maximum number of nodes in the level of binary tree

7. Height of Binary Tree (Leetcode-104)

Ex1

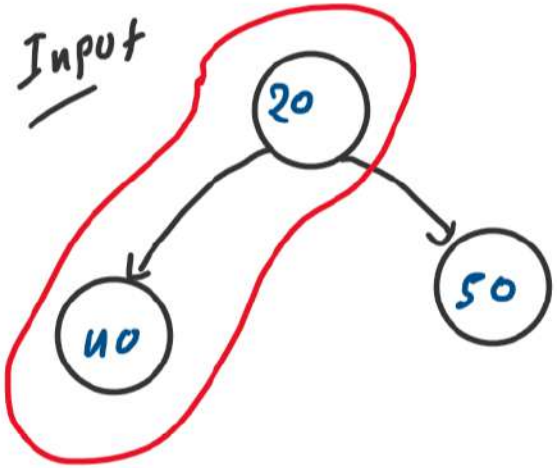
Input



Output

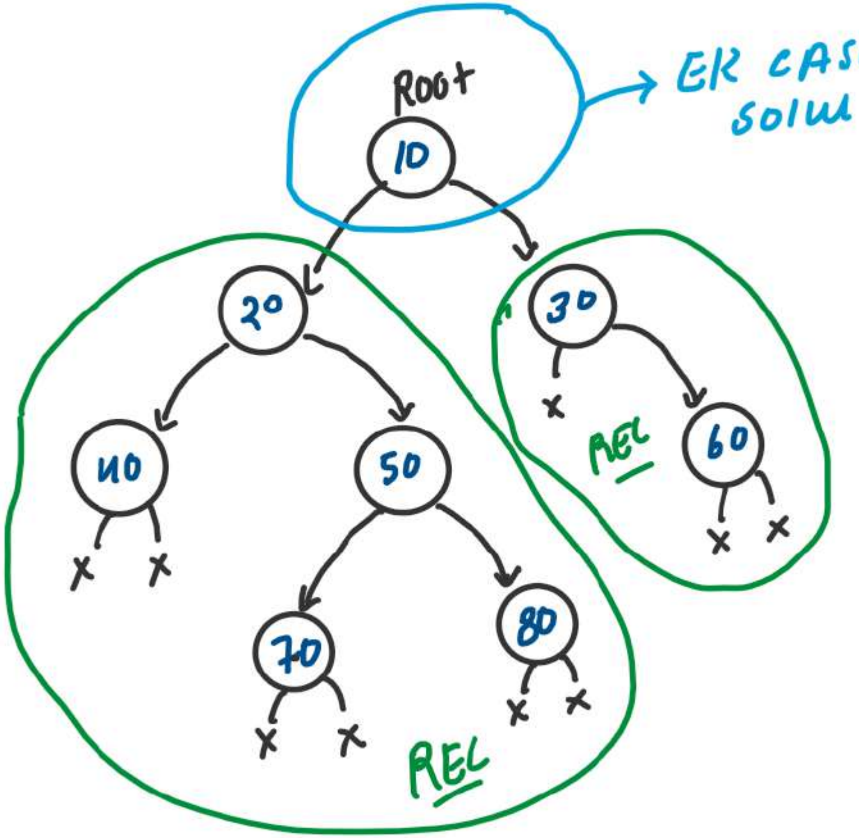
Height = 4

Input



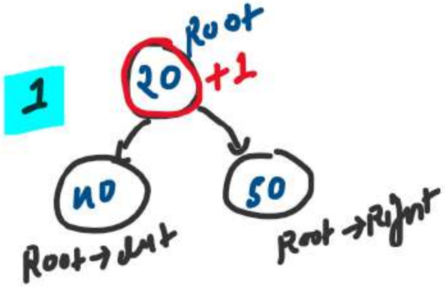
Output

Height = 2



EK CASE HUM solve kar lenge

Height = 1



Left = 1

Right = 1

Height = max(1,1) + 1

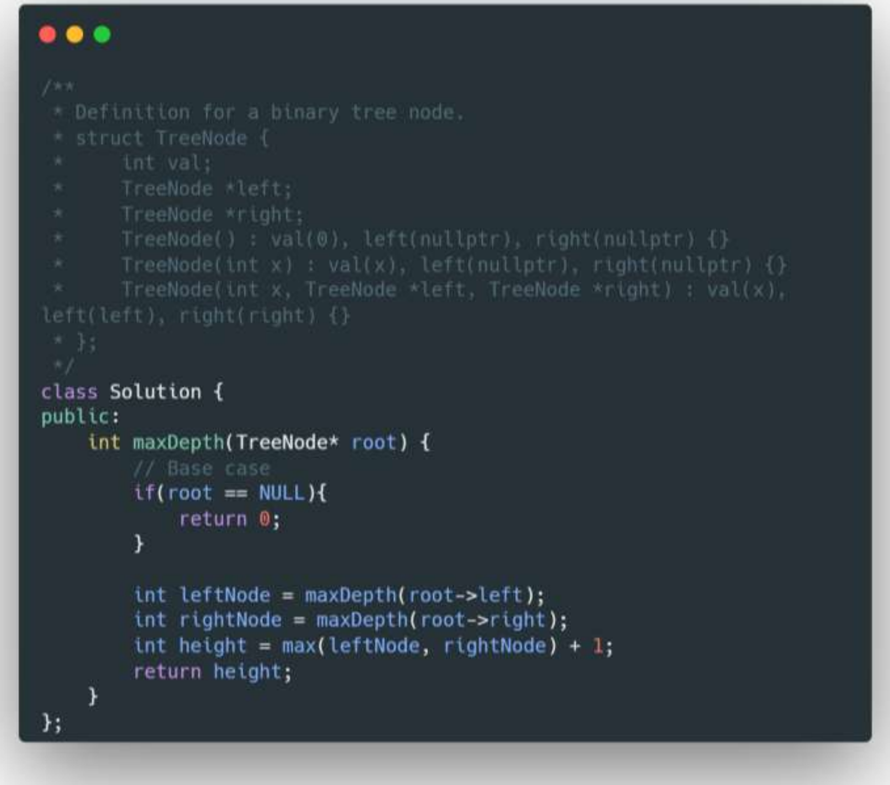
= 1 + 1

= 2

Base case

(Root == Null)

↪ return 0



```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        // Base case
        if(root == NULL){
            return 0;
        }

        int leftNode = maxDepth(root->left);
        int rightNode = maxDepth(root->right);
        int height = max(leftNode, rightNode) + 1;
        return height;
    }
};
```

Time and space complexity: *O(N),*
where N is total number of nodes in binary tree