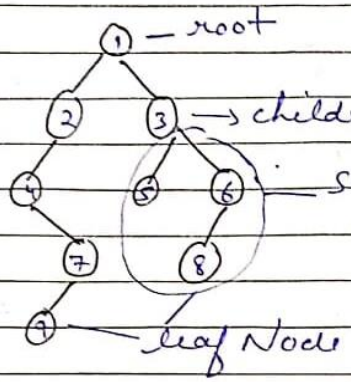
Tree (BY – Stiver + Love)

Tree → Tree is a Hierarchical data structure.



subtree of Node 3 (Both the guys of Node called subtree)

Ancestors ↓ All above connected nodes from a particular Node.

Ancestors of 9 → 7, 4, 2, 1 but 3 is not ancestor of 9.

Type of BT

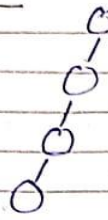
1) Full BT - either has 0 or 2 children.
2) Complete BT - i) all levels are completely field except the last level.
   - The last node has all nodes in left as possible.
3) Perfect BT - all leaf node are at same level.
4) Balanced BT - height of tree at max $\log(N)$ ∴ N is NO. of node.

If $n=8$ $\log(8) = \log(2^3) = 3\log_2 2 = 3$

Maximum height of tree = 3

used for finding complexity

5) Degenerate Tree – (skew Tree like LL)
if n = 4



Binary Representation of in C++

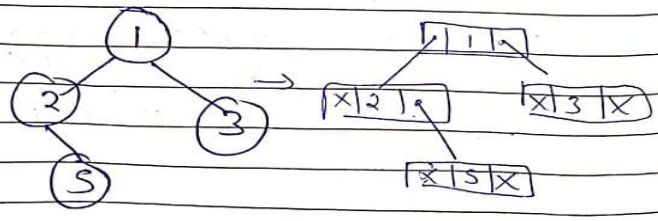
```
Struct Node{
    int data;
    Struct Node* left;
    Struct Node* Right;

    Node(int val)
    {
      data = val;
      left = Right = Null;
    }
```

Constructor

```
Main(){
struct Node* root = newnode(1)
  root -> left = newnode(2)
  root -> Right = new node(3)
  root -> left -> right = newnode(5);
  }
```

New node →

In C++ (NULL)



Traversal Technique

→ BFS (Breadth First Traversal)

↳ DFS (Depth First Traversal)

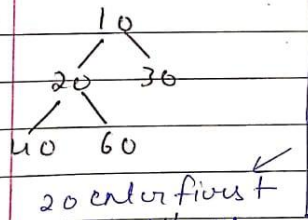
Inorder Preorder Post Order

Level Order Traversal / BFS

Type 1 - Level Order Traversal

Note - For level Order Traversal, we always use queue data Structure (FIFO)
And to show our level order Traversal in Type 1, we use vector<int> ans;

Ex -

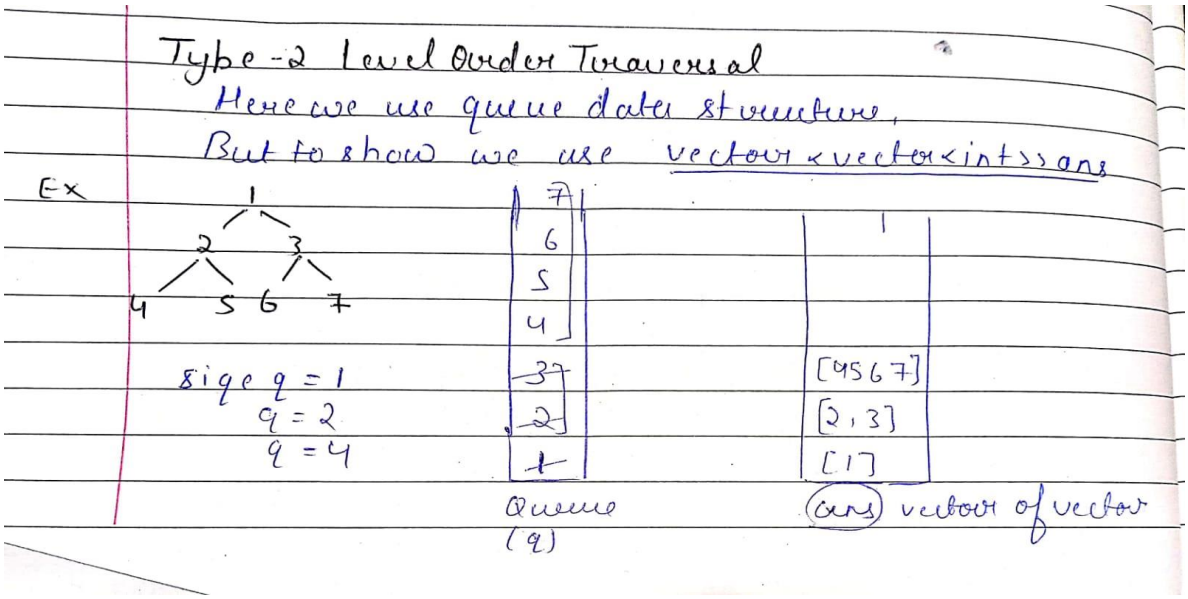


20 enter first as compare to 30

Queue
~~60~~
~~40~~
~~30~~
~~20~~
~~10~~

ans	
60	
40	
30	→ return ans
20	
10	

```
vector<int> levelOrder(Node* root)
{
  // Two show our level order Traversal
  vector<int> ans ;
  //  if root == Null means return an empty ans
  if(root==NULL)return ans ;
// Note for level order Traversal , we always use Queue Data Structure (FIFO)
  queue<Node*> q;
  // Push root into queue
  q.push(root);

  // if our queue is not empty then check for left or right child
  while(!q.empty())
  {
      // Make a node of Node* (Node pointer) data type , to stor root
       Node* node=q.front();

       ans.push_back(node->data);
            if(node->left!=NULL)q.push(node->left);
            if(node->right!=NULL)q.push(node->right);
            q.pop();
  }
  return ans ;
}
```



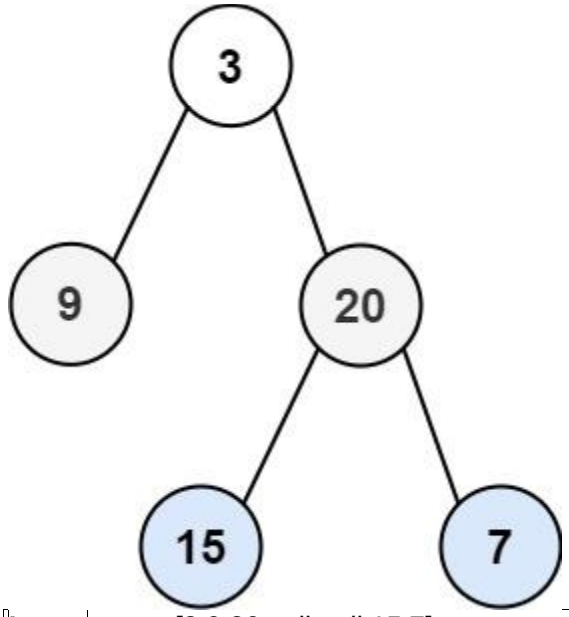
***

```
    vector<vector<int>> levelOrder(TreeNode* root)
    {
        //  Vector of vector to store level wise value
        vector<vector<int>>ans;
        if(root==NULL) return ans;

        //  Create queue because in Level order traversal we use queue
        queue<TreeNode*> q;
        // Queue me root daal denge
        q.push(root);  // Queue me root daal denge
        while(!q.empty())
        {
  /*  Find queue ka size kyuki hr new ko insert krne ke bdd hum lined up in queue
lenge , phle wali value queue se bahar ho jaegiii  */

            int size=q.size();
            //  Level defined is lie because they store lined up element
            vector<int>level;
            for(int i=0;i<size;i++)
            {
   //  first in first out , means first value ko node me store krleenge or queue
se pop krdenege
                TreeNode* node=q.front();
                q.pop();

                if(node->left!=NULL)q.push(node->left);
                if(node->right!=NULL)q.push(node->right);

  //  level - store lined up value node ki sbhi values ko hum level me daal denge
                level.push_back(node->val);
 //  Then pop hone ke bdd hum usko ans me store krdenge level - lined up value ko
            }
            // Phle hum level me pushed up krenge , thrn ans vector me level koo
            ans.push_back(level);
        }
        return ans;
    }
```

*** Reverse Level Order Traversal :

Given a binary tree of size N, find its reverse level order traversal. ie- the traversal must begin from the last level.

LeeTcode - https://leetcode.com/problems/binary-tree-level-order-traversal-ii/description/



Input: root = [3,9,20,null,null,15,7]
Output: [[15,7],[9,20],[3]]

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root)
    {
        vector<vector<int>>result;
        if(root==NULL)return result;
        queue<TreeNode*>q;

        q.push(root);

        while(!q.empty())
        {
            int n = q.size();

            vector<int>level;
            for(int i=0;i<n;i++)
            {
                TreeNode* temp = q.front();
                level.push_back(temp->val);
                if(temp->left!=NULL)q.push(temp->left);
                if(temp->right!=NULL)q.push(temp->right);
                q.pop();
```

```
            }
            result.push_back(level);
        }

        reverse(result.begin(),result.end());

        return result;

    }
};
```

Link : https://practice.geeksforgeeks.org/problems/reverse-level-order-traversal/1

GFG CODE

```
vector<int> reverseLevelOrder(Node *root)
{
     vector<int> ans;
    if(root == NULL)return ans;

     queue<Node *> q;
     q.push(root);

    while(!q.empty())
    {
        // First for root
        Node* t=q.front();
        ans.push_back(t->data);
        q.pop();
        // Then access right first as compare to left ,  because hume back se traverse krna hai
        if(t->right) q.push(t->right);
        if(t->left) q.push(t->left);
      }
      //  Reverse the vector ans
      reverse(ans.begin(), ans.end());
      return ans;
}
```

*** Count Maximum No Of Nodes :

Given an integer **i**. Print the **maximum number of nodes** on level i of a binary tree.

Link:https://practice.geeksforgeeks.org/problems/introduction-to-trees/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=introduction-to-trees

CODE

```
int countNodes(int i)
    {
        if(i==1)
        return 1;
        if(i>1)
        {
            int ans = pow(2,i-1);    bcoz level starting from 2^0 = 1
                                                              2^1 = 2
                                                              2^2 = 4

            return ans;
        }
    }
```

DFS Traversal :

1) Preorder Traversal – (Root,Left,Right)

Reccursion Code : https://practice.geeksforgeeks.org/problems/preorder-traversal/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=preorder-traversal

```
vector<int>v;
vector <int> preOrder(Node* root)
{
  if(root!=NULL)
  {
      v.push_back(root->data); // push in v
      preOrder(root->left);
```

```
      preOrder(root->right);
  }
  return v;
}
vector <int> preorder(Node* root)
{
    vector<int>res=preOrder(root);
    // we need to clear v , because here we store all elemens from v to res
    v.clear();
    return res;
}
```

Iterative Code : **https://leetcode.com/problems/binary-tree-preorder-traversal/submissions/866242110/**

```
vector<int> preorderTraversal(TreeNode* root)
    {
        // vector ans lia jisme print krna hai value koo
        vector<int>ans;
        if(root==NULL) return ans;
      //  For preorder traversal we use only stack We put root node into stack
        stack<TreeNode*>st;
        st.push(root);

        while(!st.empty())
        {
            // /Humne value ko root me daal kr usko pop krlia
            root=st.top();
            st.pop();

            ans.push_back(root->val);
            if(root->right!=NULL)st.push(root->right);
            if(root->left!=NULL)st.push(root->left);
        }
        return ans;
    }
```

2) Inorder Traversal – (Left,Root,Right)

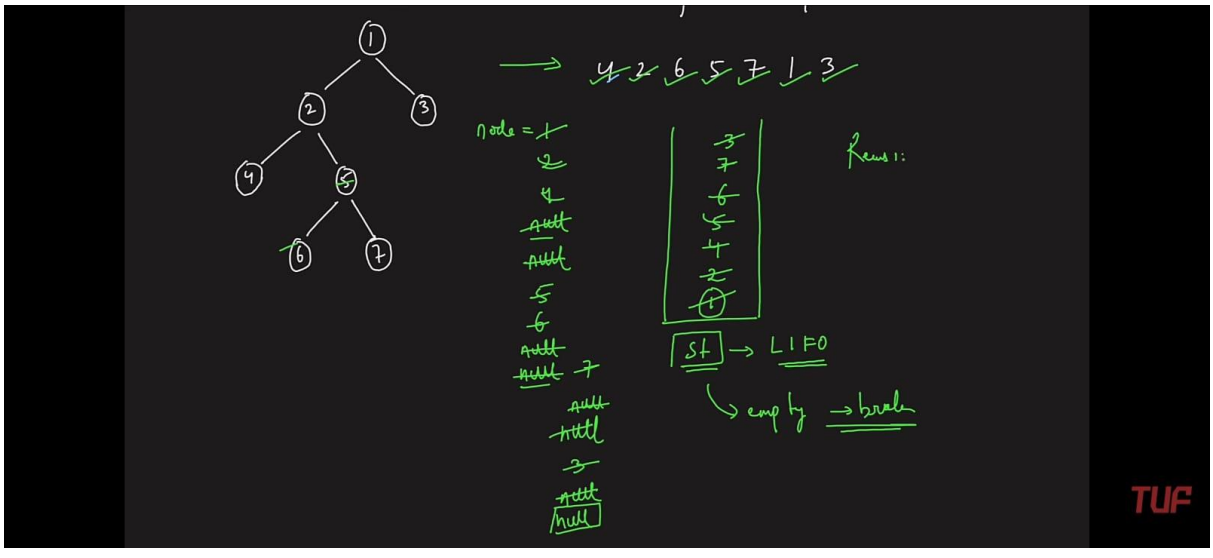
Reccursion Code : https://leetcode.com/problems/binary-tree-inorder-traversal/submissions/

```
// Funtion To find Inorder Traversal

    void inorder(TreeNode* root, vector<int>& ans){

        if(root == NULL){
            return;
        }
        inorder(root->left, ans);  //left subtree
        ans.push_back(root->val);      //pushing value of root to ans
```

```
        inorder(root->right, ans); //right subtree
    }

    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        inorder(root, ans);
        return ans;
    }
```

Iterative Code :

```
vector<int> inorderTraversal(TreeNode* root)
    {
        stack<TreeNode*> st;
        TreeNode* node=root;
        vector<int> inorder;

        while (true)
        {
            // If node ke left part not NULL so we goes to left part
            if(node!=NULL)
            {
                st.push(node);
                node = node->left;
            }
            else
            {
                if(st.empty()==true)break;  // node ka left or right both null
                node=st.top();
                st.pop();
                inorder.push_back(node->val);
                node = node->right;
            }
        }
        return inorder;
    }
```



3) PostOrder Traversal – (Left, Right, Root)

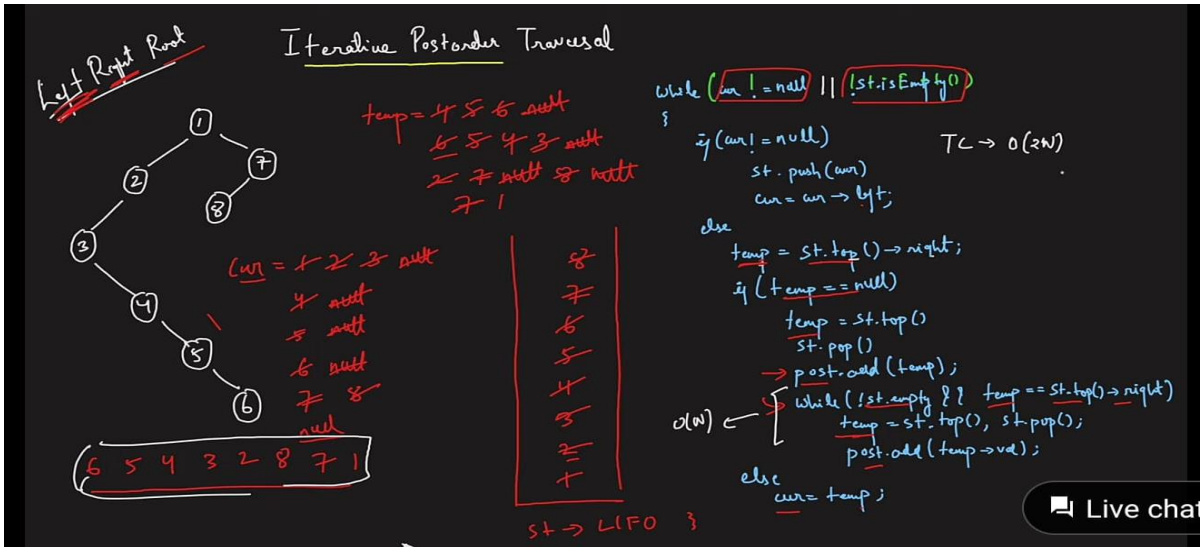
Reccursion Code :

```
//  vector<int> &ans - Means i give a address of my ans vector
   void recursive(TreeNode* root, vector<int> &ans) {
       if(root) {
           recursive(root->left, ans);
           recursive(root->right, ans);
           ans.push_back(root->val);
       }
   }

   vector<int> postorderTraversal(TreeNode* root) {
       vector<int> ans;
       recursive(root, ans);
       return ans;
   }
```

Iterative Code : Using 1 Stack

```
 vector<int> postorderTraversal(TreeNode* root)
   {
       vector<int> ans;
       TreeNode* curr = root;  // Curr me root
       stack<TreeNode*> s;
       if(root==NULL)
       {
           return ans;
       }
       //
       while(curr!=NULL || !s.empty())
       {
           if(curr!=NULL)  // if curr null nhi hai to we go on left left
           {
               s.push(curr);
               curr = curr->left;
           }

           else
           {
            // When curr left goel null , so now we goona move on right
```

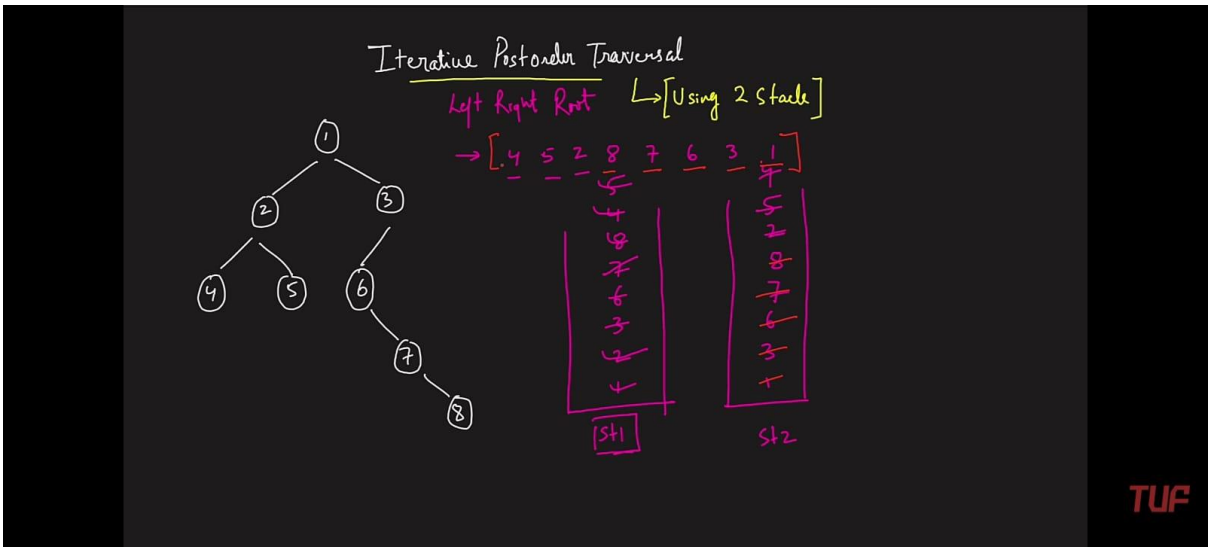
```
                // point to second last on right element
                TreeNode* temp = s.top()->right;
                // But if right also null
                if(temp ==NULL){
                    temp = s.top();
                    s.pop();
                    ans.push_back(temp->val);
/*  Actally element add hone ke bdd stack se pop ho jaate hai then , hum temp
    or stack me top elemt ka comare krenge if both equal , then we temp ko pop
    and push into vector  */

                    while(!s.empty() && temp == s.top()->right) {
                        temp = s.top();
                        s.pop();
                        ans.push_back(temp->val);
                    }
                }
                else{
                    curr = temp;
                }
            }
        }
        return ans;
    }
```



Iterative Code : Using 2 Stack

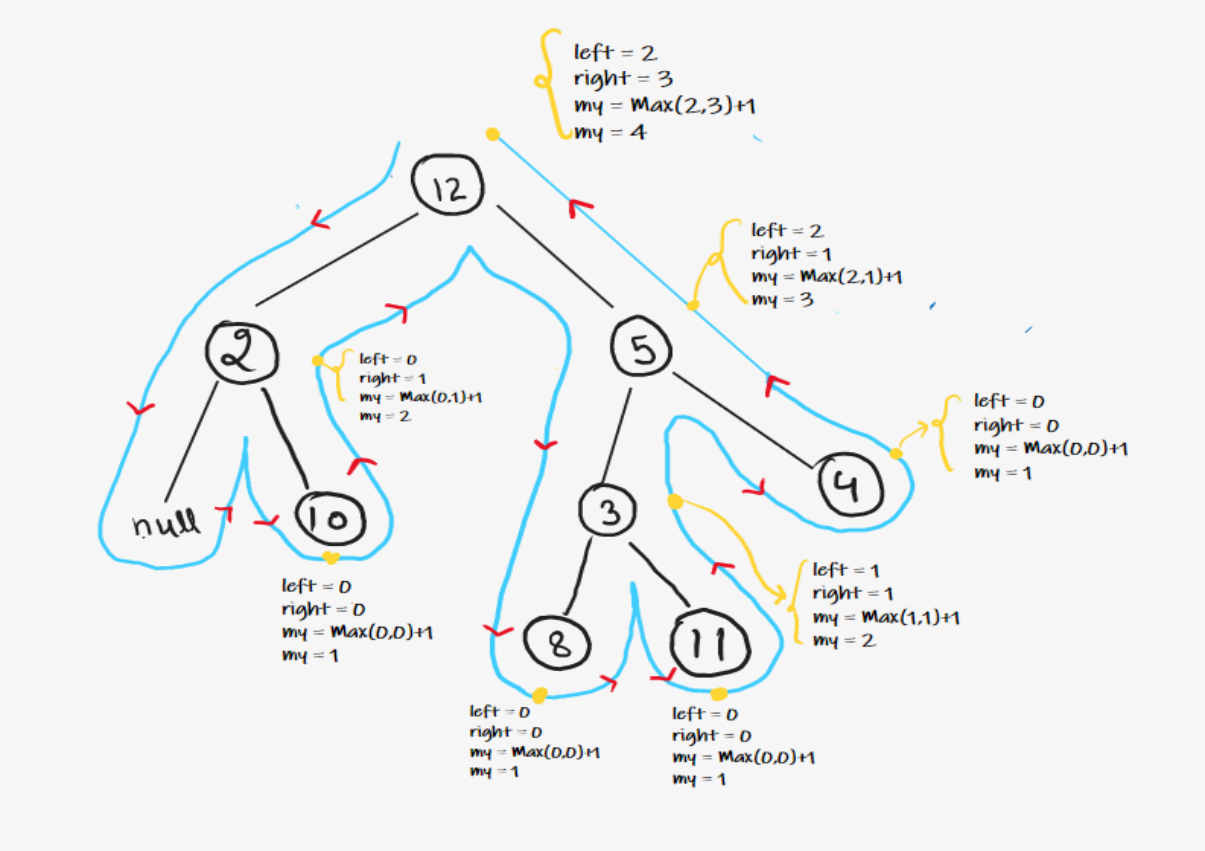
```
    vector<int> postorderTraversal(TreeNode* root)
    {
        vector<int>ans;
        if(root==NULL)return ans;
        stack<TreeNode*>st1,st2;
```

```
        st1.push(root);
        while(!st1.empty())
        {
            root=st1.top();
            st1.pop();
            st2.push(root);
           //  First insert into from left
            if(root->left!=NULL)
            {
                st1.push(root->left);
            }
            // Second insert into from right
             if(root->right!=NULL)
            {
                st1.push(root->right);
            }
        }
       //  st2 stack ke elements ko ek ek krke ans me daalte rhenge
        while(!st2.empty())
        {
            ans.push_back(st2.top()->val);
            st2.pop();
        }
        return ans;
    }
```



Problems

Q1- Maximum Depth Of Binary Tree - A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.



Code - `Using Reccursion`

```
 int maxDepth(TreeNode* root)
{
     if(root==NULL) return 0;

     int leftheight = maxDepth(root->left);
     int rightheight = maxDepth(root->right);
     int totalheight = max(leftheight,rightheight)+1;
     return totalheight;
 }
```

Intuition + Approach: Using LEVEL ORDER TRAVERSAL

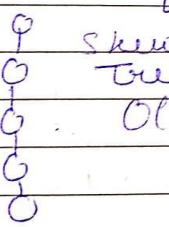
If we observe carefully, the depth of the Binary Tree is the number of levels in the binary tree. So, if we simply do a level order traversal on the binary tree and keep a count of the number of levels, it will be our answer.

In this example, if we start traversing the tree level-wise, then we can reach at max Level 4, so our answer is 4. Because the maximum depth we can achieve is indicated by the last level at which we can travel.

Maximum Depth in Binary Tree

This Problem is solved by using Recursion as well as Level order Traversal.

It does not use any space complexity, but it uses Auxiliary stack space. which after execution automatically removed from stack space.
Auxiliary Space (compl - exity O(height)
Worst Case if given



which does not any Recursion, but it uses queue data structure
It takes SC→O(N)
in worst case.
And worst Case if given