TREES QUESTION

(DAY :- 6)

VERY EASY

QUESTION 1:- Count Complete Tree Nodes
Given the root of a complete binary tree, return the number of the nodes in the tree.

According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

Design an algorithm that runs in less than O(n) time complexity.

Example 1:
Input: root = [1,2,3,4,5,6]

Output: 6

Example 2:

Input: root = []

Output: 0

Example 3:

Input: root = [1]

Output: 1

Constraints:

- **The number of nodes in the tree is in the range [0, 5 * 104].**
- **0 <= Node.val <= 5 * 104**
- **The tree is guaranteed to be complete.**

CODE:-

```
#include <iostream>
using namespace std;
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) {}
};
int countNodes(TreeNode* root) {
    if (!root) return 0;
    int leftDepth = 0, rightDepth = 0;
    TreeNode* leftNode = root;
    TreeNode* rightNode = root;

    while (leftNode) {
        leftDepth++;
        leftNode = leftNode->left;
    }

    while (rightNode) {
        rightDepth++;
        rightNode = rightNode->right;
    }
    if (leftDepth == rightDepth) return (1 << leftDepth) - 1;
    return 1 + countNodes(root->left) + countNodes(root->right);
}
int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    cout << "Number of nodes: " << countNodes(root) << endl;
    return 0;
}
```

QUESTION 2:- Binary Tree - Find Maximum Depth

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.

Example 1:

Input: [3,9,20,null,null,15,7]

Output: 3

Example 2:

Input: [1,null,2]

Output: 2

Constraints:

- **The number of nodes in the tree is in the range [0, 104].**
- **-100 <= Node.val <= 100**

CODE:-

```
#include <iostream>
using namespace std;

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) {}
};
int maxDepth(TreeNode* root) {
    if (!root) return 0;
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
    return 1 + max(leftDepth, rightDepth);
}
int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);
    cout << "Maximum Depth: " << maxDepth(root) << endl;
    return 0;
}
```

main.cpp

```cpp
#include <iostream>
using namespace std;

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) {}
};

int maxDepth(TreeNode* root) {
    if (!root) return 0;

    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
```

Output

```
Maximum Depth: 3


=== Code Execution Su
```

QUESTION 3:- Binary Tree - Sum of All Nodes

Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.

Example 1:

Input: root = [1, 2, 3, 4, 5, null, 6]

Output: 21

Explanation: The sum of all nodes is 1 + 2 + 3 + 4 + 5 + 6 = 21.

Example 2:

Input: root = [5, 2, 6, 1, 3, 4, 7]

Output: 28

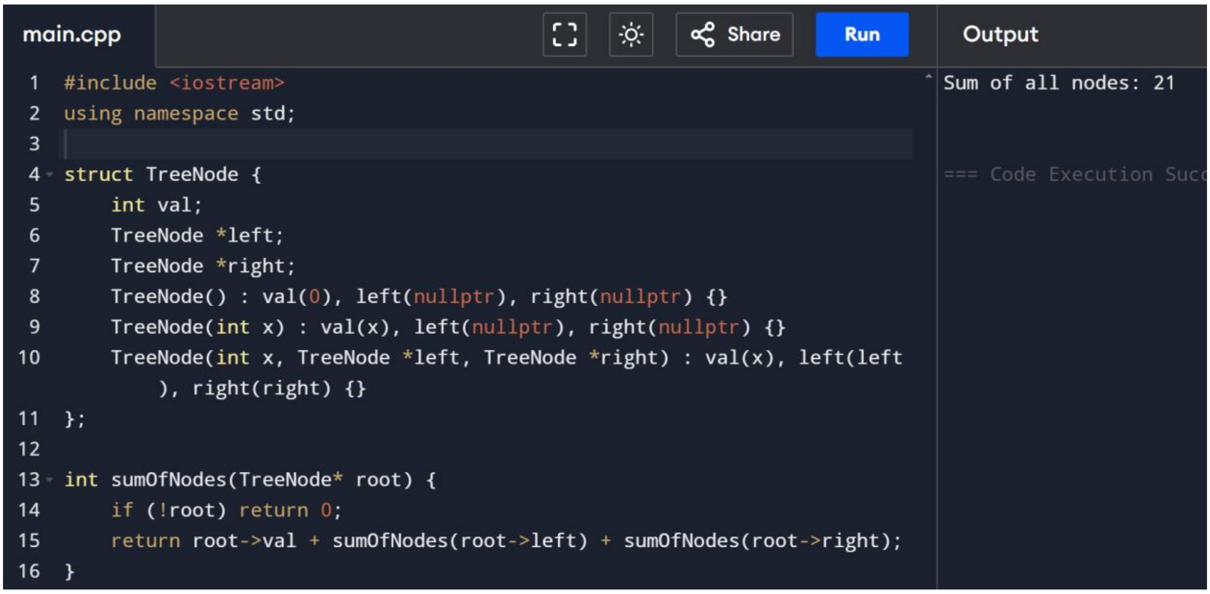
Explanation: The sum of all nodes is 5 + 2 + 6 + 1 + 3 + 4 + 7 = 28.

CODE:-

```cpp
#include <iostream>
using namespace std;
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) {}
};
```

```
int sumOfNodes(TreeNode* root) {
   if (!root) return 0; // Base case: If the tree is empty, the sum is 0.
   return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
}
int main() {
   TreeNode* root = new TreeNode(1);
   root->left = new TreeNode(2);
   root->right = new TreeNode(3);
   root->left->left = new TreeNode(4);
   root->left->right = new TreeNode(5);
   root->right->right = new TreeNode(6);

   cout << "Sum of all nodes: " << sumOfNodes(root) << endl;
   return 0;
}
```



EASY

QUESTION 1:- Same Tree

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:

Input: p = [1,2,3], q = [1,2,3]

Output: true

Example 2:

Input: p = [1,2], q = [1,null,2]

Output: false

Constraints:

- **The number of nodes in both trees is in the range [0, 100].**
- **-104 <= Node.val <= 104**

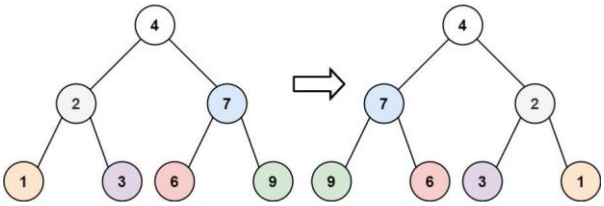
CODE:-

```
#include <iostream>
using namespace std;
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) {}
};
bool isSameTree(TreeNode* p, TreeNode* q) {
   if (!p && !q) return true;
   if (!p || !q) return false;
   return (p->val == q->val) && isSameTree(p->left, q->left) && isSameTree(p->right, q-
>right); }
int main() {
   TreeNode* p = new TreeNode(1);
   p->left = new TreeNode(2);
   p->right = new TreeNode(3);
   TreeNode* q = new TreeNode(1);
   q->left = new TreeNode(2);
   q->right = new TreeNode(3);
   cout << (isSameTree(p, q) ? "true" : "false") << endl;
   return 0;
 }
```

QUESTION 2:- Invert Binary Tree

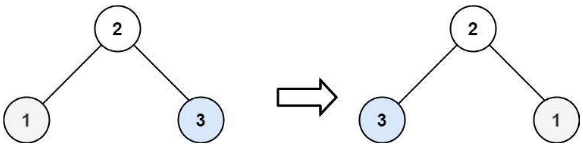
Given the root of a binary tree, invert the tree, and return its root.

Example 1:



Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]

Example 2:



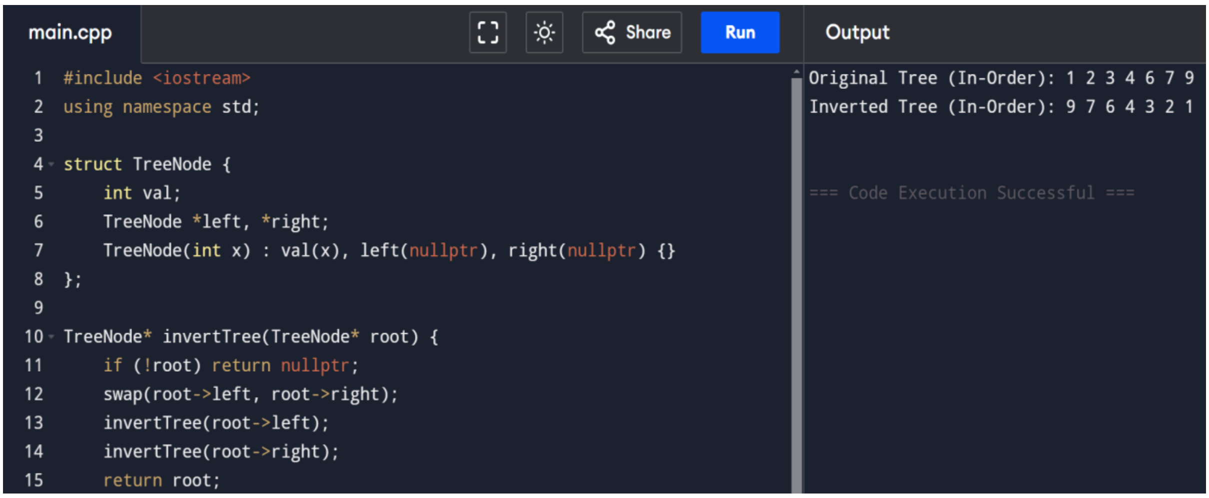
Input: root = [2,1,3]
Output: [2,3,1]
Constraints:

- **The number of nodes in the tree is in the range [0, 100].**
- **-100 <= Node.val <= 100**

CODE:-

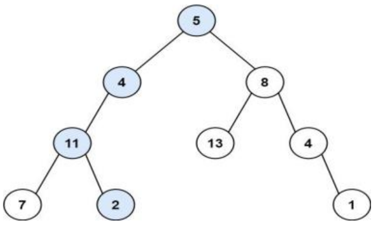
```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
TreeNode* invertTree(TreeNode* root) {
    if (!root) return nullptr;
    swap(root->left, root->right);
    invertTree(root->left);
    invertTree(root->right);
    return root; }
```

```
void printTree(TreeNode* root) {
    if (!root) return;
    printTree(root->left);
    cout << root->val << " ";
    printTree(root->right);
}
int main() {
    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(7);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(9);
    cout << "Original Tree (In-Order): ";
    printTree(root);
    cout << endl;
    invertTree(root); // Inverts the tree
    cout << "Inverted Tree (In-Order): ";
    printTree(root);
    cout << endl;
    return 0;
}
```



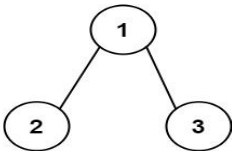
QUESTION 3:- Path Sum
Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found.

Example 1:



Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.

Example 2:



Input: root = [1,2,3], targetSum = 5
Output: false
Explanation: There are two root-to-leaf paths in the tree:

(1 --> 2): The sum is 3.

(1 --> 3): The sum is 4.

There is no root-to-leaf path with sum = 5.

Example 3:

Input: root = [], targetSum = 0
Output: false
Explanation: Since the tree is empty, there are no root-to-leaf paths.

CODE:-

```
#include <iostream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

```
bool hasPathSum(TreeNode* root, int targetSum) {
    if (!root) return false;
    if (!root->left && !root->right)
        return root->val == targetSum;
           hasPathSum(root->right, targetSum - root->val);
}

int main() {
    TreeNode* root = new TreeNode(5);
    root->left = new TreeNode(4);
    root->right = new TreeNode(8);
    root->left->left = new TreeNode(11);
    root->left->left->left = new TreeNode(7);
    root->left->left->right = new TreeNode(2);
    root->right->left = new TreeNode(13);
    root->right->right = new TreeNode(4);
    root->right->right->right = new TreeNode(1);
    int targetSum = 22;

    cout << (hasPathSum(root, targetSum) ? "true" : "false") << endl;
    return 0;

}
```

MEDIUM

QUESTION 1:- Construct Binary Tree from Preorder and Inorder Traversal
Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Example 1:

Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

Output: [3,9,20,null,null,15,7]

Example 2:

Input: preorder = [-1], inorder = [-1]

Output: [-1]

Constraints:

- **1 <= preorder.length <= 3000**
- **inorder.length == preorder.length**
- **-3000 <= preorder[i], inorder[i] <= 3000**
- **preorder and inorder consist of unique values.**
- **Each value of inorder also appears in preorder.**
- **preorder is guaranteed to be the preorder traversal of the tree.**
- **inorder is guaranteed to be the inorder traversal of the tree.**

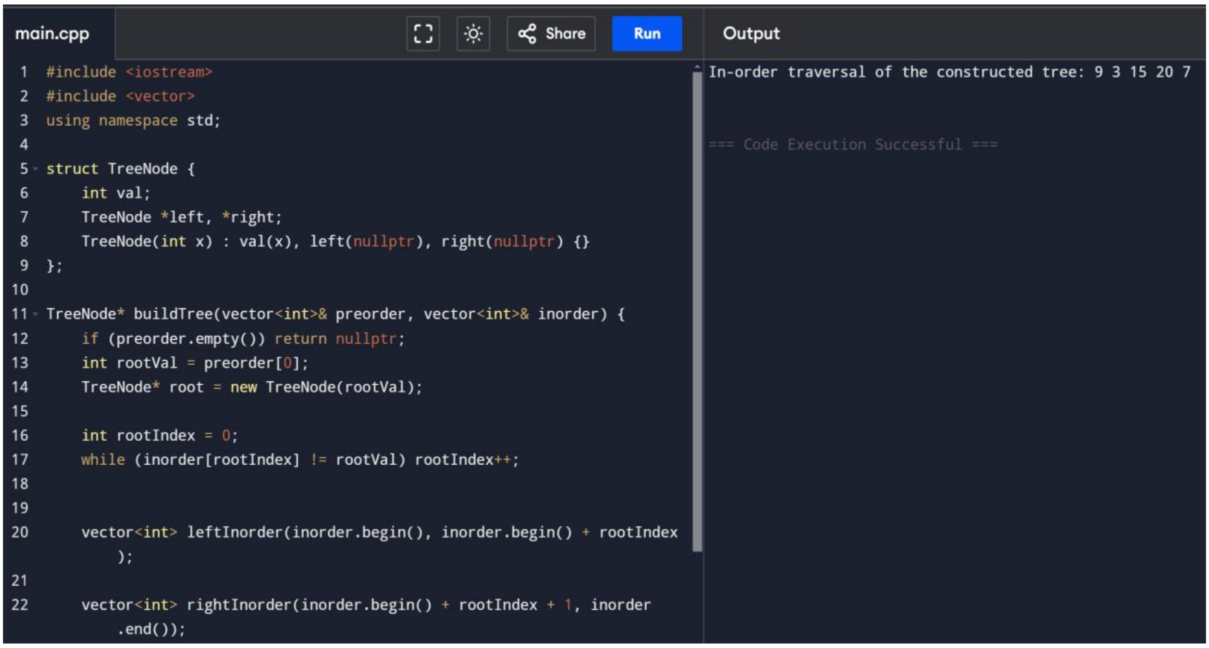
CODE:-

```
#include <iostream>
#include <vector>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
    if (preorder.empty()) return nullptr;
    int rootVal = preorder[0];
    TreeNode* root = new TreeNode(rootVal);
    int rootIndex = 0;
    while (inorder[rootIndex] != rootVal) rootIndex++;
    vector<int> leftInorder(inorder.begin(), inorder.begin() + rootIndex);
    vector<int> rightInorder(inorder.begin() + rootIndex + 1, inorder.end());
```

```
    preorder.erase(preorder.begin());
    vector<int> leftPreorder(preorder.begin(), preorder.begin() + leftInorder.size());
    vector<int> rightPreorder(preorder.begin() + leftInorder.size(), preorder.end());
    root->left = buildTree(leftPreorder, leftInorder);
    root->right = buildTree(rightPreorder, rightInorder);
    return root;
}
void printTree(TreeNode* root) {
    if (root == nullptr) return;
    printTree(root->left);
    cout << root->val << " ";
    printTree(root->right);
}

int main() {
    vector<int> preorder = {3,9,20,15,7};
    vector<int> inorder = {9,3,15,20,7};
    TreeNode* root = buildTree(preorder, inorder);
    cout << "In-order traversal of the constructed tree: ";
    printTree(root);
    cout << endl;
    return 0;
}
```



<u>QUESTION 2:-</u> Construct Binary Tree from Inorder and Postorder Traversal
Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary

tree.

Example 1:

Input: inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]

Output: [3,9,20,null,null,15,7]

Example 2:

Input: inorder = [-1], postorder = [-1]

Output: [-1]

Constraints:

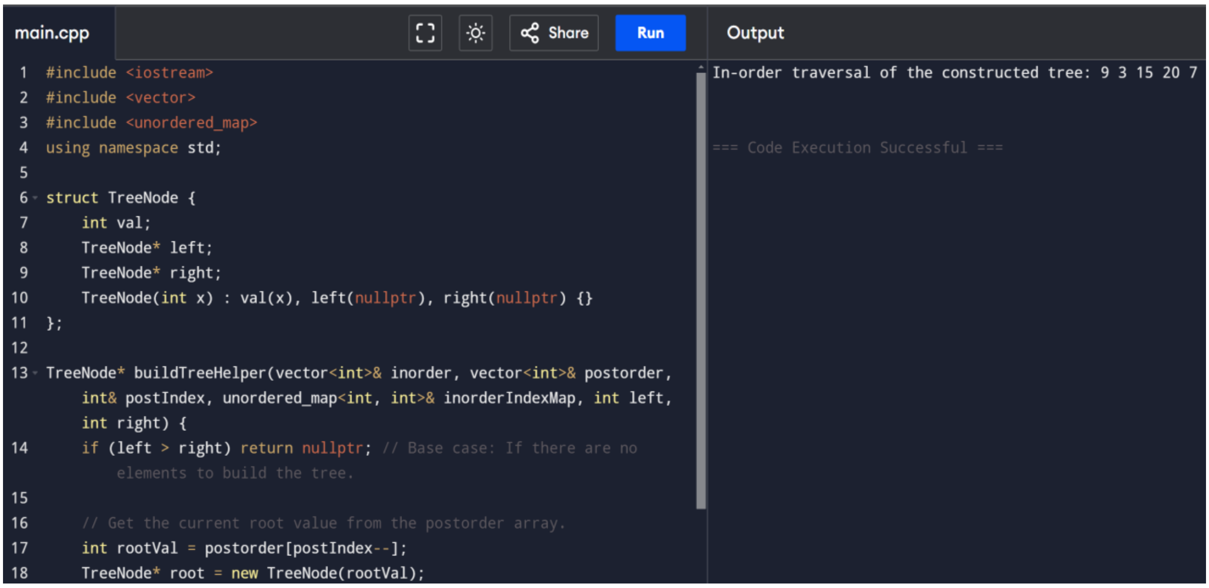
- **1 <= inorder.length <= 3000**
- **postorder.length == inorder.length**
- **-3000 <= inorder[i], postorder[i] <= 3000**
- **inorder and postorder consist of unique values.**
- **Each value of postorder also appears in inorder.**
- **inorder is guaranteed to be the inorder traversal of the tree.**
- **postorder is guaranteed to be the postorder traversal of the tree.**

CODE:-

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int& postIndex,
unordered_map<int, int>& inorderIndexMap, int left, int right) {
if (left > right) return nullptr;
int rootVal = postorder[postIndex--];
TreeNode* root = new TreeNode(rootVal);
root->right = buildTreeHelper(inorder, postorder, postIndex, inorderIndexMap,
inorderIndexMap[rootVal] + 1, right);
root->left = buildTreeHelper(inorder, postorder, postIndex, inorderIndexMap, left,
inorderIndexMap[rootVal] - 1);
return root;
}
TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
unordered_map<int, int> inorderIndexMap; // Map for storing the index of elements in
inorder traversal.
for (int i = 0; i < inorder.size(); ++i) {
inorderIndexMap[inorder[i]] = i;
}
```

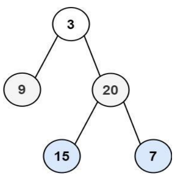
```
int postIndex = postorder.size() - 1; // Start from the last element in postorder.
    return buildTreeHelper(inorder, postorder, postIndex, inorderIndexMap, 0, inorder.size() - 1);
}
void printTree(TreeNode* root) {
    if (!root) return;
    printTree(root->left);
    cout << root->val << " ";
    printTree(root->right);
}
int main() {
    vector<int> inorder = {9, 3, 15, 20, 7};
    vector<int> postorder = {9, 15, 7, 20, 3};
    TreeNode* root = buildTree(inorder, postorder);
    cout << "In-order traversal of the constructed tree: ";
    printTree(root);
    cout << endl;
    return 0;
}
```



<u>QUESTION 3:-</u> Binary Tree Level Order Traversal

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Example 1:



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

Output: [[3],[9,20],[15,7]]

Example 2:

Input: root = [1]

Output: [[1]]

Example 3:

Input: root = []

Output: []

Constraints:

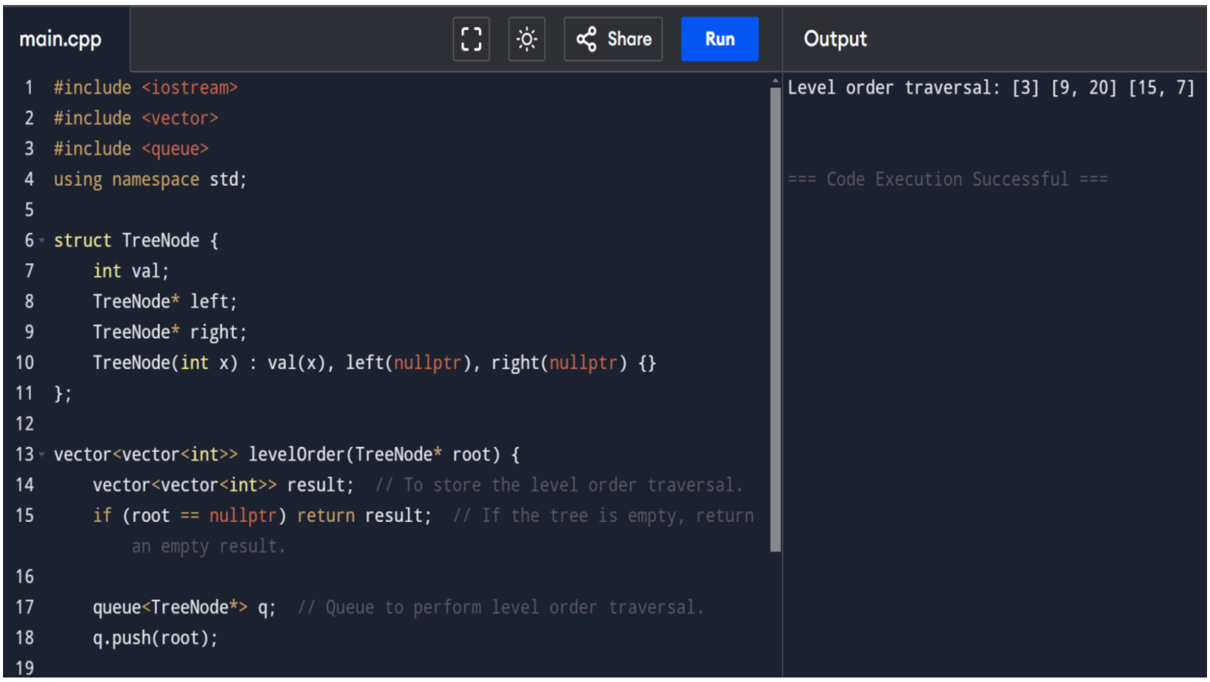
- **The number of nodes in the tree is in the range [0, 2000].**
- **-1000 <= Node.val <= 1000**

CODE:-

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
vector<vector<int>> levelOrder(TreeNode* root) {
   vector<vector<int>> result;
   if (root == nullptr) return result;
   queue<TreeNode*> q;
   q.push(root);
   while (!q.empty()) {
   int levelSize = q.size();
   vector<int> levelNodes;
   for (int i = 0; i < levelSize; ++i) {
   TreeNode* currentNode = q.front();
   q.pop();
   levelNodes.push_back(currentNode->val);
   if (currentNode->left) q.push(currentNode->left);
   if (currentNode->right) q.push(currentNode->right);    }
   result.push_back(levelNodes);   }
   return result;
```

```
}
void printLevelOrder(const vector<vector<int>>& result) {
    for (const auto& level : result) {
    cout << "[";
    for (int i = 0; i < level.size(); ++i) {
    cout << level[i];
    if (i != level.size() - 1) cout << ", ";   }
    cout << "] "; }
    cout << endl;
}
int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);
     vector<vector<int>> result = levelOrder(root);
    cout << "Level order traversal: ";
    printLevelOrder(result);
    return 0;
}
```



HARD

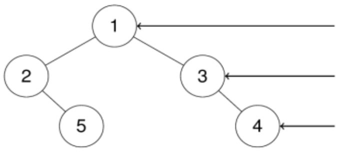
QUESTION 1:- Binary Tree Right Side View
Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Example 1:

Input: root = [1,2,3,null,5,null,4]

Output: [1,3,4]

Explanation:

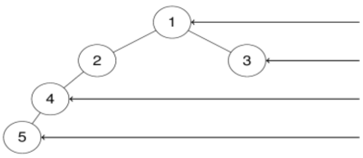


Example 2:

Input: root = [1,2,3,4,null,null,null,5]

Output: [1,3,4,5]

Explanation:



Example 3:

Input: root = [1,null,3]

Output: [1,3]

Example 4:

Input: root = []

Output: []

Constraints:

- **The number of nodes in the tree is in the range [0, 100].**
- **-100 <= Node.val <= 100**

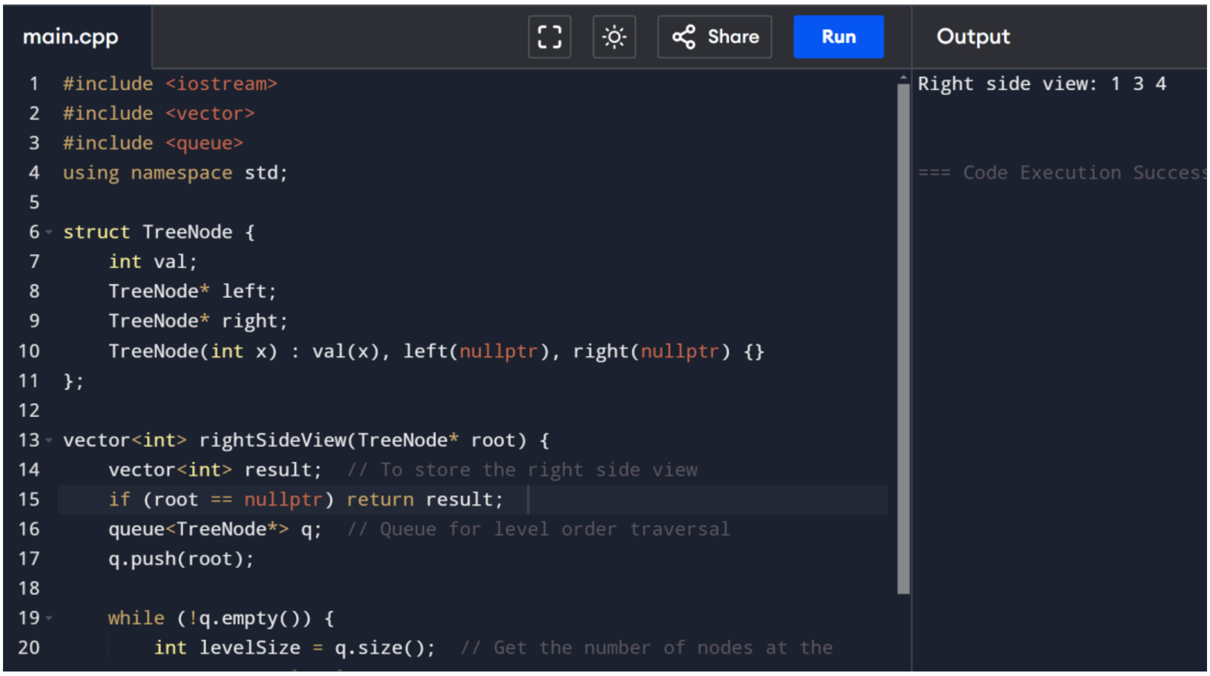
CODE:-

```
#include <iostream>
#include <vector>
```

```
#include <queue>
using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
vector<int> rightSideView(TreeNode* root) {
   vector<int> result;
   if (root == nullptr) return result;
   queue<TreeNode*> q;
   q.push(root);
   while (!q.empty()) {
   int levelSize = q.size();
   for (int i = 0; i < levelSize; ++i) {
   TreeNode* node = q.front();
   q.pop();
   if (i == levelSize - 1) {
   result.push_back(node->val);     }
   if (node->left) q.push(node->left);
   if (node->right) q.push(node->right);  }
   }
   return result;
}
void printRightSideView(const vector<int>& result) {
   for (int val : result) {
   cout << val << " ";  }
   cout << endl;
}
int main() {
   TreeNode* root = new TreeNode(1);
   root->left = new TreeNode(2);
   root->right = new TreeNode(3);
   root->left->right = new TreeNode(5);
   root->right->right = new TreeNode(4);
   vector<int> result = rightSideView(root);
   cout << "Right side view: ";
   printRightSideView(result);

   return 0;
}
```



main.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

vector<int> rightSideView(TreeNode* root) {
    vector<int> result;  // To store the right side view
    if (root == nullptr) return result;
    queue<TreeNode*> q;  // Queue for level order traversal
    q.push(root);

    while (!q.empty()) {
        int levelSize = q.size();  // Get the number of nodes at the
```

Output

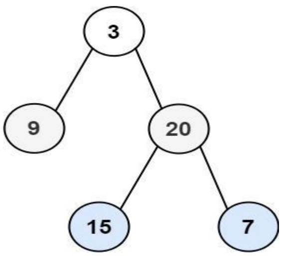
```
Right side view: 1 3 4

=== Code Execution Success
```

<u>QUESTION 2:-</u> Binary Tree Zigzag Level Order Traversal

Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

Example 1:



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

Example 2:
Input: root = [1]
Output: [[1]]

Example 3:
Input: root = []
Output: []

Constraints:

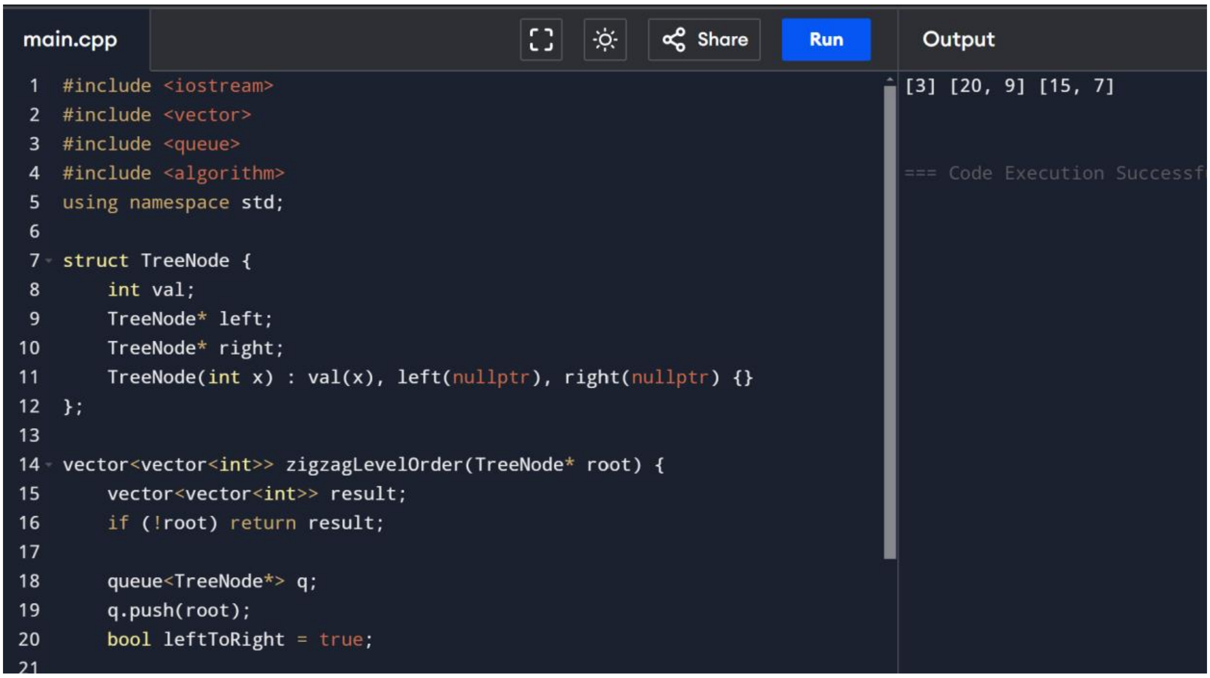
- **The number of nodes in the tree is in the range [0, 2000].**
- **-100 <= Node.val <= 100**

CODE:-

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
   vector<vector<int>> result;
   if (!root) return result;
   queue<TreeNode*> q;
   q.push(root);
   bool leftToRight = true;
   while (!q.empty()) {
   int levelSize = q.size();
   vector<int> levelNodes;
   for (int i = 0; i < levelSize; ++i) {
   TreeNode* node = q.front();
    q.pop();
    levelNodes.push_back(node->val);
    if (node->left) q.push(node->left);
    if (node->right) q.push(node->right);
    }
    if (!leftToRight) reverse(levelNodes.begin(), levelNodes.end());
    result.push_back(levelNodes);
    leftToRight = !leftToRight;    }
   return result;
}
void printResult(const vector<vector<int>>& result) {
   for (const auto& level : result) {
   cout << "[";
   for (int i = 0; i < level.size(); i++) {
   cout << level[i];
   if (i != level.size() - 1) cout << ", "; }
   cout << "] "; }
   cout << endl;
}
int main() {
   TreeNode* root = new TreeNode(3);
   root->left = new TreeNode(9);
```

```
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);
    vector<vector<int>> result = zigzagLevelOrder(root);
    printResult(result);
    return 0;
}
```



<u>QUESTION 3:-</u> Kth Smallest Element in a BST (Binary Search Tree)

Given a binary search tree (BST), write a function to find the kth smallest element in the tree.
Input: root = [3,1,4,null,2], k = 1
Output: 1
Explanation: The inorder traversal of the BST is [1, 2, 3, 4], and the 1st smallest element is 1.
Input: root = [5,3,6,2,4,null,null,1], k = 3
Output: 3
Explanation: The inorder traversal of the BST is [1, 2, 3, 4, 5, 6], and the 3rd smallest element is 3.
Constraints:

- **The number of nodes in the tree is in the range [1, 1000].**
- **-10^4 ≤ Node.val ≤ 10^4.**

<u>CODE:-</u>

```
#include <iostream>
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
int kthSmallest(TreeNode* root, int& k) {
    if (root == nullptr) return -1;
    int left = kthSmallest(root->left, k);
    if (left != -1) return left;  // If we found the kth smallest in the left subtree, return it
    k--;
    if (k == 0) return root->val;
    return kthSmallest(root->right, k);
}
int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(1);
    root->right = new TreeNode(4);
    root->left->right = new TreeNode(2);
    int k = 1;
    cout << "The " << k << "th smallest element is: " << kthSmallest(root, k) << endl;
    root = new TreeNode(5);
    root->left = new TreeNode(3);
    root->right = new TreeNode(6);
    root->left->left = new TreeNode(2);
    root->left->right = new TreeNode(4);
    root->left->left->left = new TreeNode(1);
    k = 3;
    cout << "The " << k << "th smallest element is: " << kthSmallest(root, k) << endl;
    return 0;
}
```