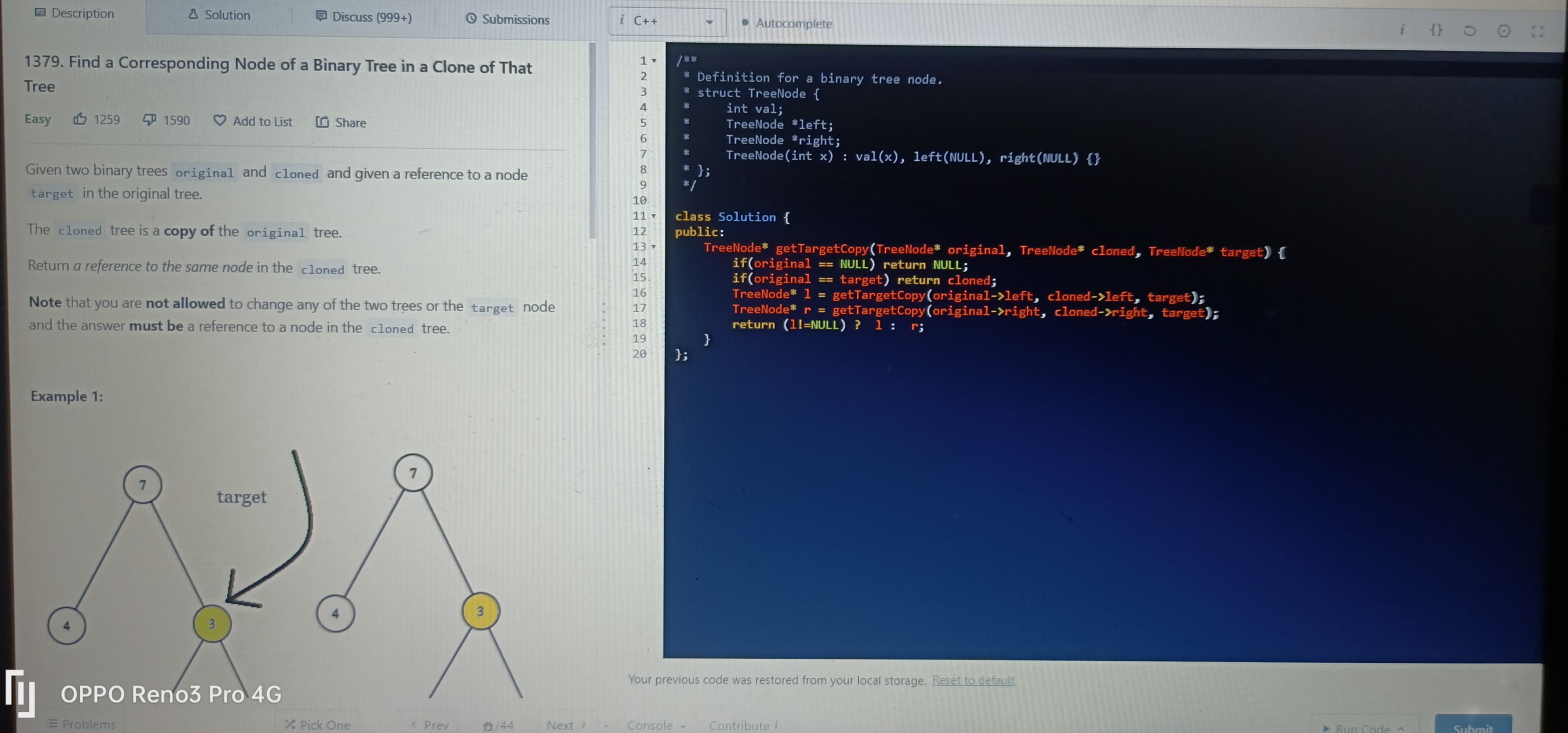


1379. Find a Corresponding Node of a Binary Tree in a Clone of That Tree

Given two binary trees `original` and `cloned` and given a reference to a node `target` in the original tree.

The `cloned` tree is a **copy of** the `original` tree.

Return *a reference to the same node* in the `cloned` tree.

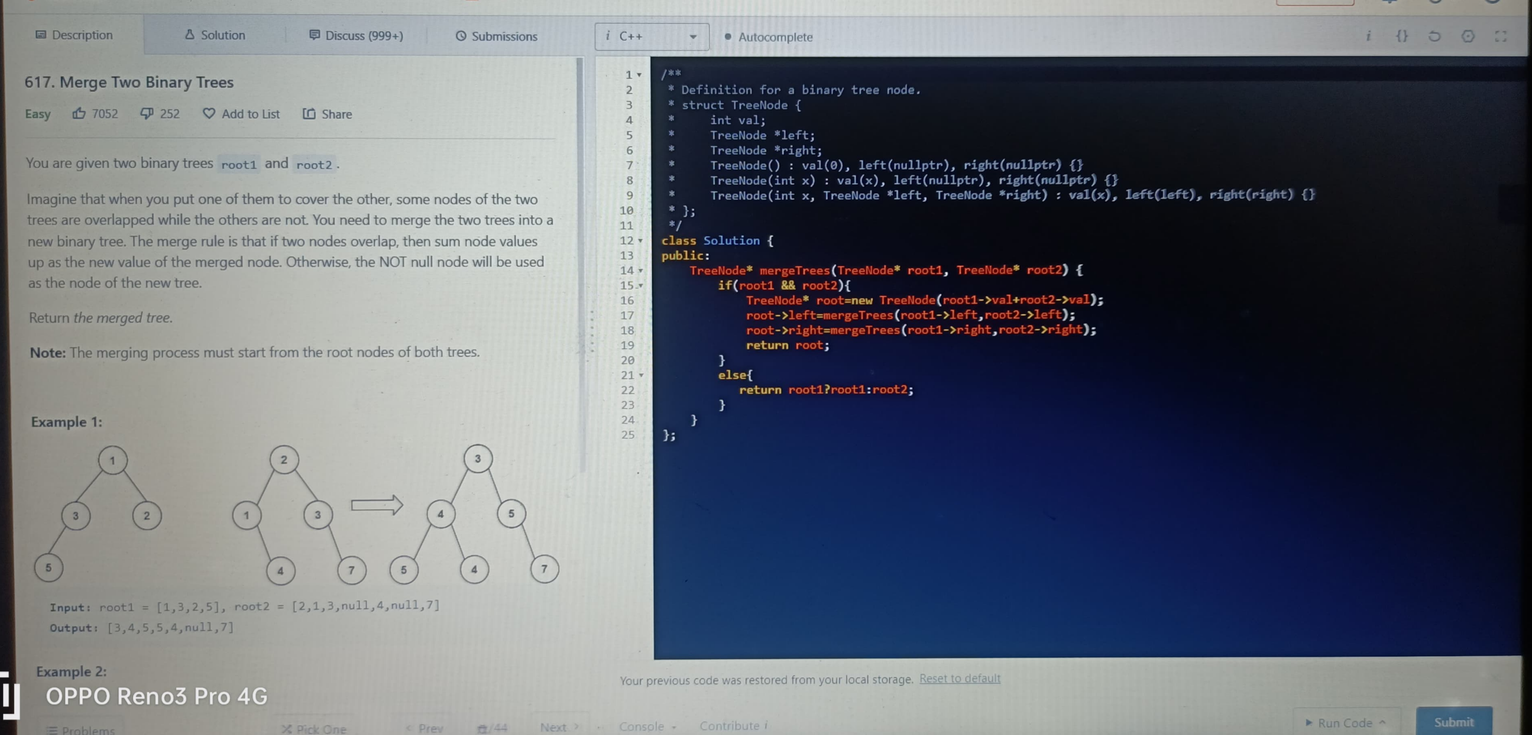
Note that you are **not allowed** to change any of the two trees or the `target` node and the answer **must be** a reference to a node in the `cloned` tree.

Example 1:

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
public:
    TreeNode* getTargetCopy(TreeNode* original, TreeNode* cloned, TreeNode* target) {
        if(original == NULL) return NULL;
        if(original == target) return cloned;
        TreeNode* l = getTargetCopy(original->left, cloned->left, target);
        TreeNode* r = getTargetCopy(original->right, cloned->right, target);
        return (l!=NULL) ?  l :  r;
    }
};
```

Your previous code was restored from your local storage. Reset to default



617. Merge Two Binary Trees

Easy 7052 252 Add to List Share

You are given two binary trees `root1` and `root2`.

Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.

Return *the merged tree.*

Note: The merging process must start from the root nodes of both trees.

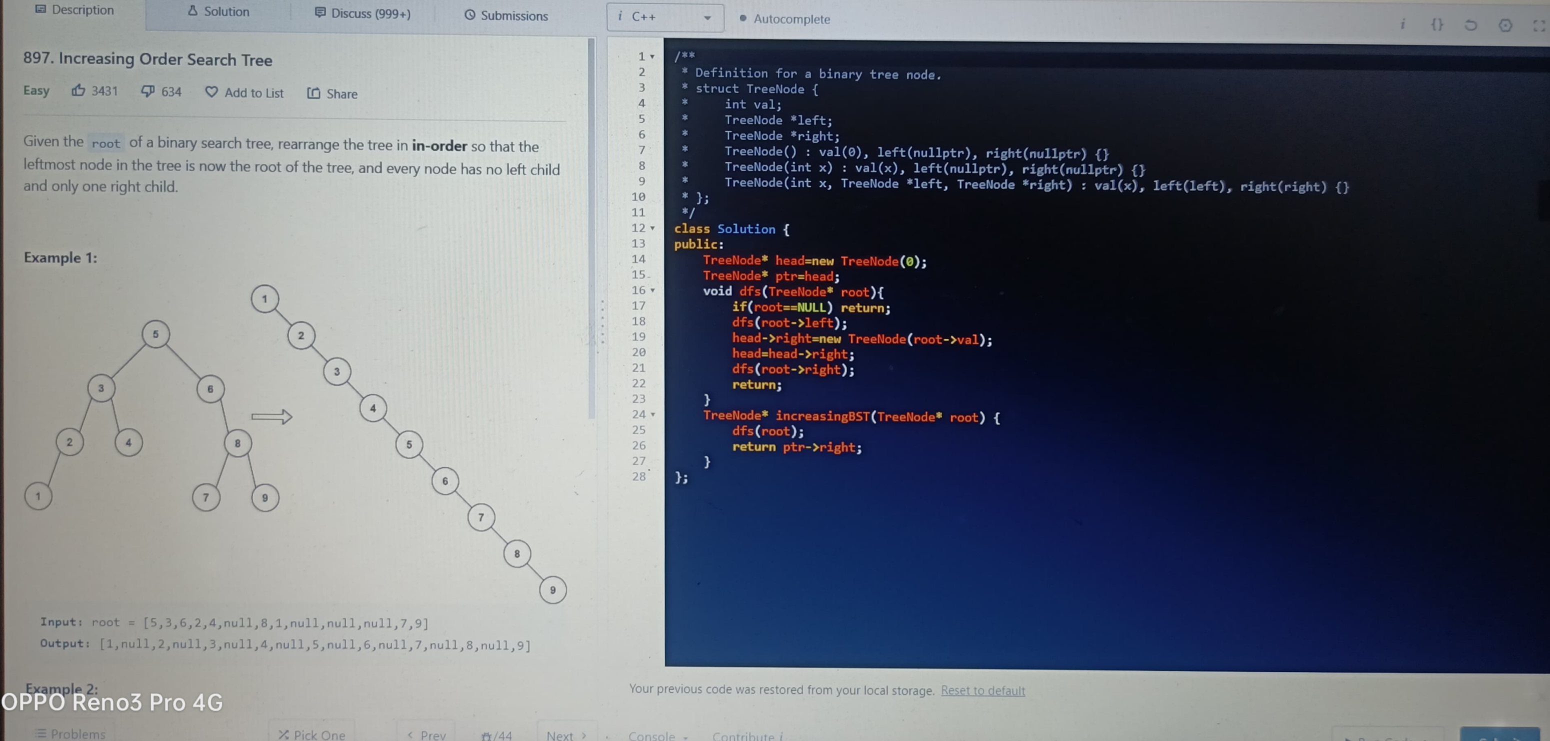
Example 1:

```
Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]
Output: [3,4,5,5,4,null,7]
```

Example 2:

```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:
    TreeNode* mergeTrees(TreeNode* root1, TreeNode* root2) {
        if(root1 && root2){
            TreeNode* root=new TreeNode(root1->val+root2->val);
            root->left=mergeTrees(root1->left,root2->left);
            root->right=mergeTrees(root1->right,root2->right);
            return root;
        }
        else{
            return root1?root1:root2;
        }
    }
};
```

Your previous code was restored from your local storage. Reset to default



897. Increasing Order Search Tree

Easy 3431 634 Add to List Share

Given the `root` of a binary search tree, rearrange the tree in **in-order** so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only one right child.

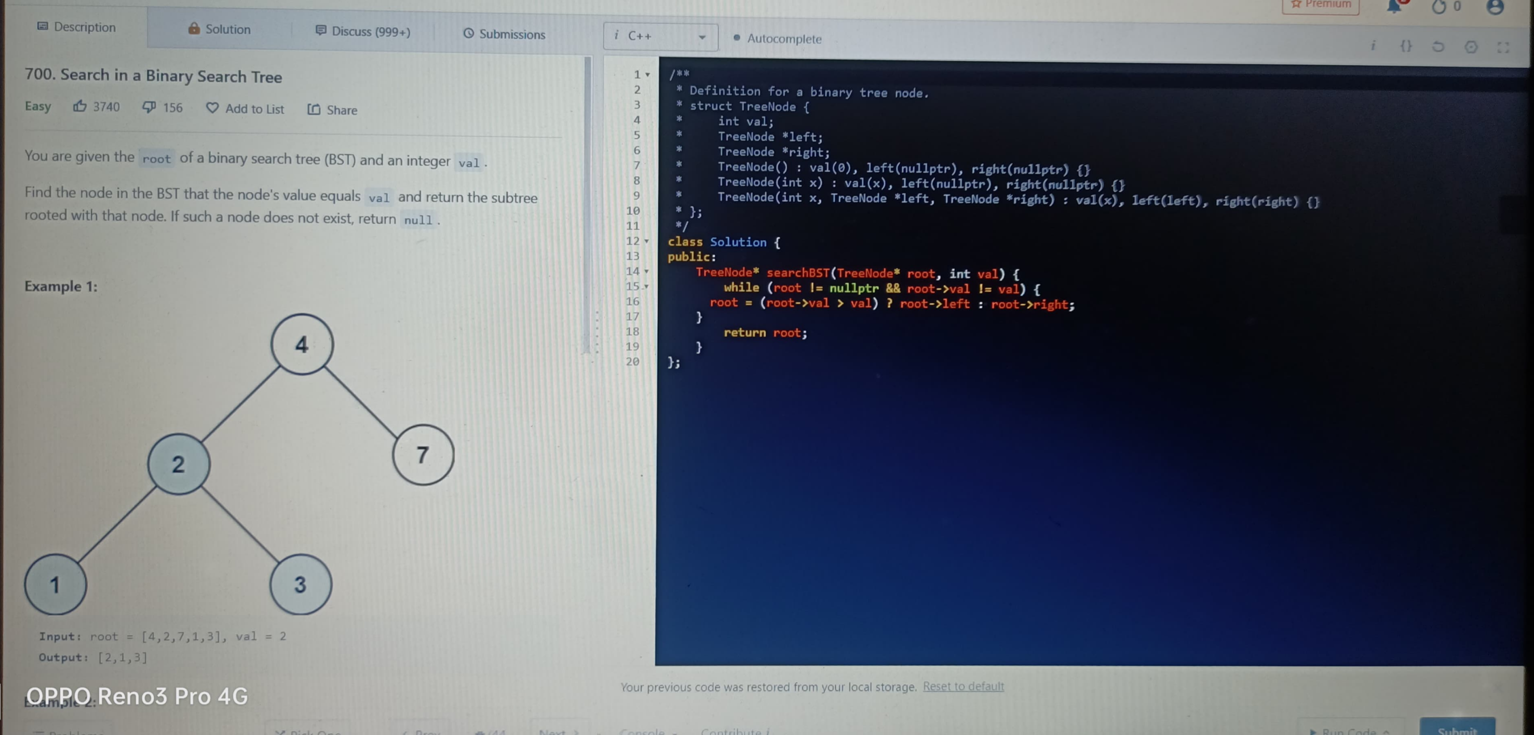
Example 1:

```
Input: root = [5,3,6,2,4,null,8,1,null,null,null,7,9]
Output: [1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]
```

Example 2:

```
/**
 * 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:
    TreeNode* head=new TreeNode(0);
    TreeNode* ptr=head;
    void dfs(TreeNode* root){
        if(root==NULL) return;
        dfs(root->left);
        head->right=new TreeNode(root->val);
        head=head->right;
        dfs(root->right);
        return;
    }
    TreeNode* increasingBST(TreeNode* root) {
        dfs(root);
        return ptr->right;
    }
};
```

Your previous code was restored from your local storage. Reset to default



700. Search in a Binary Search Tree

Easy 3740 156 Add to List Share

You are given the `root` of a binary search tree (BST) and an integer `val`.

Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node. If such a node does not exist, return `null`.

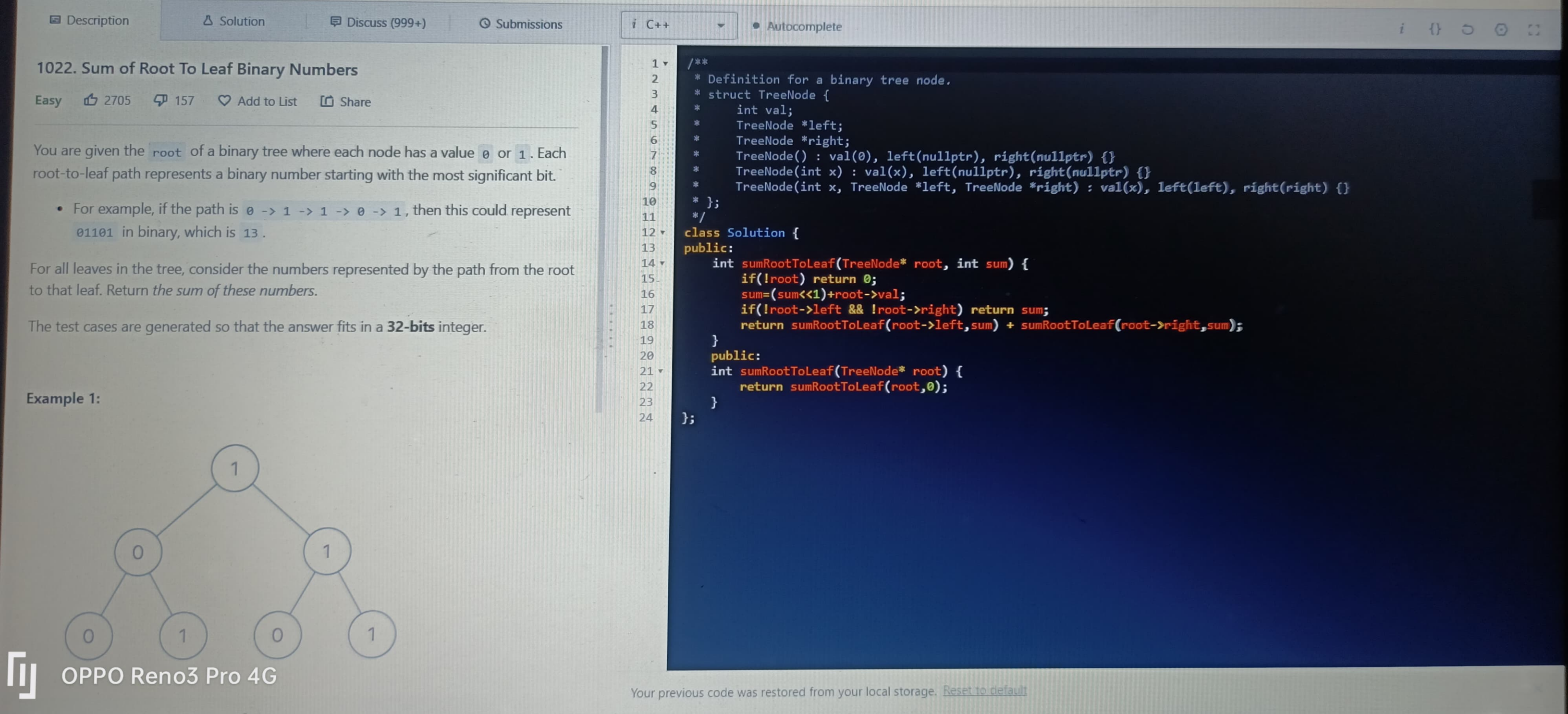
Example 1:

```
Input: root = [4,2,7,1,3], val = 2
Output: [2,1,3]
```

Example 2:

```
/**
 * 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:
    TreeNode* searchBST(TreeNode* root, int val) {
        while (root != nullptr && root->val != val) {
      root = (root->val > val) ? root->left : root->right;
    }
        return root;
    }
};
```

Your previous code was restored from your local storage. Reset to default



1022. Sum of Root To Leaf Binary Numbers

Easy 2705 157 Add to List Share

You are given the `root` of a binary tree where each node has a value `0` or `1`. Each root-to-leaf path represents a binary number starting with the most significant bit.

- For example, if the path is `0 -> 1 -> 1 -> 0 -> 1`, then this could represent `01101` in binary, which is `13`.

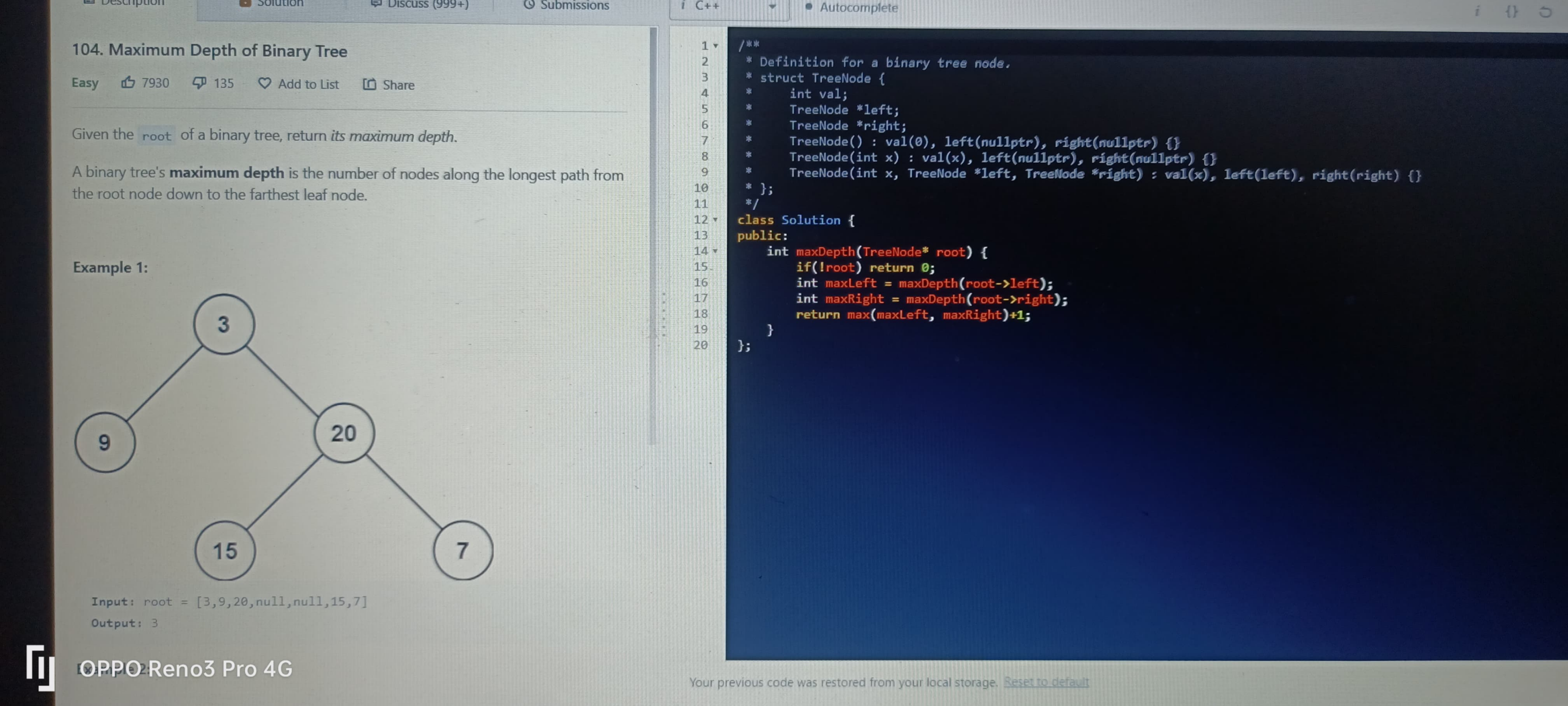
For all leaves in the tree, consider the numbers represented by the path from the root to that leaf. Return *the sum of these numbers*.

The test cases are generated so that the answer fits in a **32-bits** integer.

Example 1:

```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 sumRootToLeaf(TreeNode* root, int sum) {
        if(!root) return 0;
        sum=(sum<<1)+root->val;
        if(!root->left && !root->right) return sum;
        return sumRootToLeaf(root->left,sum) + sumRootToLeaf(root->right,sum);
    }
    public:
    int sumRootToLeaf(TreeNode* root) {
        return sumRootToLeaf(root,0);
    }
};
```

Your previous code was restored from your local storage. Reset to default



104. Maximum Depth of Binary Tree

Given the `root` of a binary tree, return *its 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.

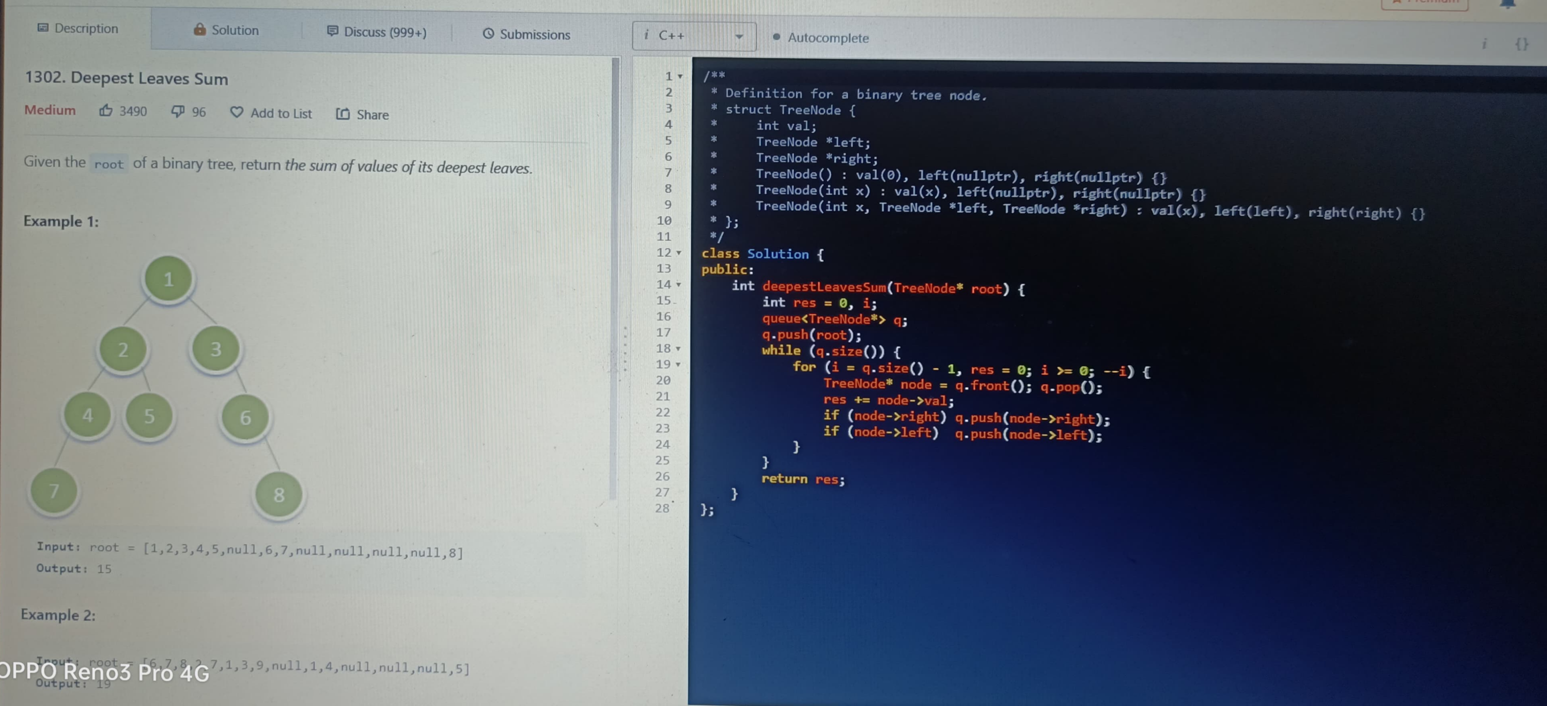
Example 1:

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

Example 2:

```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) {
        if(!root) return 0;
        int maxLeft = maxDepth(root->left);
        int maxRight = maxDepth(root->right);
        return max(maxLeft, maxRight)+1;
    }
};
```

Your previous code was restored from your local storage. Reset to default



1302. Deepest Leaves Sum

Given the `root` of a binary tree, return *the sum of values of its deepest leaves*.

Example 1:

```
Input: root = [1,2,3,4,5,null,6,7,null,null,null,null,8]
Output: 15
```

Example 2:

```
Input: root = [6,7,8,2,7,1,3,9,null,1,4,null,null,null,5]
Output: 19
```

```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 deepestLeavesSum(TreeNode* root) {
        int res = 0, i;
        queue<TreeNode*> q;
        q.push(root);
        while (q.size()) {
            for (i = q.size() - 1, res = 0; i >= 0; --i) {
                TreeNode* node = q.front(); q.pop();
                res += node->val;
                if (node->right) q.push(node->right);
                if (node->left)  q.push(node->left);
            }
        }
        return res;
    }
};
```