UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE/IT-6th Sem)

Subject Name: Advanced Programming Lab - 2

Subject Code: 22CSP-351

Submitted to:

Mr. Vishal

Submitted by:

Name: Sanya Singh

UID: 23BCS14374

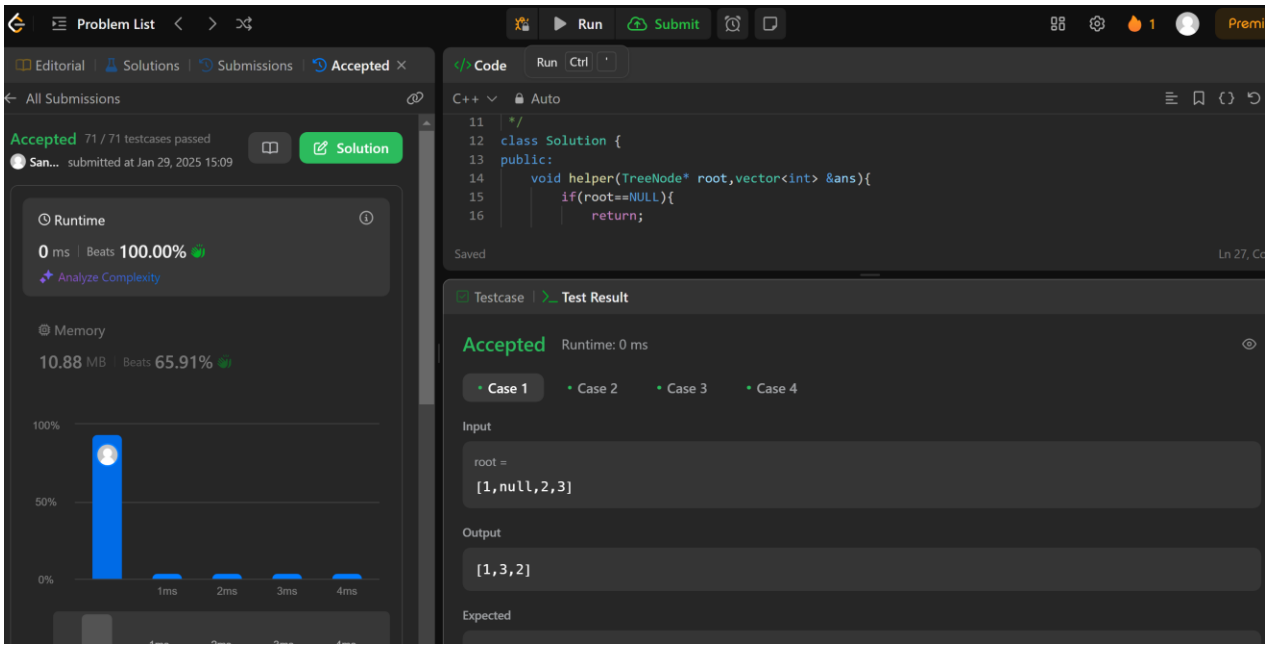
Section: FL_IOT_604

Group: A

ASSIGNMENT-3

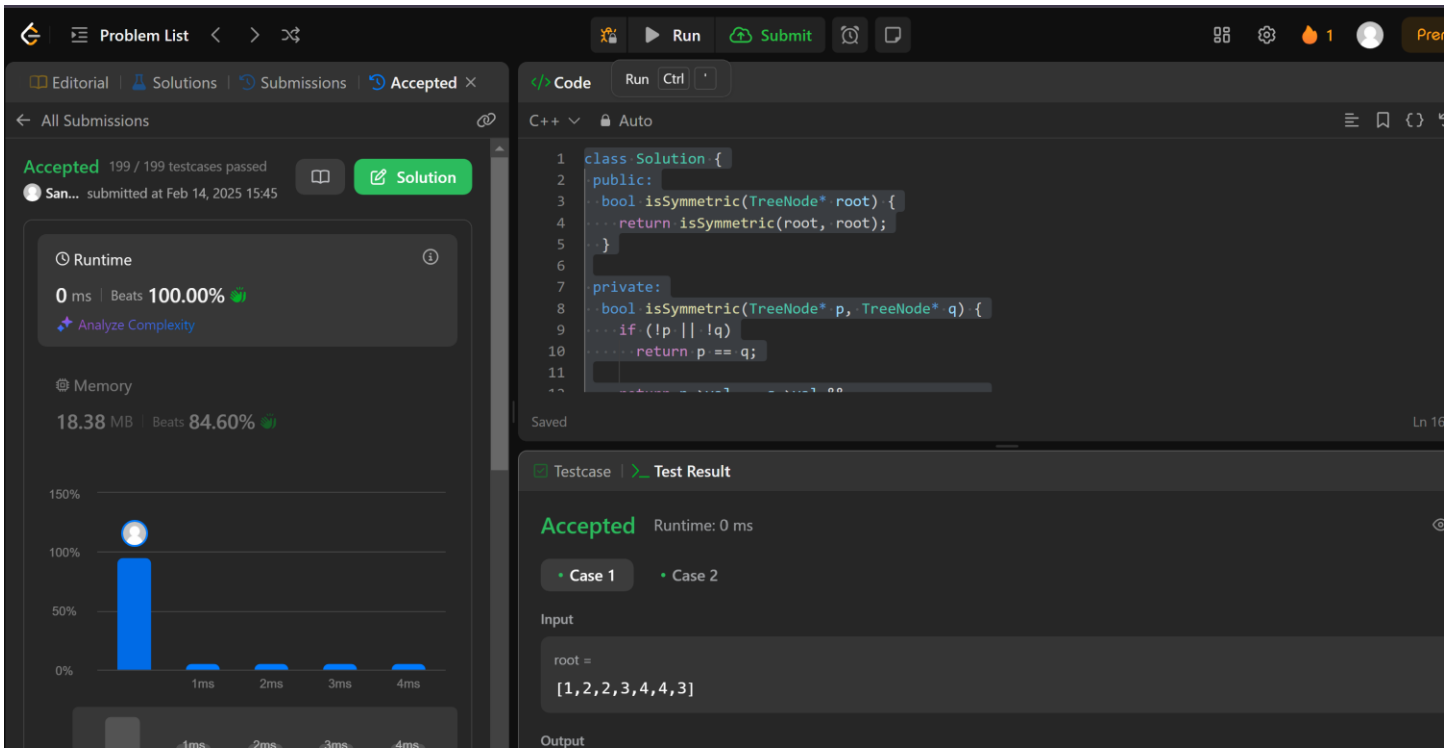
94.Binary Tree Inorder Traversal

```
class Solution {
public:
    void helper(TreeNode* root,vector<int> &ans){
        if(root==NULL){
            return;
        }
        helper(root->left,ans);
       ans.push_back(root->val);
        helper(root->right,ans);
    }
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        if(root==NULL){
            return ans;
        }
        helper(root,ans);
        return ans;
    }};
```



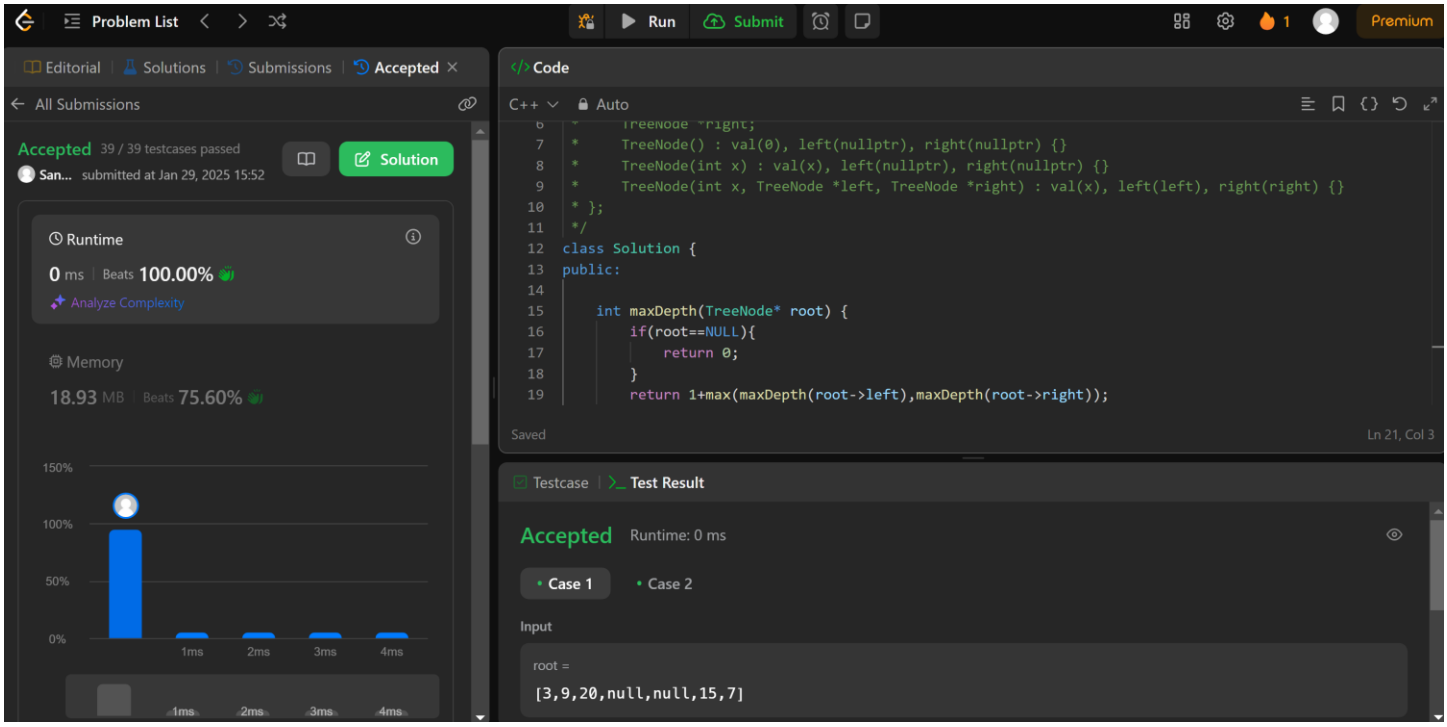
101.Symmetric Tree

```
class Solution {
 public:
  bool isSymmetric(TreeNode* root) {
    return isSymmetric(root, root);
  }
 private:
  bool isSymmetric(TreeNode* p, TreeNode* q) {
    if (!p || !q)
      return p == q;
    return p->val == q->val &&
           isSymmetric(p->left, q->right) &&
           isSymmetric(p->right, q->left);
  }
};
```



104.[Maximum Depth of Binary Tree]

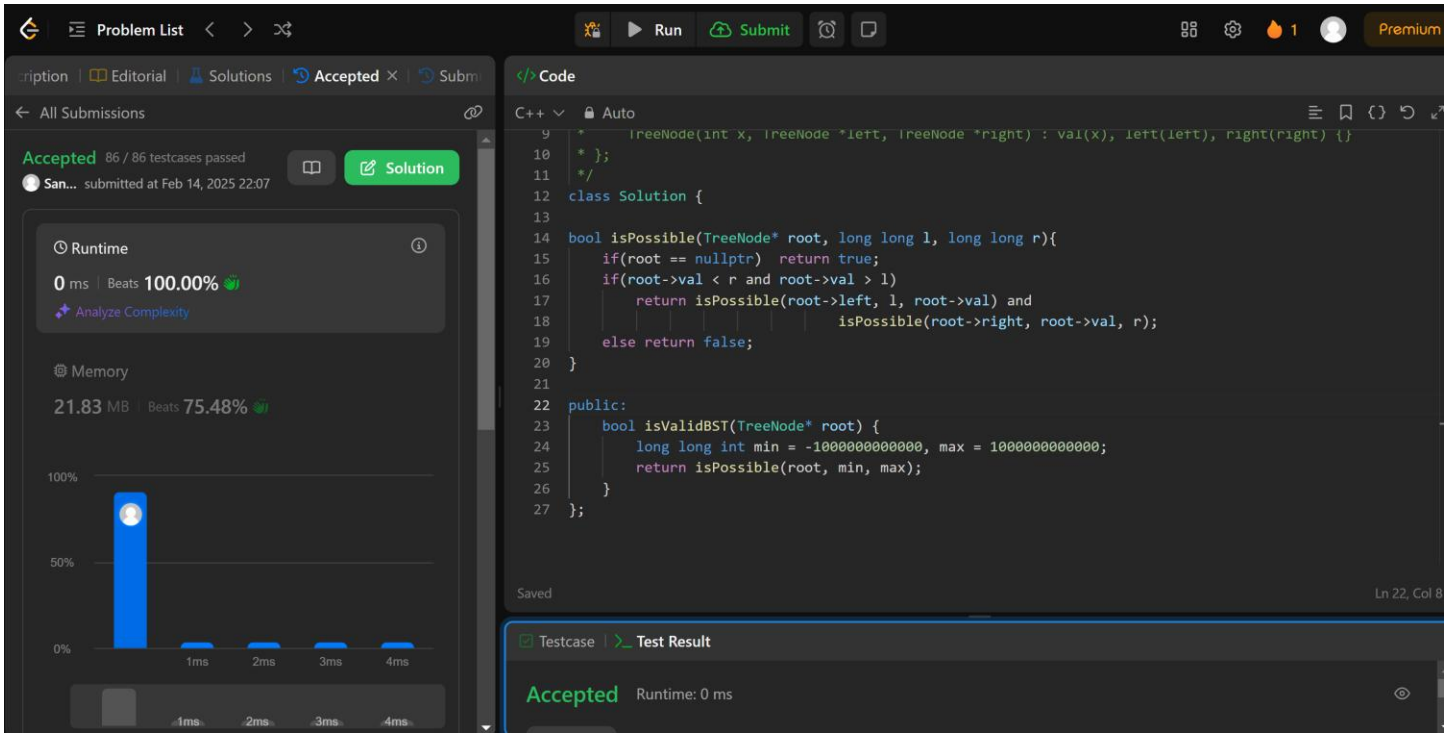
```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root==NULL){
            return 0;
        }
        return 1+max(maxDepth(root->left),maxDepth(root->right));
    }
};
```



98.[Validate Binary Search Tree]

```
class Solution {


bool isPossible(TreeNode* root, long long l, long long r){
    if(root == nullptr)  return true;
    if(root->val < r and root->val > l)
        return isPossible(root->left, l, root->val) and
```

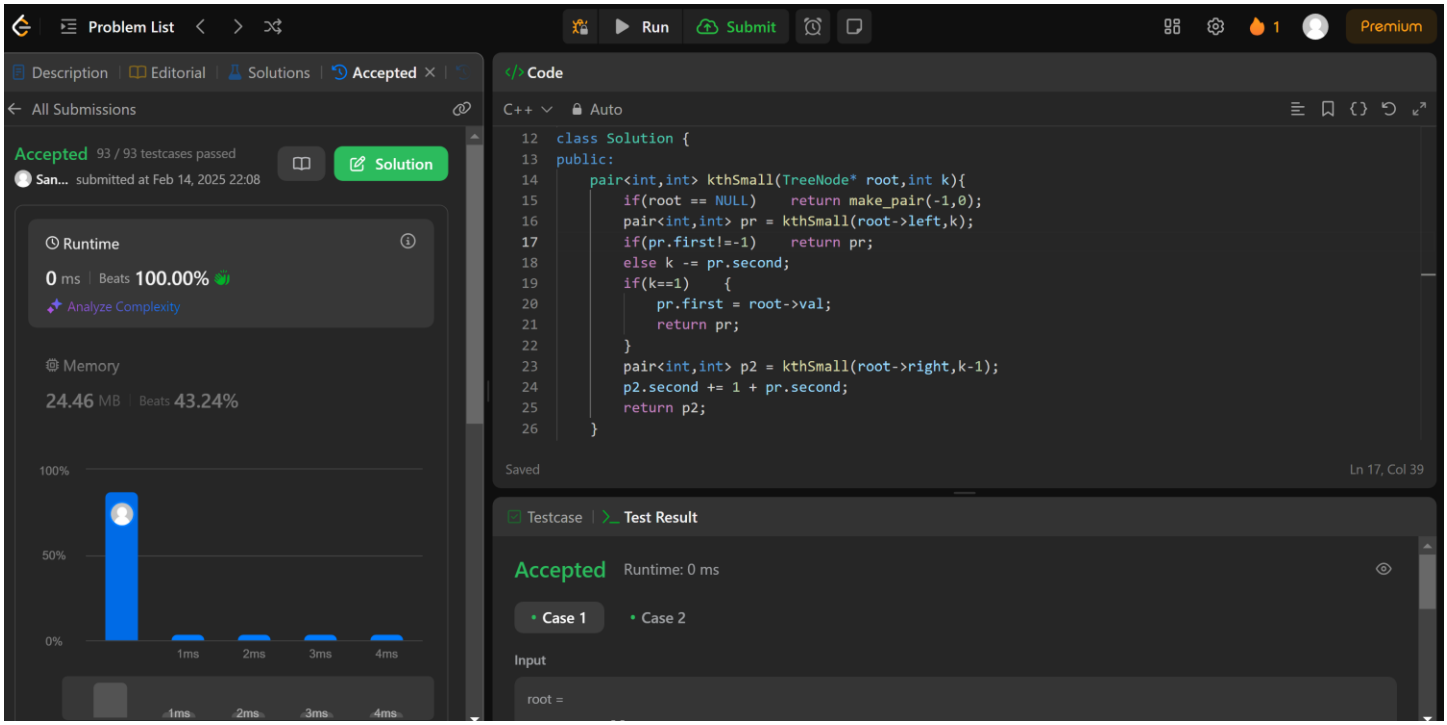
```
                isPossible(root->right, root->val, r);
    else return false;
}

public:
    bool isValidBST(TreeNode* root) {
        long long int min = -1000000000000, max = 1000000000000;
        return isPossible(root, min, max);
    }
};
```



230.[Kth Smallest Element in a BST]

```
class Solution {
public:
    pair<int,int> kthSmall(TreeNode* root,int k){
        if(root == NULL)    return make_pair(-1,0);
        pair<int,int> pr = kthSmall(root->left,k);
        if(pr.first!=-1)    return pr;
        else k -= pr.second;
```

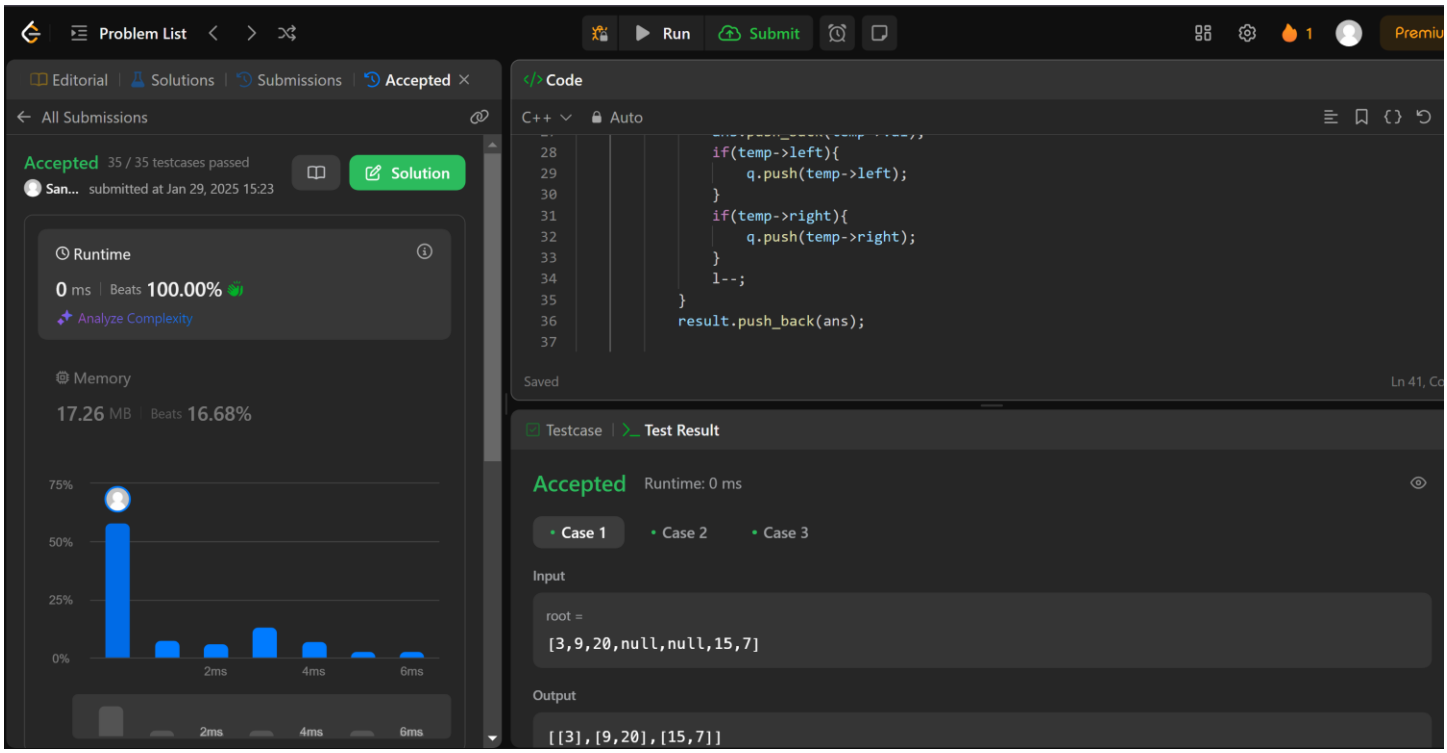
```
        if(k==1)    {
            pr.first = root->val;
            return pr;
        }
        pair<int,int> p2 = kthSmall(root->right,k-1);
        p2.second += 1 + pr.second;
        return p2;
    }
    int kthSmallest(TreeNode* root, int k) {
        return kthSmall(root,k).first;
    }
};
```



102. [Binary Tree Level Order Traversal]

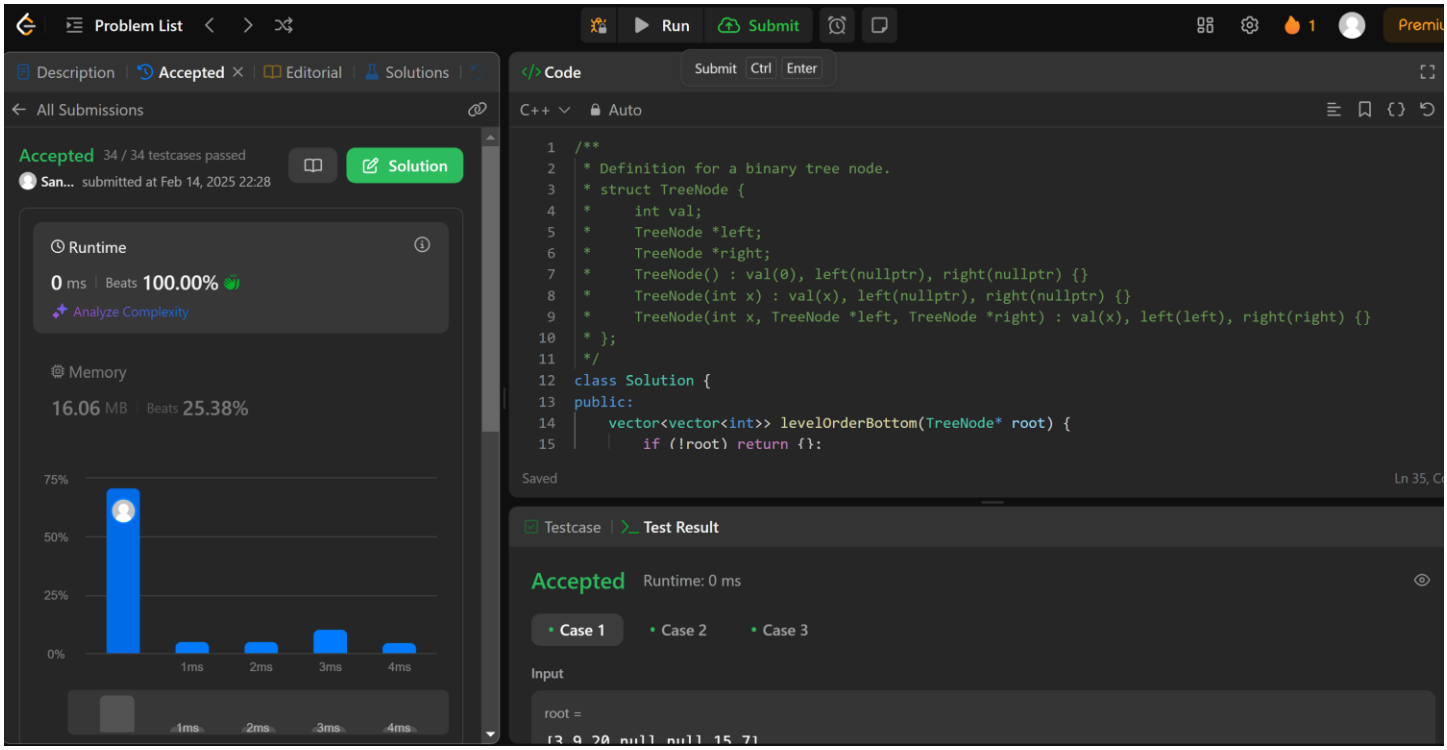
```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if(root==NULL){
            return result;
```

```
}
queue<TreeNode*> q;
q.push(root);
while(!q.empty()){
    int l=q.size();
    vector<int>  ans;
    while(l>0){
        TreeNode* temp=q.front();
        q.pop();
        ans.push_back(temp->val);
        if(temp->left){
            q.push(temp->left);
        }
        if(temp->right){
            q.push(temp->right);}
        l--;}
    result.push_back(ans);}
return result;}
```



107.Binary Tree Level Order Traversal II

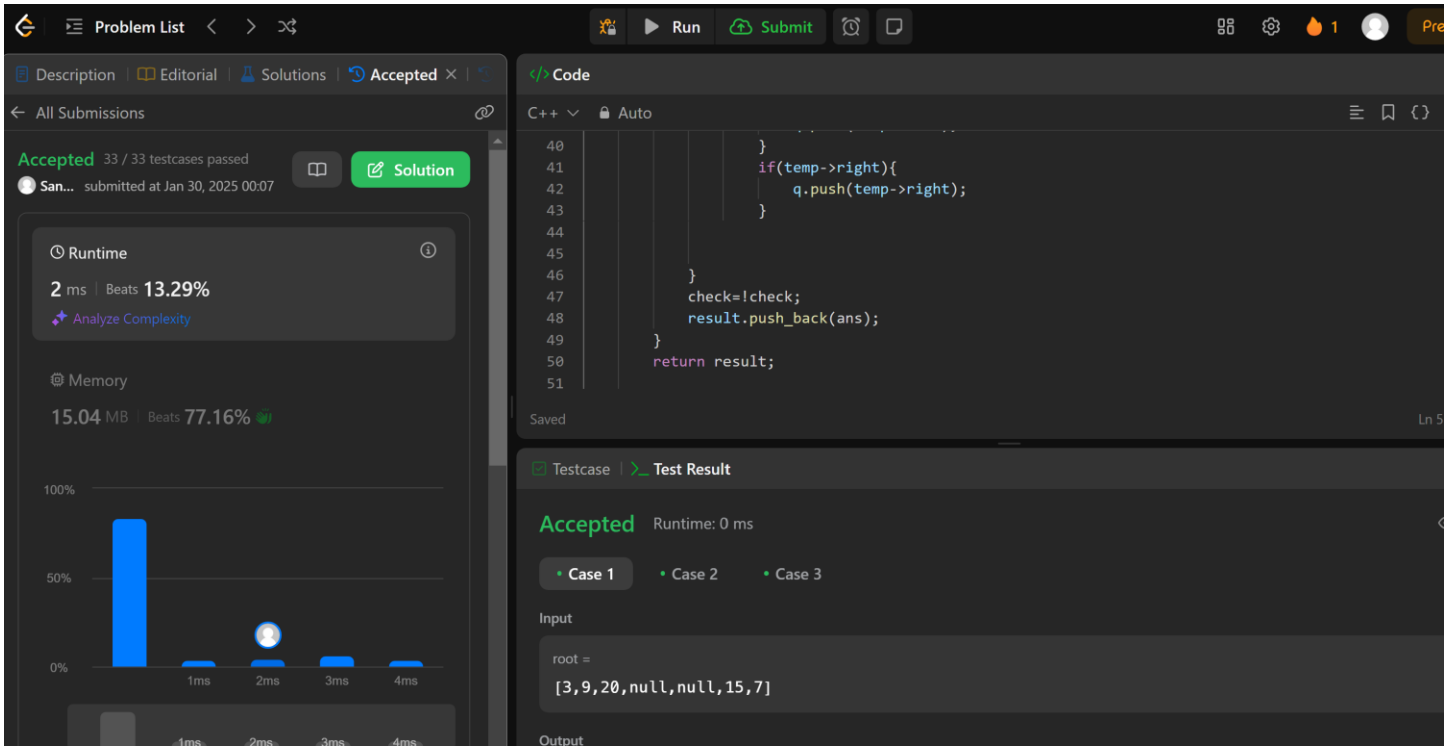
```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        if (!root) return {};
        vector<vector<int>> result;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
            vector<int> level;
            for (int i = 0; i < size; ++i) {
                TreeNode* node = q.front();
                q.pop();
                level.push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(level);
        }
        reverse(result.begin(), result.end());
        return result;
    }
};
```



103.[Binary Tree Zigzag Level Order Traversal]

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

        bool check=true;
        while(!q.empty()){
            int l=q.size();
            vector<int> ans(l);
            for(int i=0;i<l;i++){
                TreeNode* temp=q.front();
                q.pop();
```

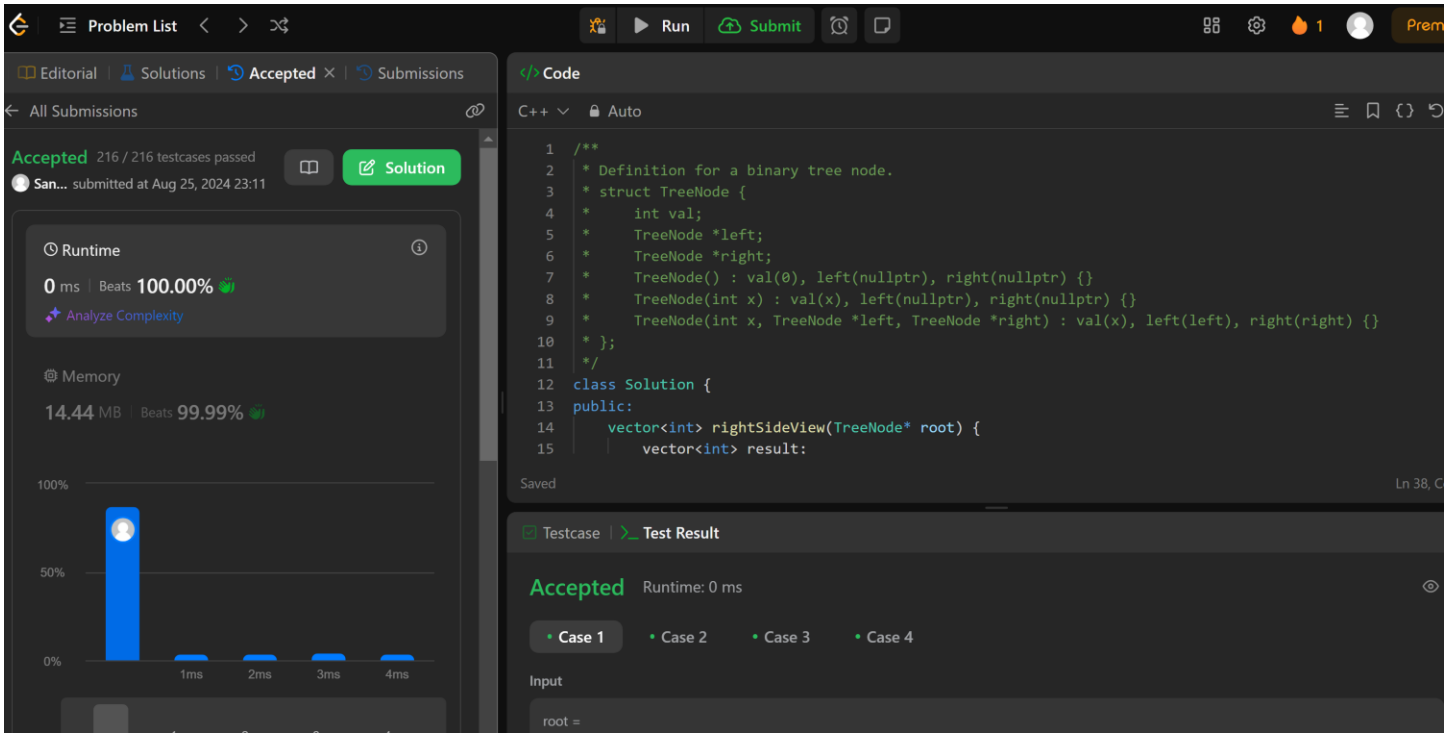
```
        int index;
        if(check==true){
            index=i;
        }
        else{
            index=l-i-1;
        }
        ans[index]=temp->val;
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);}}
    check=!check;
    result.push_back(ans);
}
return result;}}
```



199.Binary Tree Right Side View

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> result;
        if(root==NULL){
            return result;
        }
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()){
            int n=q.size();

            for(int i=0;i<n;i++){
            TreeNode* temp = q.front();
            q.pop();
            if(i == n-1) {
                result.push_back(temp->val);
            }
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);
            }
            }

        }
        return result;
    }
```

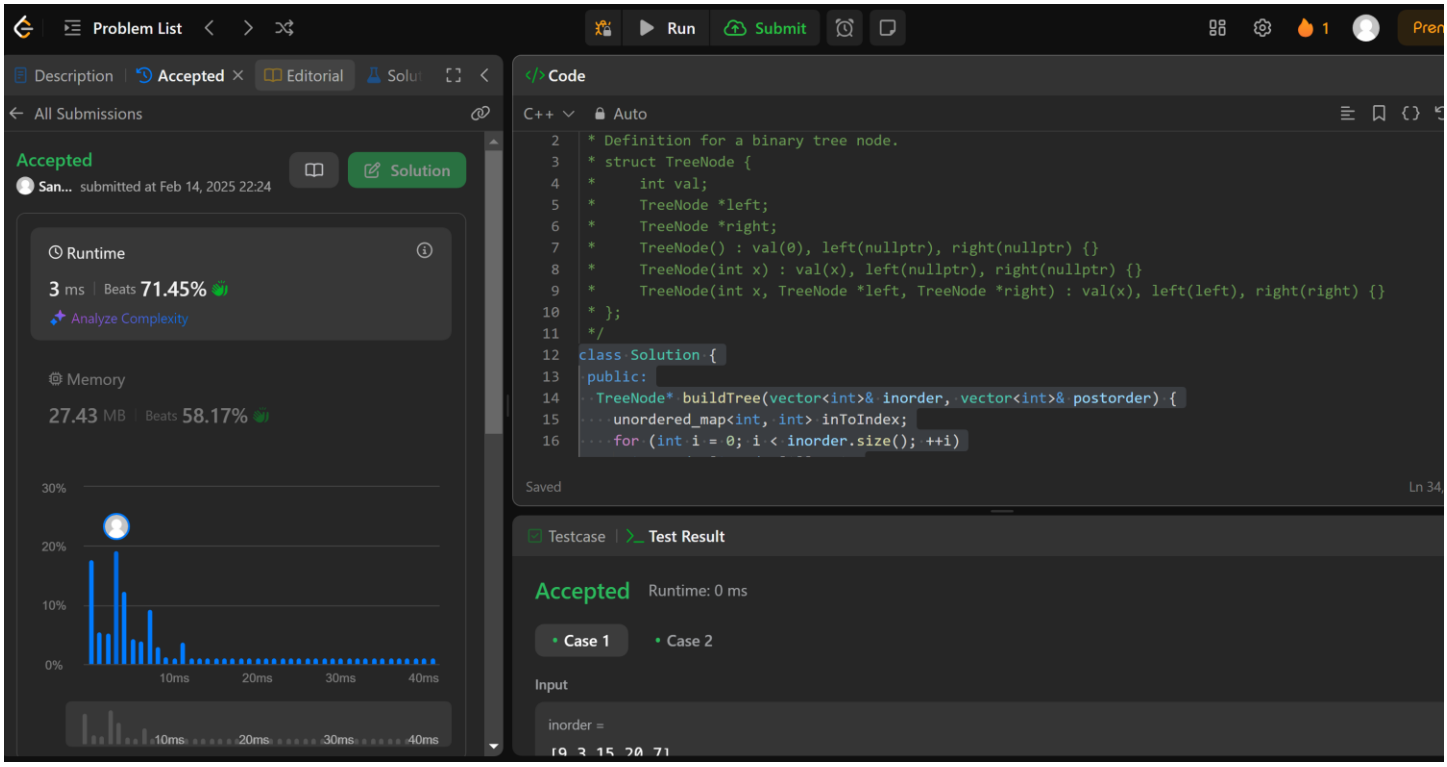
};



106.[Construct Binary Tree from Inorder and Postorder Traversal]

```
class Solution {
 public:
  TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
    unordered_map<int, int> inToIndex;
    for (int i = 0; i < inorder.size(); ++i)
      inToIndex[inorder[i]] = i;
    return build(inorder, 0, inorder.size() - 1, postorder, 0, postorder.size() - 1, inToIndex);
  }

 private:
  TreeNode* build(const vector<int>& inorder, int inStart, int inEnd, const vector<int>& postorder, int postStart,
int postEnd, const unordered_map<int, int>& inToIndex) {
    if (inStart > inEnd)
      return nullptr;

    const int rootVal = postorder[postEnd];
    const int rootInIndex = inToIndex.at(rootVal);
    const int leftSize = rootInIndex - inStart;
```

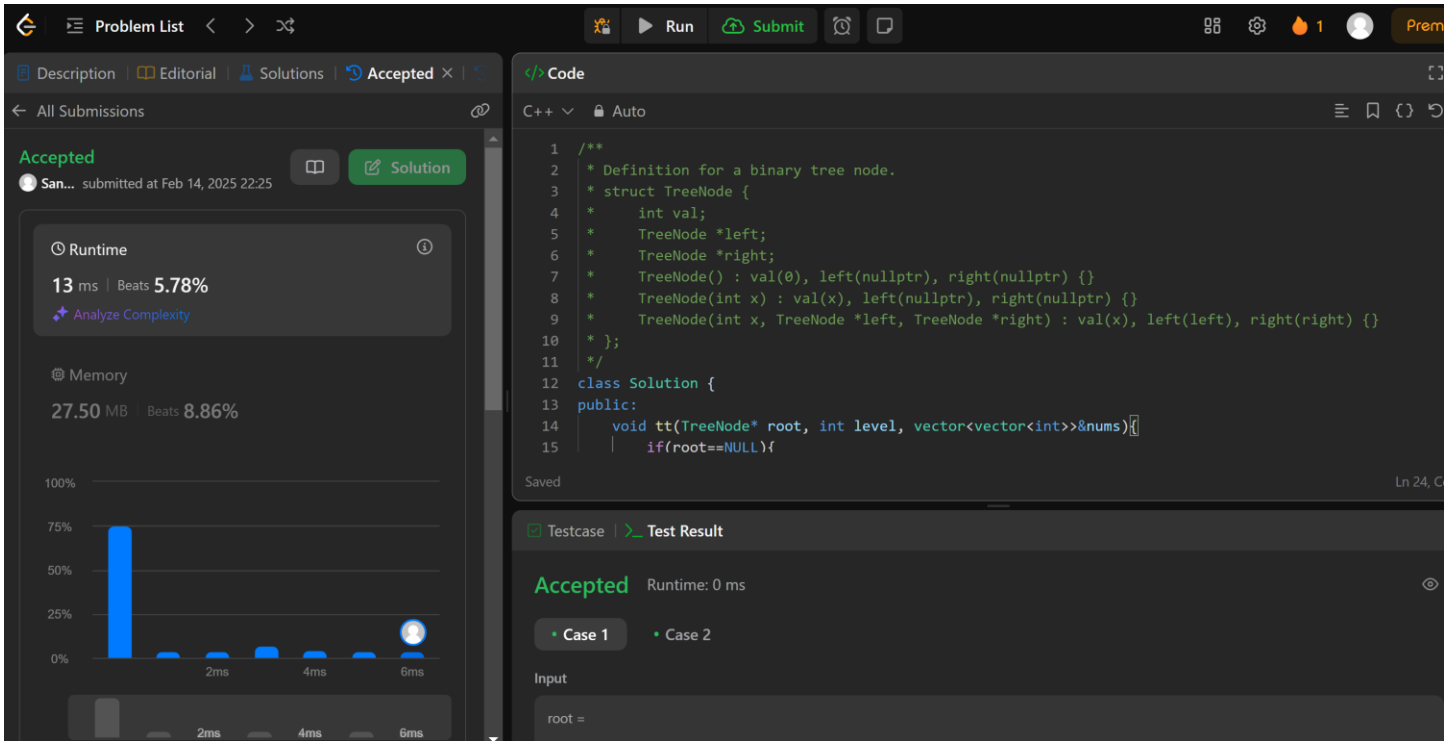
```
    TreeNode* root = new TreeNode(rootVal);
    root->left = build(inorder, inStart, rootInIndex - 1, postorder, postStart, postStart + leftSize - 1, inToIndex);
    root->right = build(inorder, rootInIndex + 1, inEnd, postorder, postStart + leftSize, postEnd - 1, inToIndex);
    return root;
  }
};
```



513.[Find Bottom Left Tree Value]

```
class Solution {
public:
    void tt(TreeNode* root, int level, vector<vector<int>>&nums){
        if(root==NULL){
            return;
        }
        if(nums.size()<=level){
            nums.push_back({});
        }

        nums[level].push_back(root->val);
```

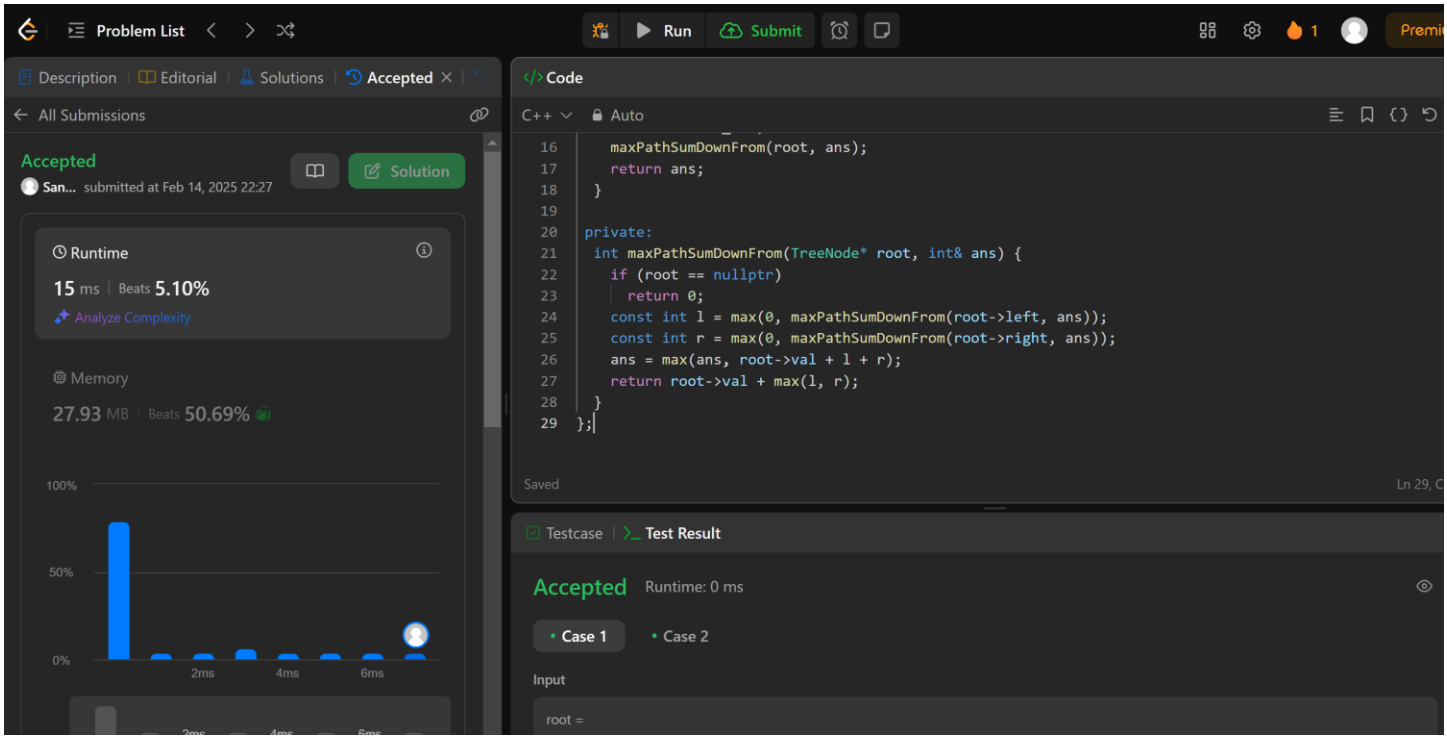
```
        tt(root->right,level+1,nums);
        tt(root->left,level+1,nums);


    }
    int findBottomLeftValue(TreeNode* root) {
       vector<vector<int>>nums;
        tt(root,0,nums);
        return nums.back().back();
    }
};
```



124. [Binary Tree Maximum Path Sum]

```
class Solution {
 public:
  int maxPathSum(TreeNode* root) {
    int ans = INT_MIN;
    maxPathSumDownFrom(root, ans);
    return ans;
  }
```

```
 private:
  int maxPathSumDownFrom(TreeNode* root, int& ans) {
    if (root == nullptr)
      return 0;
    const int l = max(0, maxPathSumDownFrom(root->left, ans));
    const int r = max(0, maxPathSumDownFrom(root->right, ans));
    ans = max(ans, root->val + l + r);
    return root->val + max(l, r);
  }
};
```

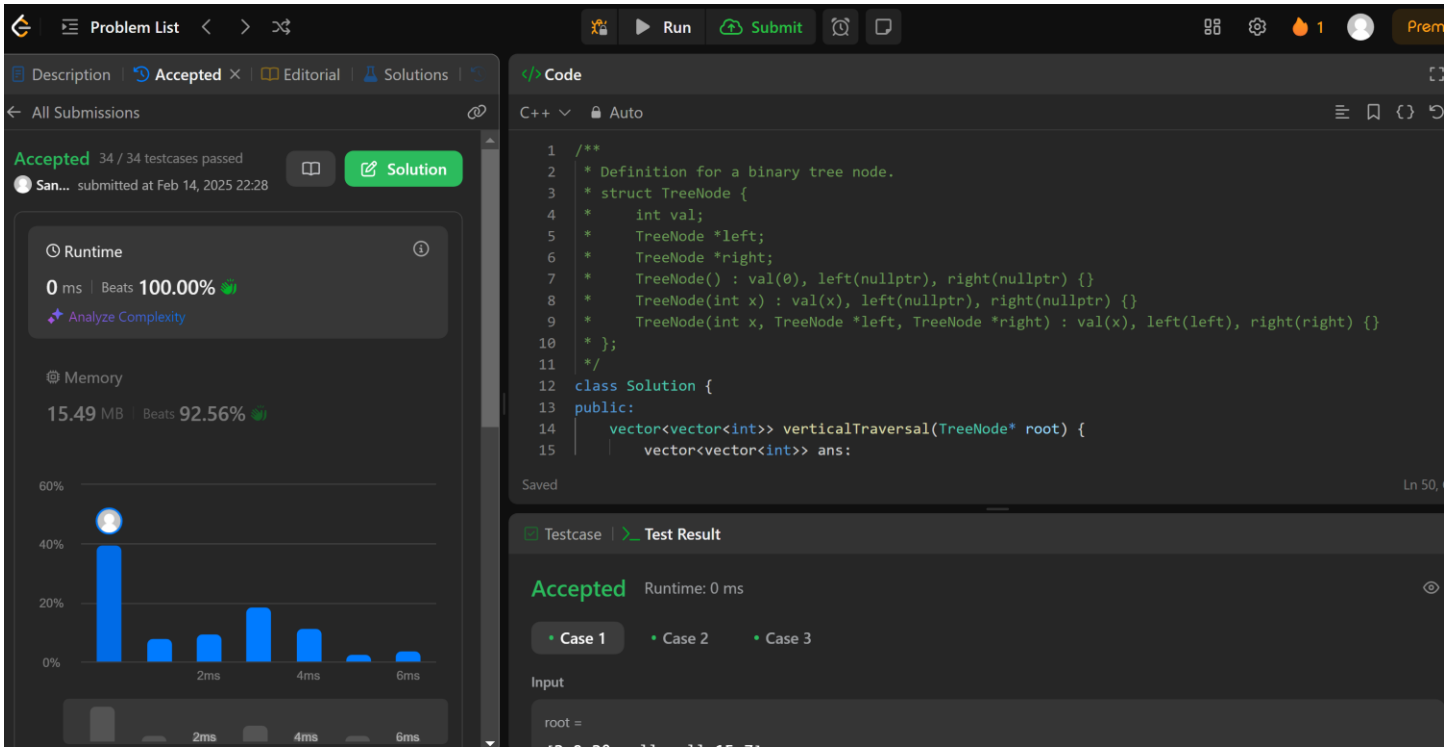


987.[Vertical Order Traversal of a Binary Tree]

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        vector<vector<int>> ans;

        queue<pair<TreeNode*,int>> Q; // node and col
        Q.push({root,0});
        int depth=0;
```

```
    while(!Q.empty()){
        int s=Q.size();
        while(s--){
            auto [node,col]=Q.front();
            Q.pop();
            ans.push_back({col,depth,node->val});
            if(node->left!=nullptr) Q.push({node->left,col-1});
            if(node->right!=nullptr) Q.push({node->right,col+1});
        }
        depth++;
    }

    sort(ans.begin(),ans.end());

    vector<vector<int>> final;
    vector<int> temp;
    int curr=ans[0][0];
    for(int i=0;i<ans.size();i++){
        if(ans[i][0]==curr) temp.push_back(ans[i][2]);
        else{
            final.push_back(temp);
            temp.clear();
            curr=ans[i][0];
            temp.push_back(ans[i][2]);
        }
    }
    if(!temp.empty()) final.push_back(temp);
    return final;

}
```

};


Problem List
Run
Submit
Description
Accepted
Editorial
Solutions
All Submissions
Accepted 34 / 34 testcases passed
San... submitted at Feb 14, 2025 22:28
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
15.49 MB | Beats 92.56%
60%
40%
20%
0%
2ms
4ms
6ms
Code
C++
Auto
/**
* 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:
vector<vector<int>> verticalTraversal(TreeNode* root) {
vector<vector<int>> ans:
Saved
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
root =