**ADVANCED PROGRAMMING LAB-2 ASSIGNMENT**

**Submitted By:**

**Name- Tanvi Chhabra**

**UID- 22BCS15679**

**Section-22BCS\_IOT\_605-B**

1. **Binary Tree Inorder Traversal**

class Solution {

public:

    void traverse(TreeNode\* root, vector<int>&arr){

        if(root==NULL) return;

        traverse(root->left, arr);

        arr.push\_back(root->val);

        traverse(root->right, arr);

    }

    vector<int> inorderTraversal(TreeNode\* root) {

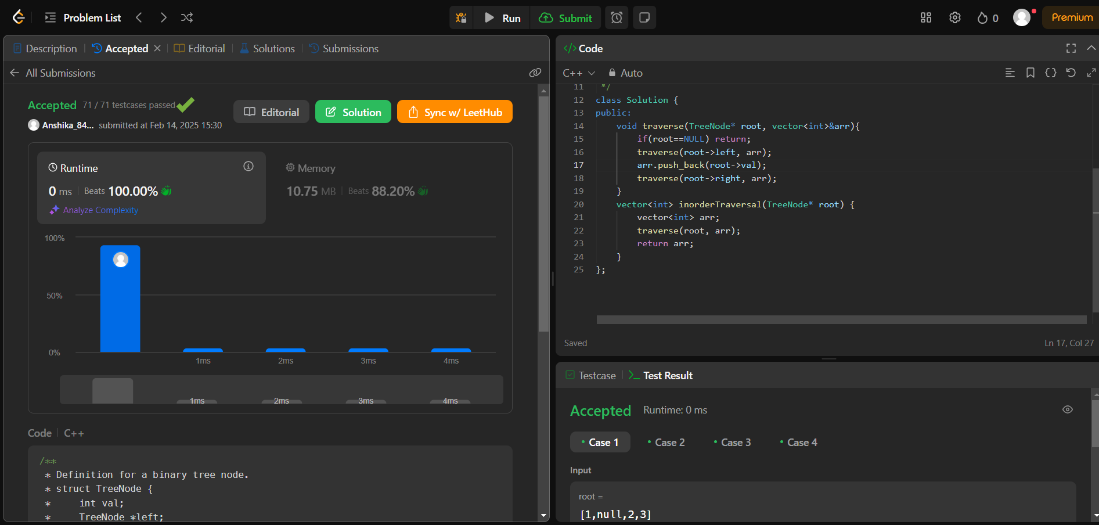
        vector<int> arr;

        traverse(root, arr);

        return arr;

    }

};



1. **Binary Tree Level Order traversal**

class Solution {

public:

    vector<vector<int>> levelOrder(TreeNode\* root) {

    vector<vector<int>> result;

    if(!root) return result;

    queue<TreeNode\*> q;

    q.push(root);

    while(!q.empty()){

        int n= q.size();

        vector<int> arr;

        while(n){

            n--;

            TreeNode\* temp= q.front();

            q.pop();

            if(!temp) continue;

            arr.push\_back(temp->val);

            if(temp->left){

                q.push(temp->left);

            }

            if(temp->right){

                q.push(temp->right);

            }

        }

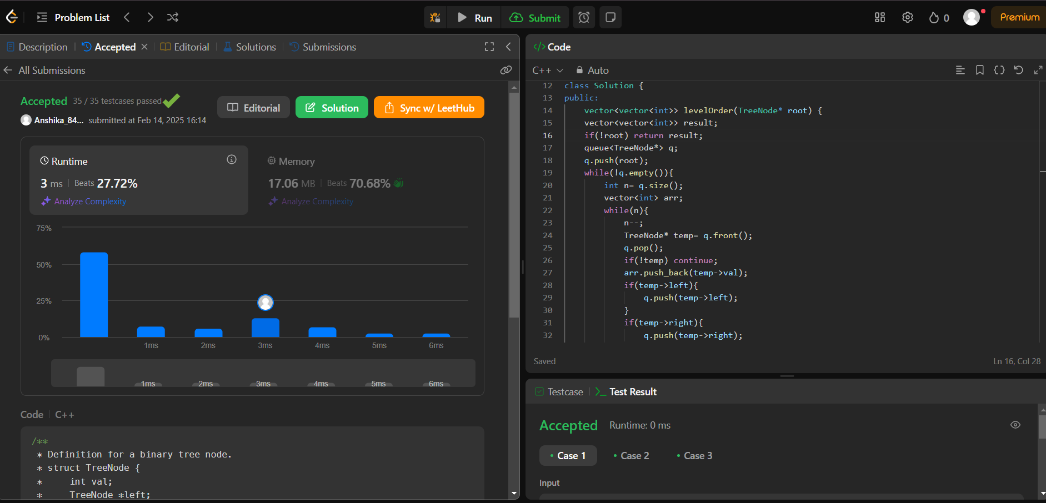
        result.push\_back(arr);

    }

    return result;

    }

};



1. **Symmetric tree**

class Solution {

public:

bool isMirror(TreeNode\* t1, TreeNode\* t2) {

if (!t1 && !t2) return true;

if (!t1 || !t2) return false;

return (t1->val == t2->val) && isMirror(t1->left, t2->right) && isMirror(t1->right, t2->left);

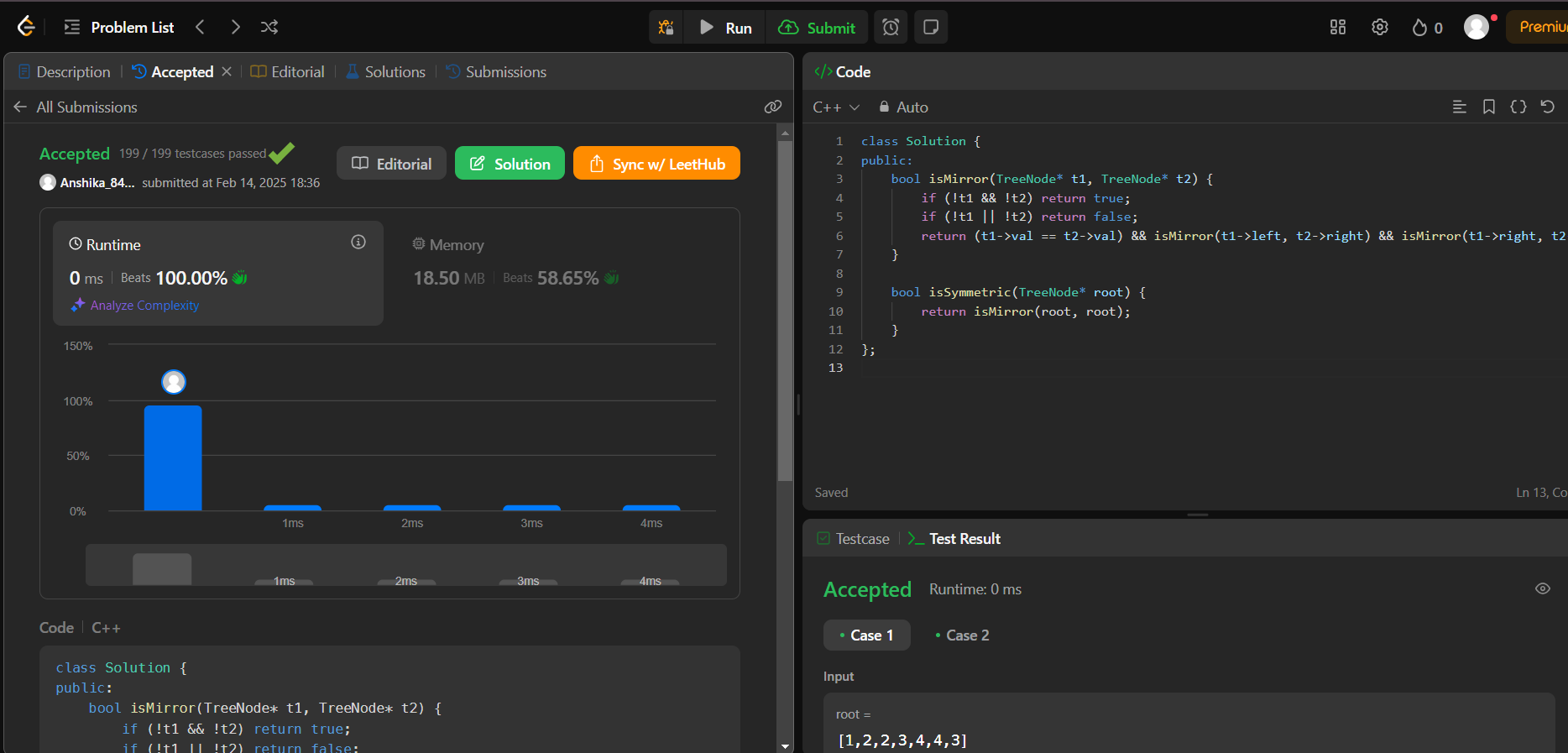
}

bool isSymmetric(TreeNode\* root) {

return isMirror(root, root);

}

};



1. **Maximum Depth of Binary Tree**

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if (root == nullptr) {

            return 0;

        }

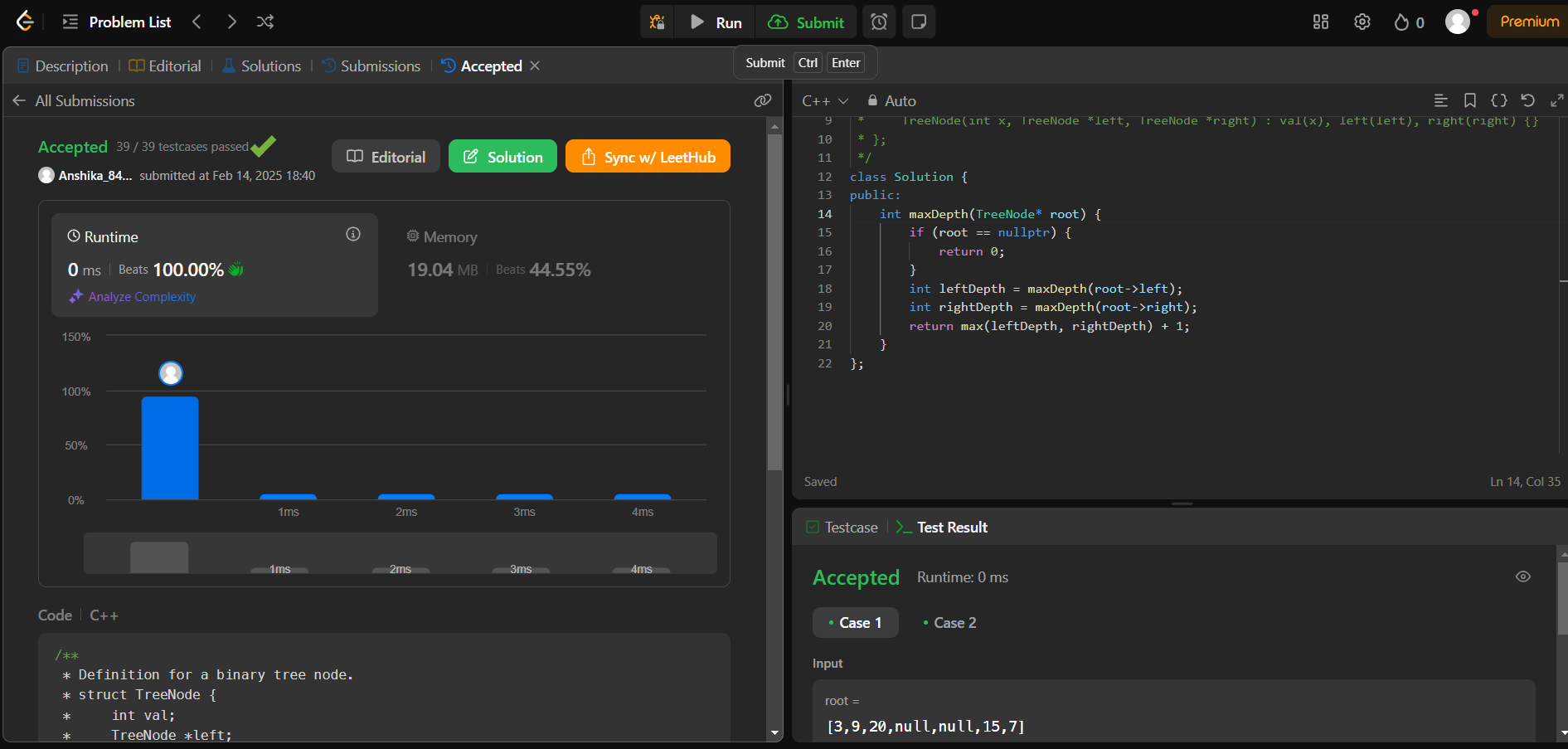
        int leftDepth = maxDepth(root->left);

        int rightDepth = maxDepth(root->right);

        return max(leftDepth, rightDepth) + 1;

    }

};



1. **Validate Binary Search Tree**

class Solution {

public:

bool isValidBST(TreeNode\* root) {

stack<TreeNode\*> st;

TreeNode\* prev = nullptr;

while (!st.empty() || root) {

while (root) {

st.push(root);

root = root->left;

}

root = st.top();

st.pop();

if (prev && root->val <= prev->val) return false;

prev = root;

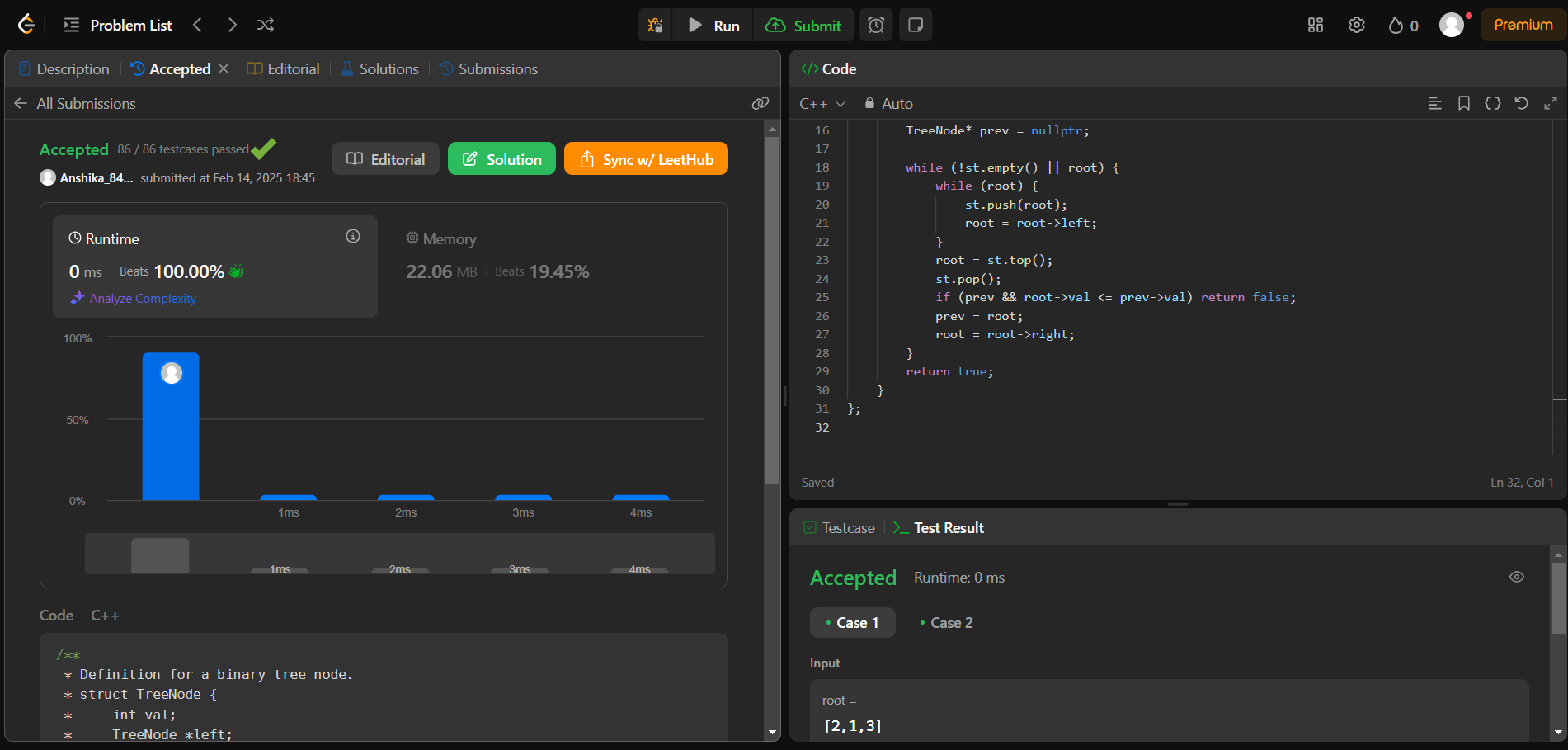
root = root->right;

}

return true;

}

};



1. **Kth smallest element in a BST**

class Solution {

public:

int count = 0, result = 0;

void inorder(TreeNode\* root, int k) {

if (!root) return;

inorder(root->left, k);

count++;

if (count == k) {

result = root->val;

return;

}

inorder(root->right, k);

}

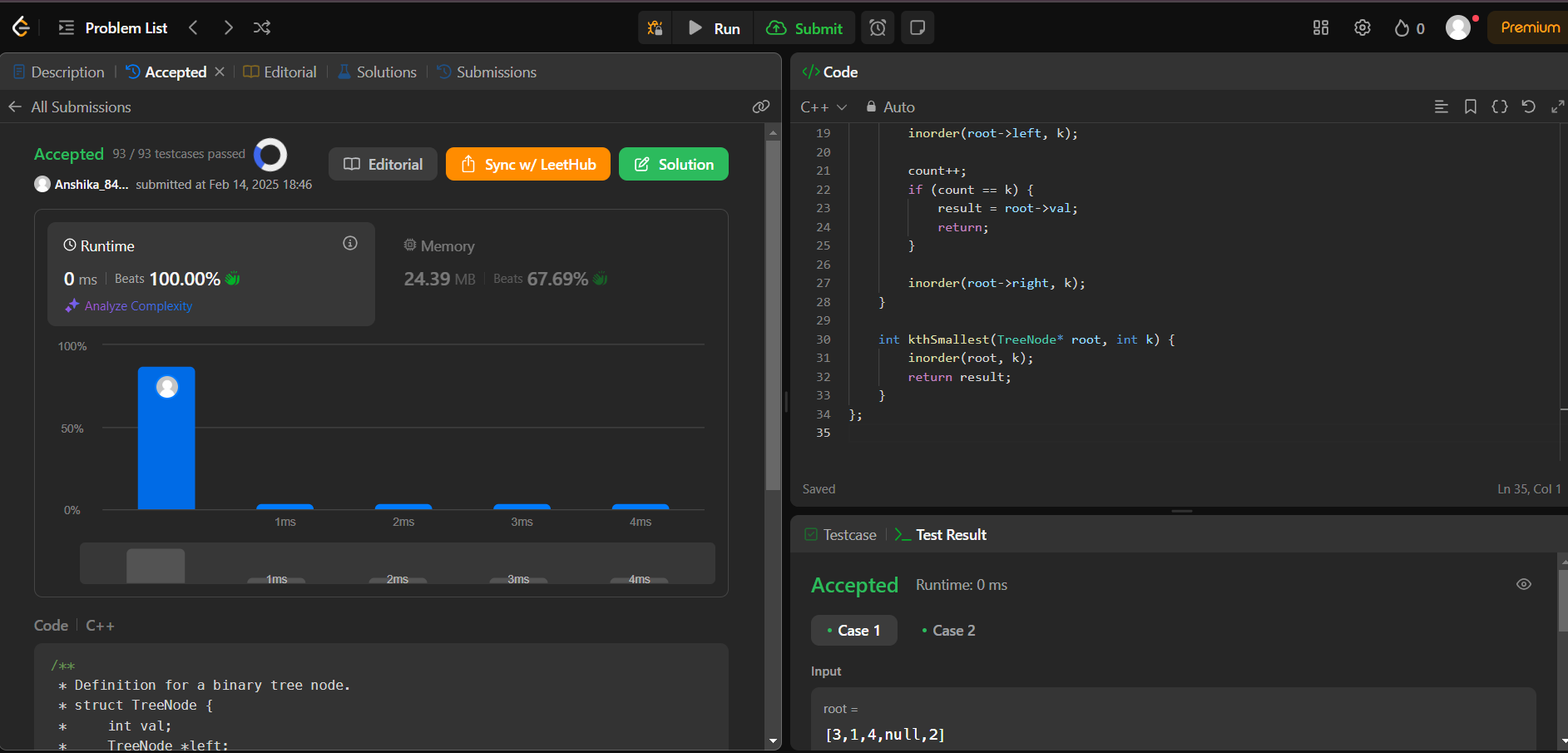
int kthSmallest(TreeNode\* root, int k) {

inorder(root, k);

return result;

}

};



1. **Binary Tree Level Order traversal II**

class Solution {

public:

vector<vector<int>> levelOrderBottom(TreeNode\* root) {

vector<vector<int>> result;

if (!root) return 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);

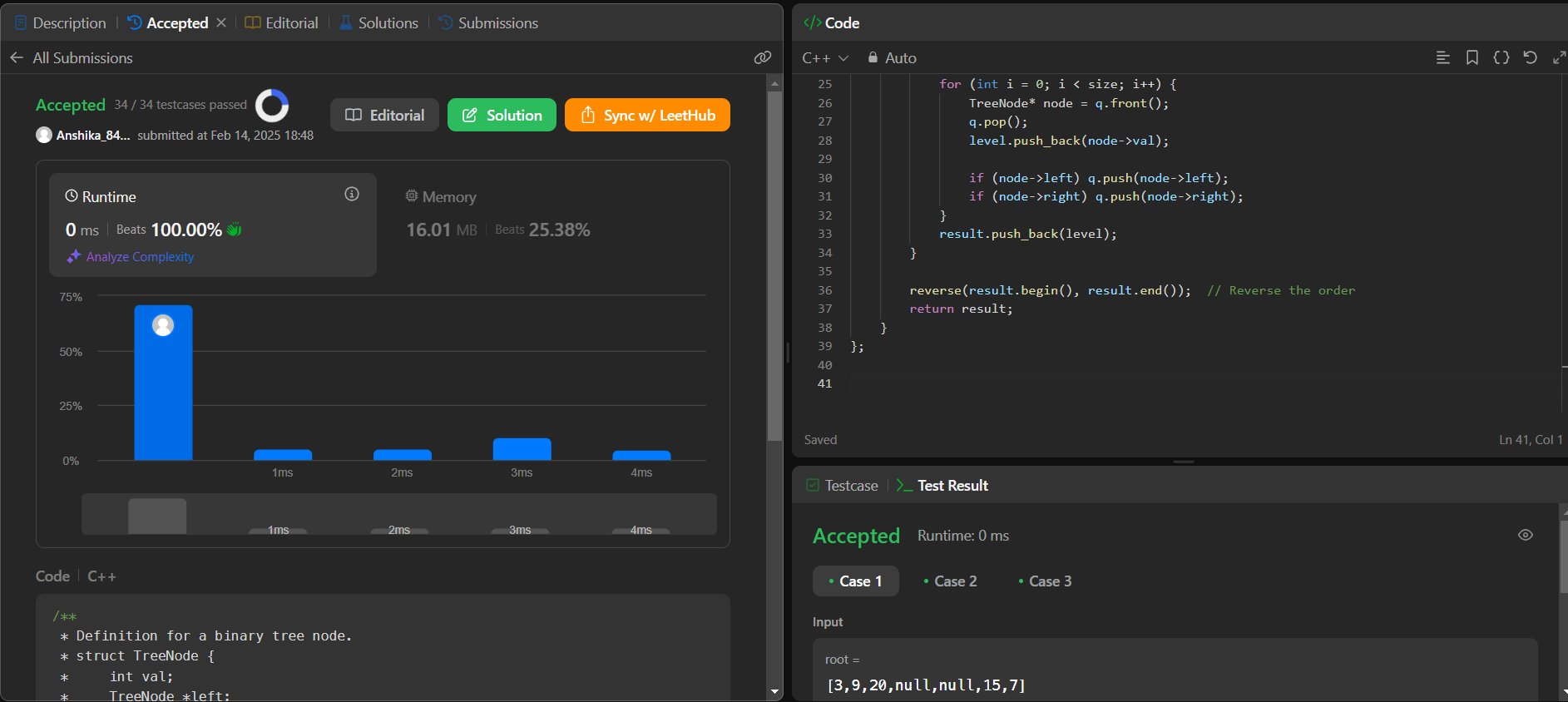
}

reverse(result.begin(), result.end()); // Reverse the order

return result;

}

};



1. **Binary Tree Zigzag Level Oder Traversal**

class Solution {

public:

vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

vector<vector<int>> result;

if (!root) return result;

queue<TreeNode\*> q;

q.push(root);

bool leftToRight = true; // Flag to toggle direction

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);

}

if (!leftToRight) reverse(level.begin(), level.end());

result.push\_back(level);

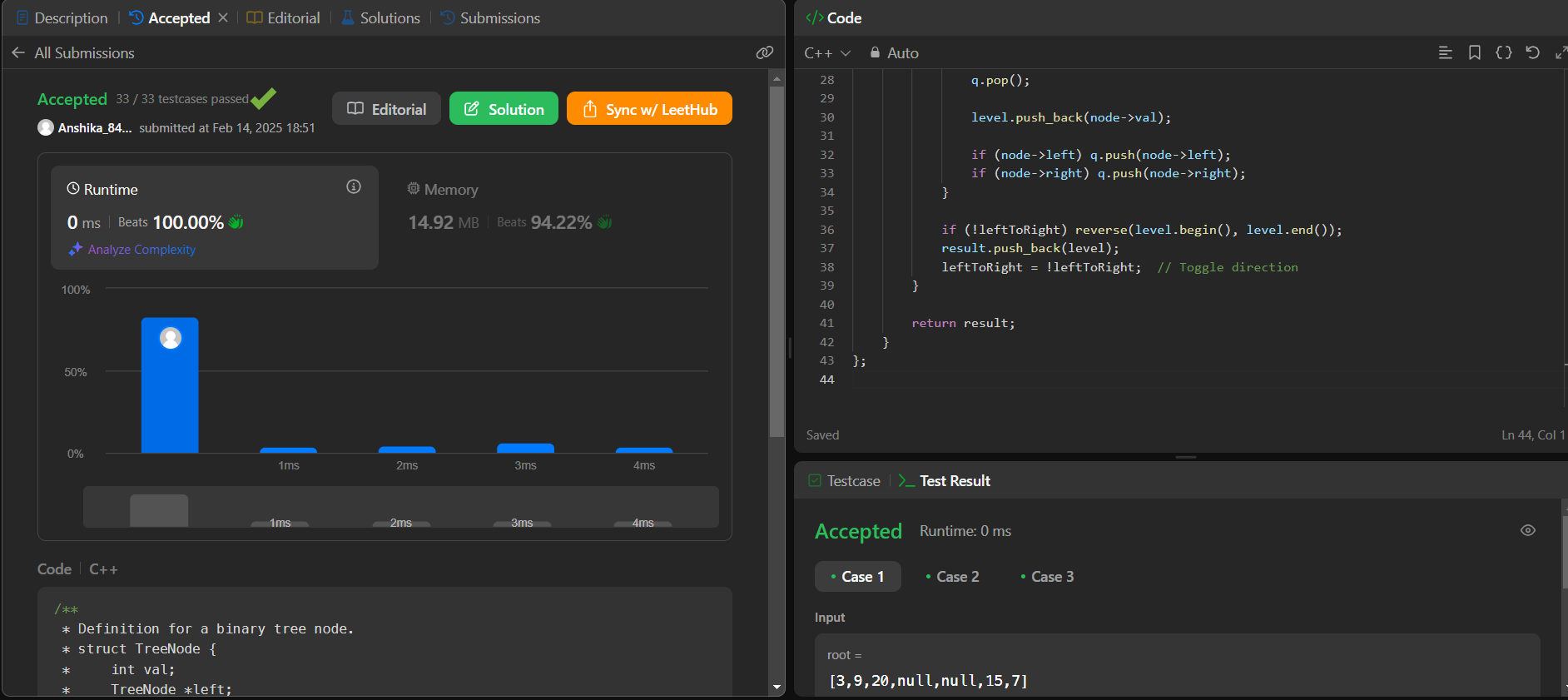
leftToRight = !leftToRight; // Toggle direction

}

return result;

}

};



1. **Binary Tree Right Side View**

class Solution {

public:

void dfs(TreeNode\* node, int level, vector<int>& result) {

if (!node) return;

if (level == result.size()) result.push\_back(node->val);

dfs(node->right, level + 1, result);

dfs(node->left, level + 1, result);

}

vector<int> rightSideView(TreeNode\* root) {

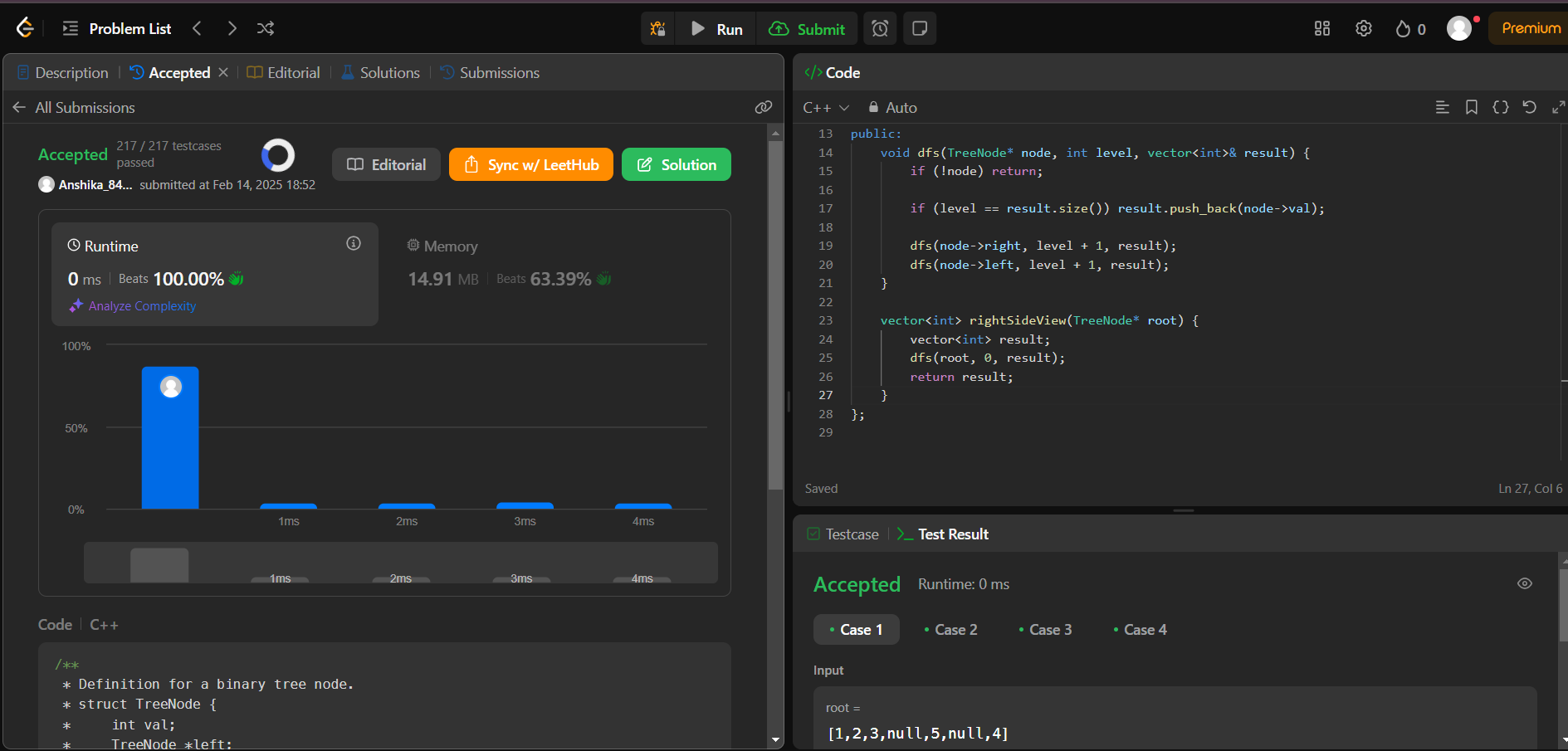
vector<int> result;

dfs(root, 0, result);

return result;

}

};



1. **Construct Binary Tree From Inorder And Postorder Traversal**

class Solution {

public:

unordered\_map<int, int> inorderMap;

int postIndex;

TreeNode\* build(vector<int>& inorder, vector<int>& postorder, int inLeft, int inRight) {

if (inLeft > inRight) return nullptr;

int rootVal = postorder[postIndex--];

TreeNode\* root = new TreeNode(rootVal);

int inIndex = inorderMap[rootVal];

root->right = build(inorder, postorder, inIndex + 1, inRight);

root->left = build(inorder, postorder, inLeft, inIndex - 1);

return root;

}

TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

postIndex = postorder.size() - 1;

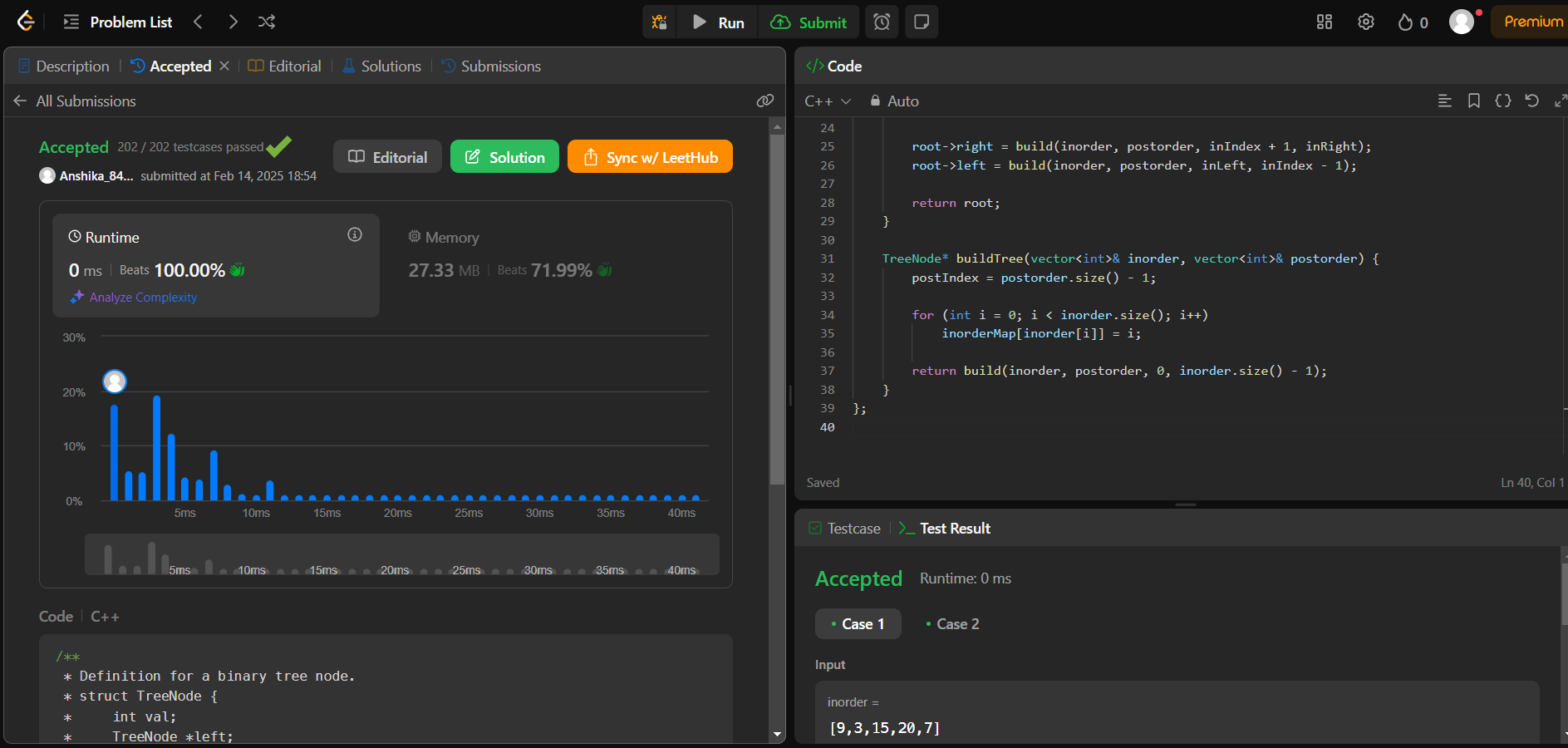
for (int i = 0; i < inorder.size(); i++)

inorderMap[inorder[i]] = i;

return build(inorder, postorder, 0, inorder.size() - 1);

}

};



1. **Find Bottom Left Tree Value**

class Solution {

public:

    int findBottomLeftValue(TreeNode\* root) {

        queue<TreeNode\*> q;

        q.push(root);

        int leftmostValue = 0;

        while (!q.empty()) {

            int size = q.size();

            leftmostValue = q.front()->val;

            for (int i = 0; i < size; i++) {

                TreeNode\* node = q.front();

                q.pop();

                if (node->left) q.push(node->left);

                if (node->right) q.push(node->right);

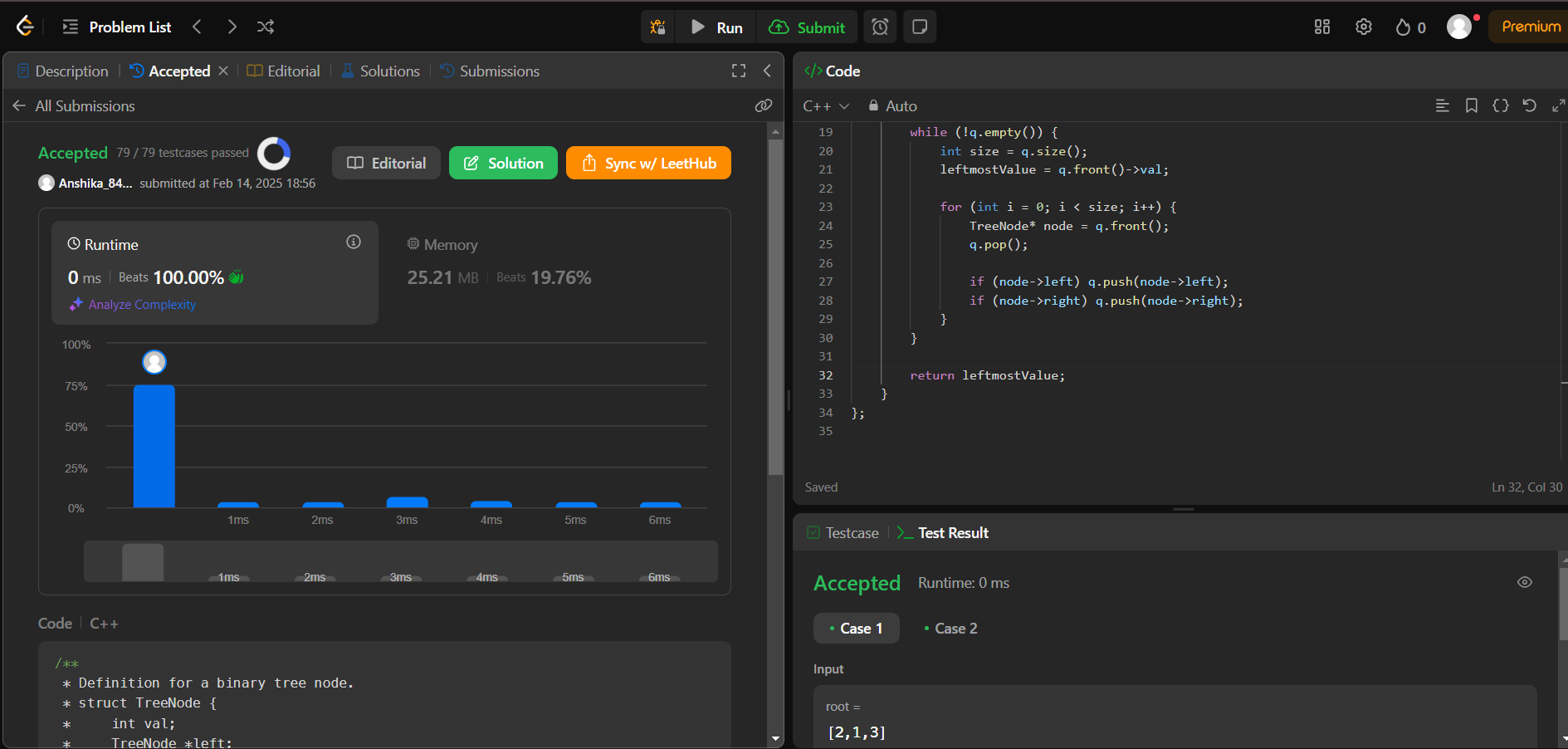
            }

        }

        return leftmostValue;

    }

};



1. **Binary Tree Maximum Path Sum**

class Solution {

public:

int maxSum = INT\_MIN;

int dfs(TreeNode\* node) {

if (!node) return 0;

int left = max(0, dfs(node->left));

int right = max(0, dfs(node->right));

maxSum = max(maxSum, left + right + node->val);

return node->val + max(left, right);

}

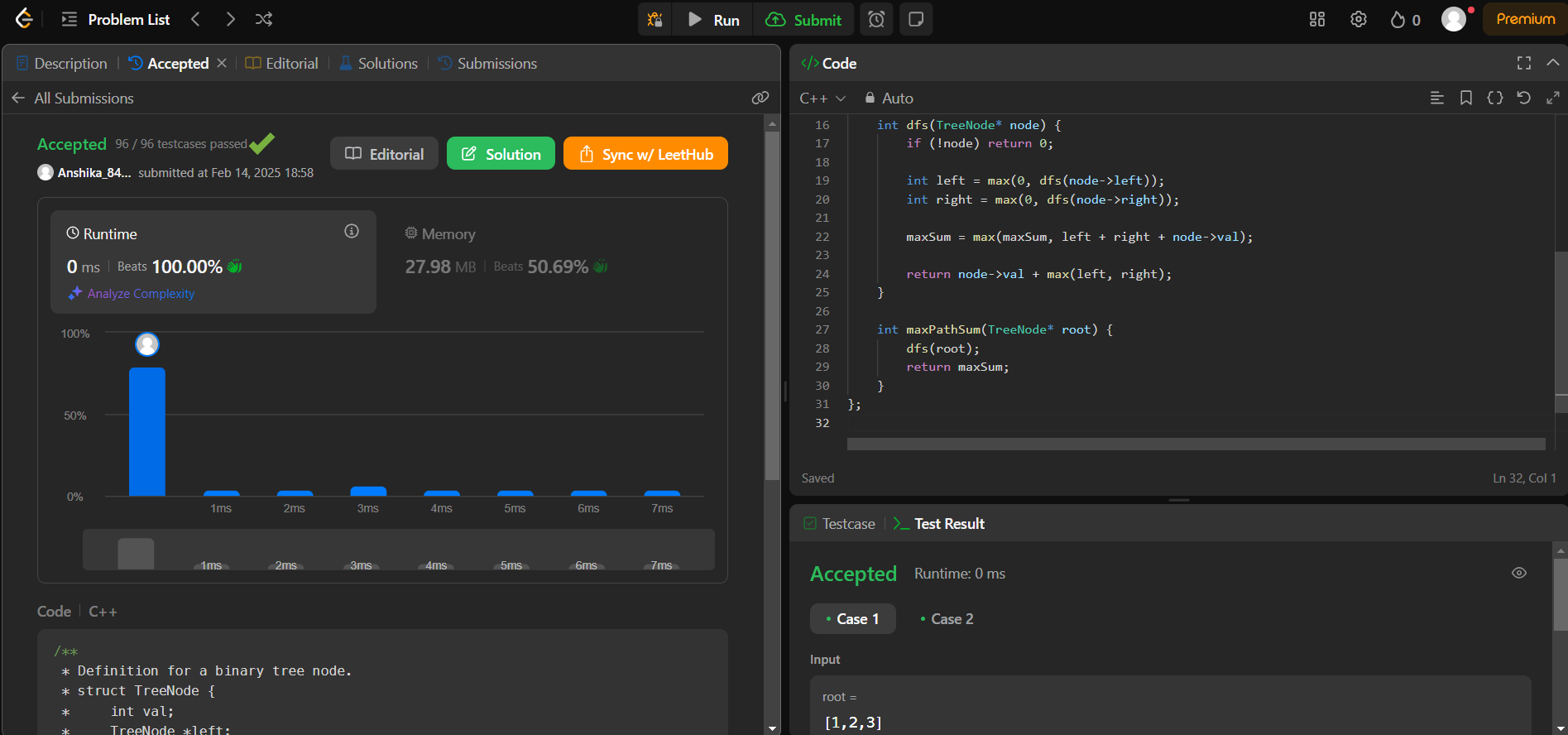
int maxPathSum(TreeNode\* root) {

dfs(root);

return maxSum;

}

};



1. **Vertical Order Traversal Of A Binary Tree**

class Solution {

public:

vector<vector<int>> verticalTraversal(TreeNode\* root) {

map<int, map<int, multiset<int>>> nodes;

queue<pair<TreeNode\*, pair<int, int>>> q;

q.push({root, {0, 0}});

while (!q.empty()) {

auto [node, pos] = q.front();

q.pop();

int x = pos.first, y = pos.second;

nodes[x][y].insert(node->val);

if (node->left) q.push({node->left, {x - 1, y + 1}});

if (node->right) q.push({node->right, {x + 1, y + 1}});

}

vector<vector<int>> result;

for (auto& [x, col] : nodes) {

vector<int> level;

for (auto& [y, values] : col)

level.insert(level.end(), values.begin(), values.end());

result.push\_back(level);

}

return result;

}

};

