**22BCS14693 Priyanshu Tak**

**94.Binary Tree Inorder Traversal**

**class Solution {**

**private:**

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

**if(root==NULL) return ;**

**inorder(root->left,res);**

**res.push\_back(root->val);**

**inorder(root->right,res);**

**}**

**public:**

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

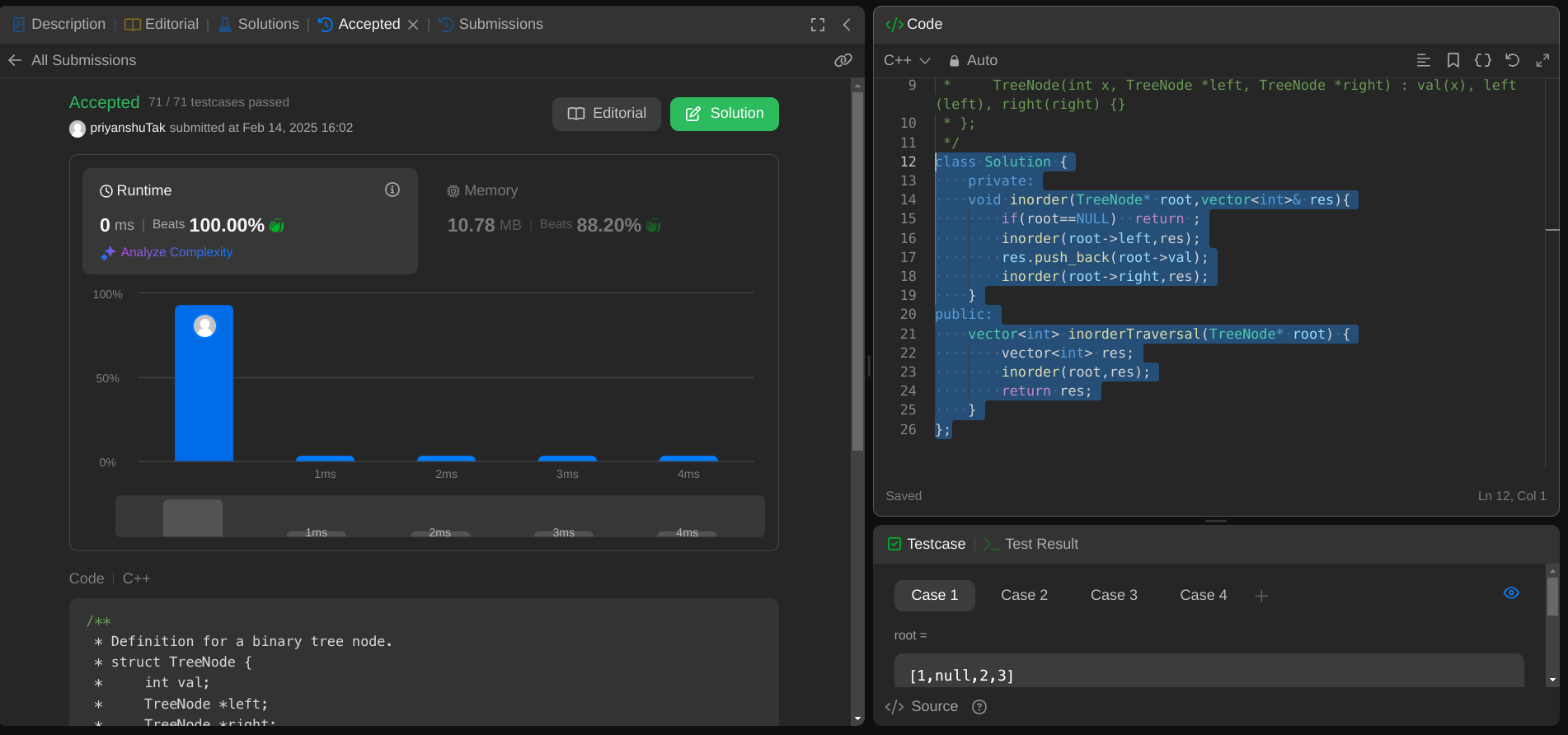
**vector<int> res;**

**inorder(root,res);**

**return res;**

**}**

**};**

****

**101.Symmetric Tree**

**class Solution {**

**public:**

**bool isSymmetric(TreeNode\* root) {**

**return isSame(root->left,root->right);**

**}**

**private:**

**bool isSame(TreeNode\*l,TreeNode\*r)**

**{**

**if(l==nullptr && r==nullptr)**

**return true;**

**if(l==nullptr || r==nullptr)**

**return false;**

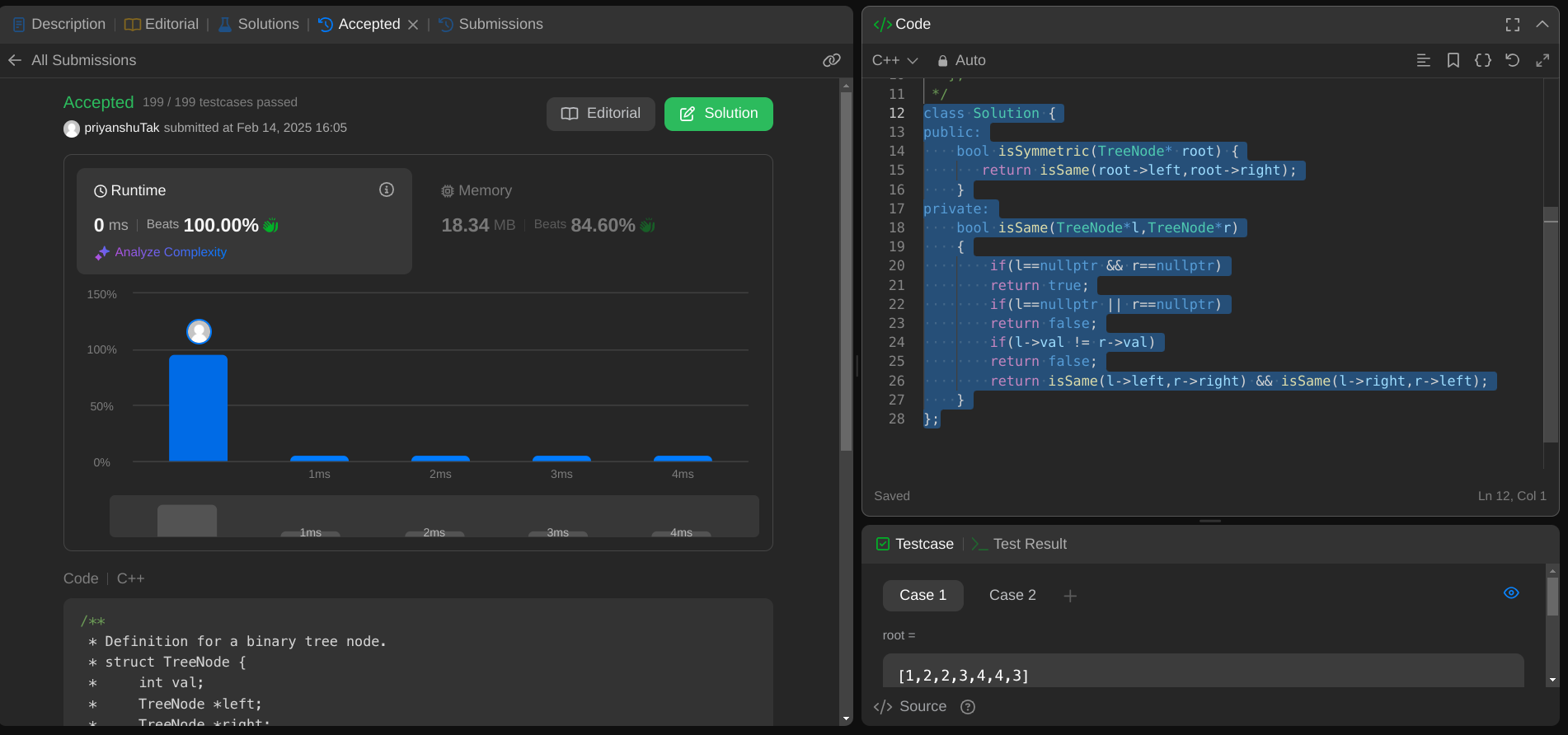
**if(l->val != r->val)**

**return false;**

**return isSame(l->left,r->right) && isSame(l->right,r->left);**

**}**

**};**

****

**104.Maximum Depth of Binary Tree**

**class Solution {**

**public:**

**int maxDepth(TreeNode\* root) {**

**if(root==NULL){**

**return 0;**

**}**

**int h1=maxDepth(root->left);**

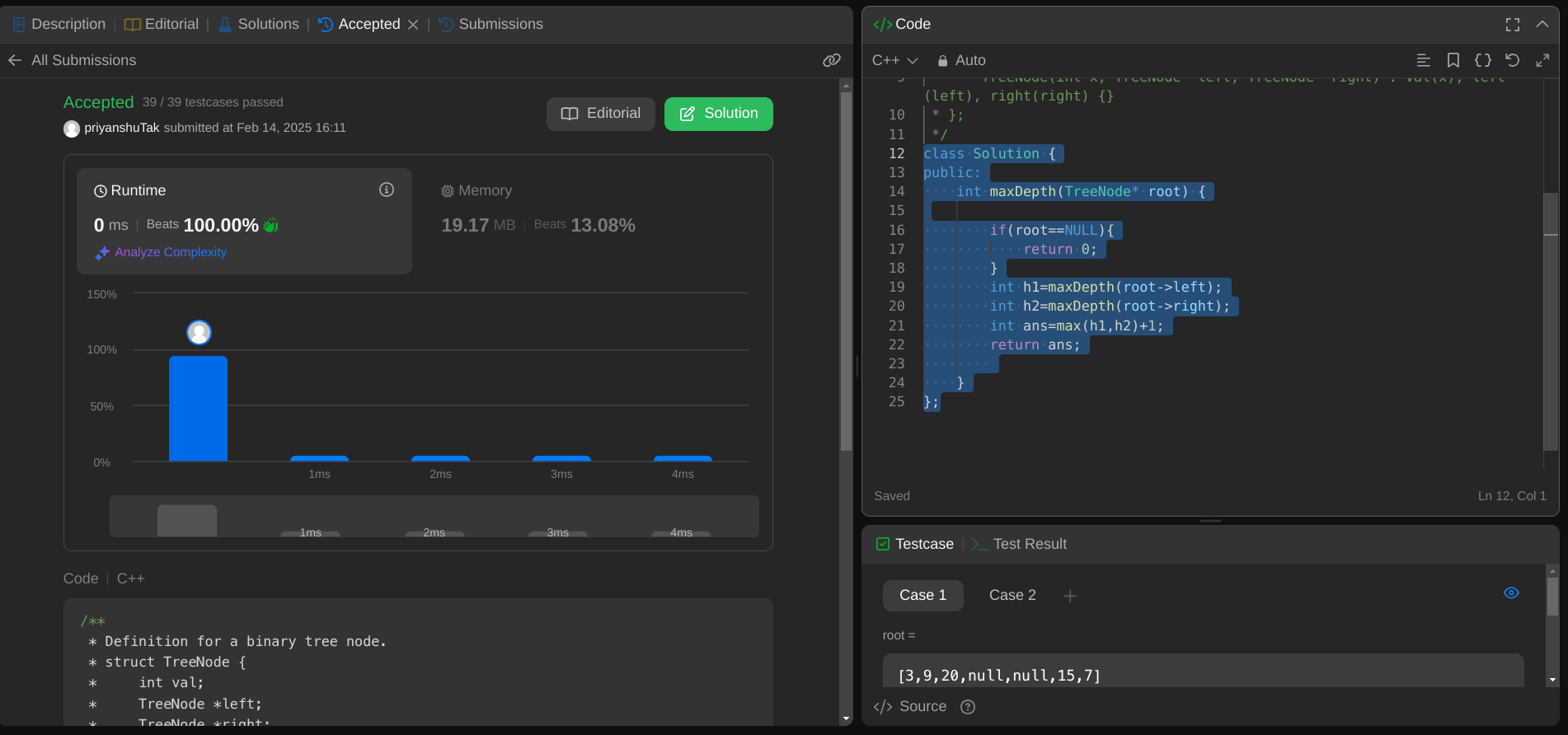
**int h2=maxDepth(root->right);**

**int ans=max(h1,h2)+1;**

**return ans;**

**}**

**};**

****

**98.Validate Binary Search Tree**

**class Solution {**

**public:**

**bool isValid(TreeNode\* root, long long maxVal, long long minVal)**

**{**

**if(root == NULL) return true;**

**if(root -> val >= maxVal || root -> val <= minVal) return false;**

**return isValid(root -> left, root->val, minVal) && isValid(root->right, maxVal, root -> val);**

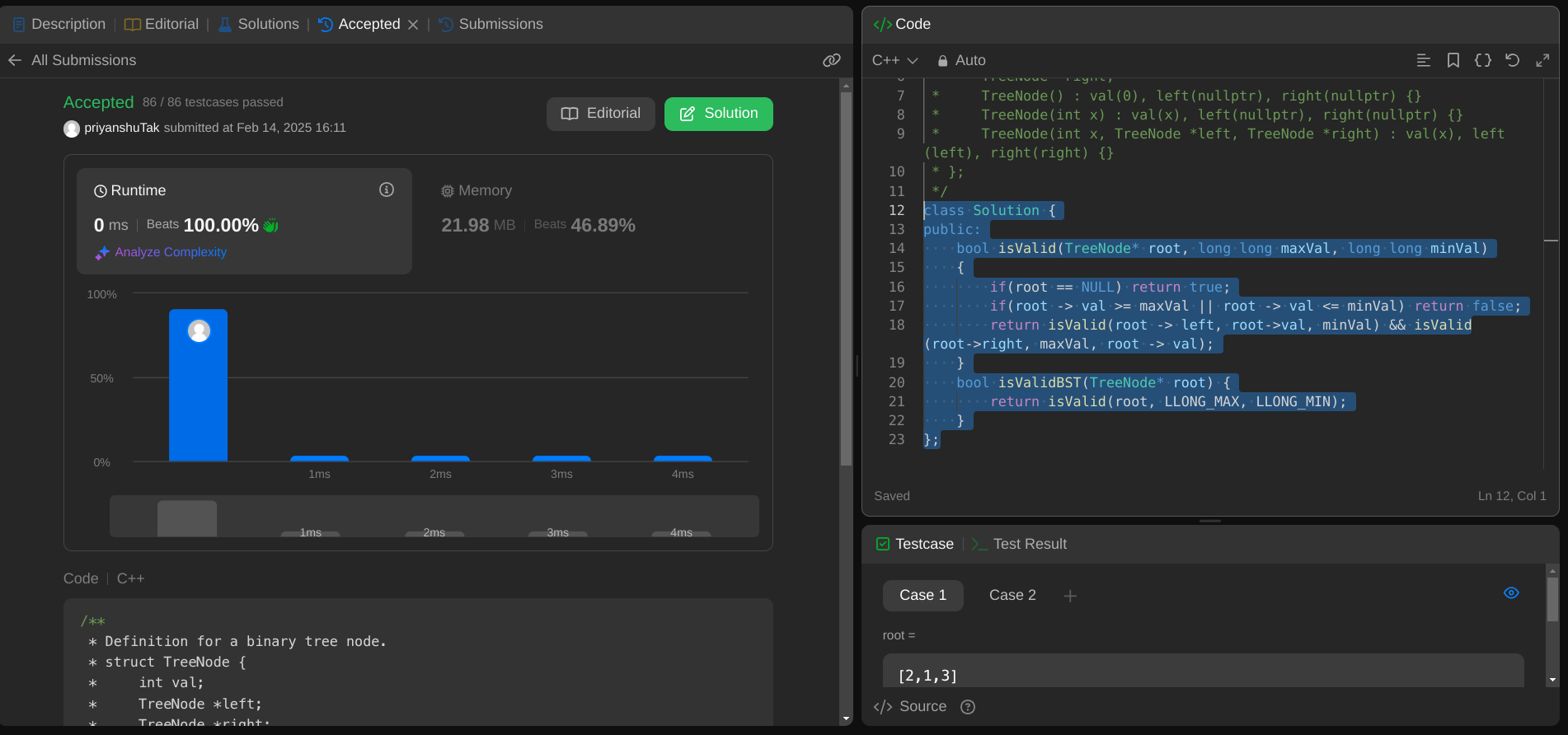
**}**

**bool isValidBST(TreeNode\* root) {**

**return isValid(root, LLONG\_MAX, LLONG\_MIN);**

**}**

**};**

****

**230.Kth Smallest Element in a BST**

**class Solution {**

**public:**

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

**if(!root) return ;**

**inorder(ans,root->left);**

**ans.push\_back(root->val);**

**inorder(ans,root->right);**

**}**

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

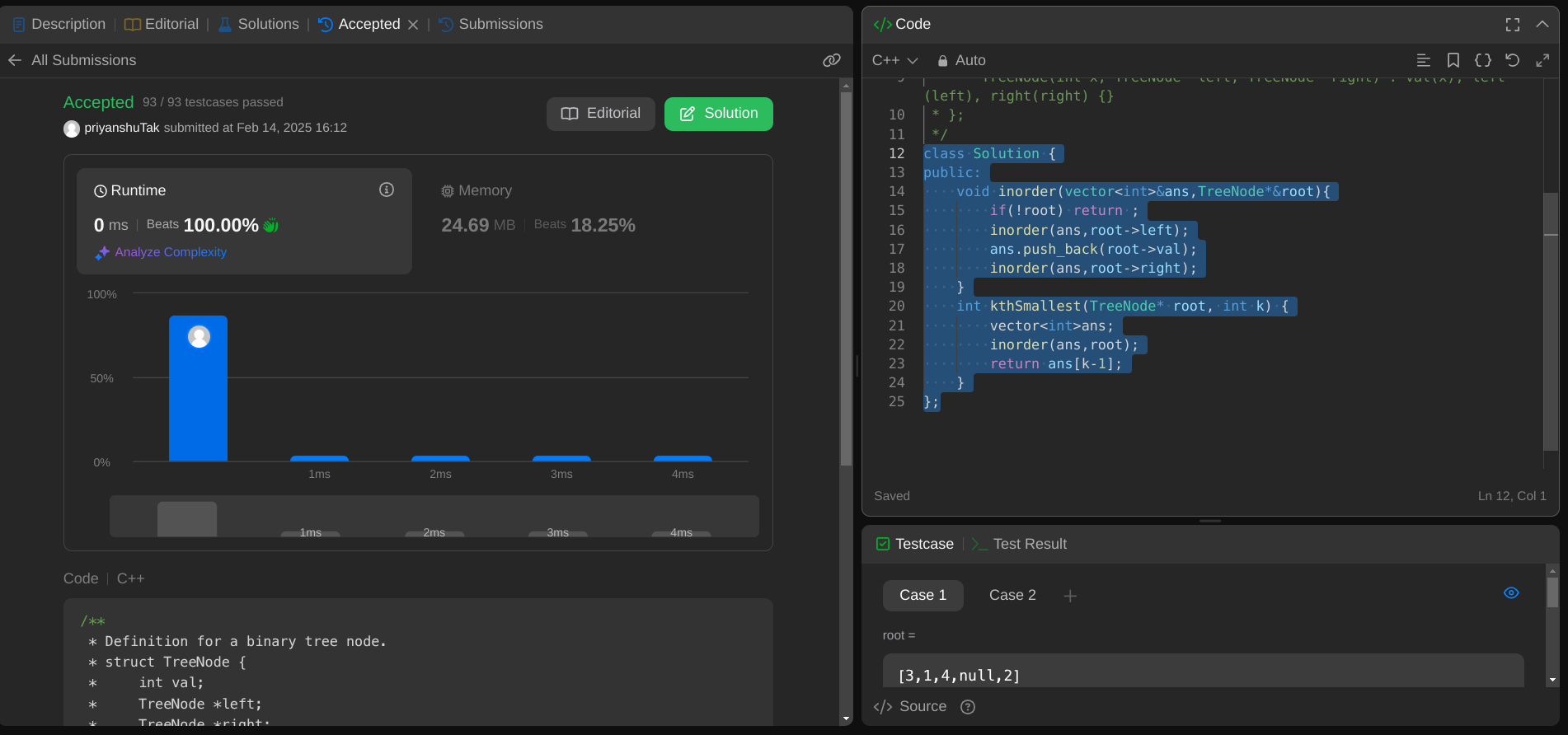
**vector<int>ans;**

**inorder(ans,root);**

**return ans[k-1];**

**}**

**};**

****

**102. Binary Tree Level Order Traversal**

**class Solution {**

**public:**

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

**if (root == nullptr)**

**return {};**

**vector<vector<int>> ans;**

**queue<TreeNode\*> q{{root}};**

**while (!q.empty()) {**

**vector<int> currLevel;**

**for (int sz = q.size(); sz > 0; --sz) {**

**TreeNode\* node = q.front();**

**q.pop();**

**currLevel.push\_back(node->val);**

**if (node->left)**

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

**if (node->right)**

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

**}**

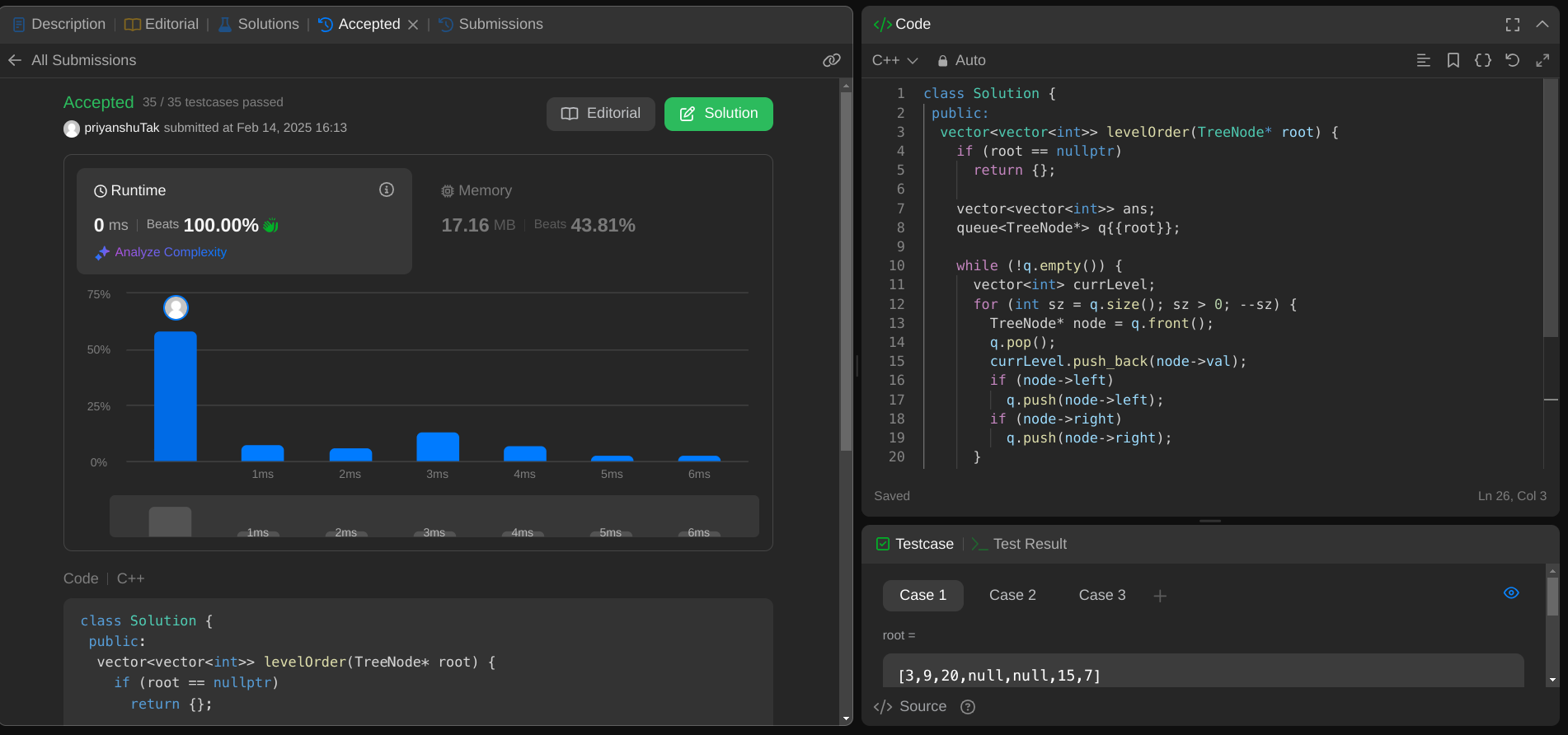
**ans.push\_back(currLevel);**

**}**

**return ans;**

**}**

**};**

****

**107.Binary Tree Level Order Traversal II**

**class Solution {**

**public:**

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

**vector<vector<int>> result;**

**int height = findHeight(root);**

**for (int i = 0; i < height; ++i) {**

**result.push\_back(vector<int>());**

**}**

**fillLevels(root, result, height - 1);**

**return result;**

**}**

**private:**

**int findHeight(TreeNode\* node) {**

**if (node == nullptr) {**

**return 0;**

**}**

**return max(findHeight(node->left), findHeight(node->right)) + 1;**

**}**

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

**if (node == nullptr) {**

**return;**

**}**

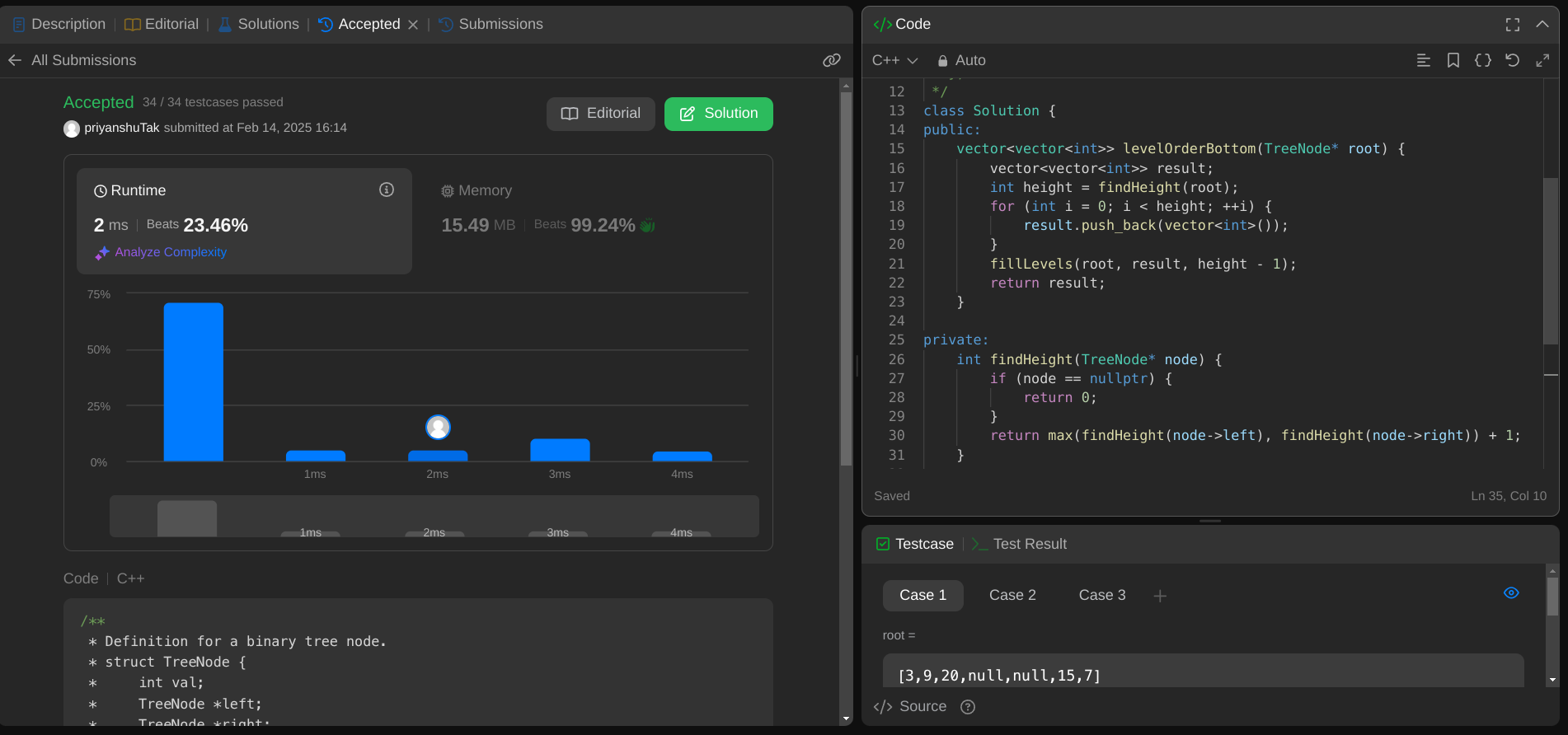
**result[level].push\_back(node->val);**

**fillLevels(node->left, result, level - 1);**

**fillLevels(node->right, result, level - 1);**

**}**

**};**

****

**103.Binary Tree Zigzag Level Order Traversal**

**class Solution {**

**public:**

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

**if (root == nullptr)**

**return {};**

**vector<vector<int>> ans;**

**deque<TreeNode\*> dq{{root}};**

**bool isLeftToRight = true;**

**while (!dq.empty()) {**

**vector<int> currLevel;**

**for (int sz = dq.size(); sz > 0; --sz)**

**if (isLeftToRight) {**

**TreeNode\* node = dq.front();**

**dq.pop\_front();**

**currLevel.push\_back(node->val);**

**if (node->left)**

**dq.push\_back(node->left);**

**if (node->right)**

**dq.push\_back(node->right);**

**} else {**

**TreeNode\* node = dq.back();**

**dq.pop\_back();**

**currLevel.push\_back(node->val);**

**if (node->right)**

**dq.push\_front(node->right);**

**if (node->left)**

**dq.push\_front(node->left);**

**}**

**ans.push\_back(currLevel);**

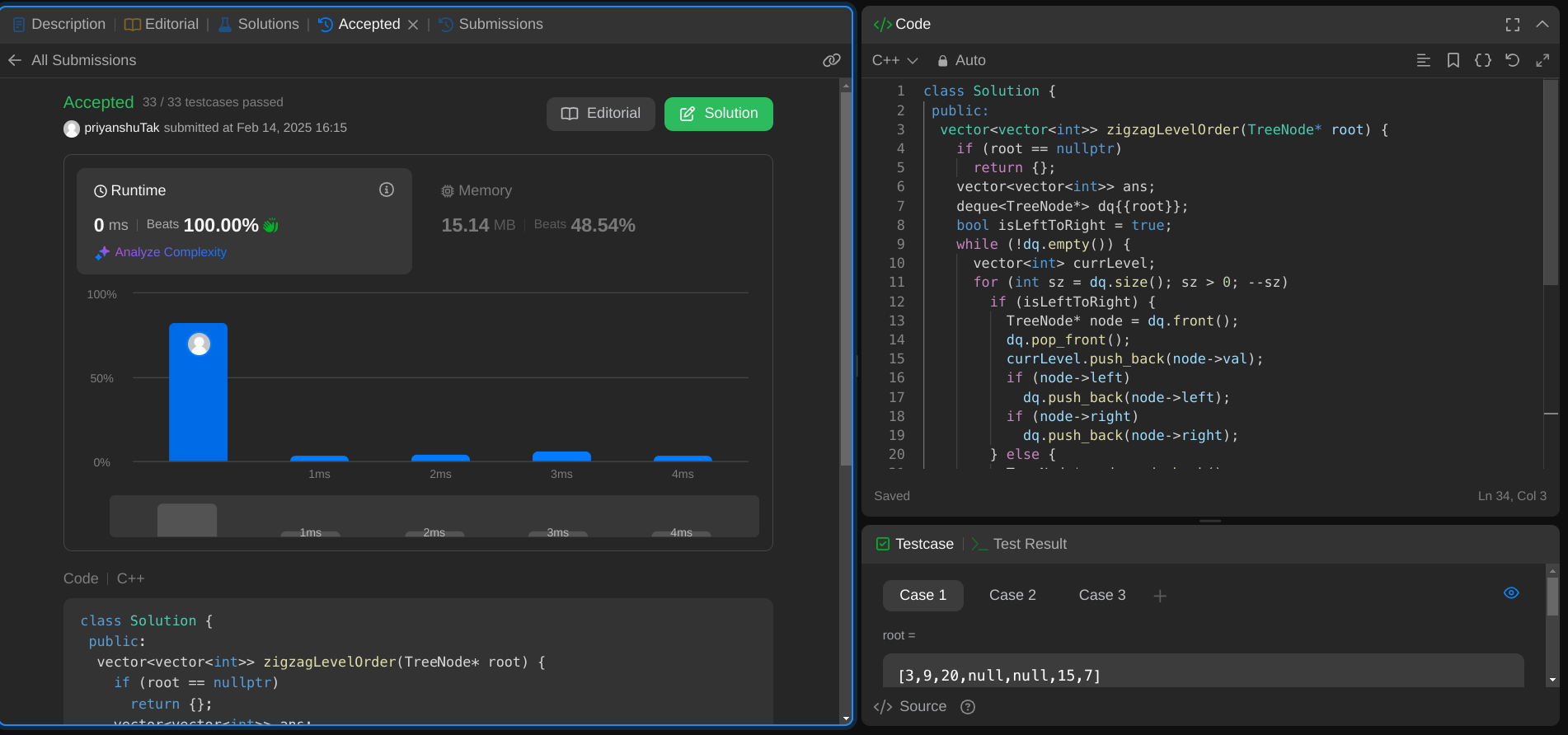
**isLeftToRight = !isLeftToRight;**

**}**

**return ans;**

**}**

**};**

****

**199.Binary Tree Right Side View**

**class Solution {**

**public:**

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

**if(root==NULL){**

**return;**

**}**

**if(result.size() == level){**

**result.push\_back(root->val);**

**}**

**traverse(root->right, level+1, result);**

**traverse(root->left, level+1, result);**

**}**

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

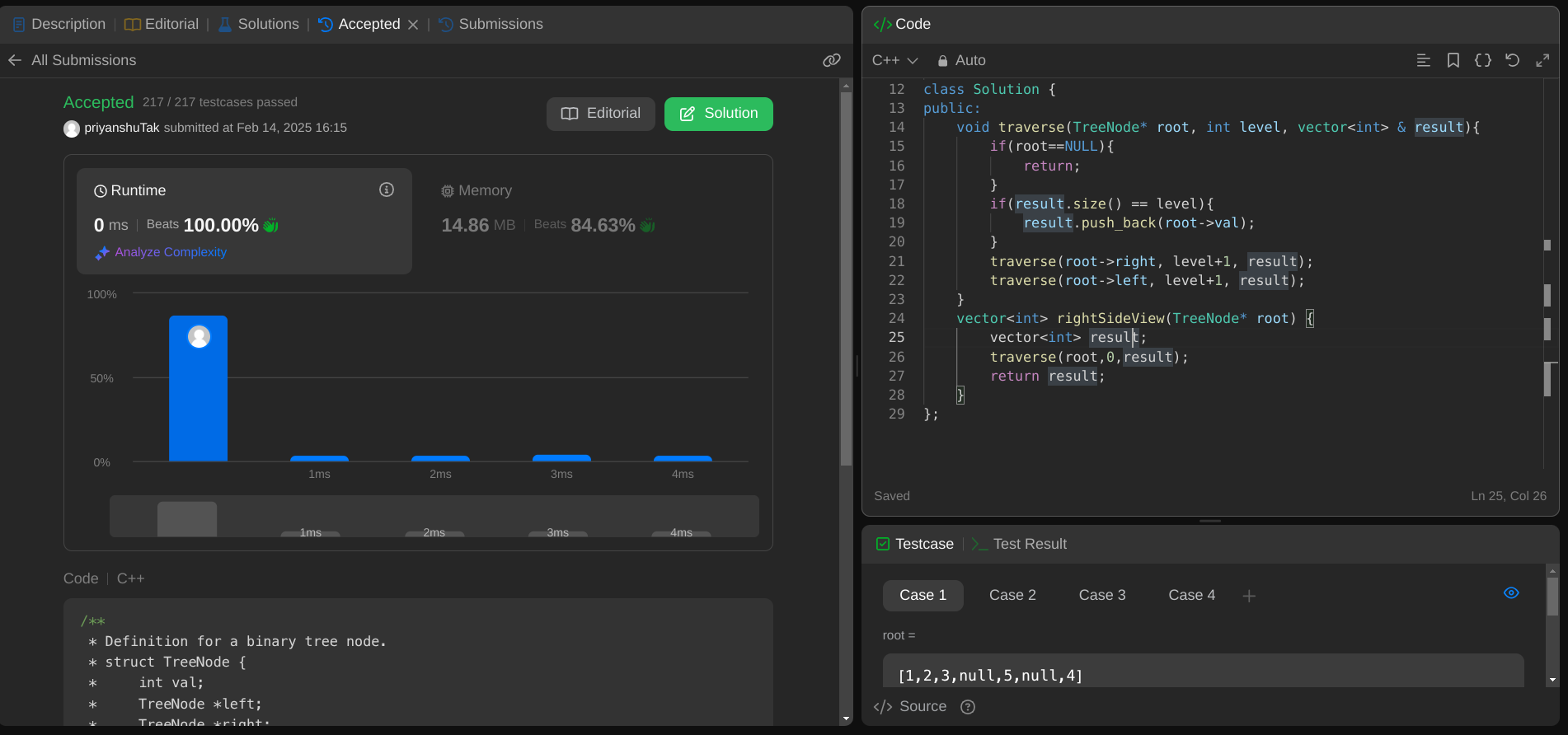
**vector<int> result;**

**traverse(root,0,result);**

**return result;**

**}**

**};**

****

**106.Construct Binary Tree from Inorder and Postorder Traversal**

**class Solution {**

**public:**

**TreeNode\* build(vector<int>& in,int inlo,int inhi,vector<int>& post,int postlo,int posthi){**

**if(postlo>posthi) return NULL;**

**TreeNode \* root = new TreeNode(post[posthi]);**

**if(postlo==posthi) return root;**

**int i = inlo;**

**while(i<=inhi){**

**if(in[i]==post[posthi]) break;**

**i++;**

**}**

**int leftcount = i - inlo;**

**root->left = build(in , inlo ,i-1, post , postlo , postlo+leftcount-1 );**

**root->right = build(in ,i+1 ,inhi, post ,postlo+leftcount,posthi-1 );**

**return root;**

**}**

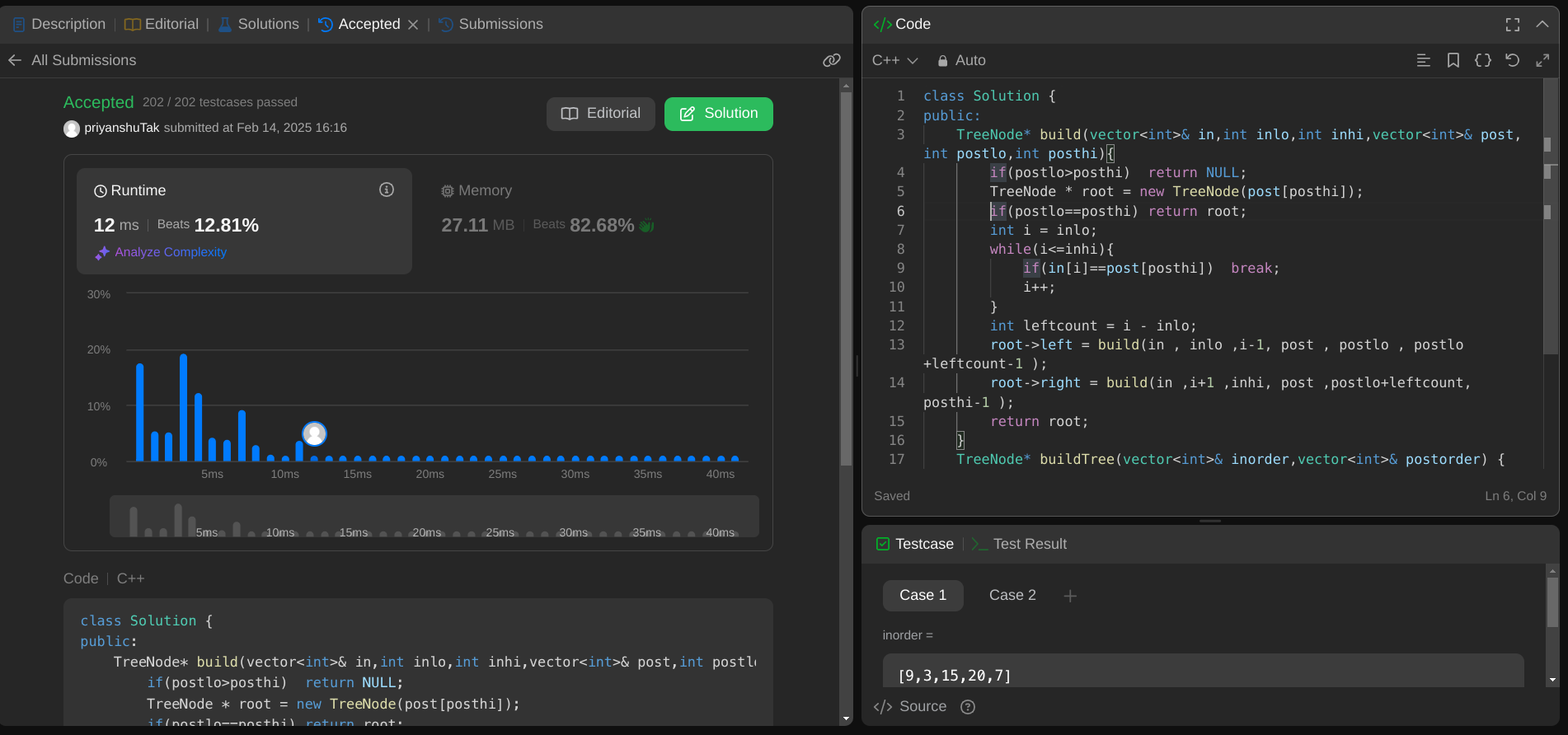
**TreeNode\* buildTree(vector<int>& inorder,vector<int>& postorder) {**

**int n = postorder.size();**

**return build(inorder,0,n-1,postorder,0,n-1);**

**}**

**};**

****

**513.Find Bottom Left Tree Value**

**class Solution {**

**public:**

**int findBottomLeftValue(TreeNode\* root) {**

**queue<TreeNode\*> q;**

**q.push(root);**

**int ans;**

**while(!q.empty()){**

**TreeNode\* node = q.front();**

**q.pop();**

**ans = node->val;**

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

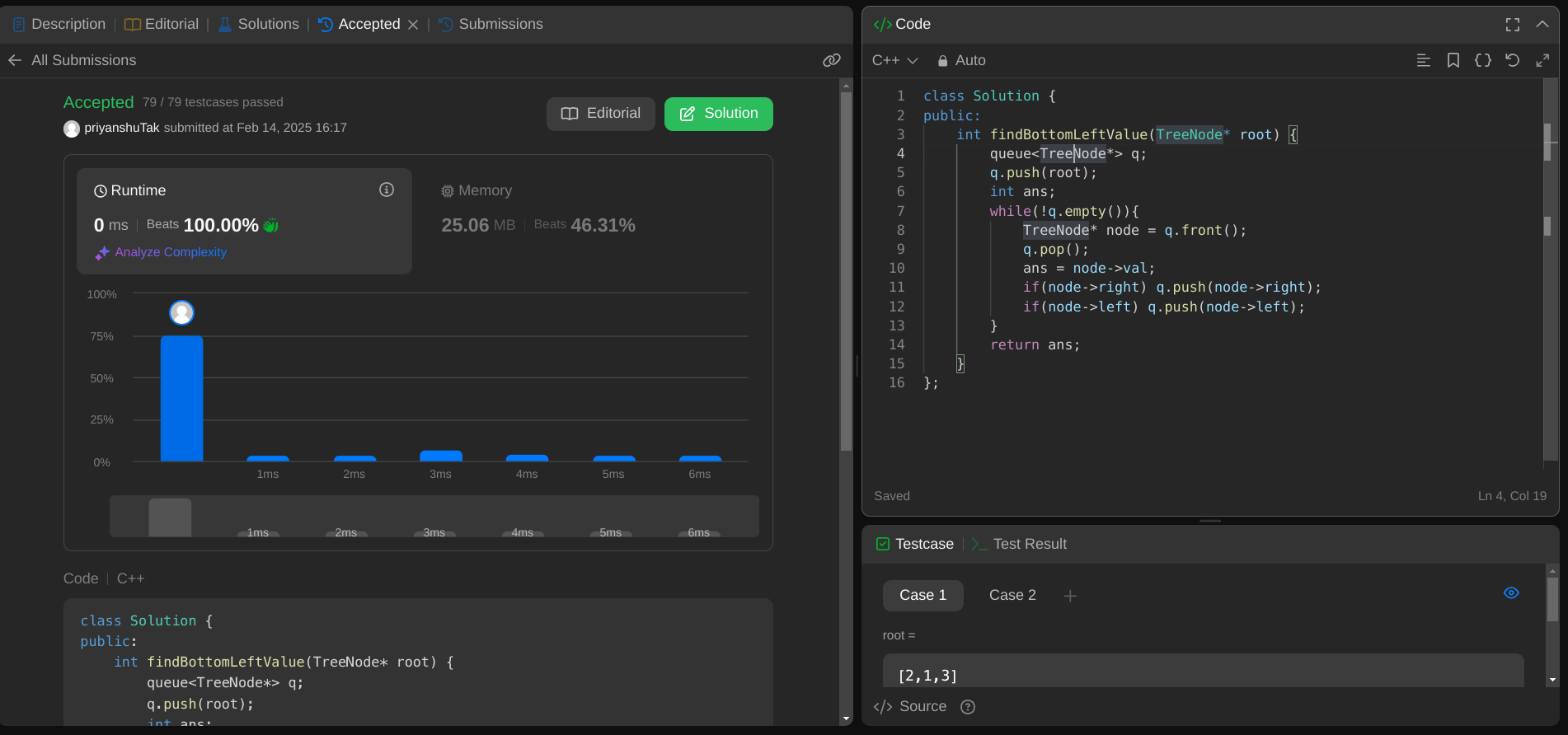
**if(node->left) q.push(node->left);**

**}**

**return ans;**

**}**

**};**

****

**124. Binary Tree Maximum Path Sum**

**class Solution {**

**public:**

**int maxPathSum(TreeNode\* root) {**

**int maxi=INT\_MIN;**

**dia(root,maxi);**

**return maxi;**

**}**

**int dia(TreeNode\* root , int & maxi){**

**if(root==NULL)return 0;**

**int leftt=max(0,dia(root->left,maxi));**

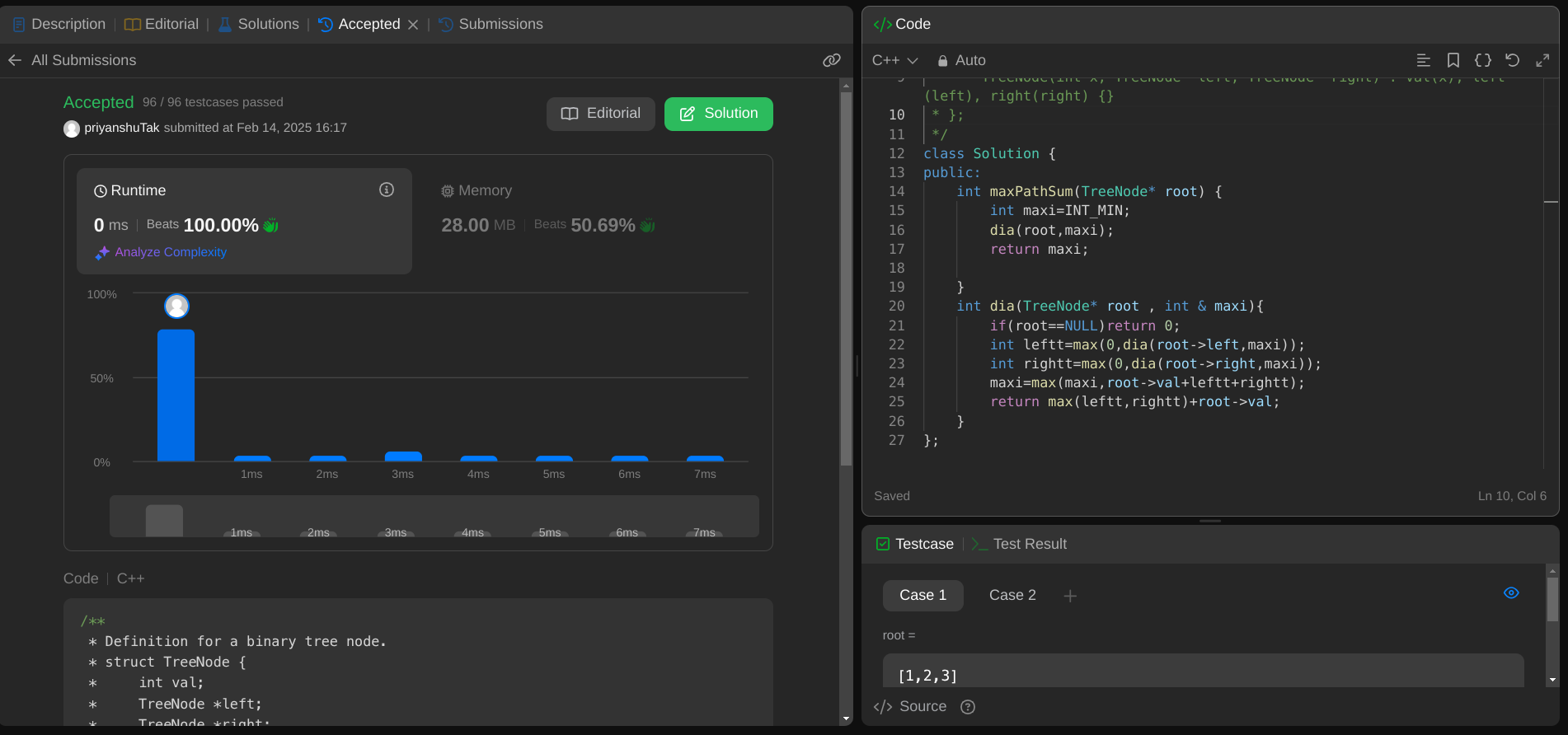
**int rightt=max(0,dia(root->right,maxi));**

**maxi=max(maxi,root->val+leftt+rightt);**

**return max(leftt,rightt)+root->val;**

**}**

**};**

****

**987.Vertical Order Traversal of a Binary Tree**

class Solution {

public:

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

vector<vector<int>> ans;

if(!root) return ans;

map<int, map<int, multiset<int>>> m1;

queue<pair<TreeNode\*, pair<int,int >>> q1;

q1.push({root,{0,0}});

while(!q1.empty())

{

TreeNode\* temp = q1.front().first;

int hd = q1.front().second.first;

int lvl = q1.front().second.second;

q1.pop();

m1[hd][lvl].insert(temp->val);

if(temp->left)

{

q1.push({temp->left,{hd-1,lvl+1}});

}

if(temp->right)

{

q1.push({temp->right,{hd+1,lvl+1}});

}

}

for(auto it: m1)

{

vector<int> col;

for(auto itt: it.second)

{

col.insert(col.end(), itt.second.begin(), itt.second.end());

}

ans.push\_back(col);

}

return ans;

}

};

