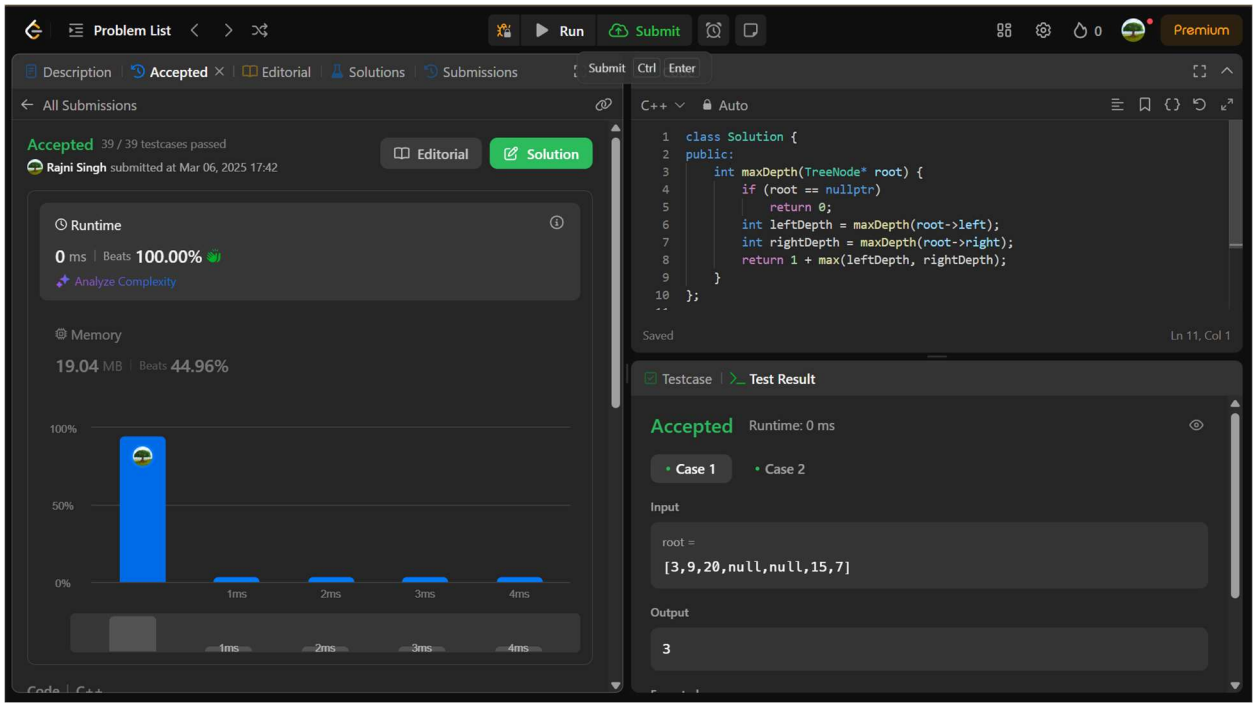
Experiment 06

Student Name: Rajni Gandha
UID: 22BET10080
Branch: BE-IT
Section/Group: BET_701/A
Semester: 06th
Date of Performance: 05-03-2025
Subject Name: Advanced Programming
Subject Code: 22ITP-351

1. **Problem: Maximum Depth of Binary Tree**

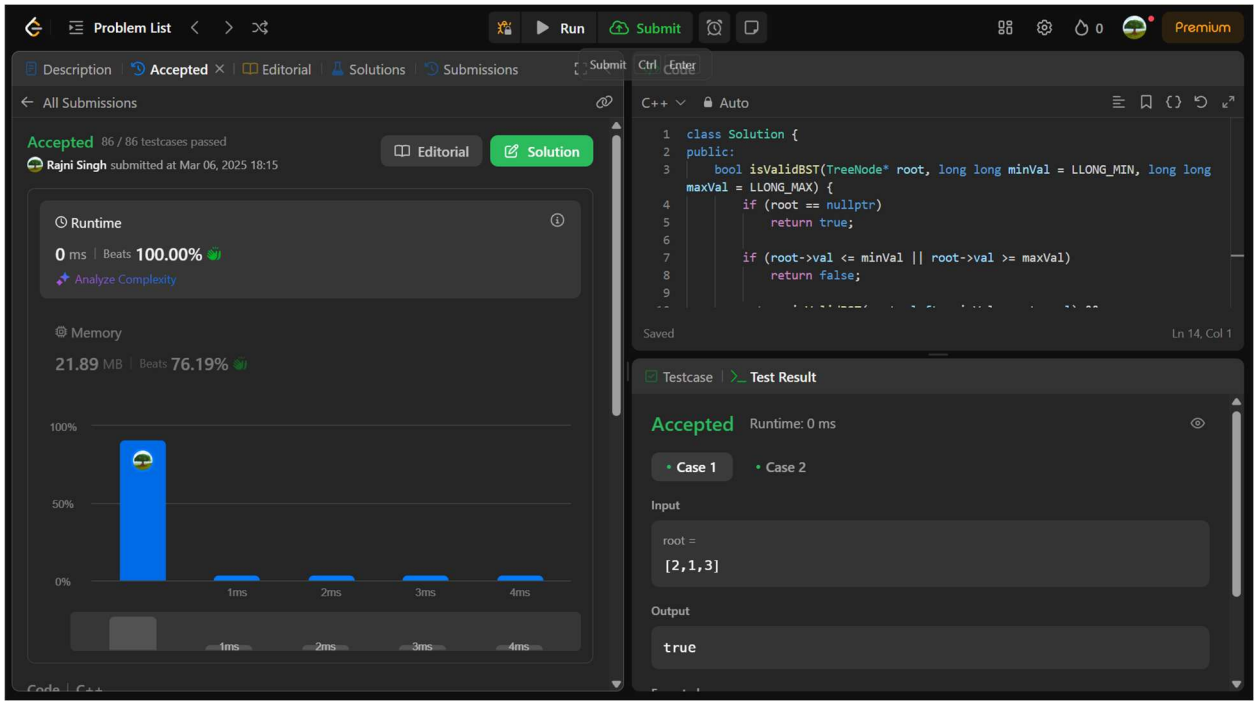
```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (root == nullptr)
            return 0;
        int leftDepth = maxDepth(root->left);
        int rightDepth = maxDepth(root->right);
        return 1 + max(leftDepth, rightDepth);
    }
};
```



2. **Problem: Validate Binary Search Tree**

```
class Solution {
public:
```

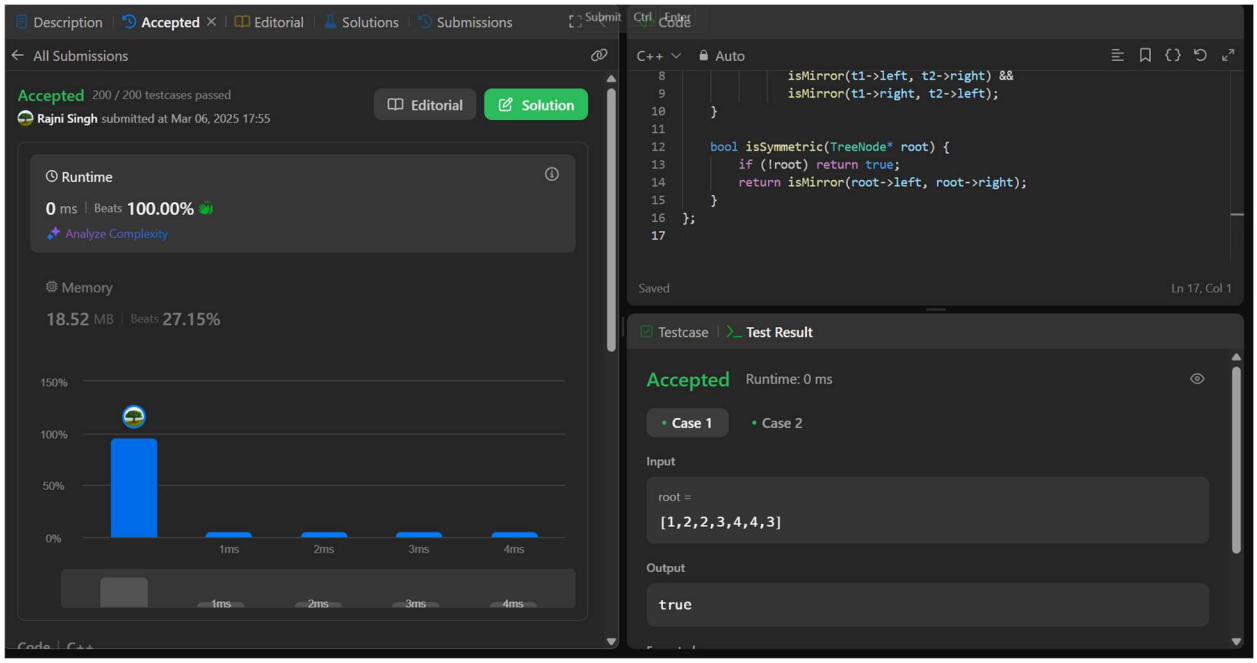
```
    bool isValidBST(TreeNode* root, long long minVal = LLONG_MIN, long long
maxVal = LLONG_MAX) {
        if (root == nullptr)
            return true;
        if (root->val <= minVal || root->val >= maxVal)
            return false;
        return isValidBST(root->left, minVal, root->val) &&
               isValidBST(root->right, root->val, maxVal);
    }
};
```



3. **Problem: 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);
```

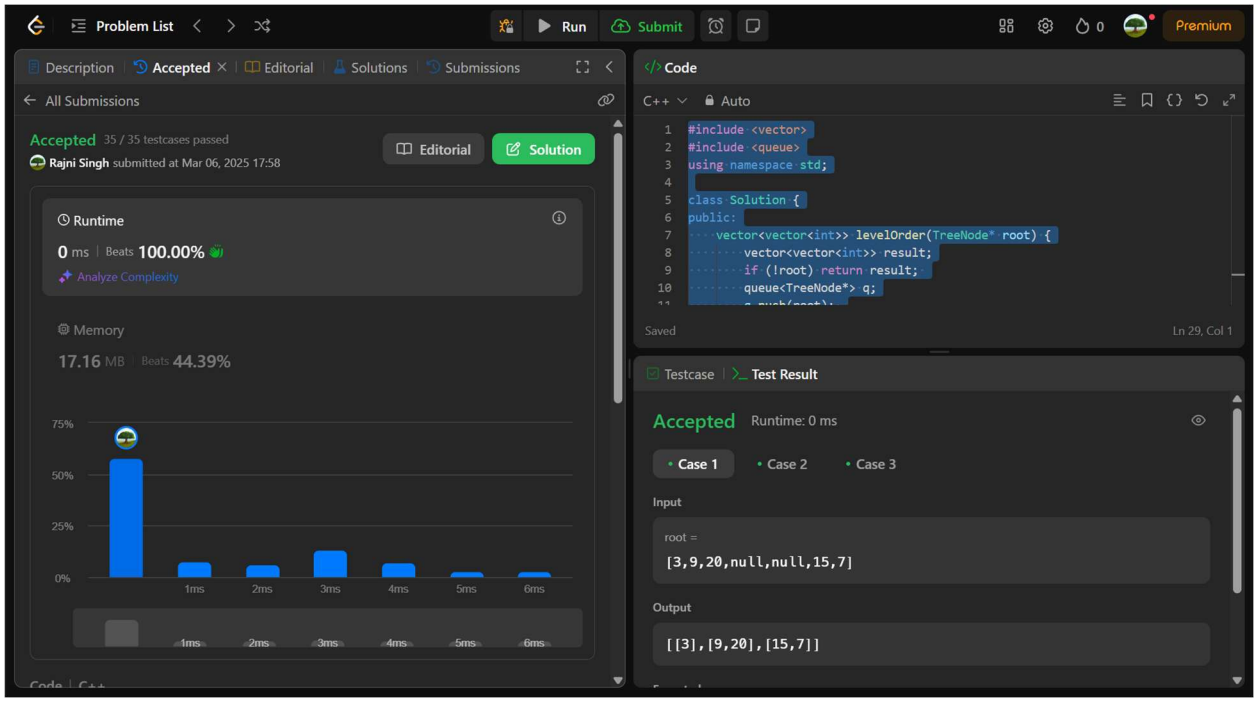
```
    }
    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        return isMirror(root->left, root->right);
    }
};
```



4. **Problem: Binary Tree Level Order Traversal**

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

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 levelSize = q.size();
            vector<int> level;
```

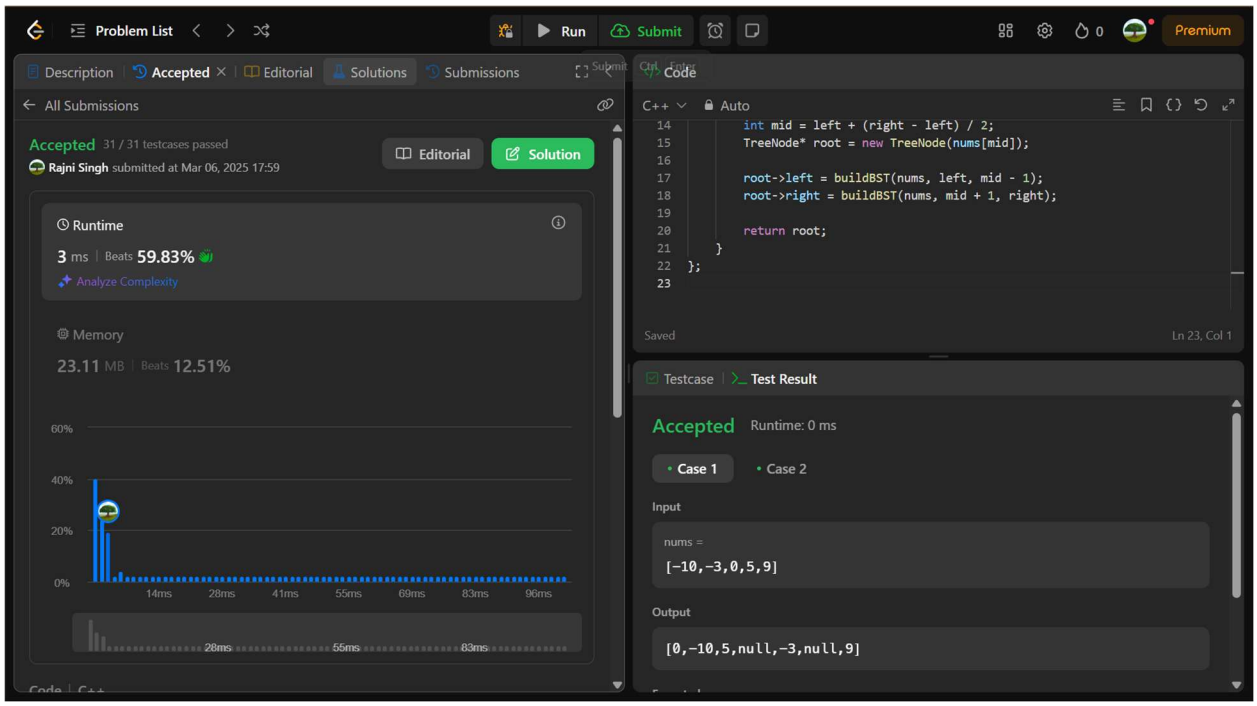
```
            for (int i = 0; i < levelSize; i++) {
                TreeNode* node = q.front();
                q.pop();
                level.push_back(node->val);

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(level);
        }
        return result;
    }
};
```



5. **Problem: Convert Sorted Array to Binary Search Tree**

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return buildBST(nums, 0, nums.size() - 1);
    }
```

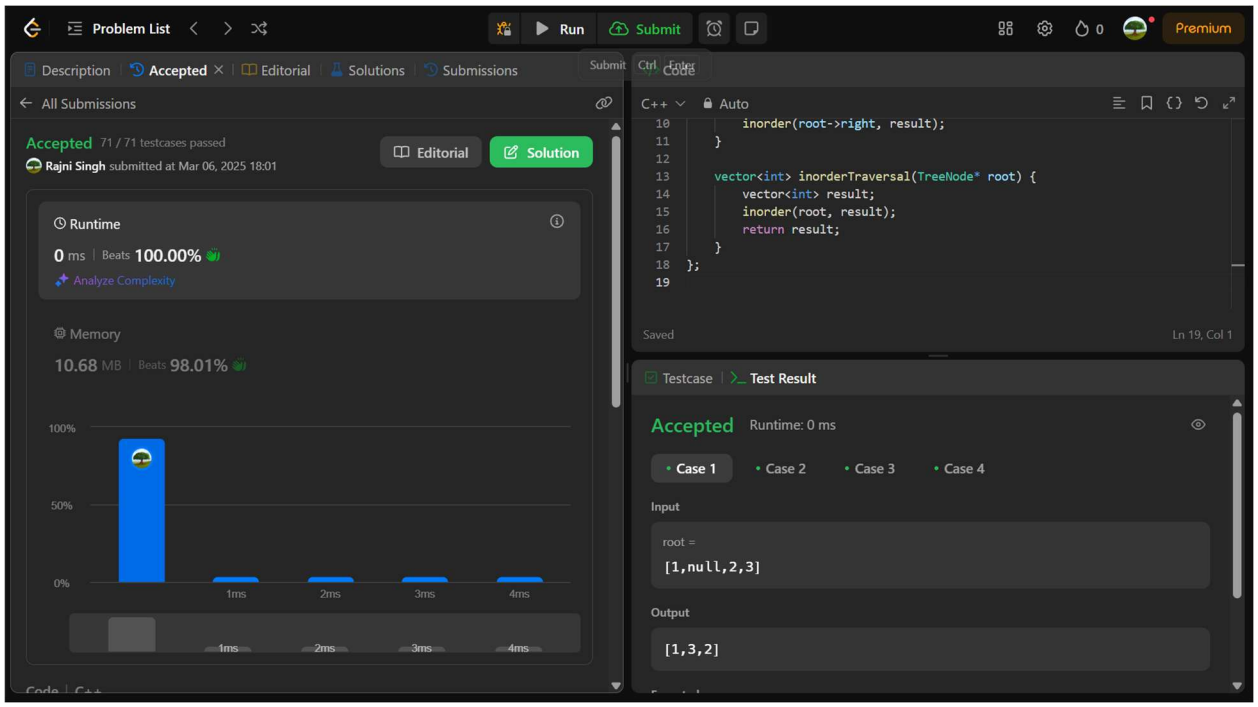
```
private:
    TreeNode* buildBST(vector<int>& nums, int left, int right) {
        if (left > right) return nullptr;  // Base case

        int mid = left + (right - left) / 2;
        TreeNode* root = new TreeNode(nums[mid]);

        root->left = buildBST(nums, left, mid - 1);
        root->right = buildBST(nums, mid + 1, right);
        return root;
    }
};
```



6. **Problem: Binary Tree Inorder Traversal**

```
class Solution {
public:
    void inorder(TreeNode* root, vector<int>& result) {
        if (!root) return;
        inorder(root->left, result);
        result.push_back(root->val);
        inorder(root->right, result);
```

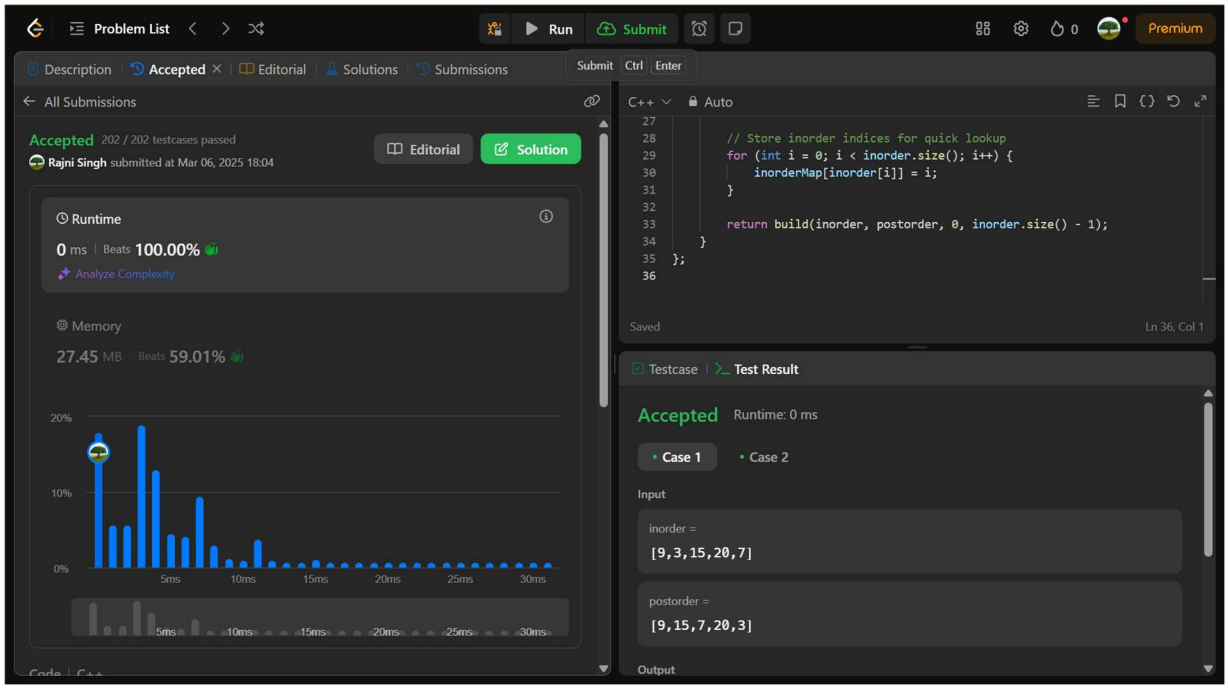
```
    }

    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorder(root, result);
        return result;
    }
};
```



7. **Problem: Construct Binary Tree from Inorder and Postorder Traversal**

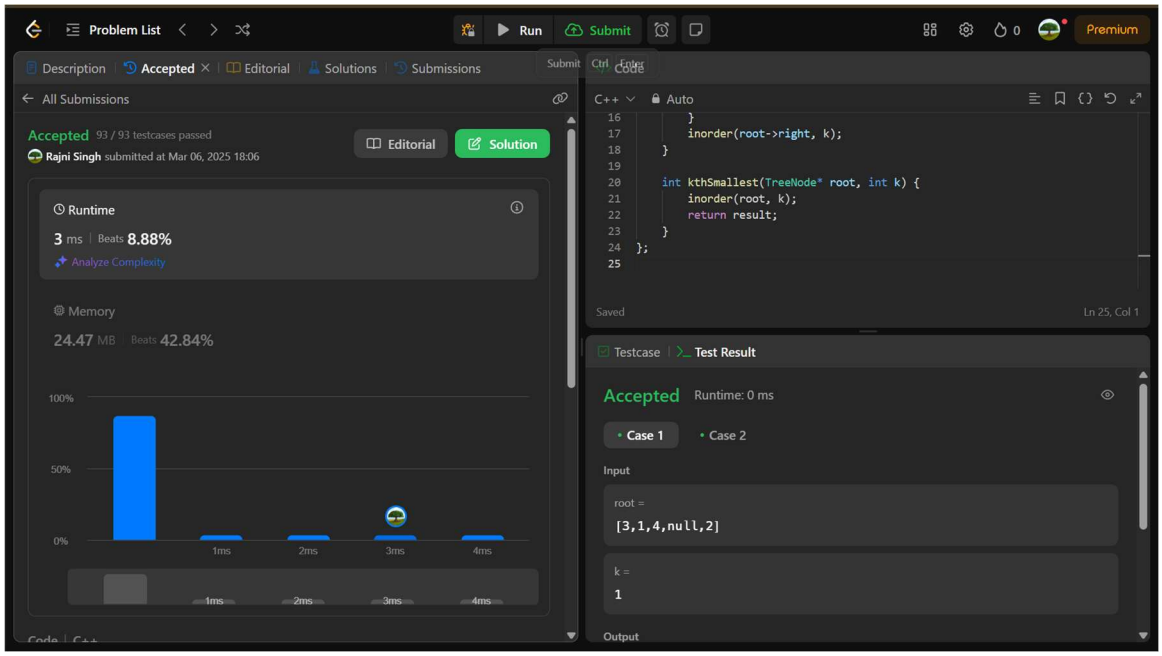
```
#include <vector>
#include <unordered_map>
using namespace std;

class Solution {
public:
    unordered_map<int, int> inorderMap;
    int postIndex;

    TreeNode* build(vector<int>& inorder, vector<int>& postorder, int left, int right) {
        if (left > right) return nullptr; // Base case
```

```
        int rootVal = postorder[postIndex--];
        TreeNode* root = new TreeNode(rootVal);

        int inIndex = inorderMap[rootVal];

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

        return root;
    }

    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        postIndex = postorder.size() - 1; // Start from last element in postorder

        for (int i = 0; i < inorder.size(); i++) {
            inorderMap[inorder[i]] = i;
        }

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



8. Problem: Kth Smallest element in a BST

```
class Solution {
public:
    int count = 0, result = -1;

    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;
    }
};
```



9. Problem: Populating Next Right Pointers in Each Node

```
class Solution {
public:
    Node* connect(Node* root) {
        if (!root) return nullptr;
        queue<Node*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                Node* node = q.front();
                q.pop();
                if (i < size - 1) {
                    node->next = q.front();
                }
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return root;
    }
};
```