**ADVANCED PROGRAMMING-II**

**ASSIGNMENT-05**

**Q1.** [**Maximum Depth of Binary Tree**](https://leetcode.com/problems/maximum-depth-of-binary-tree/description/)**:**

**CODE:**

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if (!root) return 0;

        return 1 + max(maxDepth(root->left), maxDepth(root->right));

}};

TreeNode\* createSampleTree() {

    TreeNode\* root = new TreeNode(3);

    root->left = new TreeNode(9);

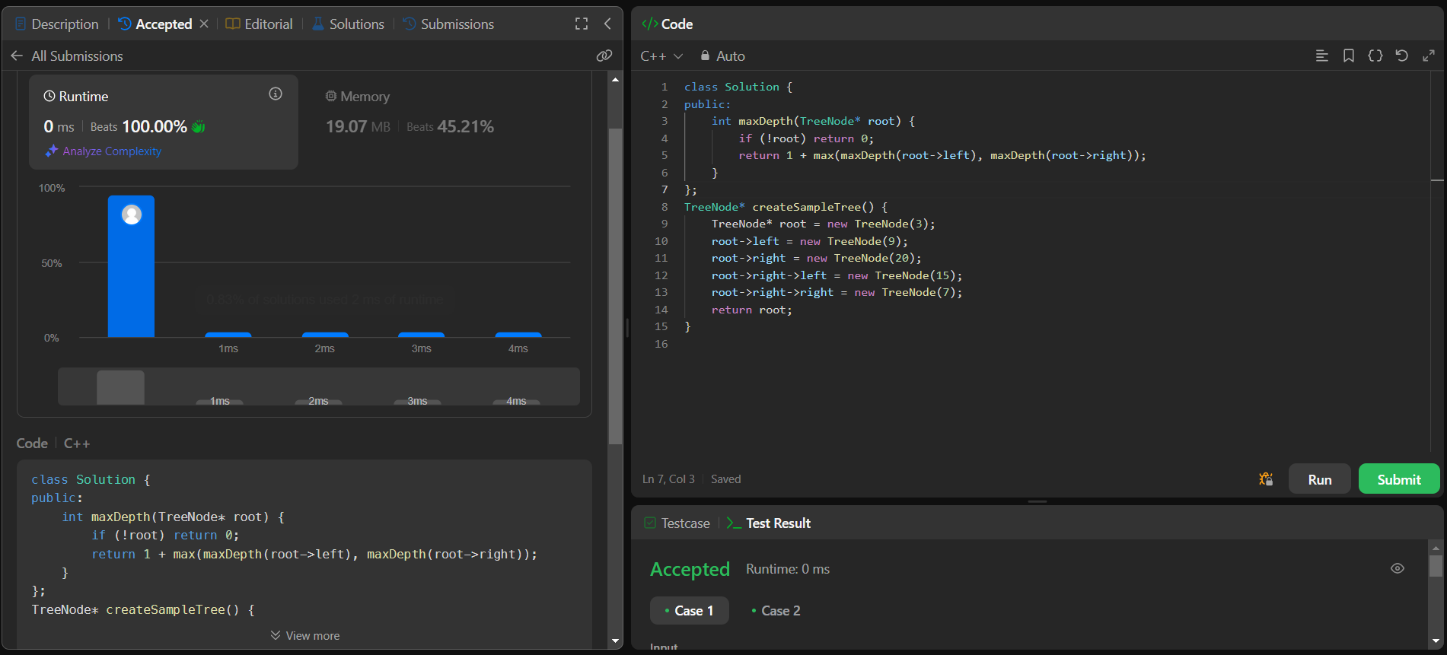
    root->right = new TreeNode(20);

    root->right->left = new TreeNode(15);

    root->right->right = new TreeNode(7);

    return root;}

**Screenshot:**



**Q2.** [**Validate Binary Search Tree**](https://leetcode.com/problems/validate-binary-search-tree/description/)**:**

**Code:**

class Solution {

public:

    bool isValidBST(TreeNode\* root) {

        return helper(root, LLONG\_MIN, LLONG\_MAX);

    }

    bool helper(TreeNode\* node, long long lower, long long upper) {

        if (!node) return true;

        long long currentVal = node->val;

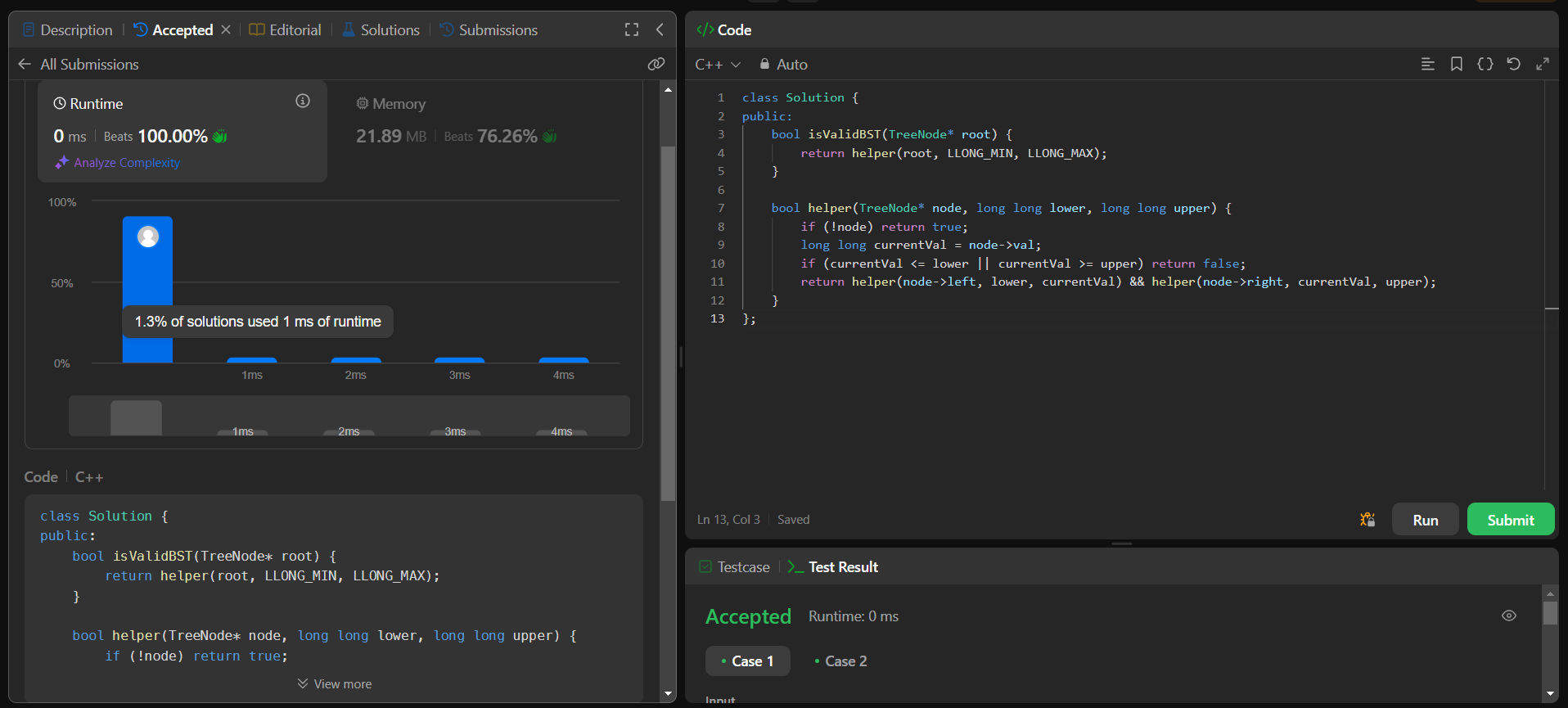
        if (currentVal <= lower || currentVal >= upper) return false;

        return helper(node->left, lower, currentVal) && helper(node->right, currentVal, upper);

    }

};

**Screenshot:**

****

**Q3.** [**Symmetric Tree**](https://leetcode.com/problems/symmetric-tree/description/)**:**

**Code:**

class Solution {

public:

    bool isSymmetric(TreeNode\* root) {

        if (!root) return true;

        return isMirror(root->left, root->right);

    }

    bool isMirror(TreeNode\* leftNode, TreeNode\* rightNode) {

        if (!leftNode && !rightNode) return true;

        if (!leftNode || !rightNode) return false;

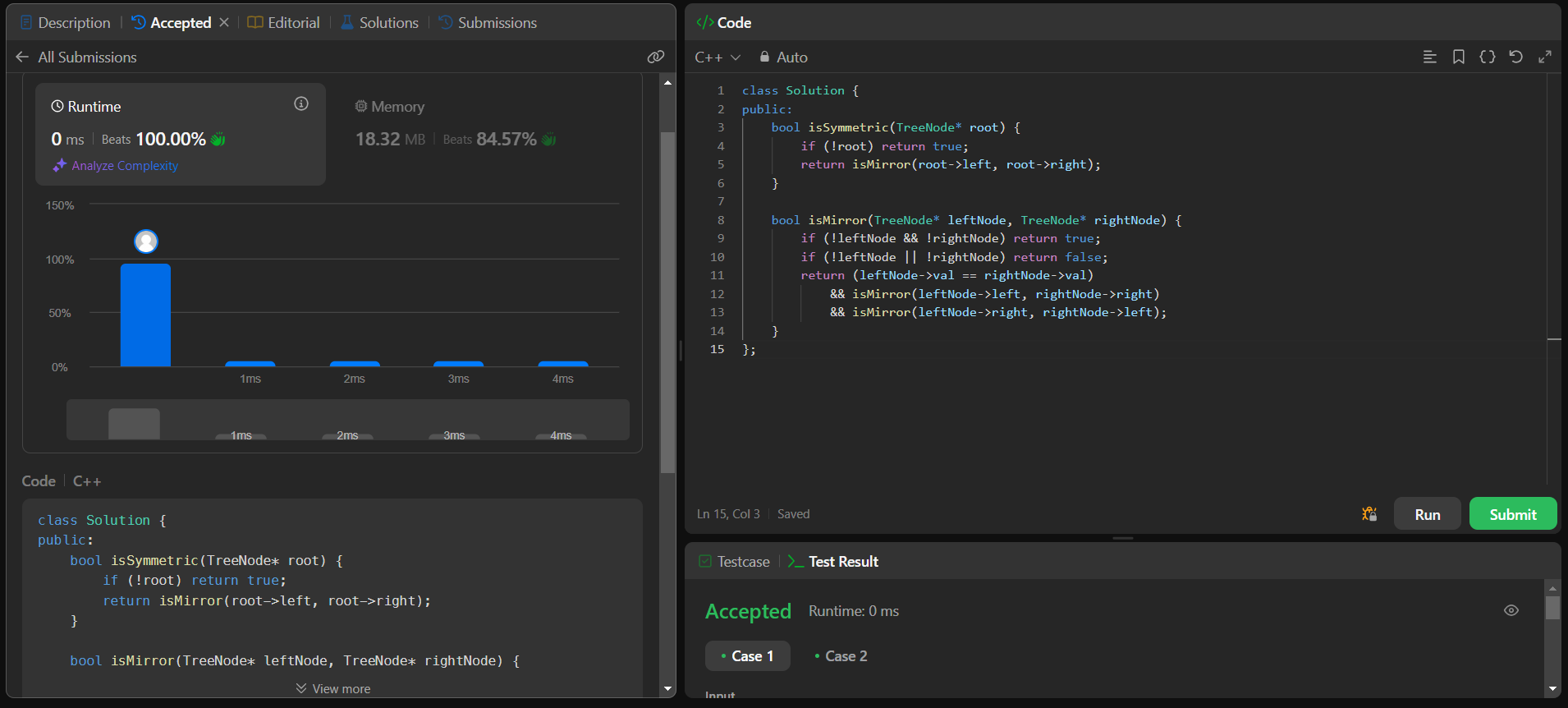
        return (leftNode->val == rightNode->val)

            && isMirror(leftNode->left, rightNode->right)

            && isMirror(leftNode->right, rightNode->left);

    }};

**Screenshot:**

****

**Q4.** [**Binary Tree Zigzag Level Order Traversal**](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/)**:**

**Code:**

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;

        while (!q.empty()) {

            int size = q.size();

            deque<int> level;

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

                TreeNode\* node = q.front();

                q.pop();

                if (leftToRight) {

                    level.push\_back(node->val);

                } else {

                    level.push\_front(node->val);

                }

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

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

            }

            result.push\_back(vector<int>(level.begin(), level.end()));

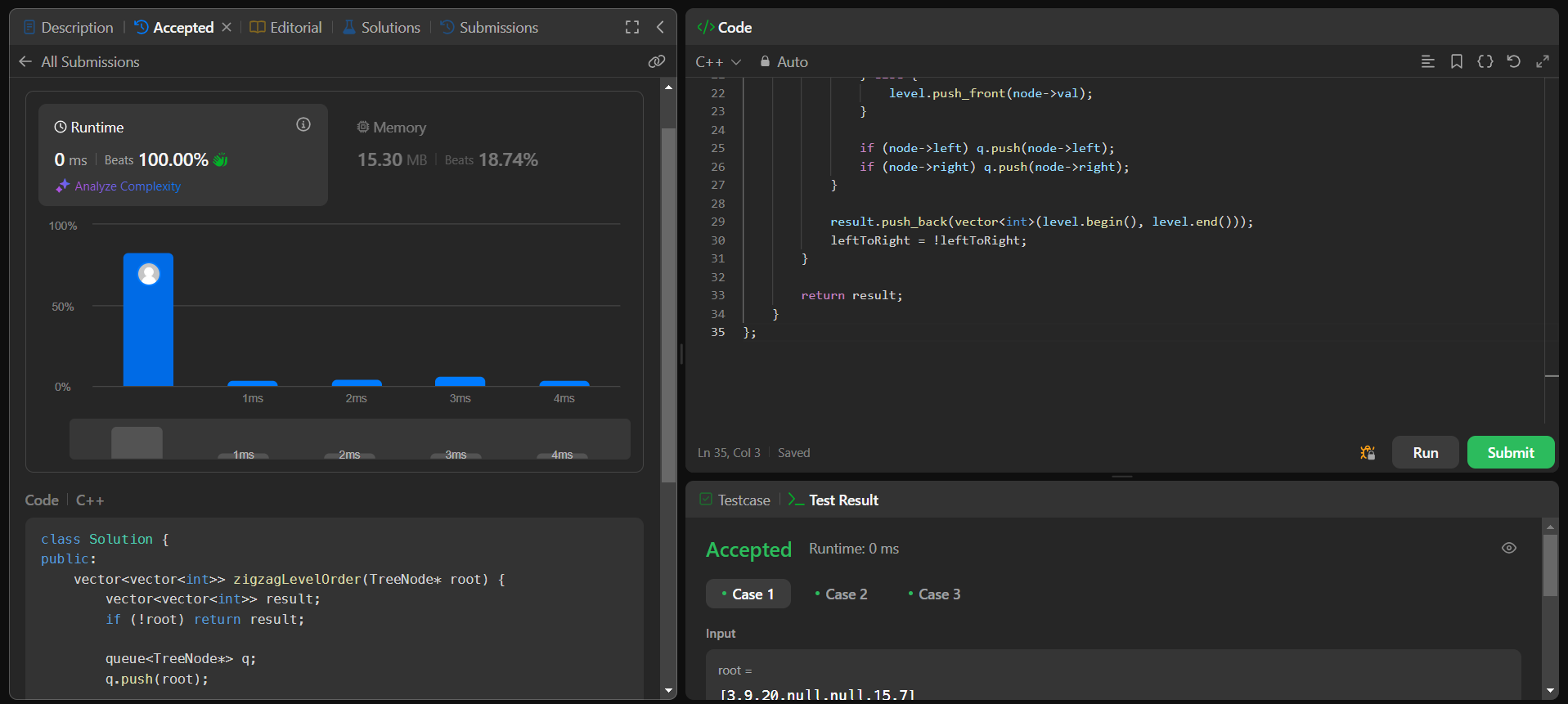
            leftToRight = !leftToRight;

        }

        return result;

    }

};

**Screenshot**:

**Q5.**[**Lowest Common Ancestor of a Binary Tree**](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/)**:**

**Code:**

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if (!root || root == p || root == q) return root;

        TreeNode\* left = lowestCommonAncestor(root->left, p, q);

        TreeNode\* right = lowestCommonAncestor(root->right, p, q);

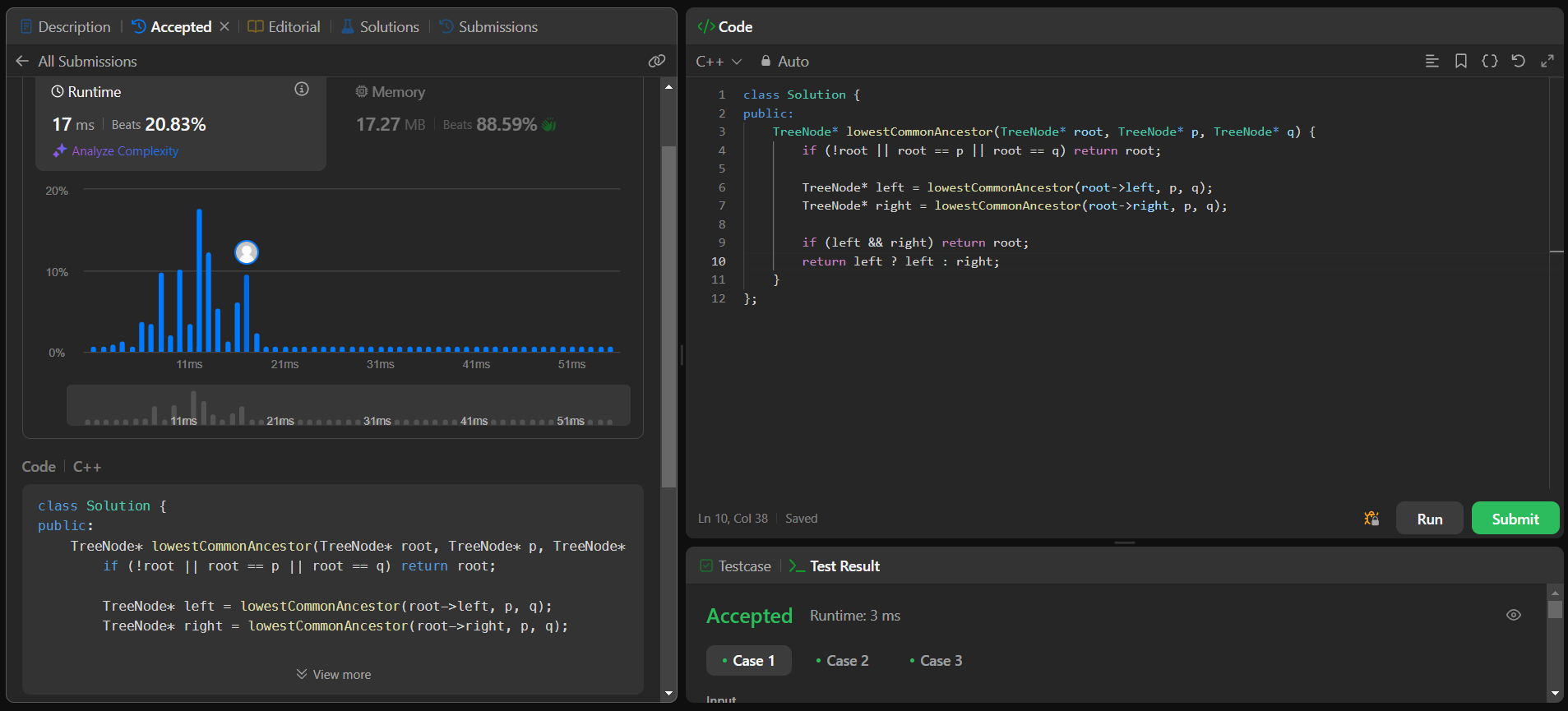
        if (left && right) return root;

        return left ? left : right;

    }

};

**Screenshot:**



**Q6.**[**Binary Tree Inorder Traversal**](https://leetcode.com/problems/binary-tree-inorder-traversal/description/)**:**

**Code:**

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;

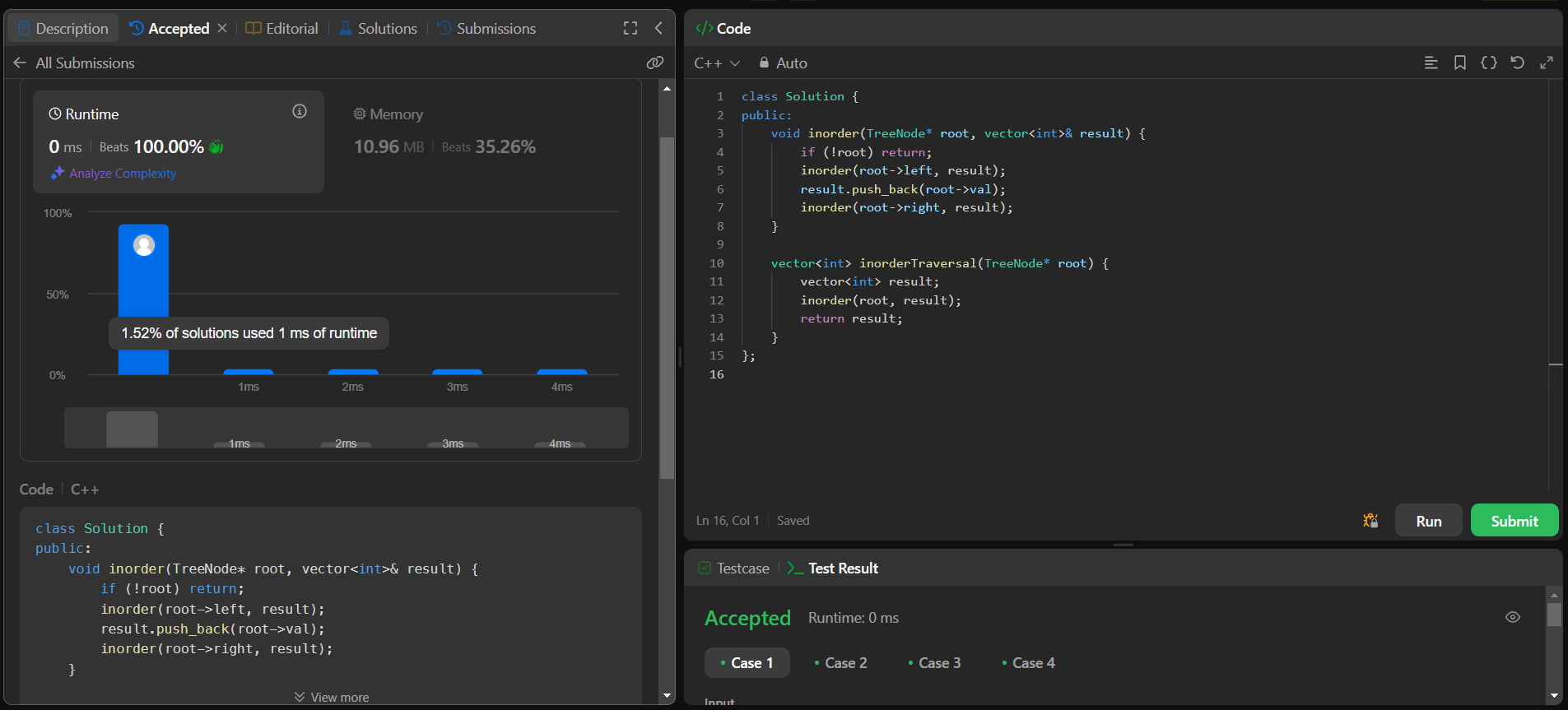
        inorder(root, result);

        return result;

    }

};

**Screenshot:**

****

**Q7.** [**Binary Tree Level Order Traversal**](https://leetcode.com/problems/binary-tree-level-order-traversal/description/)**:**

**Code:**

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 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);

            }

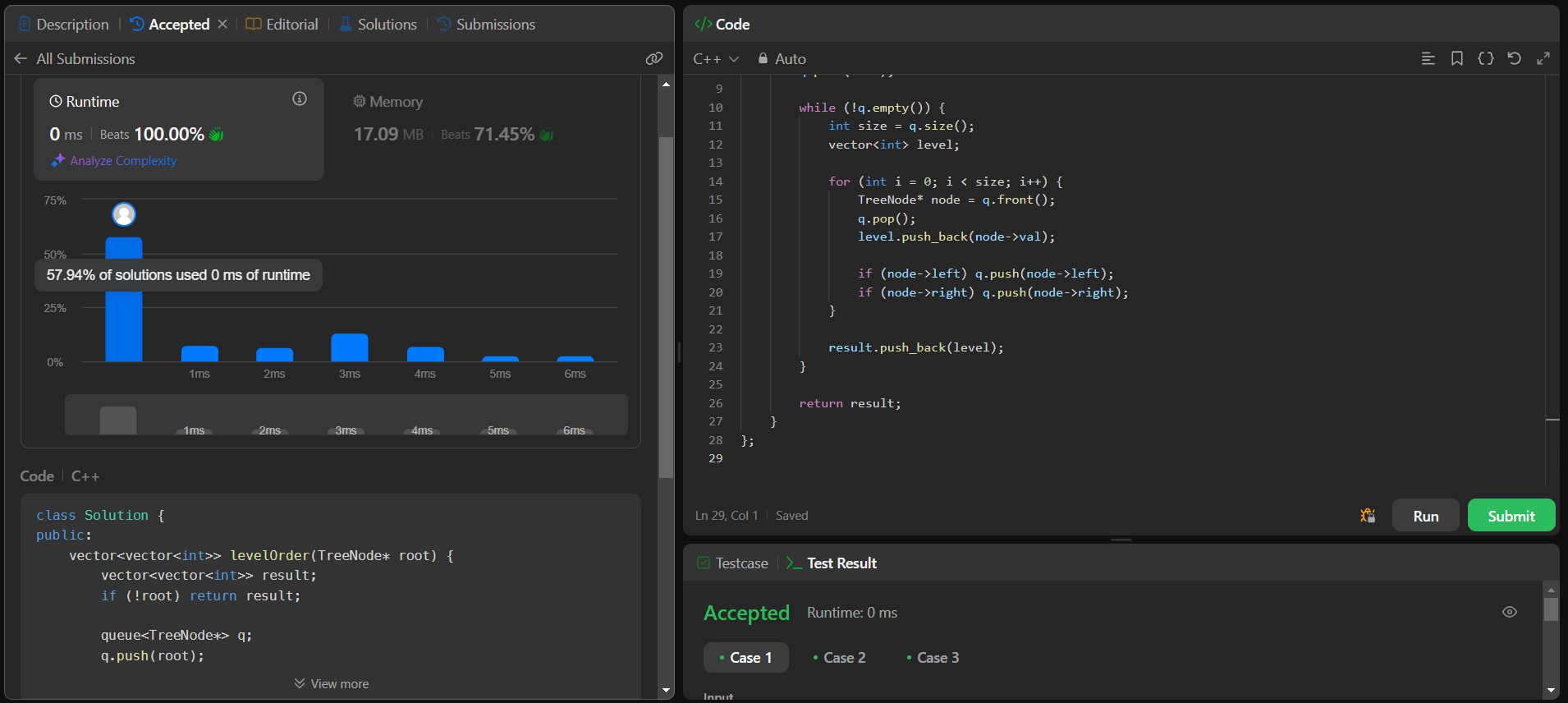
            result.push\_back(level);

        }

        return result;

    }

};

**Screenshot:**

**Q8.**[**Kth Smallest Element in a BST**](https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/)**:**

**Code:**

class Solution {

public:

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

        int count = 0, result = -1;

        inorder(root, k, count, result);

        return result;

    }

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

        if (!root) return;

        inorder(root->left, k, count, result);

        count++;

        if (count == k) {

            result = root->val;

            return;

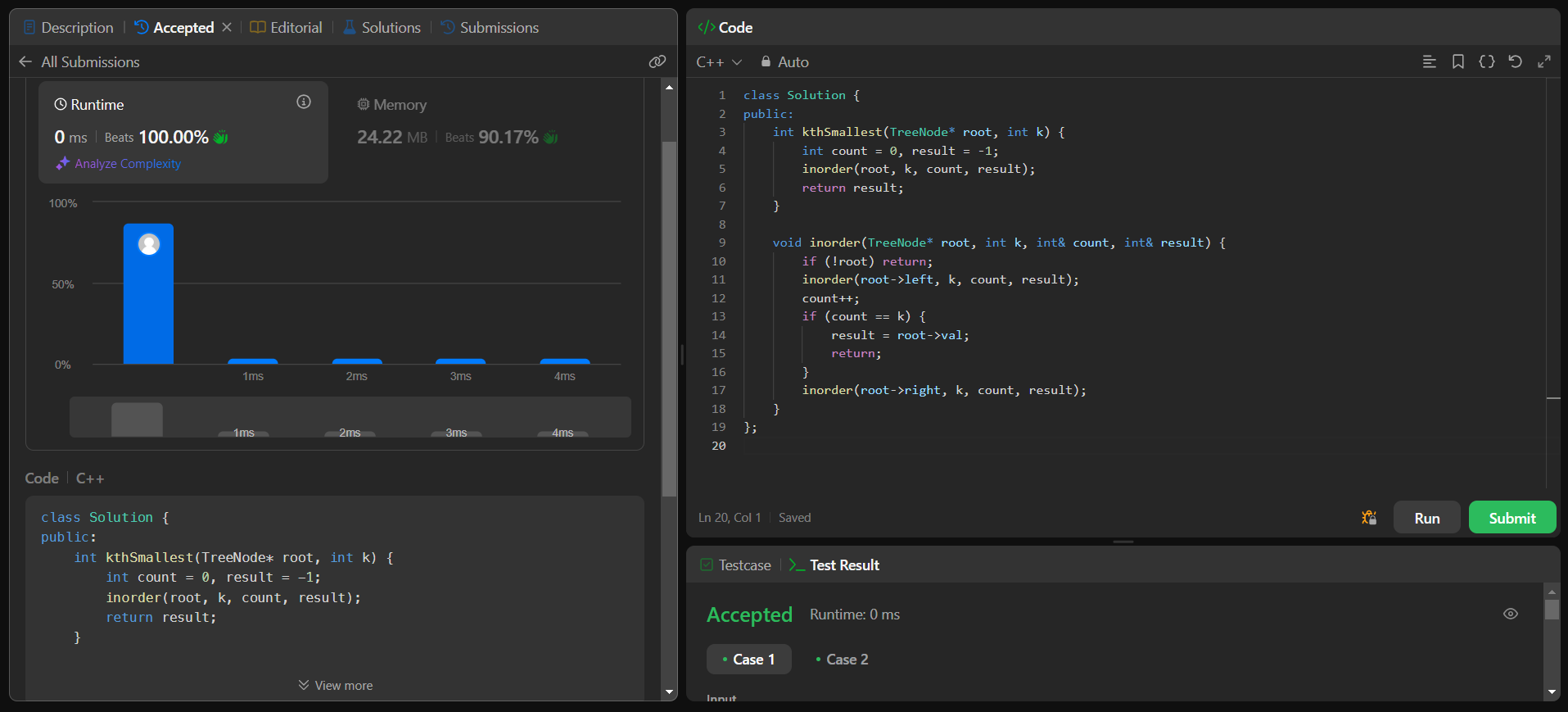
        }

        inorder(root->right, k, count, result);

    }

};

**Screenshot:**

****

**Q9.** [**Populating Next Right Pointers in Each Node**](https://leetcode.com/problems/populating-next-right-pointers-in-each-node/description/)**:**

**Code:**

class Solution {

public:

    Node\* connect(Node\* root) {

        if (!root) return nullptr;

        queue<Node\*> q;

        q.push(root);

        while (!q.empty()) {

            int size = q.size();

            Node\* prev = nullptr;

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

                Node\* curr = q.front();

                q.pop();

                if (prev) prev->next = curr;

                prev = curr;

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

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

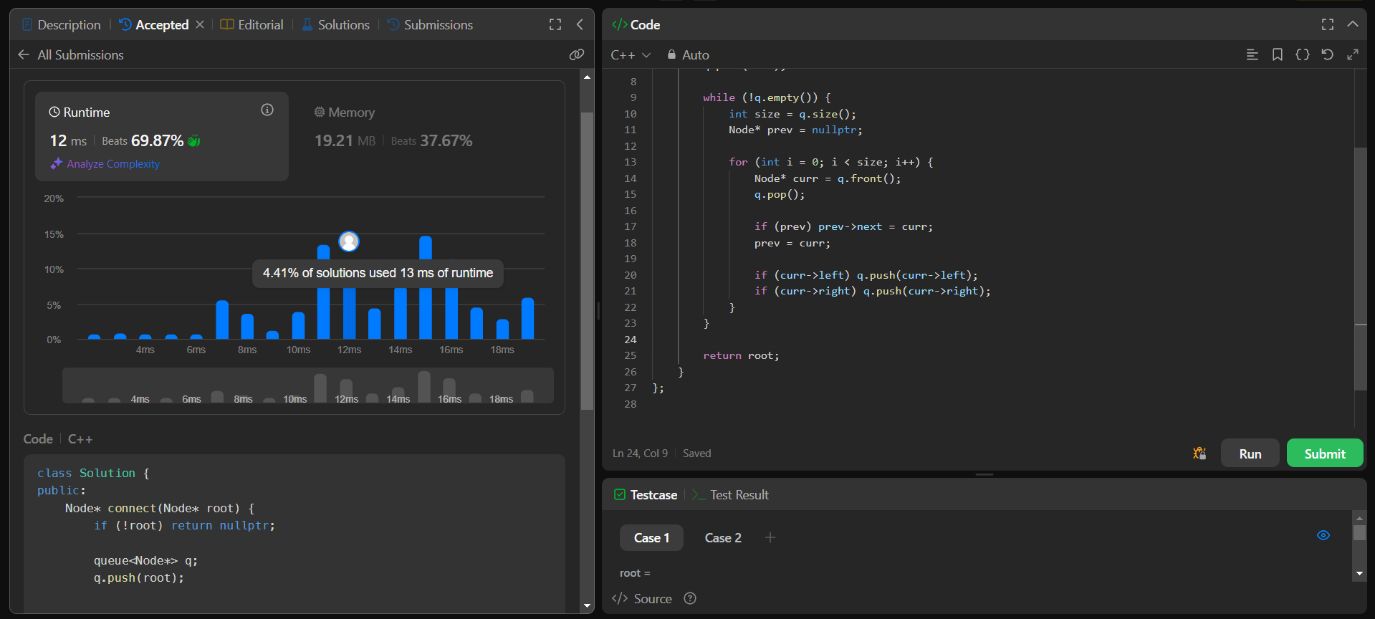
            } }

        return root;

    }

};

**Screenshot:**

****

**Q10:** [**Sum of Left Leaves**](https://leetcode.com/problems/sum-of-left-leaves/description/)**:**

**Code:**

class Solution {

public:

    int sumOfLeftLeaves(TreeNode\* root) {

        if (!root) return 0;

        int sum = 0;

        if (root->left && !root->left->left && !root->left->right) {

            sum += root->left->val;

        }

        sum += sumOfLeftLeaves(root->left);

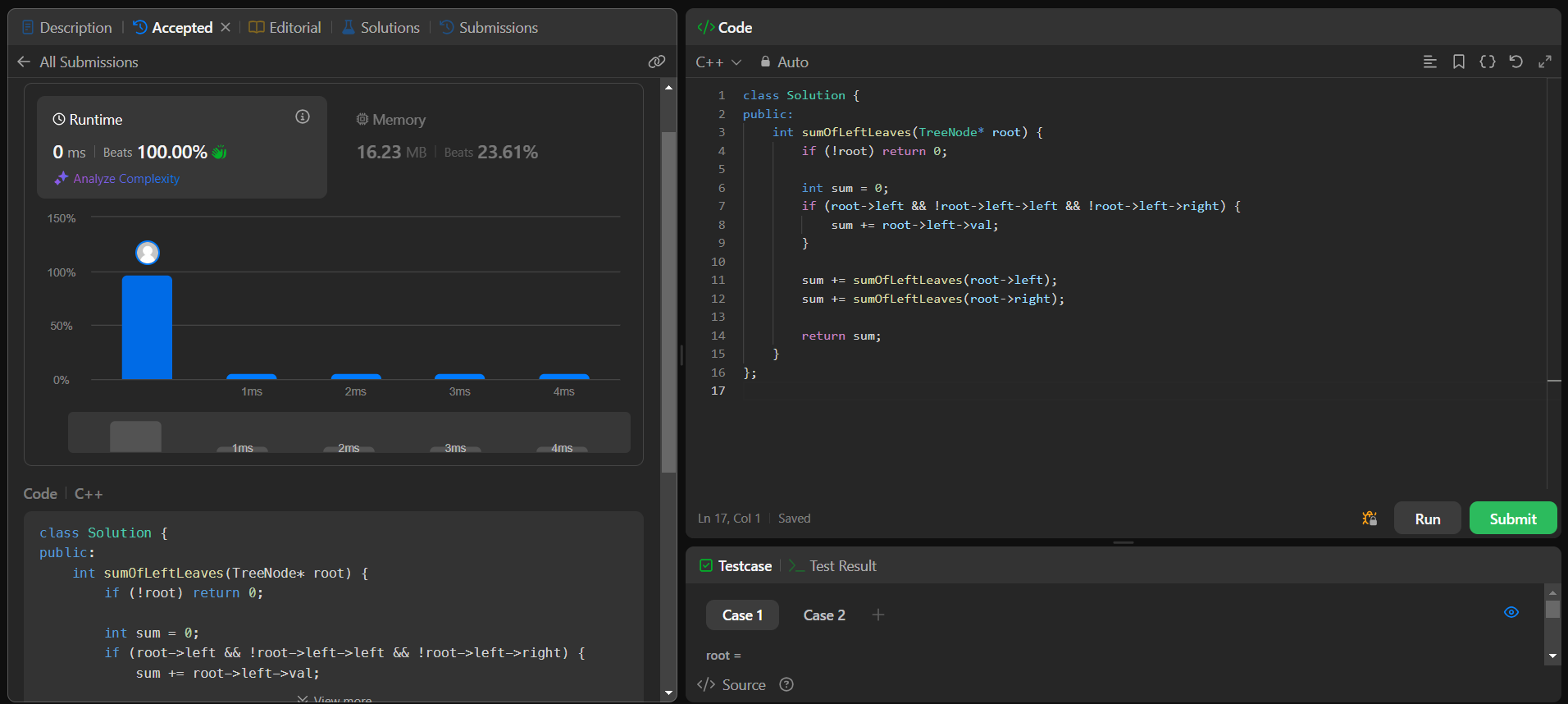
        sum += sumOfLeftLeaves(root->right);

        return sum;

    }

};

**Screenshot:**

****