**Assignment 3**

**Name – Aditya Taragi Uid – 22BCS10422.**

**Problem 1: Binary Tree Inorder Traversal (**<https://leetcode.com/problems/binary-tree-inorder-traversal/> **)**

**Code:**class Solution {

public:

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

        vector<int> result;

        traverse(root,result);

        return result;

    }

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

        if(root == NULL) return;

        traverse(root->left,result);

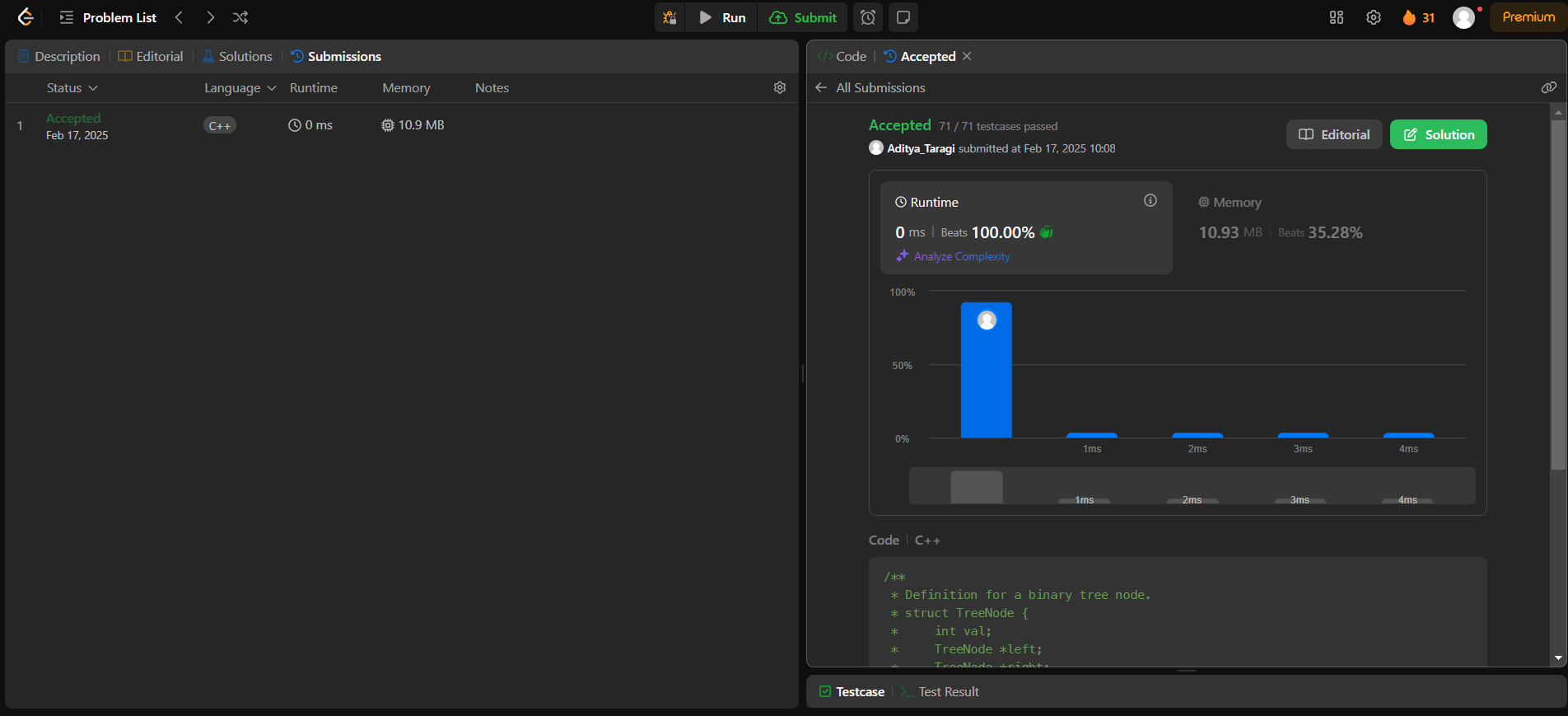
        result.push\_back(root->val);

        traverse(root->right,result);

    }

};

**Screenshot:**

****

**Problem 2: Symmetric Tree (**<https://leetcode.com/problems/symmetric-tree/> )

**Code:**class Solution {

public:

    bool areMirrImg(TreeNode\* root1, TreeNode\* root2){

        if(!root1 && !root2){

            return true;

        }

        if(!root1 || !root2){

            return false;

        }

        return (root1->val == root2->val) && (areMirrImg(root1->left,root2->right)) && (areMirrImg(root1->right,root2->left));

    }

    bool isSymmetric(TreeNode\* root) {

        if(!root){

            return true;

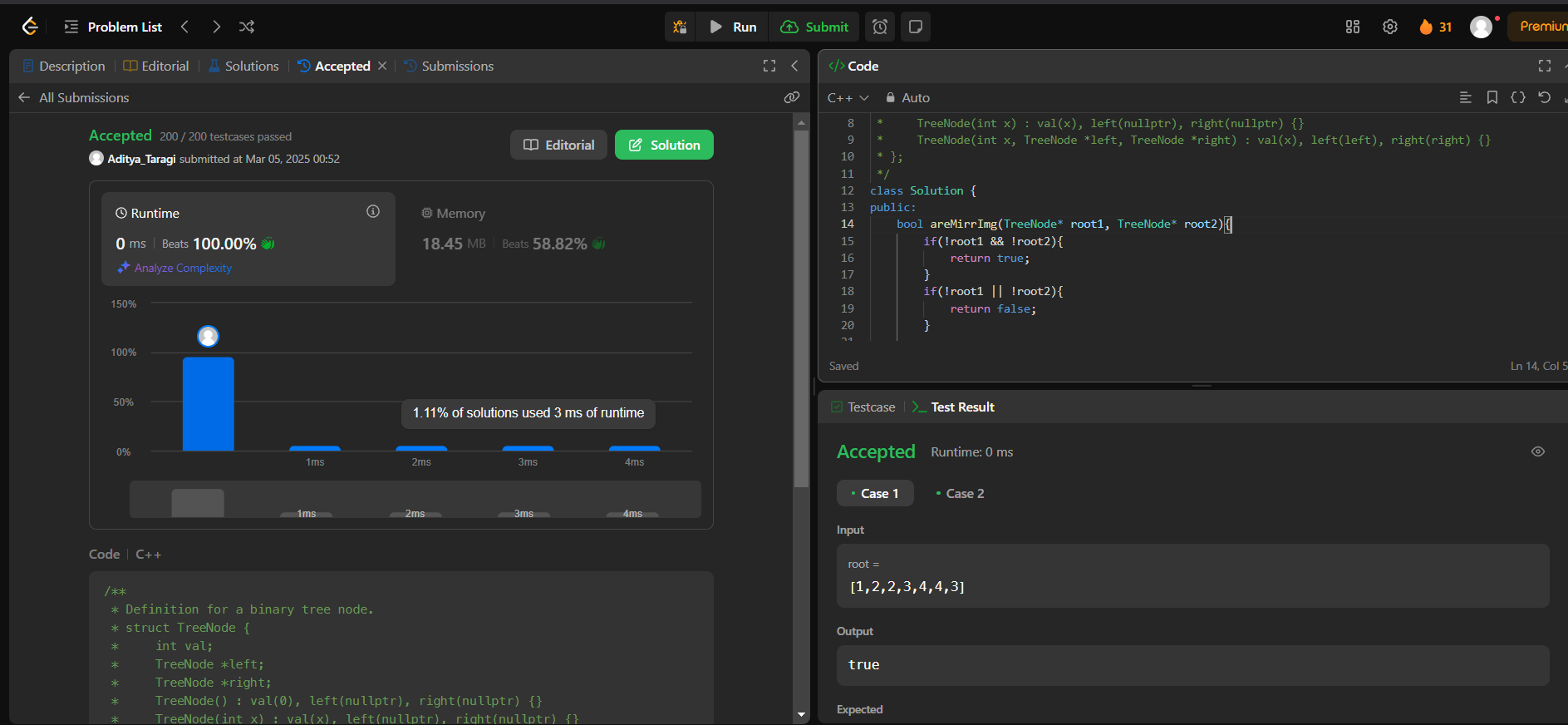
        }

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

    }

};

**Screenshot:**



**Problem 3: Maximum Depth of Binary Tree (**<https://leetcode.com/problems/maximum-depth-of-binary-tree/> **)**

**Code:**class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if(root == NULL) return 0;

        int L = maxDepth(root->left);

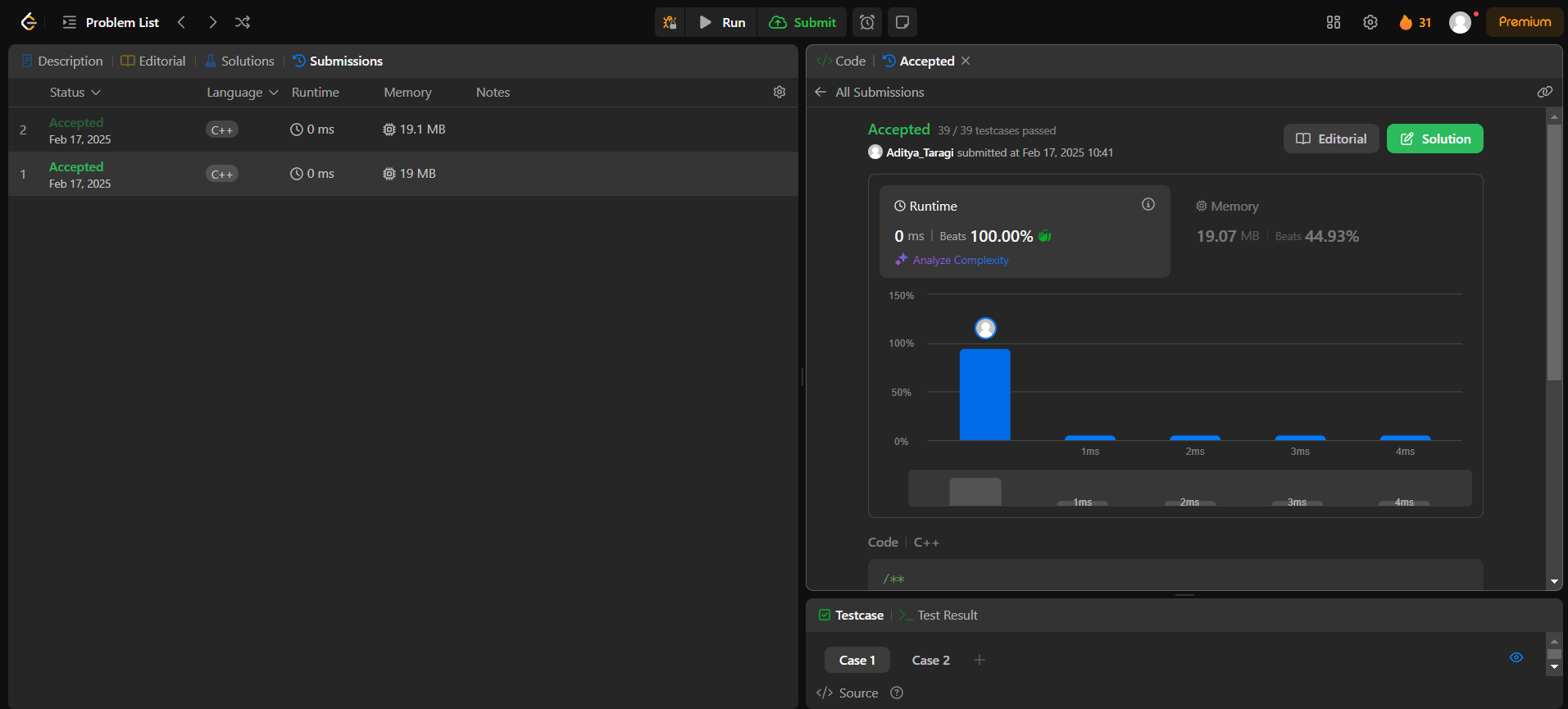
        int R = maxDepth(root->right);

        return max(L,R) + 1;

    }

};

**Screenshot:**



**Problem 4: Validate Binary Search Tree (**<https://leetcode.com/problems/validate-binary-search-tree/> **)**

**Code:**class Solution {

public:

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

        if (!root) return ;

        findInorder(root->left, inorder);

        inorder.push\_back(root->val);

        findInorder(root->right, inorder);

    bool isValidBST(TreeNode\* root) {

        vector<int> inorder;

        findInorder(root, inorder);

        for (int i = 1; i < inorder.size(); i++) {

            if (inorder[i - 1] >= inorder[i]) return false;

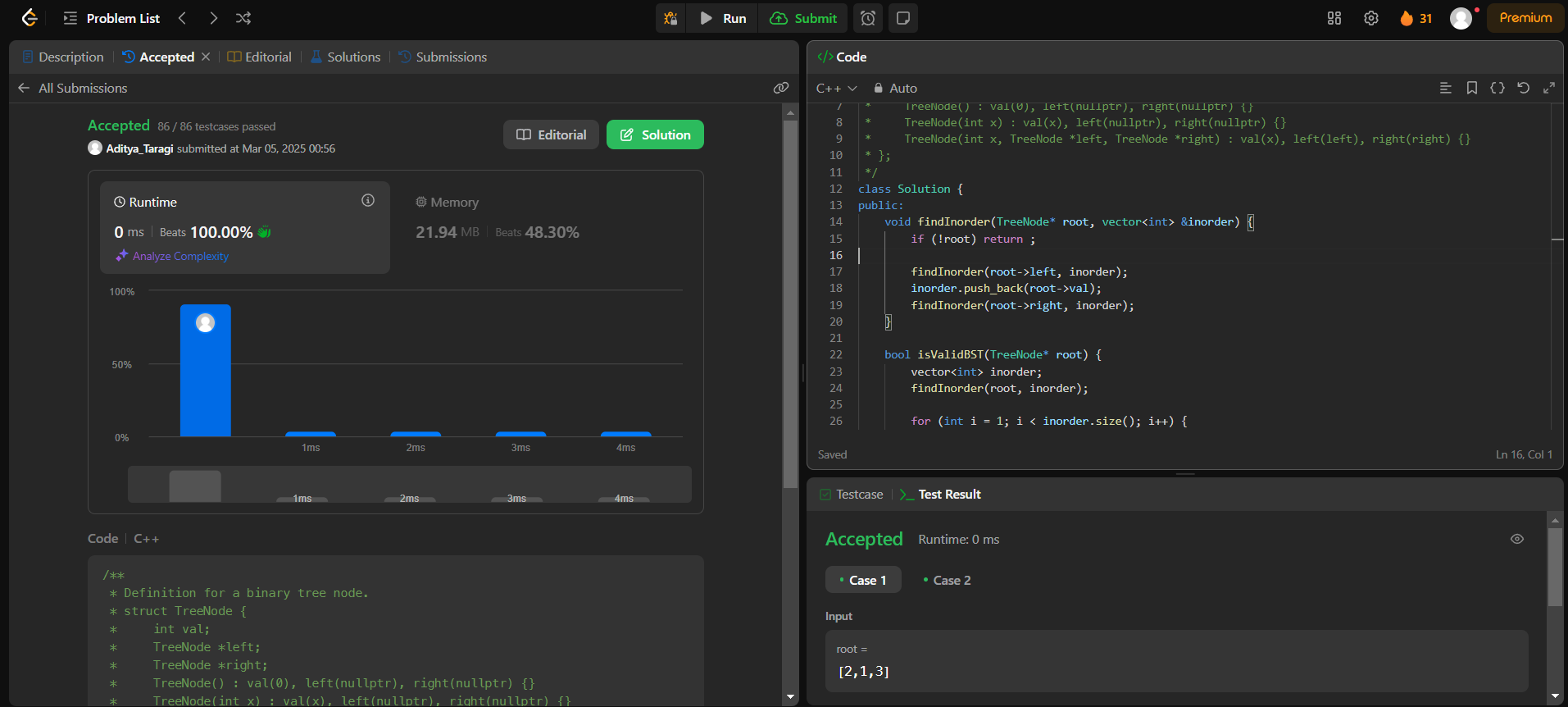
        }

        return true;

    }

};

**Screenshot:**



**Problem 5: kth Smallest Element in a BST (**<https://leetcode.com/problems/kth-smallest-element-in-a-bst/> **)**

**Code:**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;

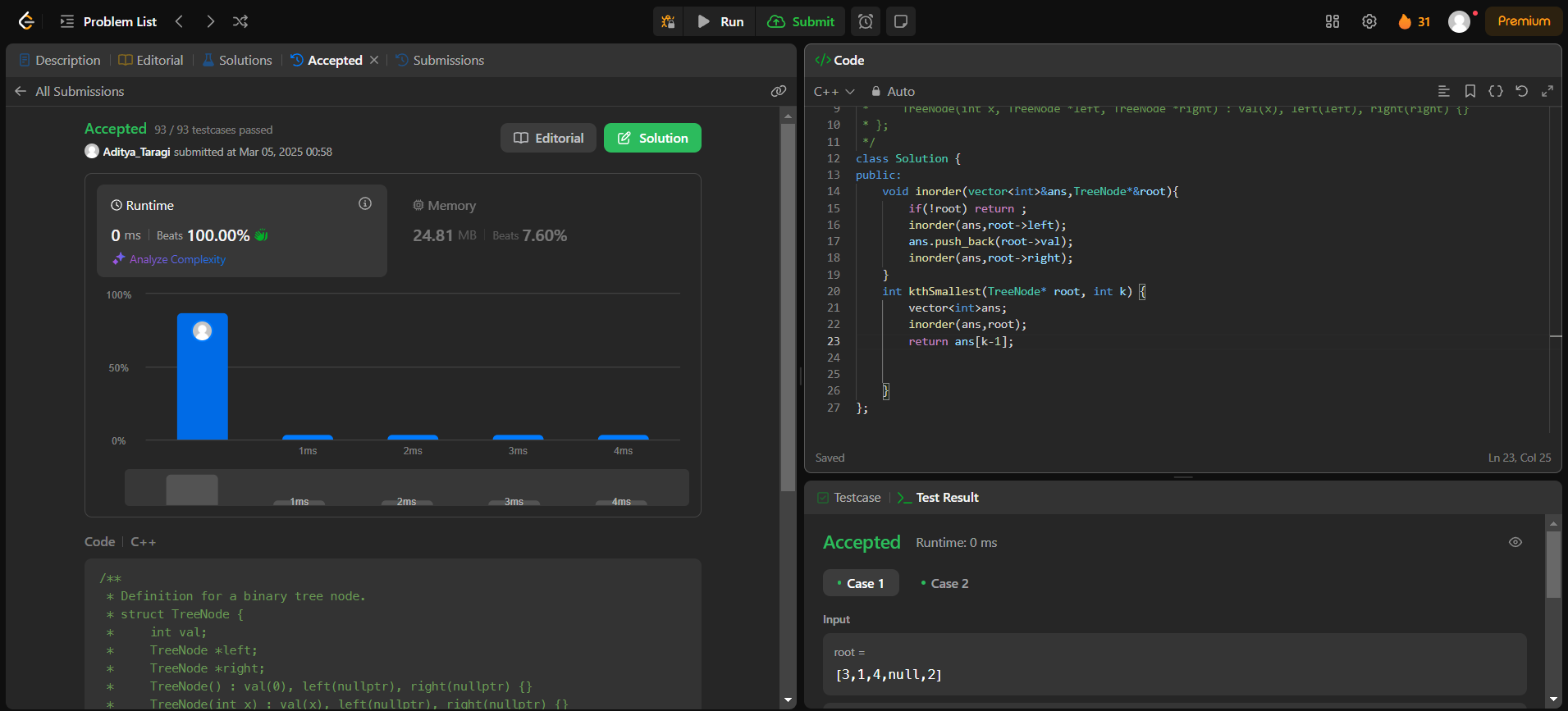
        inorder(ans,root);

        return ans[k-1];

    }

};

**Screenshot:**



**Problem 6: Binary Tree Level Order Traversal (**[https://leetcode.com/problems/binary-tree-level-order-traversal/](https://leetcode.com/problems/binary-tree-level-order-traversal/submissions/1562957007/) **)**

**Code:**class Solution {

public:

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

        vector<vector<int>> ans;

        if(root==NULL){

            return ans;

        }

        queue<TreeNode\*> q;

        q.push(root);

        while (q.size()>0) {

            vector<int> level;

            int size = q.size();

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

                TreeNode\* node = q.front();

                q.pop();

                if (node->left != NULL) {

                    q.push(node->left);

                }

                if (node->right != NULL) {

                    q.push(node->right);

                }

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

            }

            ans.push\_back(level);

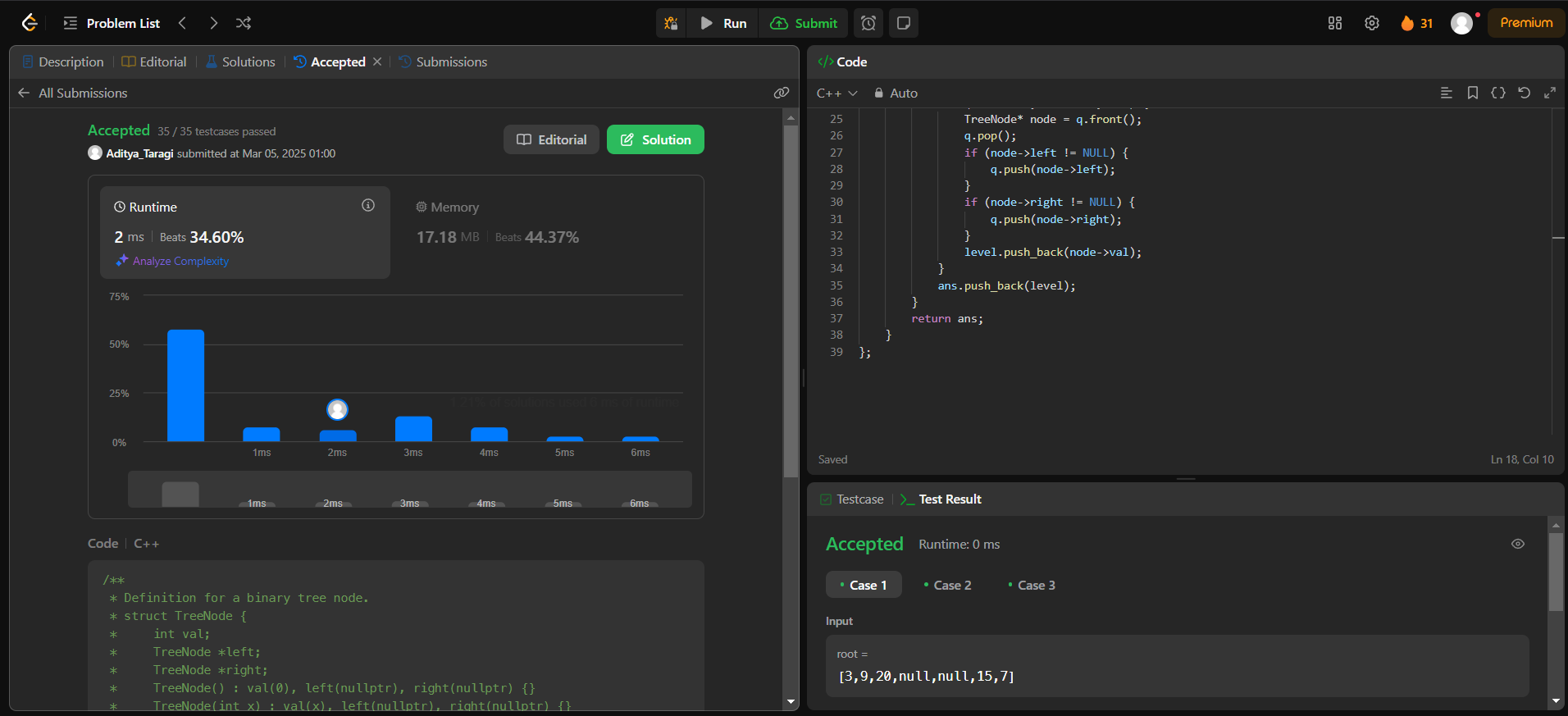
        }

        return ans;

    }

};

**Screenshot:**



**Problem 7: Binary Tree Level Order Traversal II (**<https://leetcode.com/problems/binary-tree-level-order-traversal-ii/> **)**

**Code:**class Solution {

public:

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

        vector<vector<int>> levels;

        if (!root) return levels;

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            int n = q.size();

            vector<int> level;

            for (int i = 0; i < n; ++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);

            }

            levels.insert(levels.begin(), level);

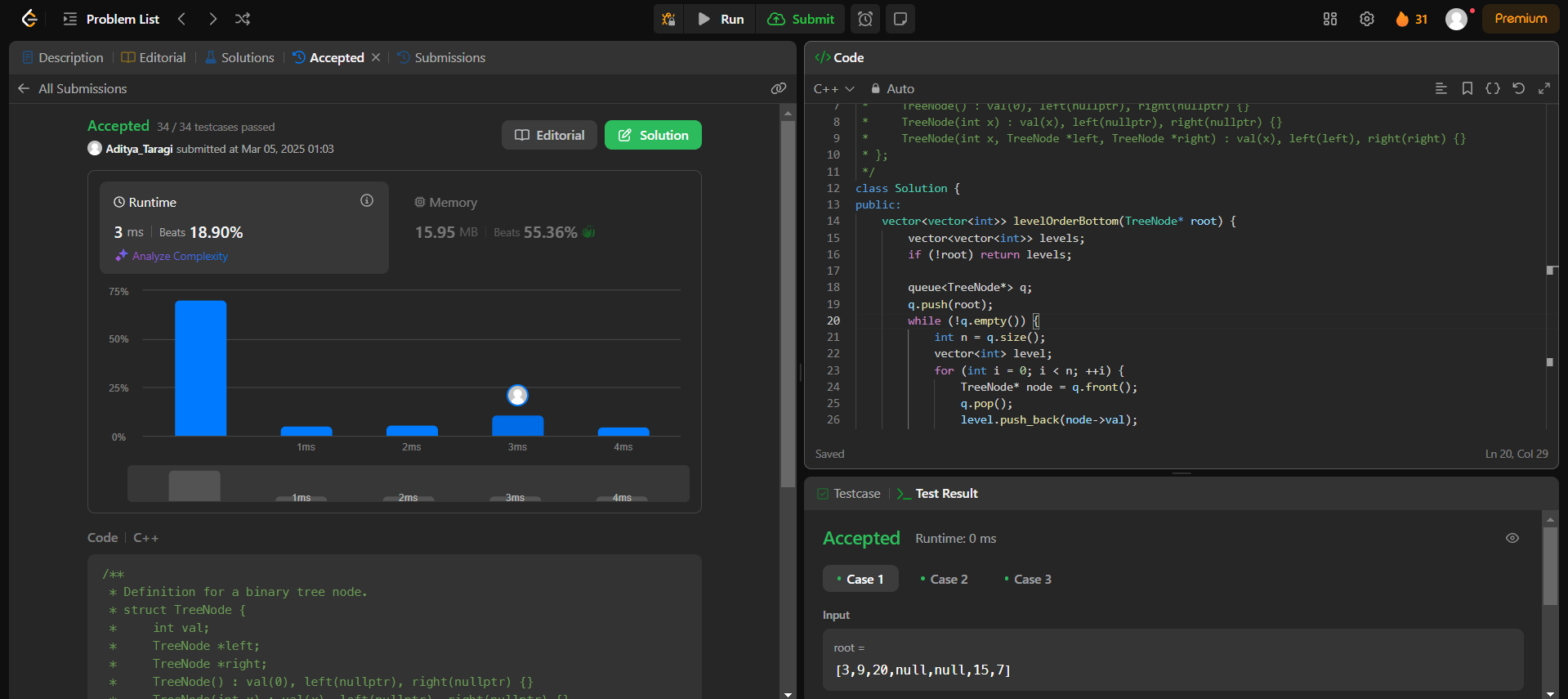
        }

        return levels;

    }

};

**Screenshot:**



**Problem 8: Binary Tree Right Side View (**<https://leetcode.com/problems/binary-tree-right-side-view/> **)**

**Code:**class Solution {

public:

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

        vector<int> ans;

        if (!root) return ans;

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            int n = q.size();

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

                TreeNode\* curr = q.front();

                q.pop();

                if (i == n - 1) ans.push\_back(curr->val);

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

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

            }

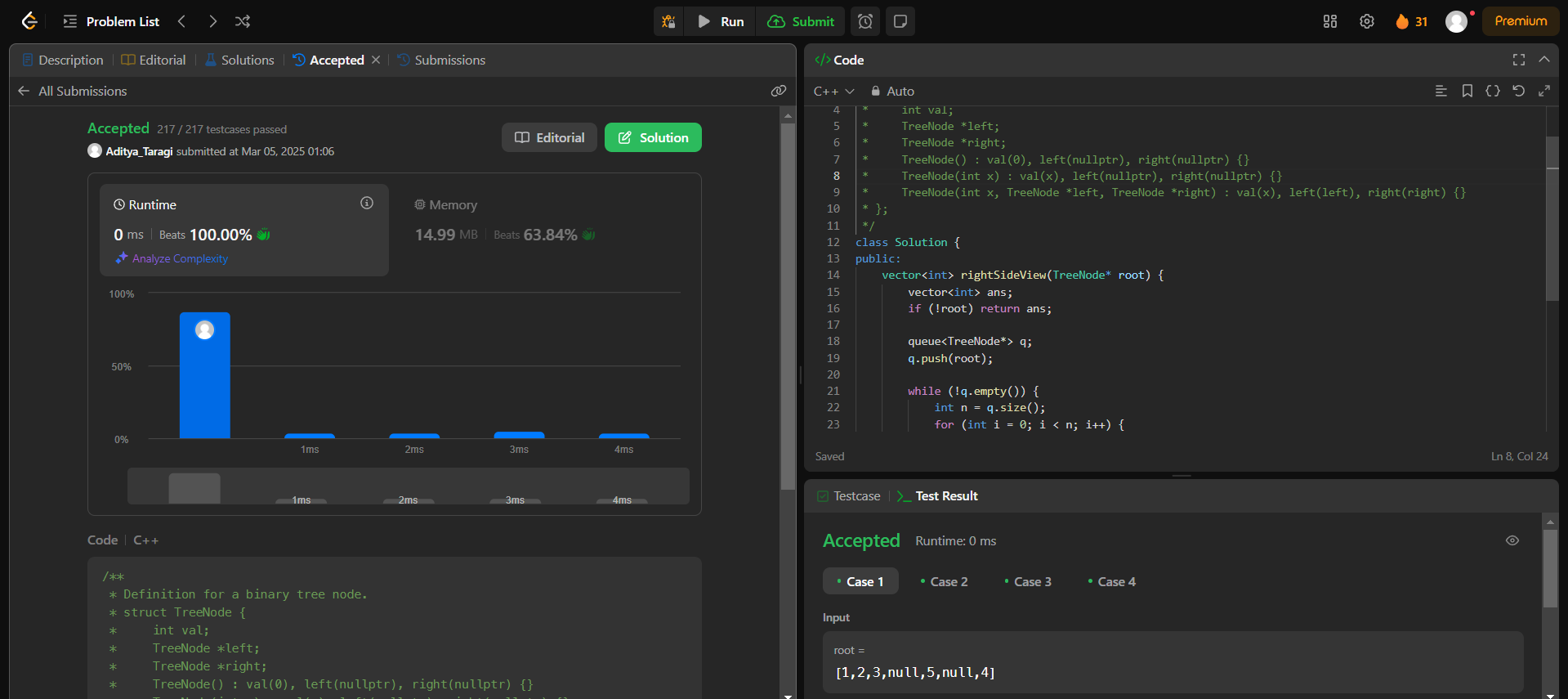
        }

        return ans;

    }

};

**Screenshot:**



**Problem 9: Binary Tree Zigzag Level Order Traversal (**<https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/> **)**

**Code:**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;

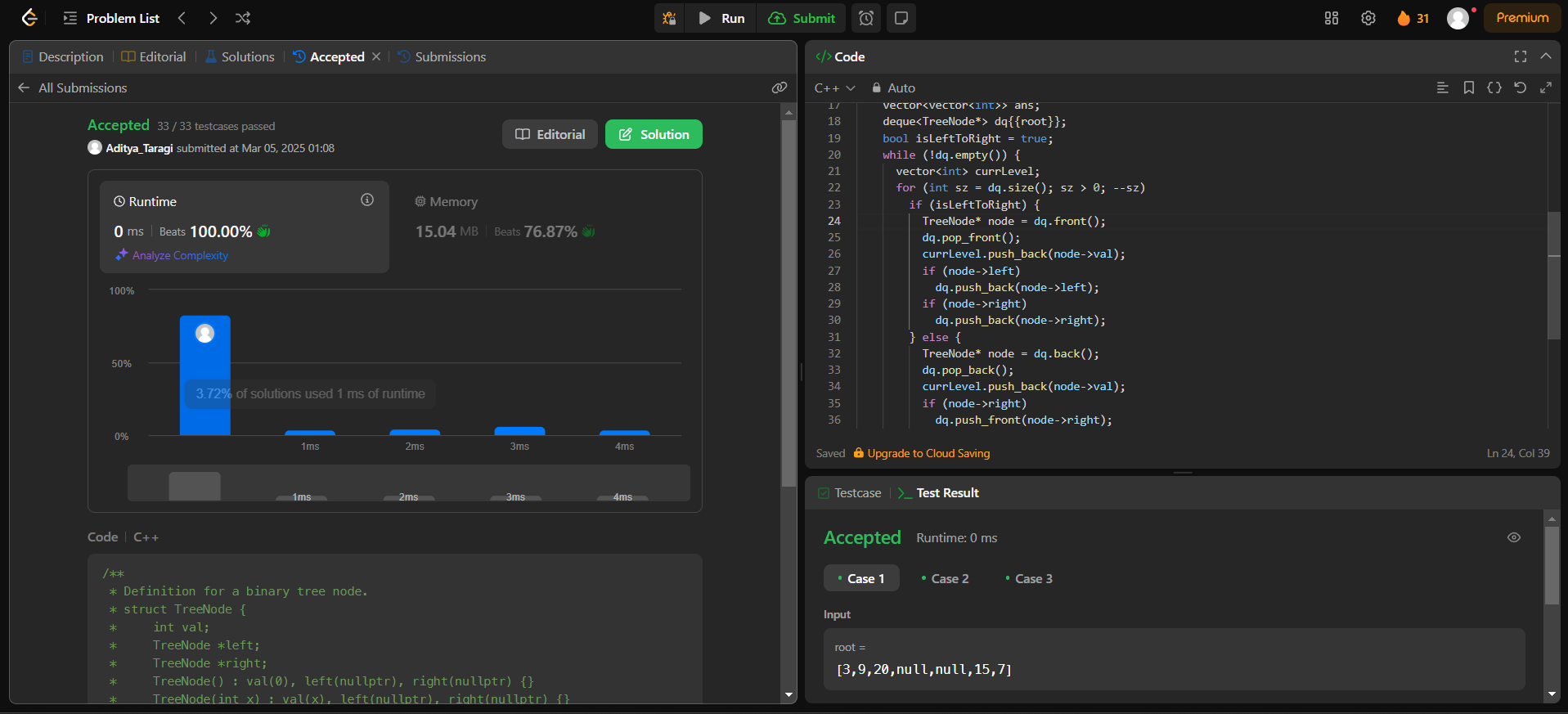
    }

    return ans;

    }

};

**Screenshot:**



**Problem 10: Construct Binary Tree From Inorder and postorder traversal (** [**https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/**](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/) **)**

**Code:**class Solution {

public:

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

        unordered\_map<int, int> inorderIndexMap;

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

            inorderIndexMap[inorder[i]] = i;

        }

        int postIndex = postorder.size() - 1;

        return constructTree(inorder, postorder, inorderIndexMap, postIndex, 0, inorder.size() - 1);

    }

    TreeNode\* constructTree(vector<int>& inorder, vector<int>& postorder, unordered\_map<int, int>& inorderIndexMap, int& postIndex, int inStart, int inEnd) {

        if (inStart > inEnd) return nullptr;

        int rootVal = postorder[postIndex--];

        TreeNode\* root = new TreeNode(rootVal);

        int rootIndex = inorderIndexMap[rootVal];

        root->right = constructTree(inorder, postorder, inorderIndexMap, postIndex, rootIndex + 1, inEnd);

        root->left = constructTree(inorder, postorder, inorderIndexMap, postIndex, inStart, rootIndex - 1);

        return root;

    }

};

**Screenshot:**

