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

* **Solution:**

class Solution {

public:

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

        vector<int> res;

        inorder(root, res);

        return res;

    }

private:

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

        if (!node) {

            return;

        }

        inorder(node->left, res);

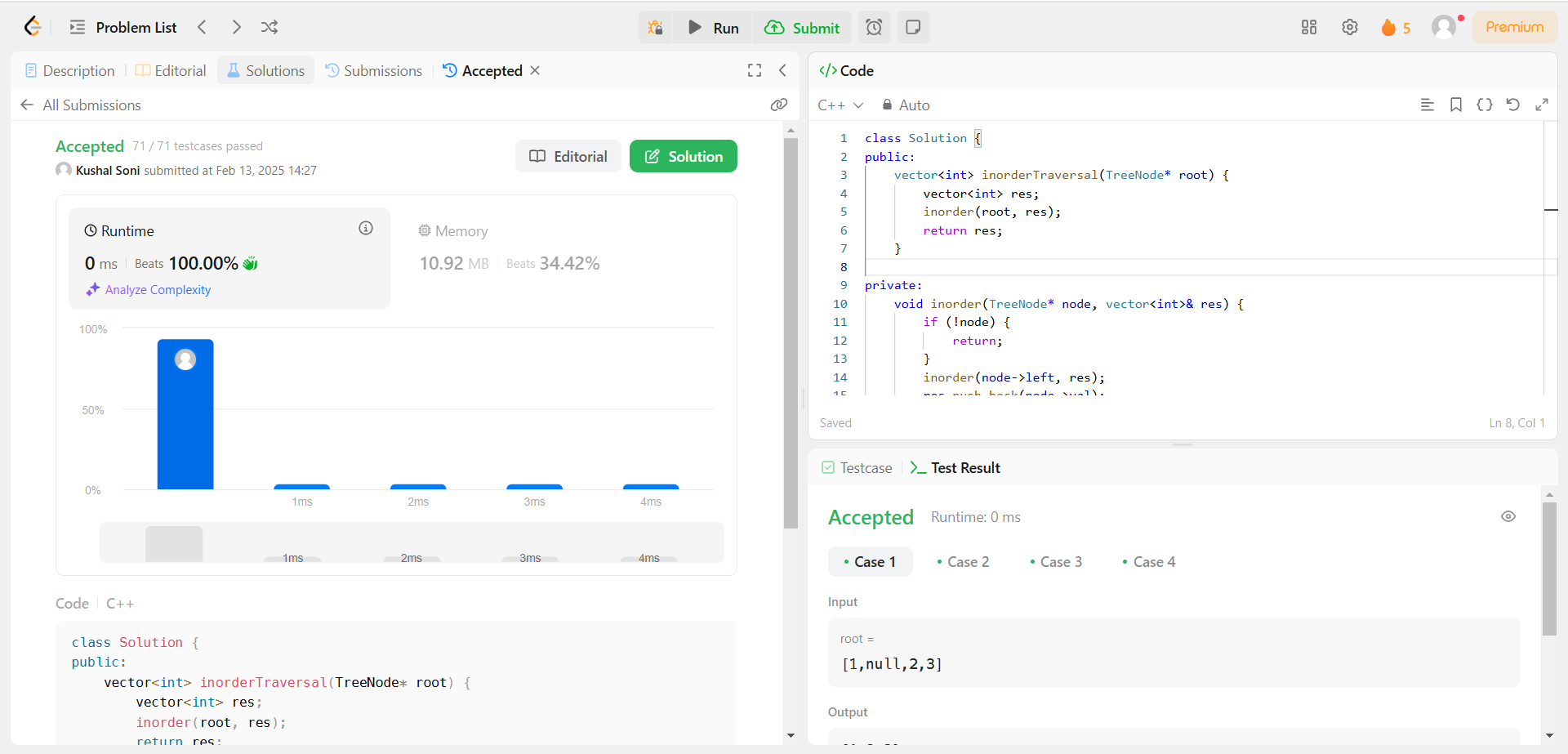
        res.push\_back(node->val);

        inorder(node->right, res);

    }

};

* **Screenshot:**

****

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

* **Solution:**

class Solution {

public:

    bool isSymmetric(TreeNode\* root) {

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

    }

private:

    bool isMirror(TreeNode\* n1, TreeNode\* n2) {

        if (n1 == nullptr && n2 == nullptr) {

            return true;

        }

        if (n1 == nullptr || n2 == nullptr) {

            return false;

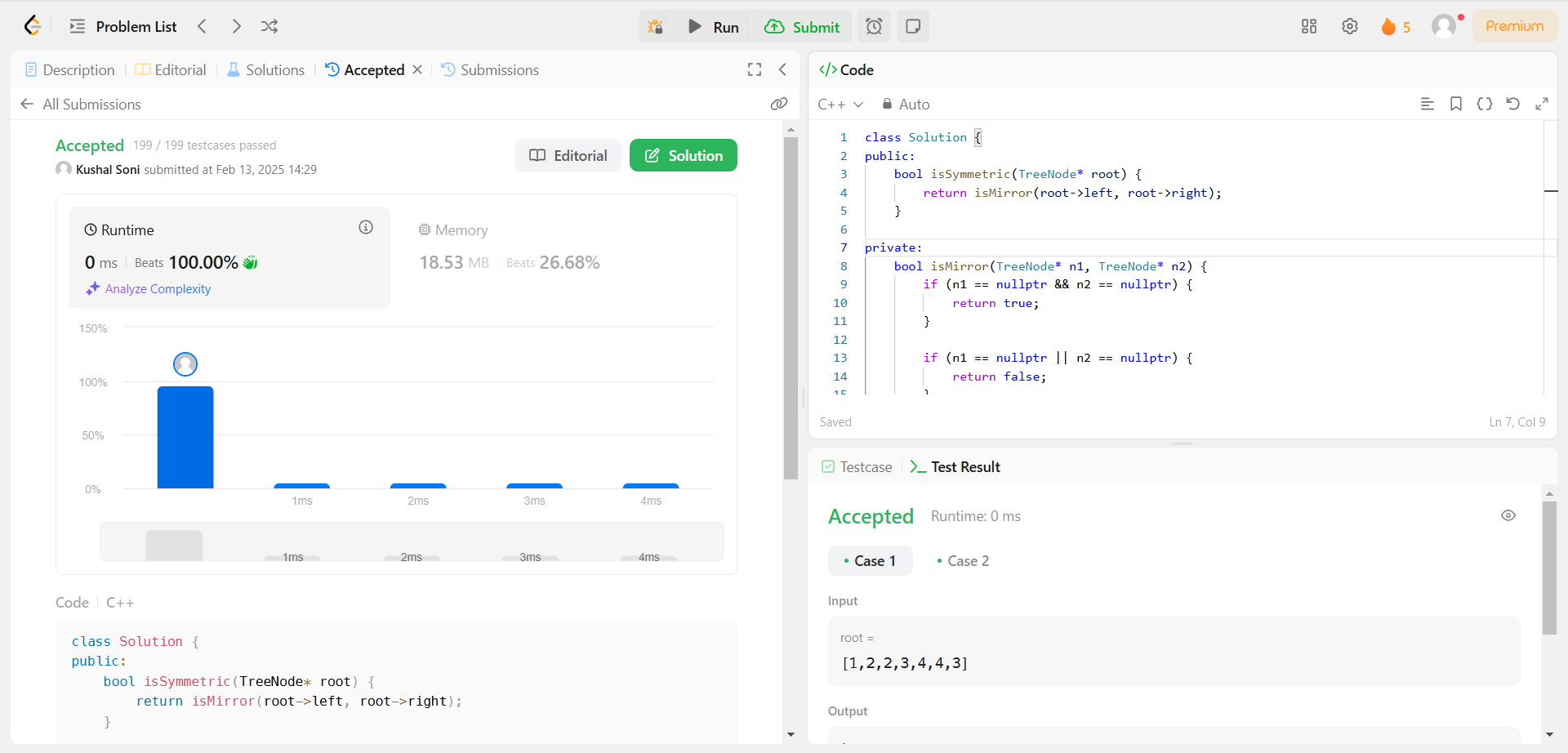
        }

        return n1->val == n2->val && isMirror(n1->left, n2->right) && isMirror(n1->right, n2->left);

    }

};

* **Screenshot:**

****

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

* **Solution:**

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if (!root) {

            return 0;

        }

        queue<TreeNode\*> q;

        q.push(root);

        int depth = 0;

        while (!q.empty()) {

            depth++;

            int levelSize = q.size();

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

                TreeNode\* node = q.front();

                q.pop();

                if (node->left) {

                    q.push(node->left);

                }

                if (node->right) {

                    q.push(node->right);

                }

            }

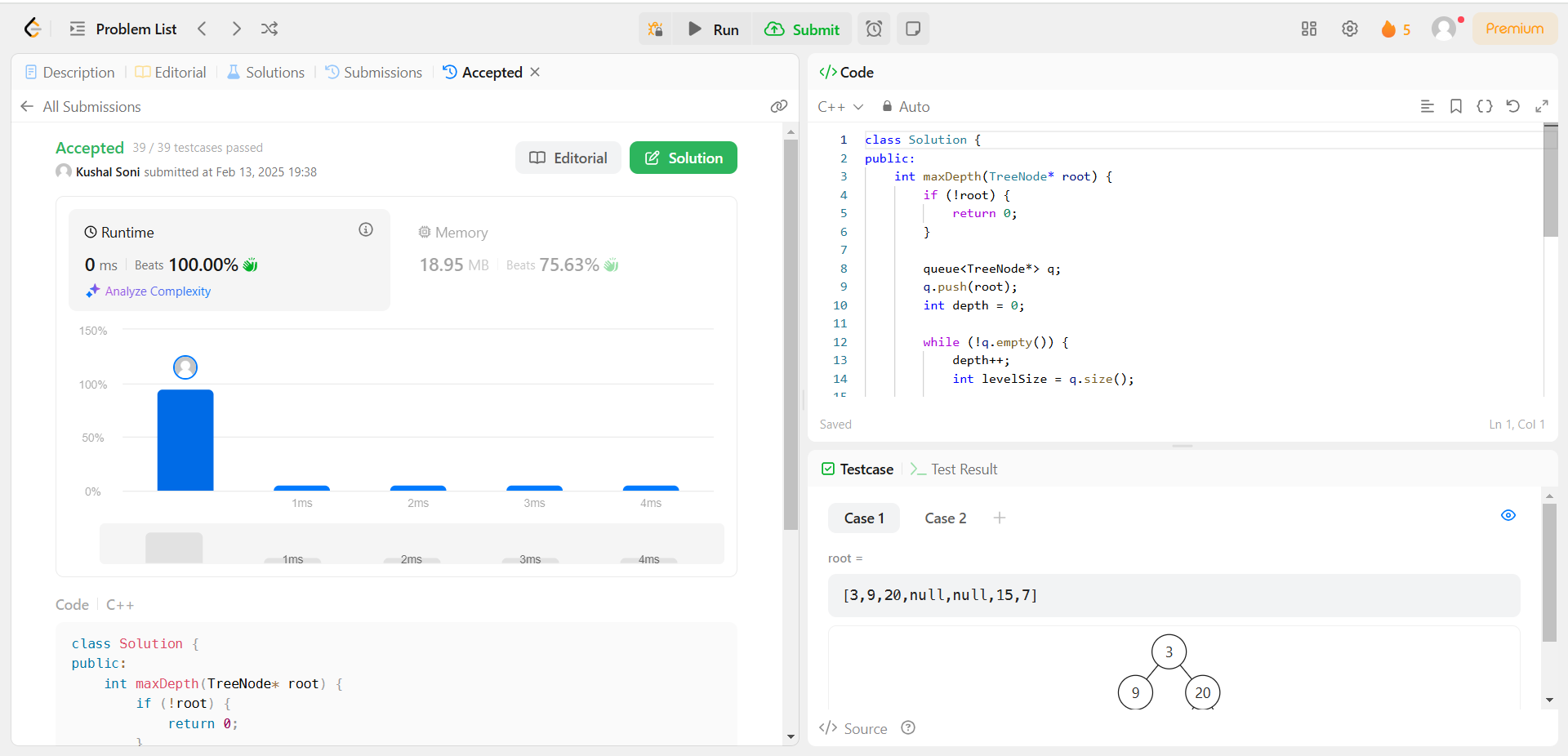
        }

        return depth;

    }

};

* **Screenshot:**

****

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

* **Solution:**

class Solution {

public:

    bool isValidBST(TreeNode\* root) {

        return valid(root, LONG\_MIN, LONG\_MAX);

    }

private:

    bool valid(TreeNode\* node, long minimum, long maximum) {

        if (!node) return true;

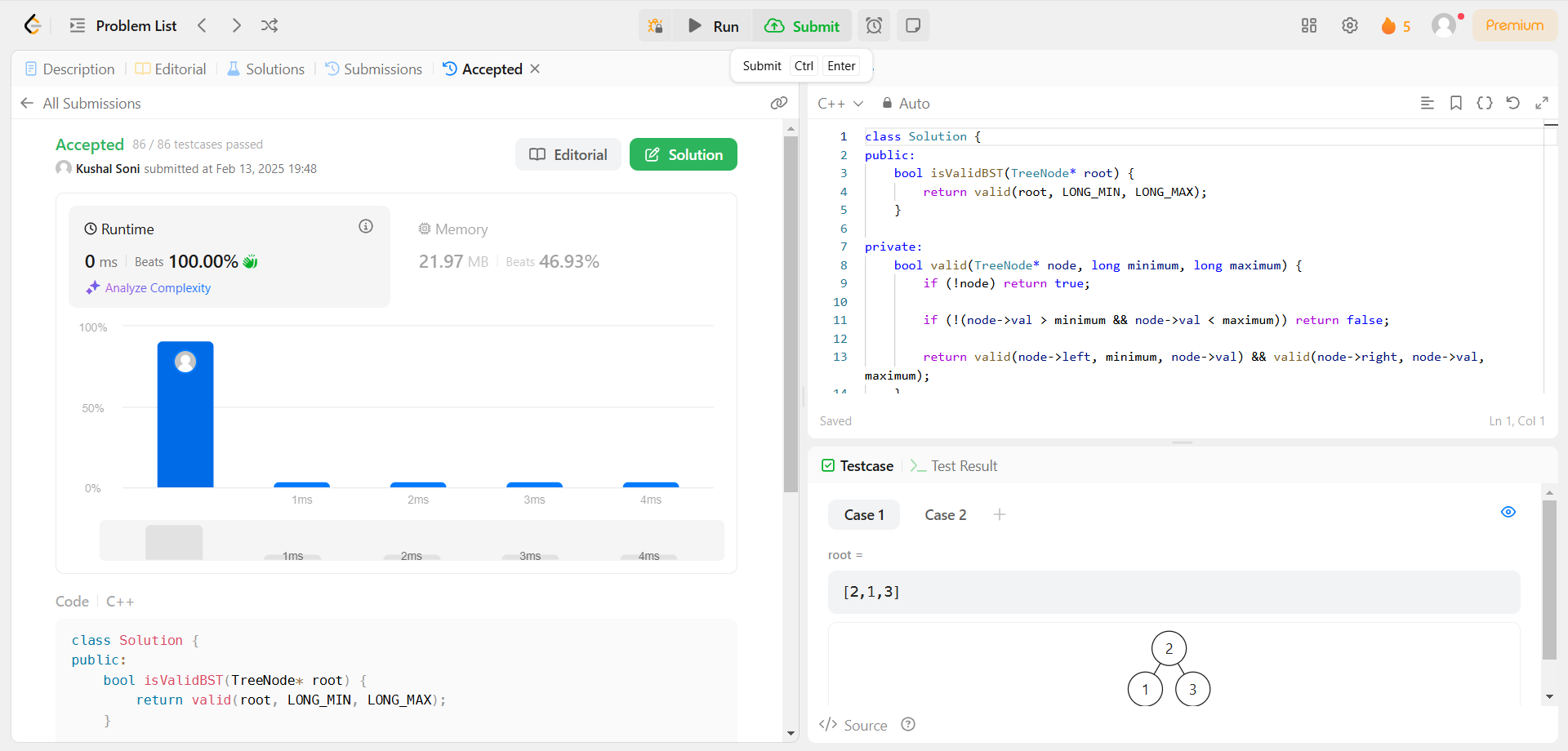
        if (!(node->val > minimum && node->val < maximum)) return false;

        return valid(node->left, minimum, node->val) && valid(node->right, node->val, maximum);

    }

};

* **Screenshot:**

****

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

* **Solution:**

class Solution {

public:

    int count = 0; // Counter for visited nodes

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

        TreeNode\* result = helper(root, k);

        return result ? result->val : 0; // Return value or 0 if not found

    }

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

        if (root == nullptr) return nullptr;

        // Traverse left subtree

        TreeNode\* left = helper(root->left, k);

        if (left != nullptr) return left; // If found in left subtree

        count++; // Increment count for current node

        if (count == k) return root; // Found k-th smallest

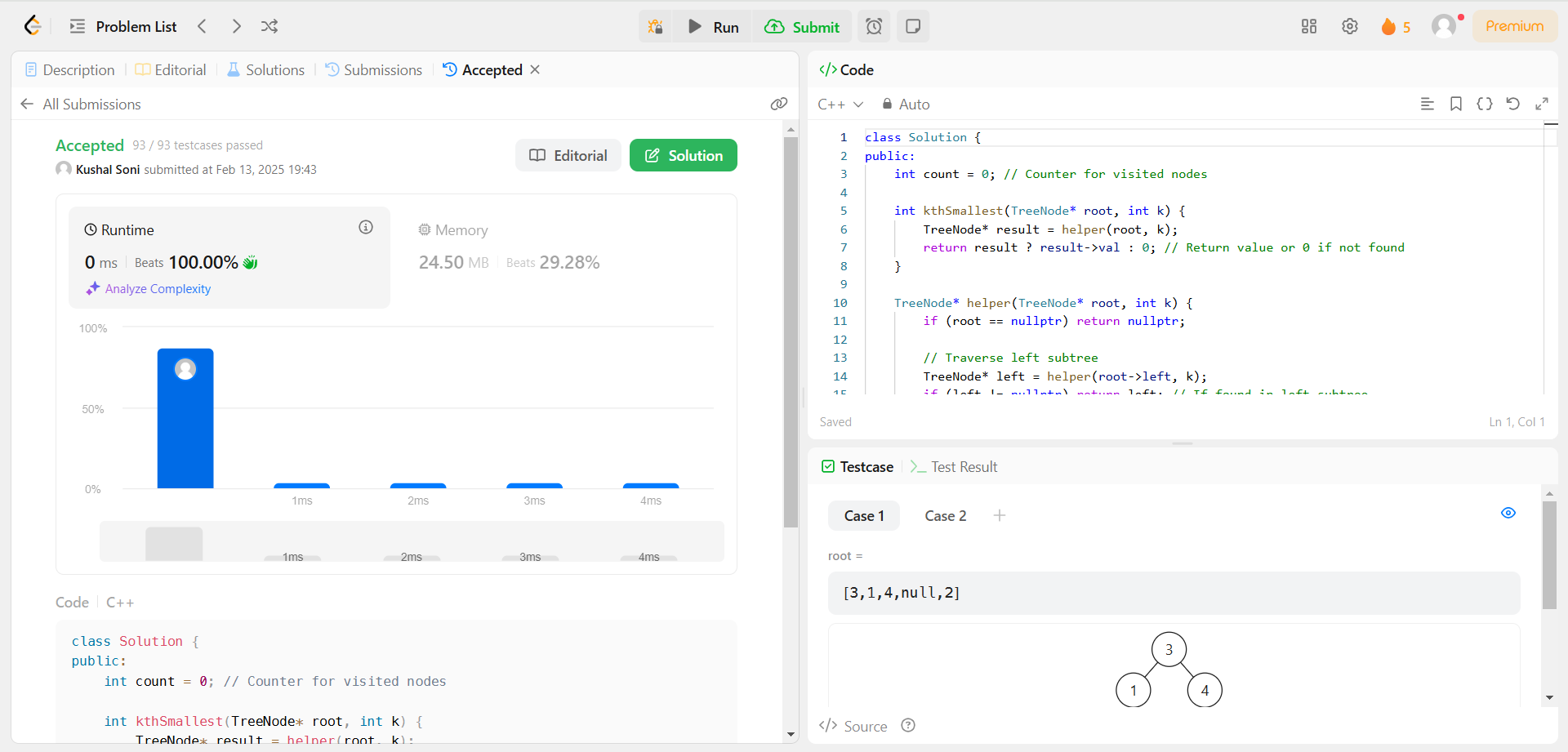
        // Traverse right subtree

        return helper(root->right, k);

    }

};

* **Screenshot:**

****

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

* **Solution:**

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.empty()){

            int s=q.size();

            vector<int>v;

            for(int i=0;i<s;i++){

                TreeNode \*node=q.front();

                q.pop();

                if(node->left!=NULL)q.push(node->left);

                if(node->right!=NULL)q.push(node->right);

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

            }

            ans.push\_back(v);

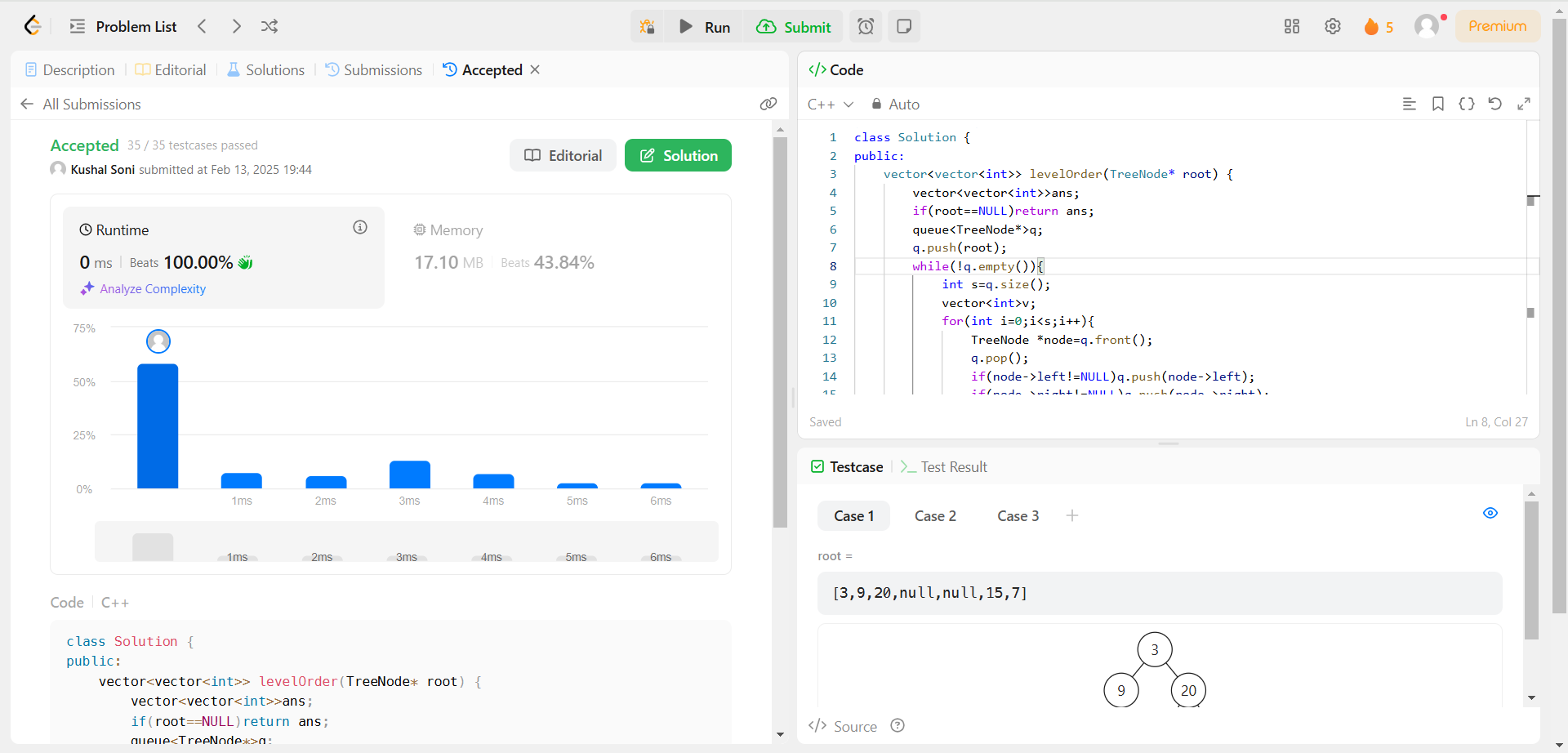
        }

        return ans;

    }

};

* **Screenshot:**

****

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

* **Solution:**

class Solution {

public:

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

        if (!root) return {};

        vector<vector<int>> 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);

        }

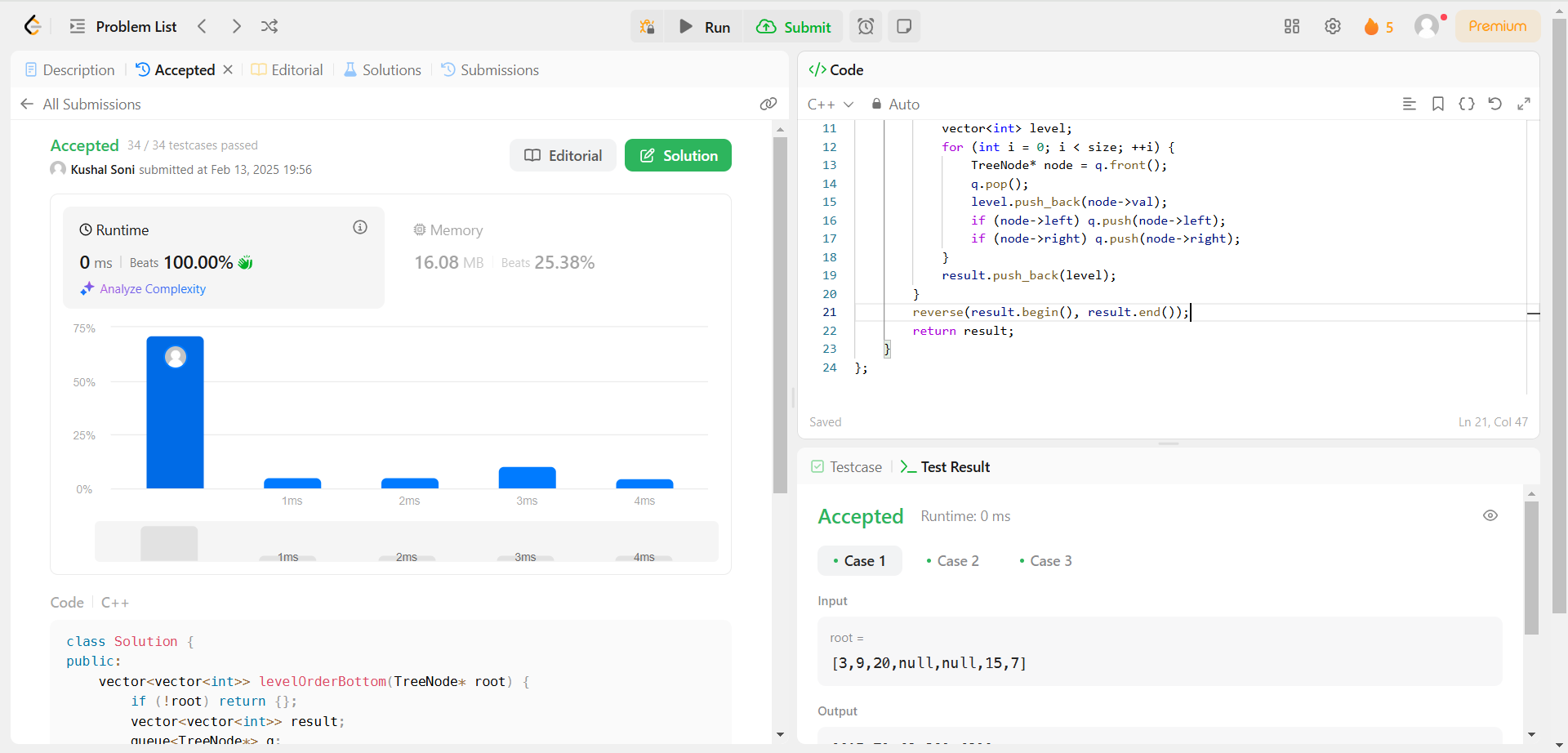
        reverse(result.begin(), result.end());

        return result;

    }

};

* **Screenshot:**

****

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

* **Solution:**

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode() : val(0), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

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

        vector<vector<int>>vec;

        if(!root){

            return {};

        }

        queue<TreeNode\*>q;

        q.push(root);

        bool flag=1;

        while(!q.empty()){

            int n=q.size();

            vector<int>ans(n);

            for(int i=0;i<n;i++){

                TreeNode\* node=q.front();

                q.pop();

                if(flag){

                    ans[i]=node->val;

                }else{

                    ans[n-1-i]=node->val;

                }

                if(node->left){

                    q.push(node->left);

                }

                if(node->right){

                    q.push(node->right);

                }

            }

            flag=!flag;

            vec.push\_back(ans);

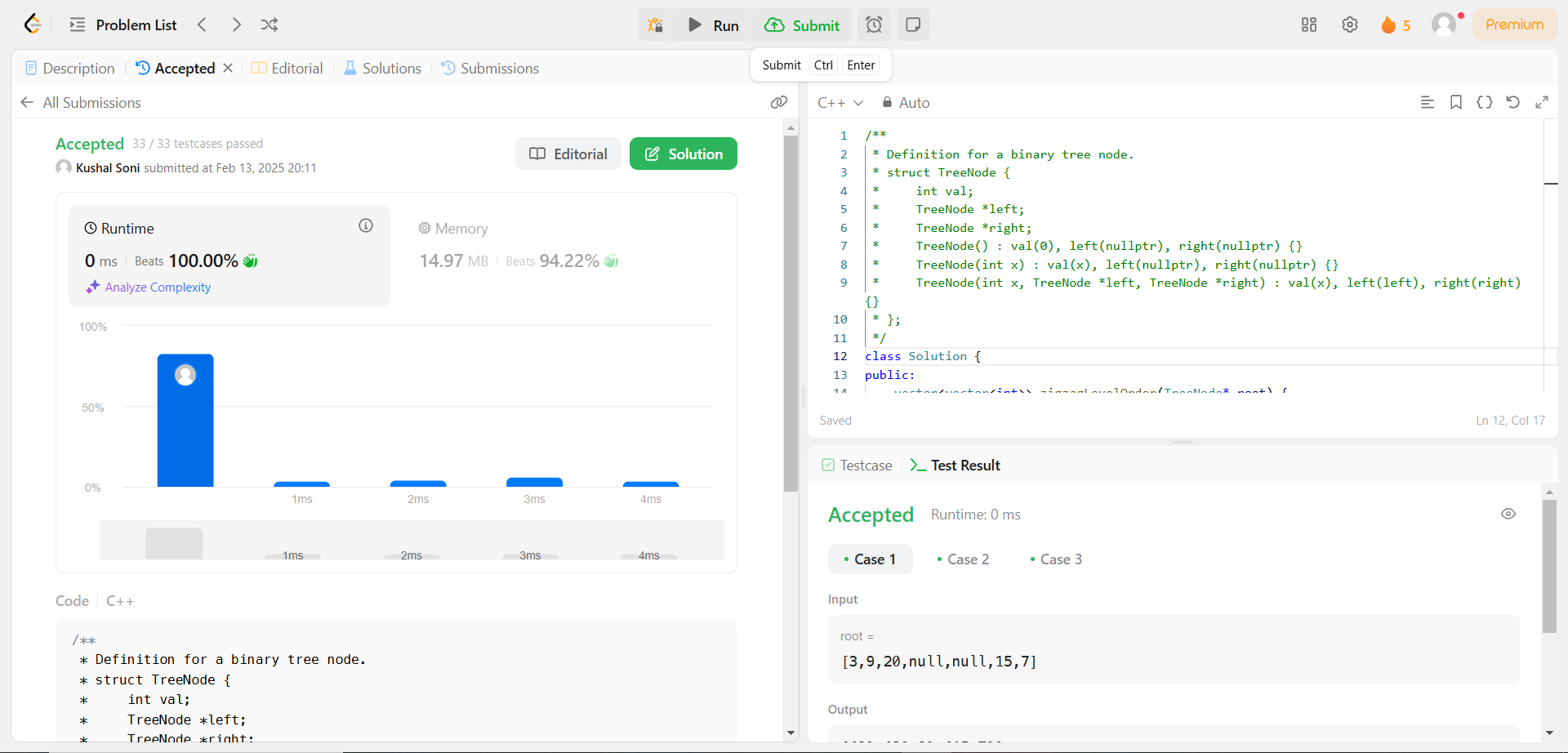
        }

        return vec;

    }

};

* **Screenshot:**

****

**199.**[**Binary Tree Right Side View**](https://leetcode.com/problems/binary-tree-right-side-view/description/)

* **Solution:**

class Solution {

 public:

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

    if (root == nullptr)

      return {};

    vector<int> ans;

    queue<TreeNode\*> q{{root}};

    while (!q.empty()) {

      const int size = q.size();

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

        TreeNode\* node = q.front();

        q.pop();

        if (i == size - 1)

          ans.push\_back(node->val);

        if (node->left)

          q.push(node->left);

        if (node->right)

          q.push(node->right);

      }

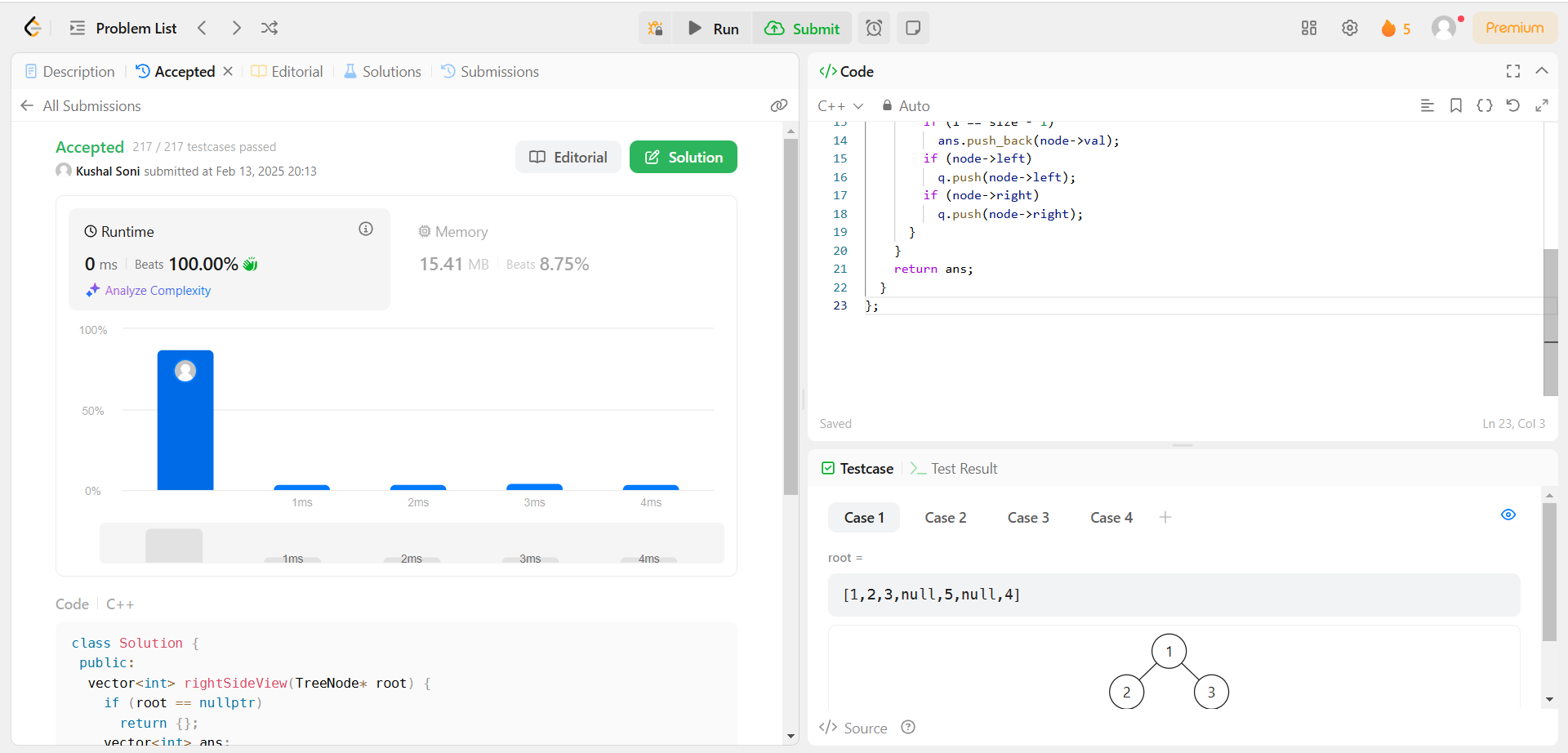
    }

    return ans;

  }

};

* **Screenshot:**

****

**106.**[**Construct Binary Tree from Inorder and Postorder Traversal**](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/description/)

* **Solution:**

class Solution {

public:

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

        unordered\_map<int, int> rec;

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

            rec[inorder[i]] = i;

        }

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

    }

    TreeNode\* helper(vector<int>& inorder, vector<int>& postorder,

                     int inStart, int inEnd,

                     int postStart, int postEnd,

                     unordered\_map<int, int>& rec) {

        if (inStart > inEnd || postStart > postEnd) return nullptr;

        int val = postorder[postEnd];

        TreeNode\* root = new TreeNode(val);

        int idx = rec[val];

        int leftSubtreeSize = idx - inStart;

        root->left = helper(inorder, postorder,

                             inStart, idx - 1,

                             postStart, postStart + leftSubtreeSize - 1,

                             rec);

        root->right = helper(inorder, postorder,

                              idx + 1, inEnd,

                              postStart + leftSubtreeSize, postEnd - 1,

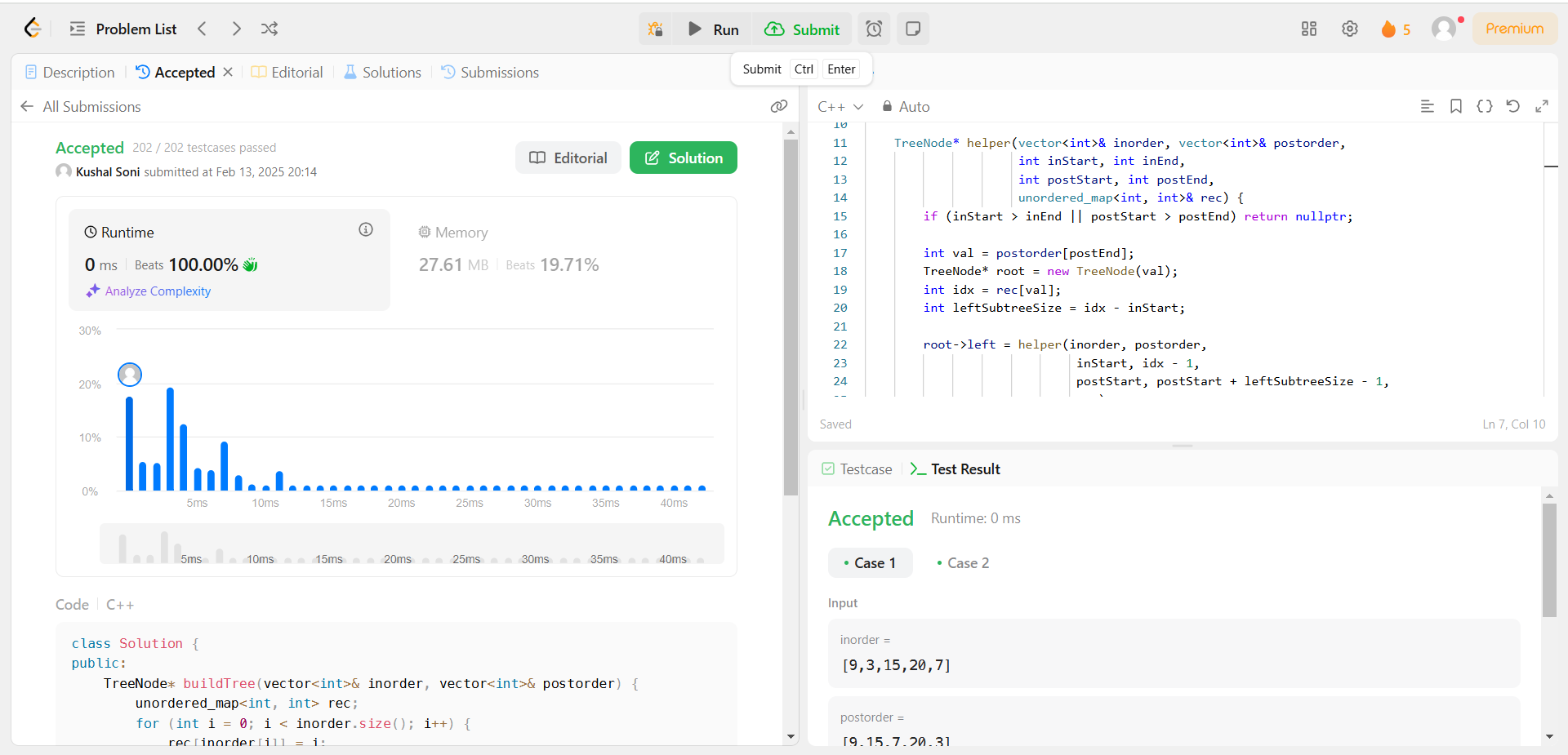
                              rec);

        return root;

    }

};

* **Screenshot:**

****

**513.**[**Find Bottom Left Tree Value**](https://leetcode.com/problems/find-bottom-left-tree-value/description/)

* **Solution:**

class Solution {

public:

    int findBottomLeftValue(TreeNode\* root) {

        queue<TreeNode\*> q;

        q.push(root);

        int leftmost\_value;

        while (!q.empty()) {

            TreeNode\* node = q.front();

            q.pop();

            leftmost\_value = node->val;

            if (node->right) {

                q.push(node->right);

            }

            if (node->left) {

                q.push(node->left);

            }

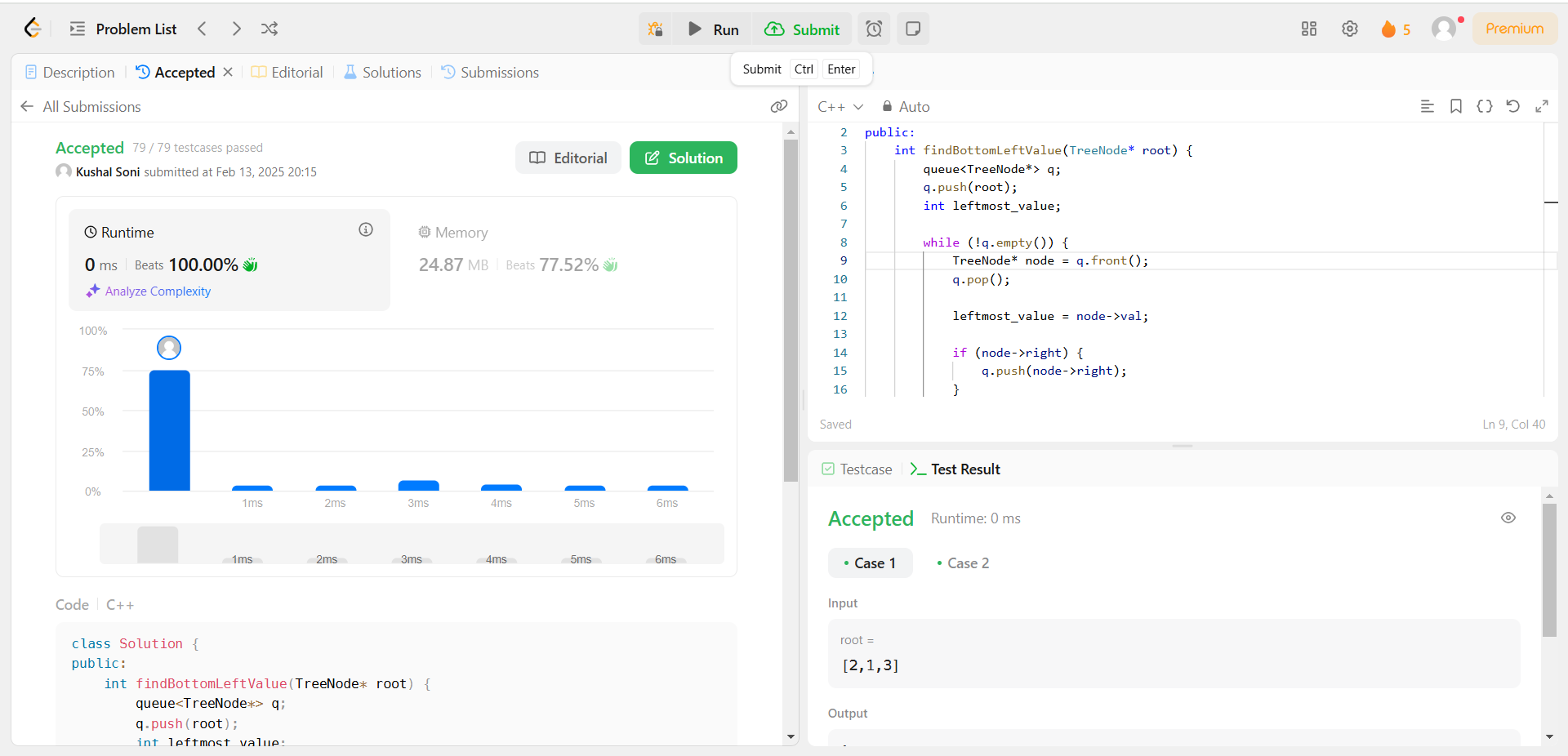
        }

        return leftmost\_value;

    }

};

* **Screenshot:**

****

**124.**[**Binary Tree Maximum Path Sum**](https://leetcode.com/problems/binary-tree-maximum-path-sum/description/)

* **Solution:**

class Solution {

public:

    int ans = INT\_MIN;

    int maxPathSum(TreeNode\* root) {

        helper(root);

        return ans;

    }

private:

    int helper(TreeNode\* root) {

        if (!root) return 0;

        int left = max(0, helper(root->left));

        int right = max(0, helper(root->right));

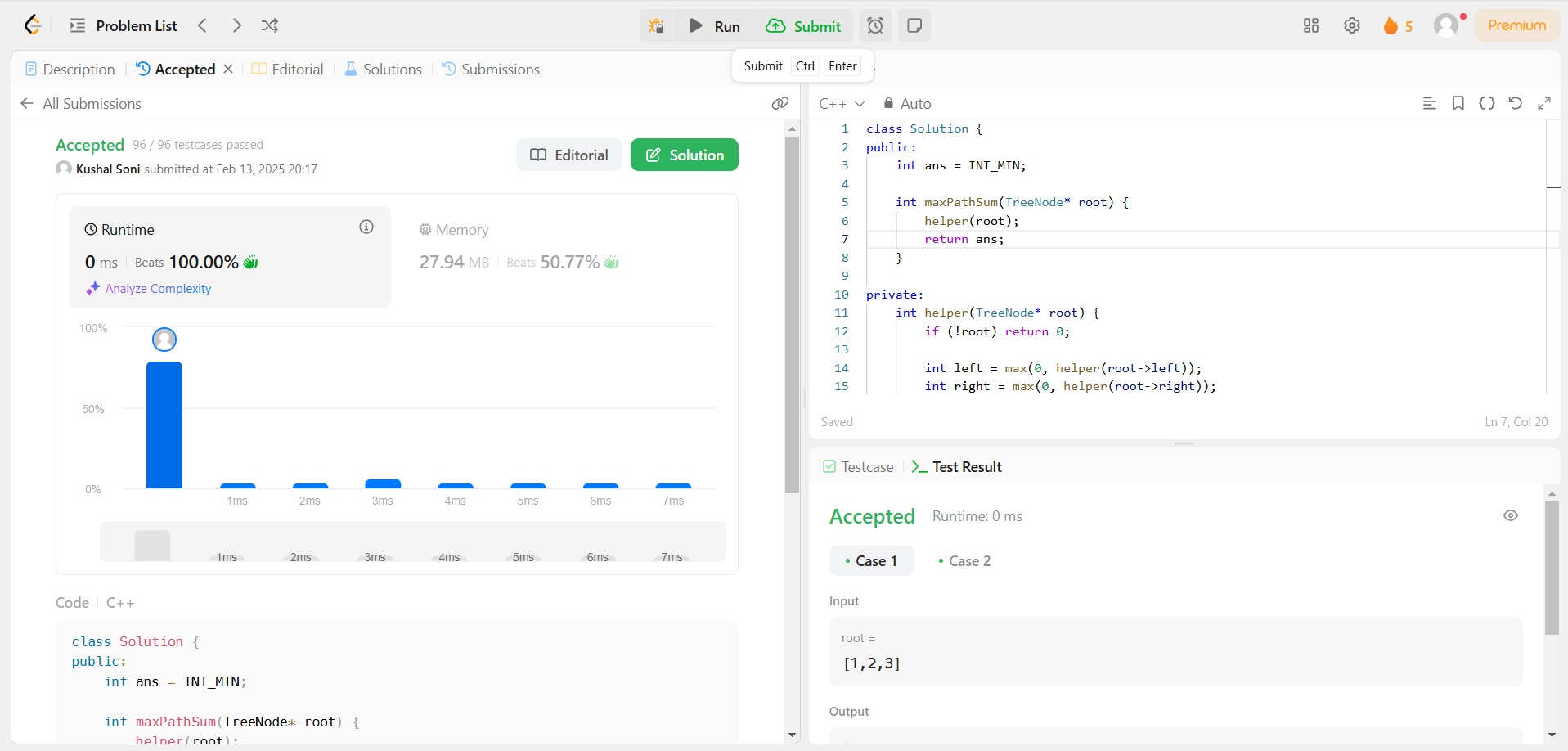
        ans = max(ans, root->val + left + right);

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

    }

};

* **Screenshot:**

****

**987.**[**Vertical Order Traversal of a Binary Tree**](https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/description/)

* **Solution:**

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode() : val(0), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

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

        vector<vector<int>> out;

        map<int, vector<int>> final\_mp;

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

        q.push({0, root});

        while (!q.empty()) {

            int n = q.size();

            map<int, vector<int>> mp;

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

                auto it = q.front();

                int index = it.first;

                TreeNode\* node = it.second;

                mp[index].push\_back(node->val);

                q.pop();

                if (node->left != NULL)

                    q.push({index - 1, node->left});

                if (node->right != NULL)

                    q.push({index + 1, node->right});

            }

            for (auto it : mp) {

                sort(it.second.begin(), it.second.end());

                vector<int> temp = final\_mp[it.first];

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

                    temp.push\_back(it.second[i]);

                }

                final\_mp[it.first] = temp;

            }

        }

        for (auto it : final\_mp) {

            out.push\_back(it.second);

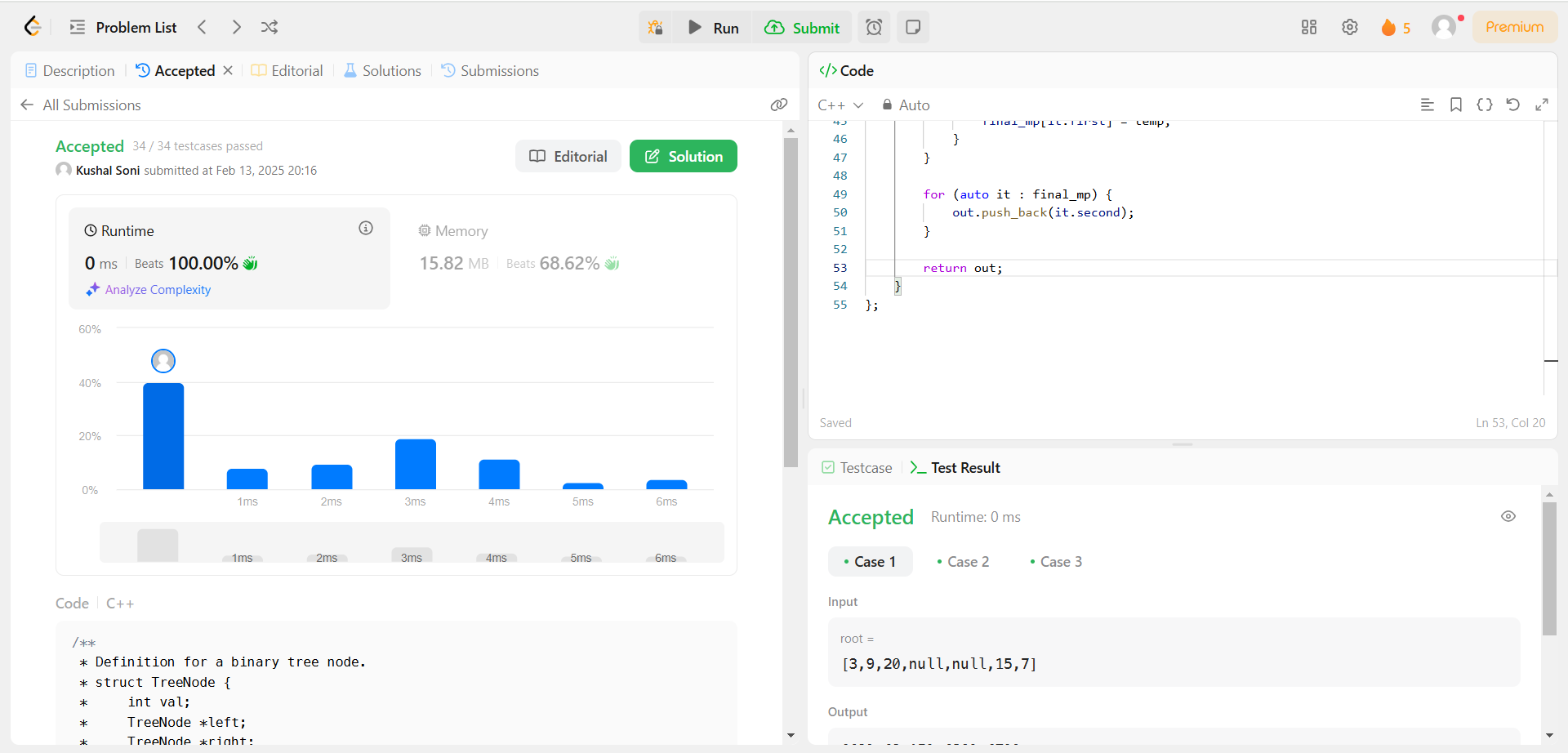
        }

        return out;

    }

};

* **Screenshot:**

****