94. Binary Tree Inorder Traversal

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;

        }

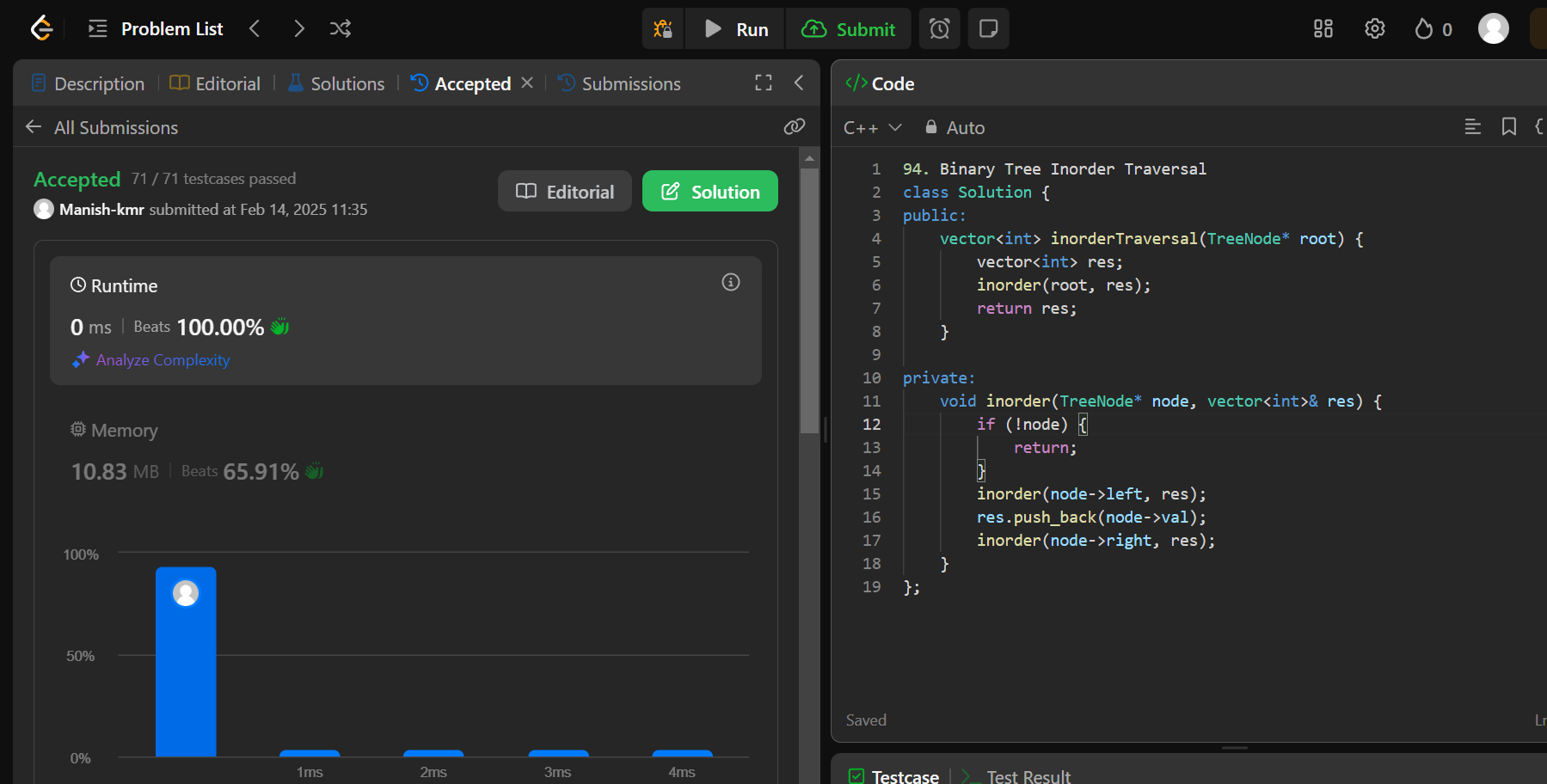
        inorder(node->left, res);

        res.push\_back(node->val);

        inorder(node->right, res);

    }

};

// 101. Symmetric Tree

class Solution {

public:

    bool isMirror(TreeNode\* left, TreeNode\* right) {

    if (!left && !right) return true;

    if (!left || !right) return false;

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

}

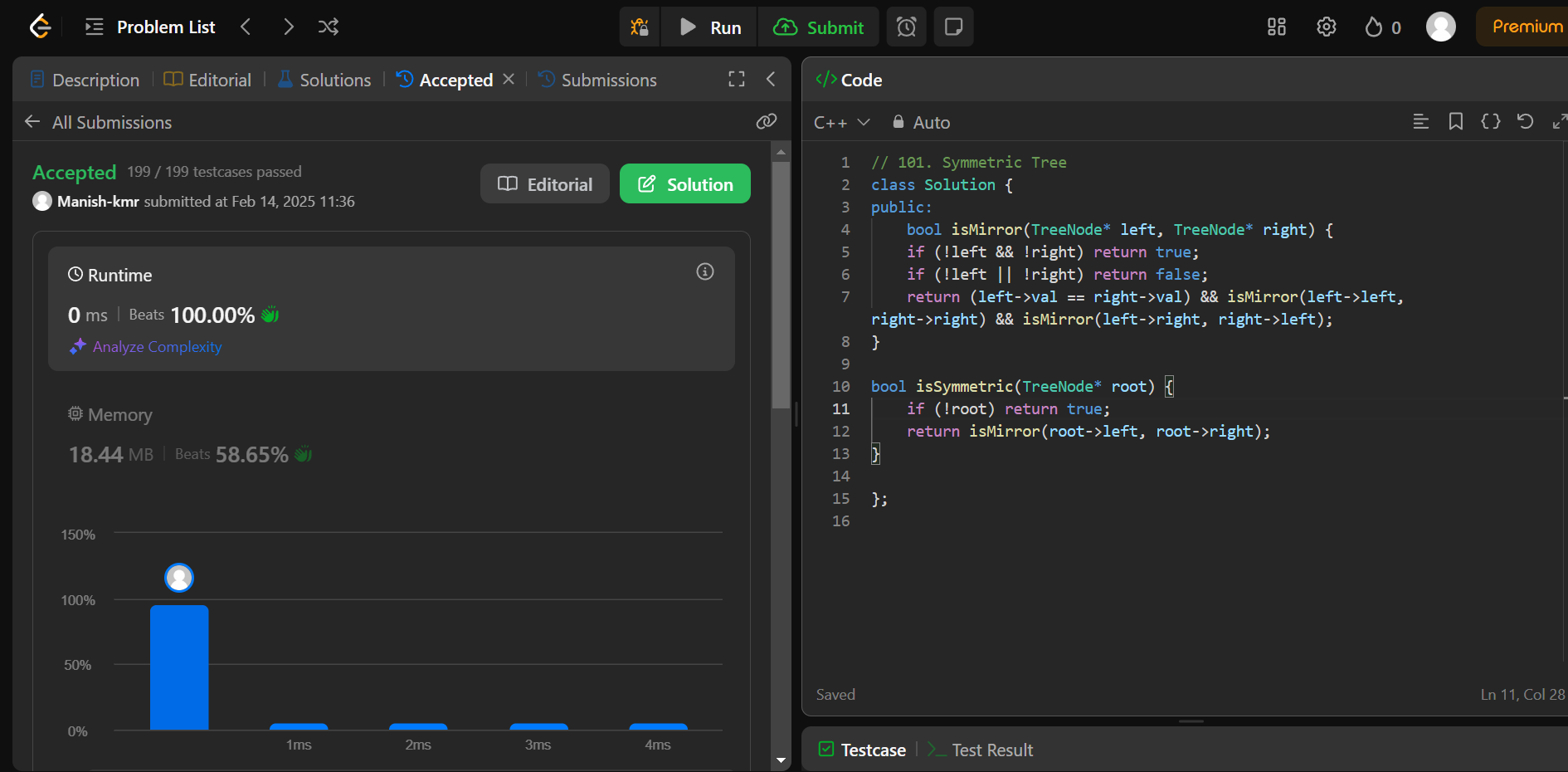
bool isSymmetric(TreeNode\* root) {

    if (!root) return true;

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

}

};

// 104. Maximum Depth of Binary Tree

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if (!root) {

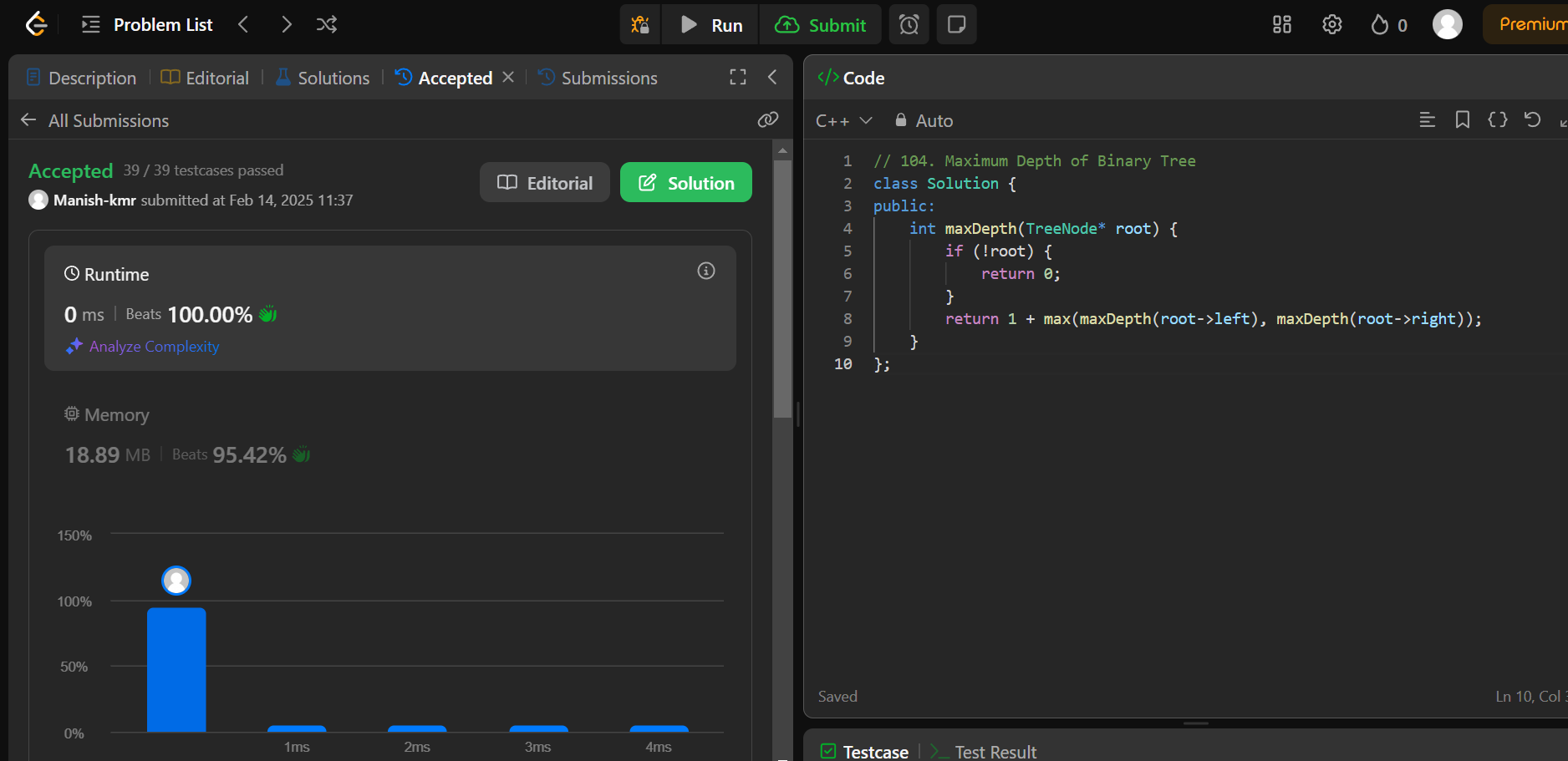
            return 0;

        }

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

    }

};

// 98. Validate Binary Search Tree

class Solution {

public:

    bool isValidBST(TreeNode\* root) {

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

    }

private:

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

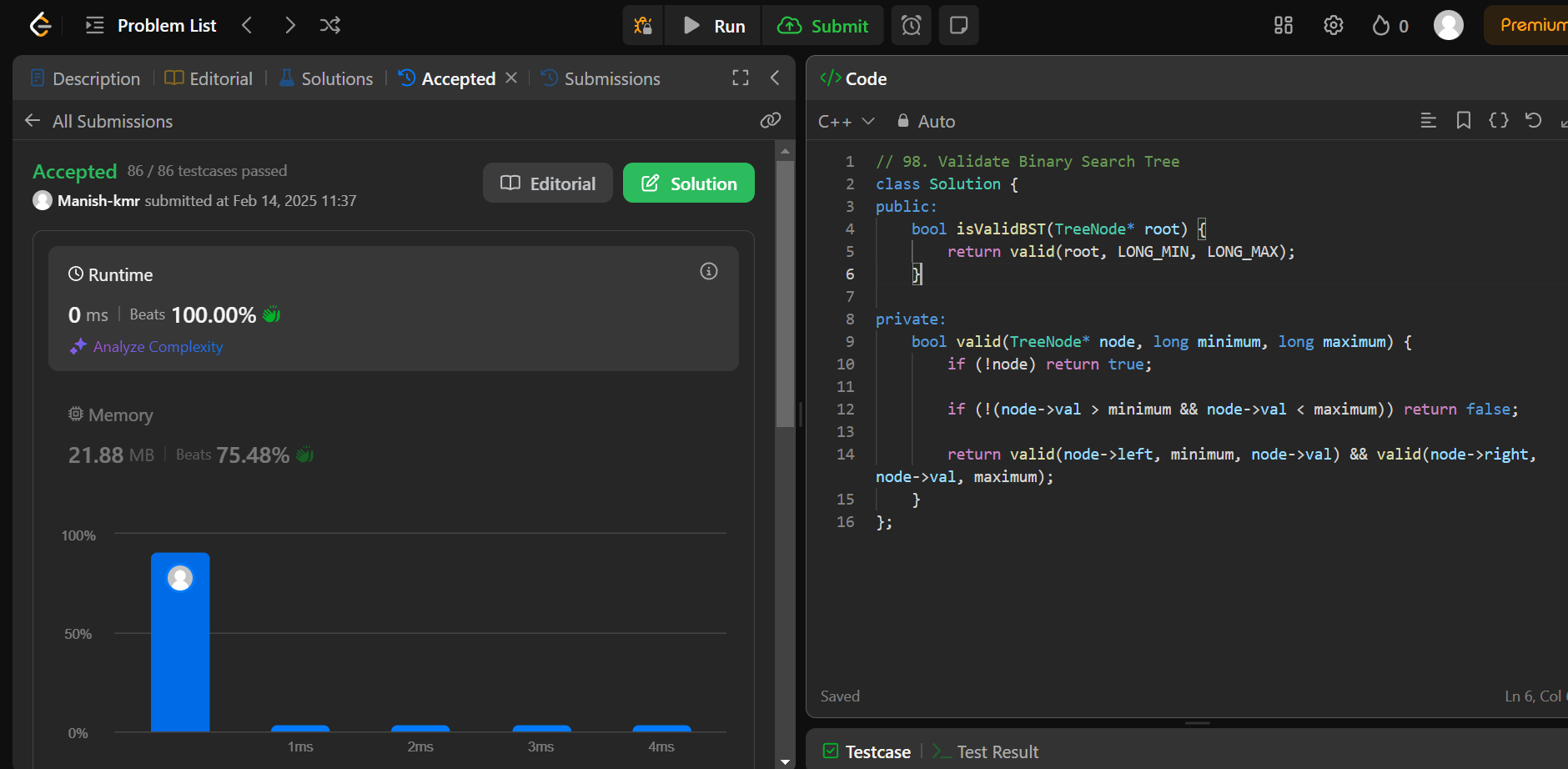
        if (!node) return true;

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

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

    }

};

// 230. Kth Smallest Element in a BST

/\*\*

 \* 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:

    pair<int,int> kthSmall(TreeNode\* root,int k){

        if(root == NULL)    return make\_pair(-1,0);

        pair<int,int> pr = kthSmall(root->left,k);

        if(pr.first!=-1)    return pr;

        else k -= pr.second;

        if(k==1)    {

            pr.first = root->val;

            return pr;

        }

        pair<int,int> p2 = kthSmall(root->right,k-1);

        p2.second += 1 + pr.second;

        return p2;

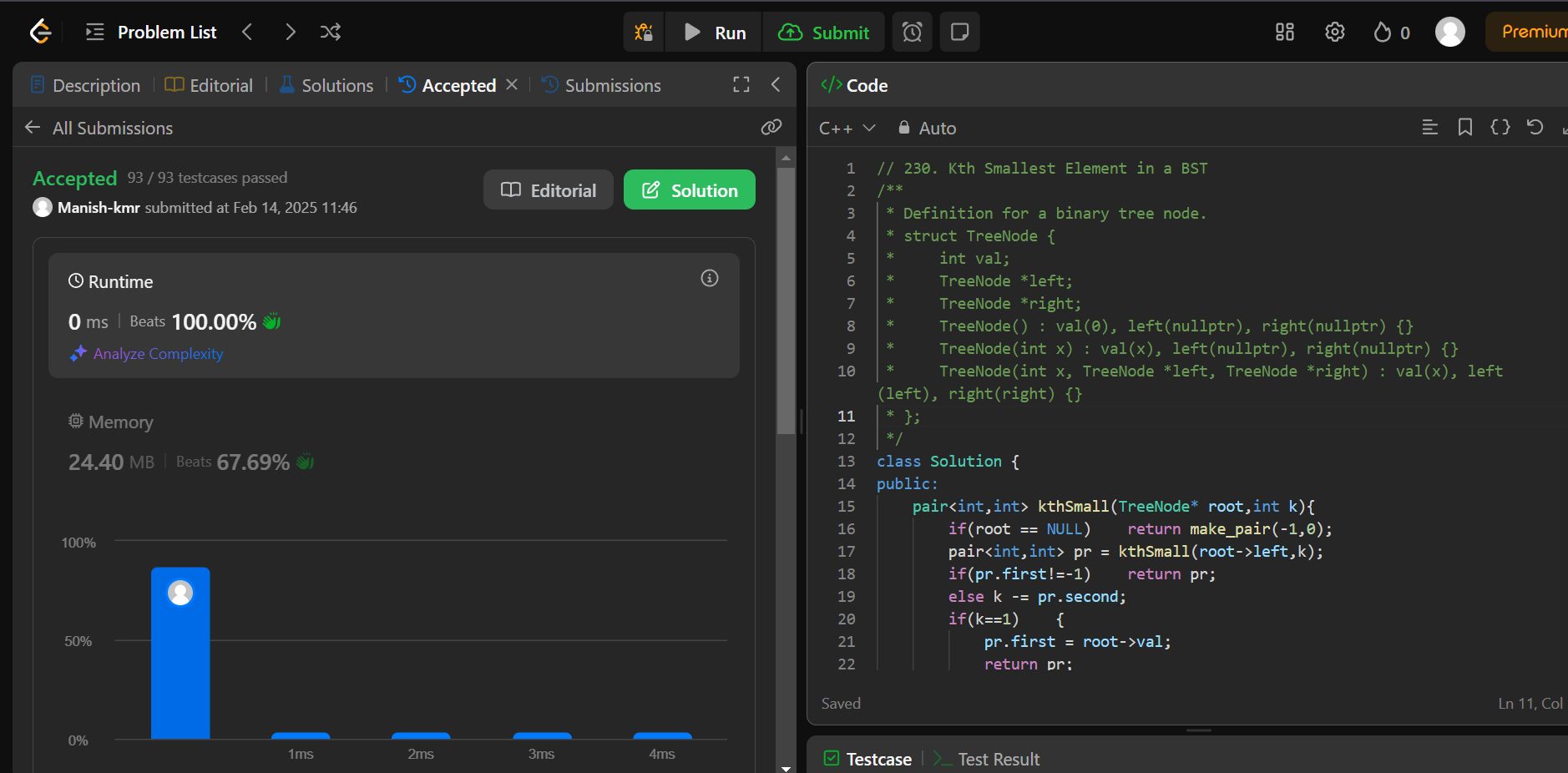
    }

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

        return kthSmall(root,k).first;

    }

};

// 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);

      }

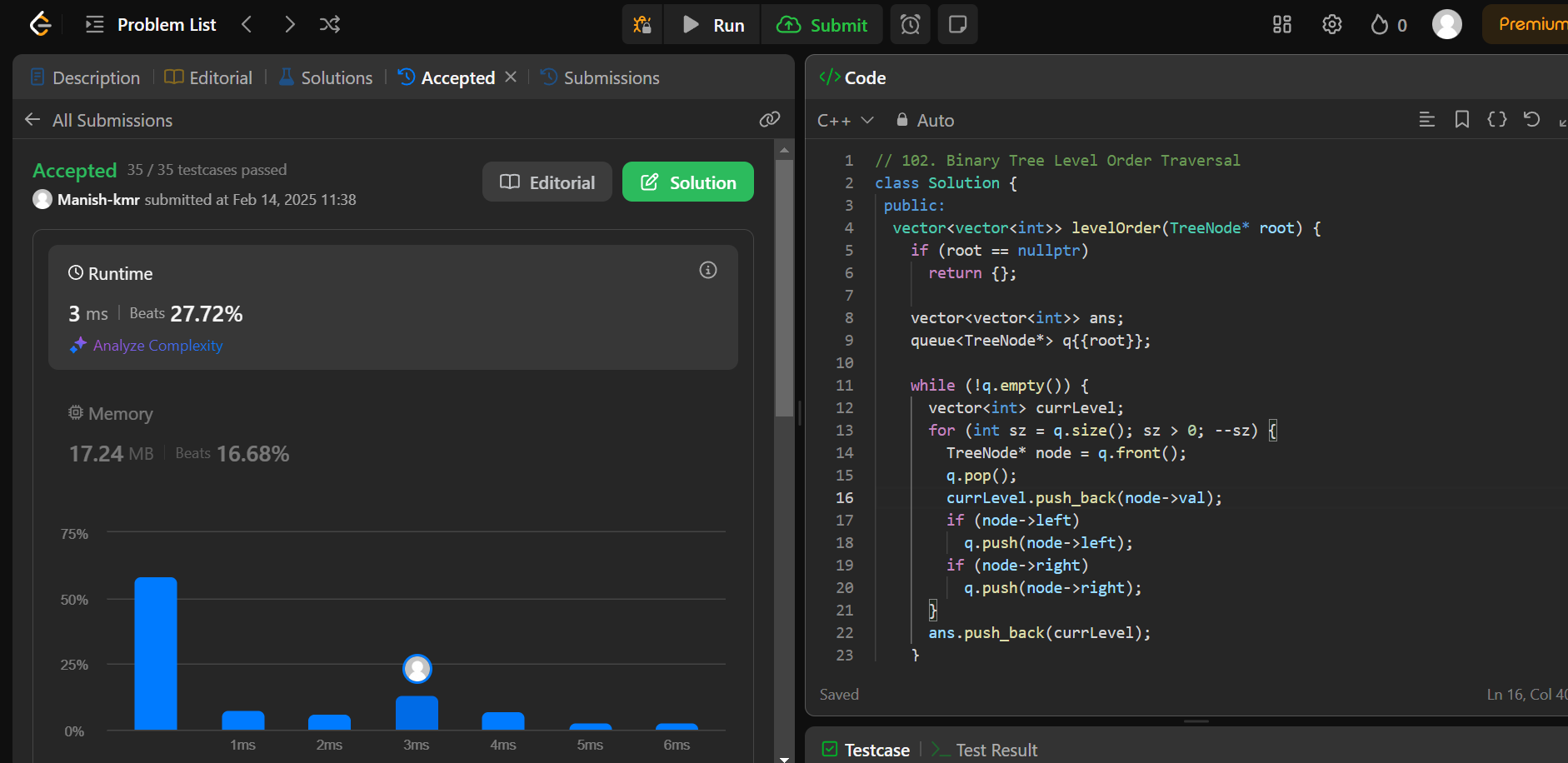
      ans.push\_back(currLevel);

    }

    return ans;

  }

};

// 107. Binary Tree Level Order Traversal II

class Solution {

public:

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

        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);

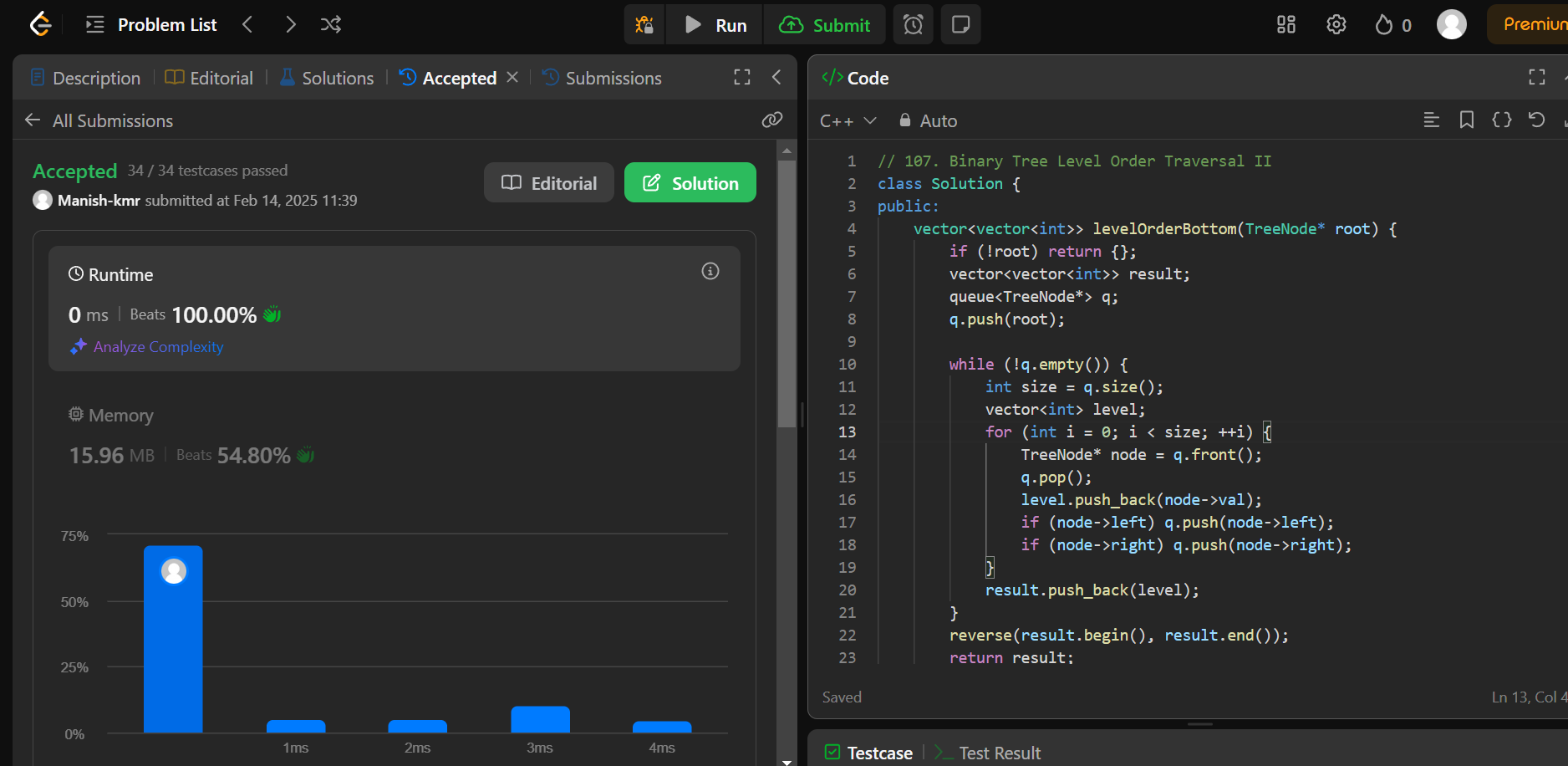
        }

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

        return result;

    }

};

// 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);

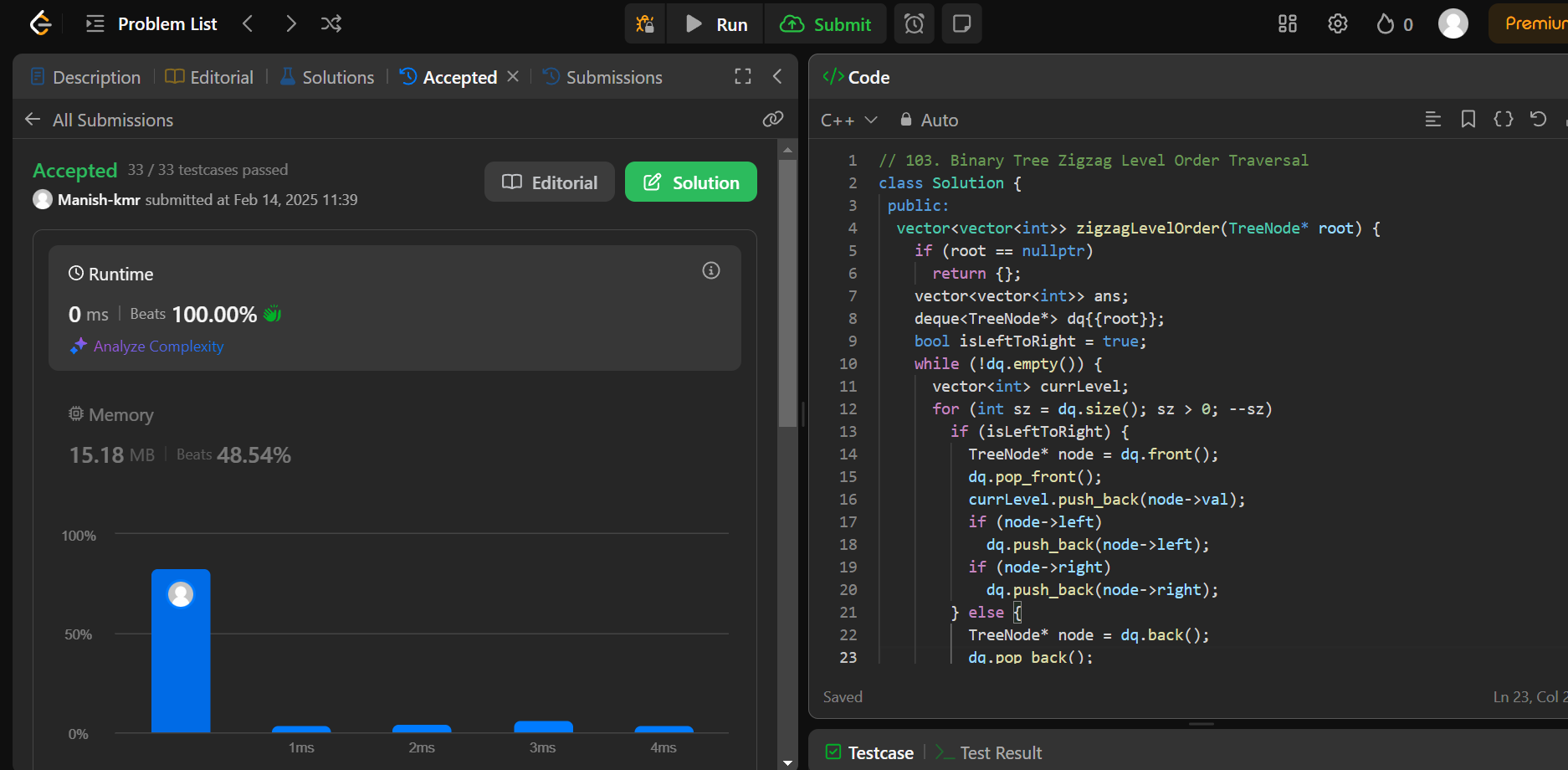
      isLeftToRight = !isLeftToRight;

    }

    return ans;

  }

};

// 199. Binary Tree Right Side View

/\*\*

 \* 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<int> res;

    unordered\_map<int,int> mp;

    void check(TreeNode\* root,int n){

        if(!root){

            return;

        }

        if(!(mp.find(n) != mp.end())){

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

            mp[n]++;

        }

        check(root->right,n+1);

        check(root->left,n+1);

    }

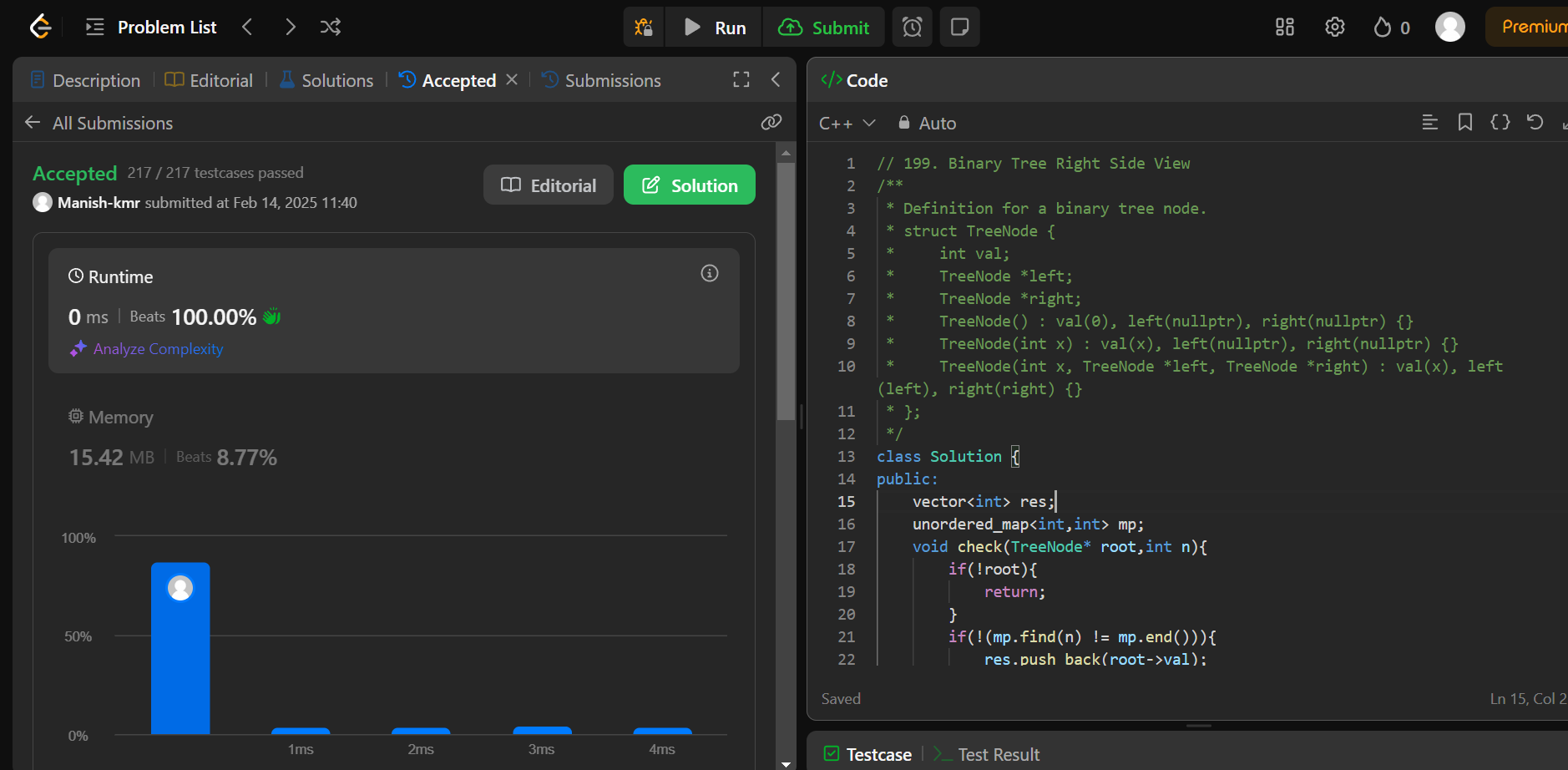
    vector<int> rightSideView(TreeNode\* root) {

      check(root,0);

      return res;

    }

};

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

/\*\*

 \* 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:

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

    unordered\_map<int, int> inToIndex;

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

      inToIndex[inorder[i]] = i;

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

  }

 private:

  TreeNode\* build(const vector<int>& inorder, int inStart, int inEnd, const vector<int>& postorder, int postStart, int postEnd, const unordered\_map<int, int>& inToIndex) {

    if (inStart > inEnd)

      return nullptr;

    const int rootVal = postorder[postEnd];

    const int rootInIndex = inToIndex.at(rootVal);

    const int leftSize = rootInIndex - inStart;

    TreeNode\* root = new TreeNode(rootVal);

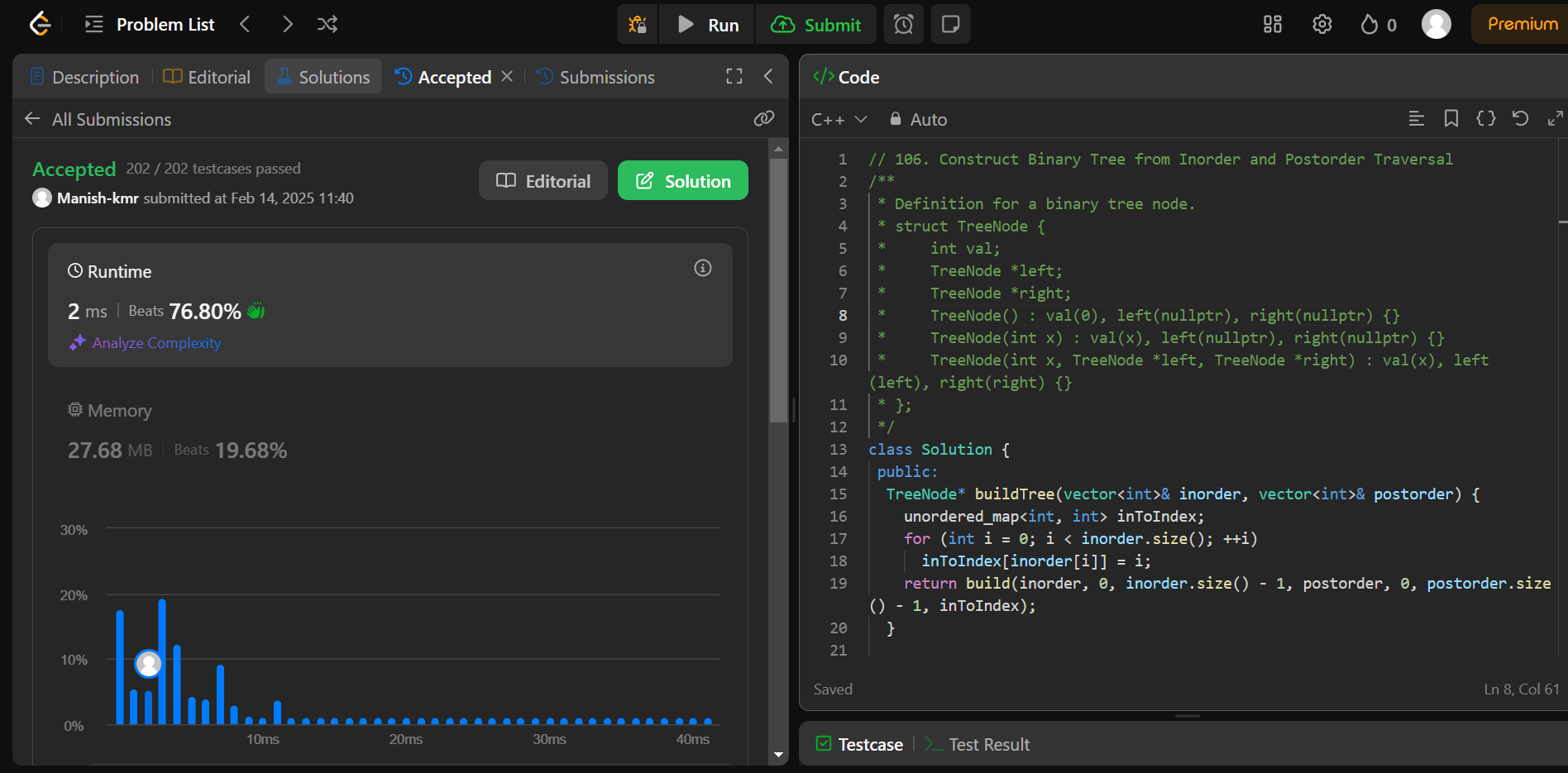
    root->left = build(inorder, inStart, rootInIndex - 1, postorder, postStart, postStart + leftSize - 1, inToIndex);

    root->right = build(inorder, rootInIndex + 1, inEnd, postorder, postStart + leftSize, postEnd - 1, inToIndex);

    return root;

  }

};

// 513. Find Bottom Left Tree Value

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);

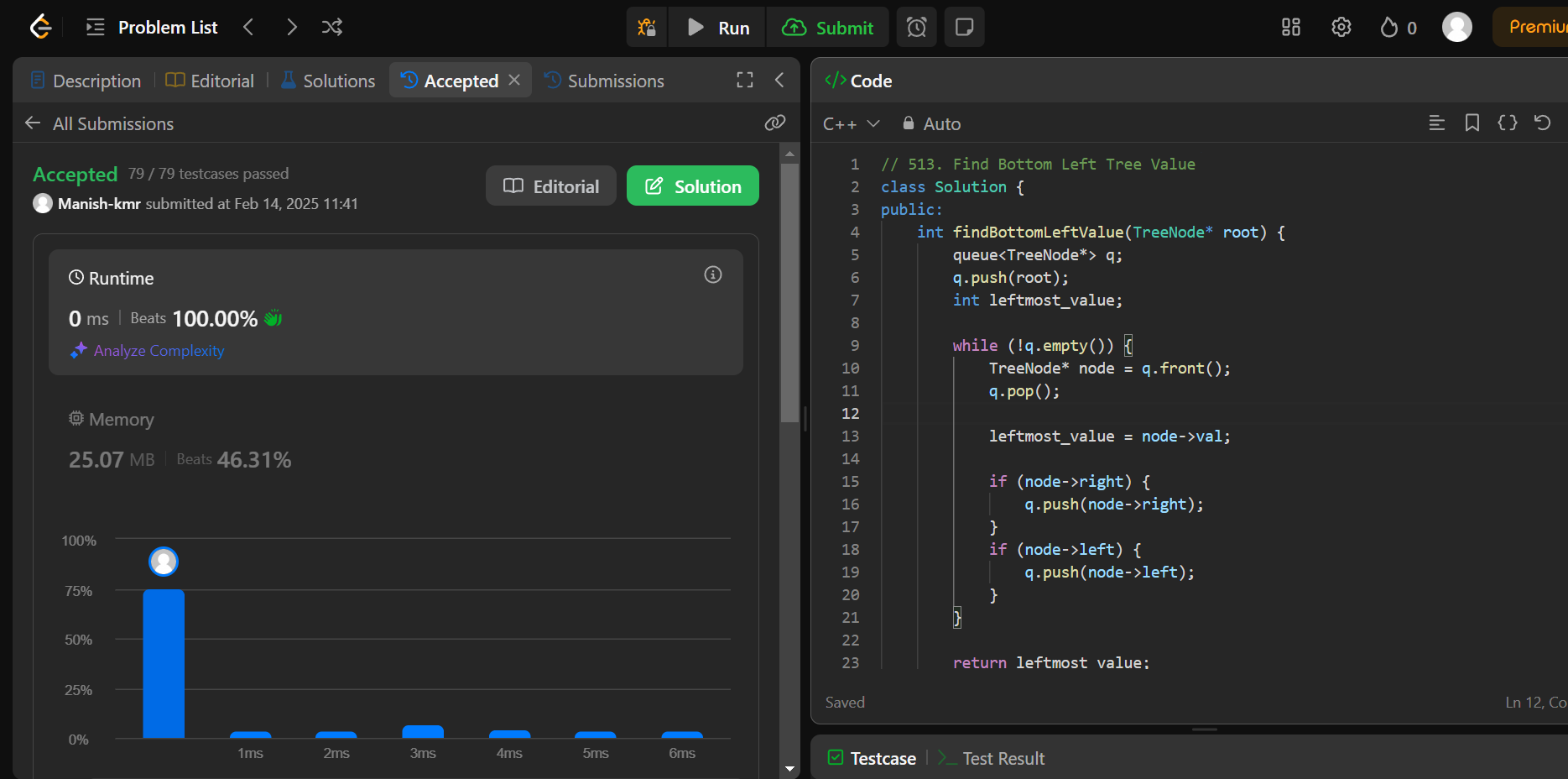
            }

        }

        return leftmost\_value;

    }

};

// 124. Binary Tree Maximum Path Sum

class Solution {

 public:

  int maxPathSum(TreeNode\* root) {

    int ans = INT\_MIN;

    maxPathSumDownFrom(root, ans);

    return ans;

  }

 private:

  int maxPathSumDownFrom(TreeNode\* root, int& ans) {

    if (root == nullptr)

      return 0;

    const int l = max(0, maxPathSumDownFrom(root->left, ans));

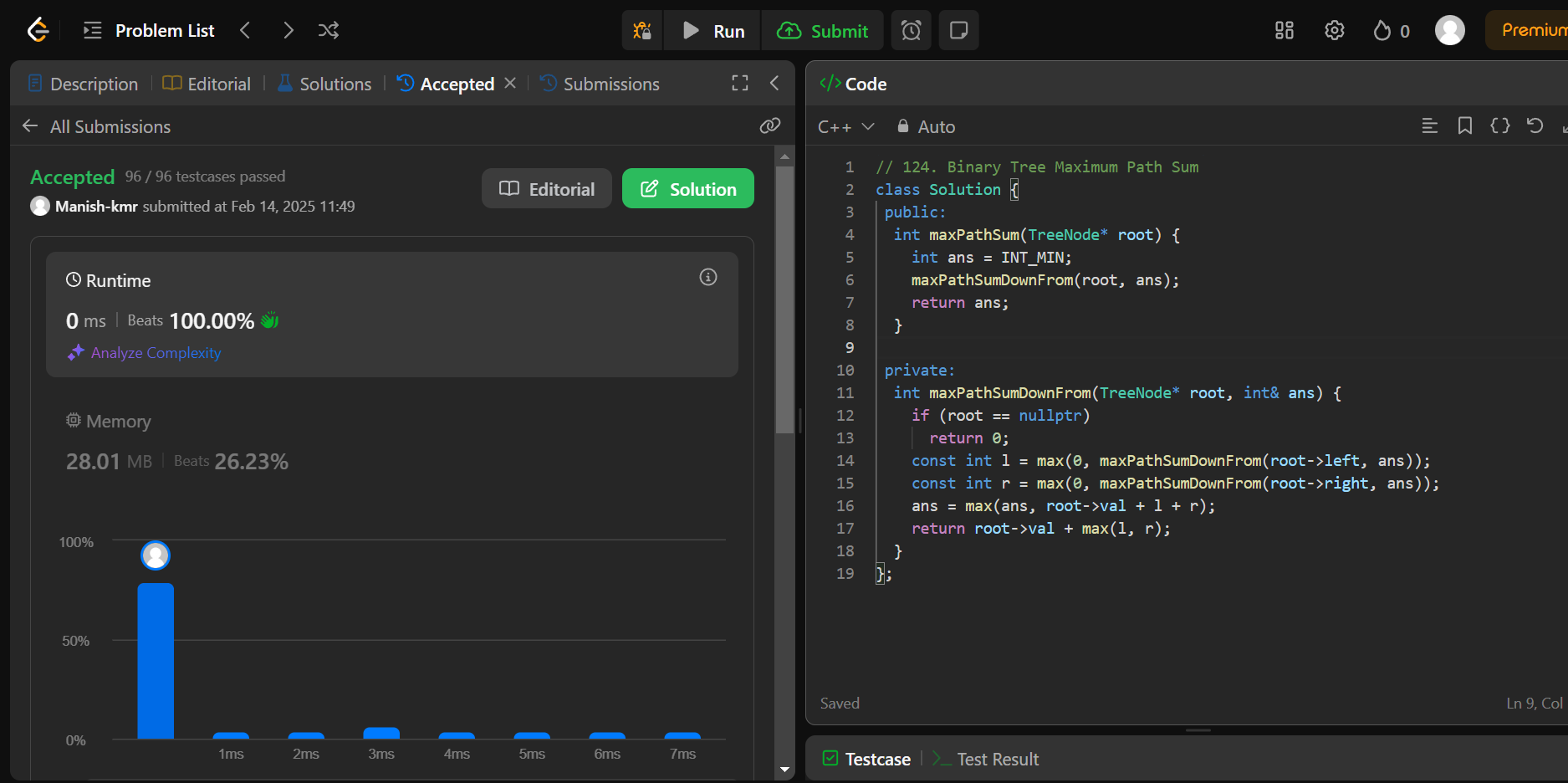
    const int r = max(0, maxPathSumDownFrom(root->right, ans));

    ans = max(ans, root->val + l + r);

    return root->val + max(l, r);

  }

};

// 987. Vertical Order Traversal of a Binary Tree

/\*\*

 \* 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>> ans;

        queue<pair<TreeNode\*,int>> Q; // node and col

        Q.push({root,0});

        int depth=0;

        while(!Q.empty()){

            int s=Q.size();

            while(s--){

                auto [node,col]=Q.front();

                Q.pop();

                ans.push\_back({col,depth,node->val});

                if(node->left!=nullptr) Q.push({node->left,col-1});

                if(node->right!=nullptr) Q.push({node->right,col+1});

            }

            depth++;

        }

        sort(ans.begin(),ans.end());

        vector<vector<int>> final;

        vector<int> temp;

        int curr=ans[0][0];

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

            if(ans[i][0]==curr) temp.push\_back(ans[i][2]);

            else{

                final.push\_back(temp);

                temp.clear();

                curr=ans[i][0];

                temp.push\_back(ans[i][2]);

            }

        }

        if(!temp.empty()) final.push\_back(temp);

        return final;

    }

};

