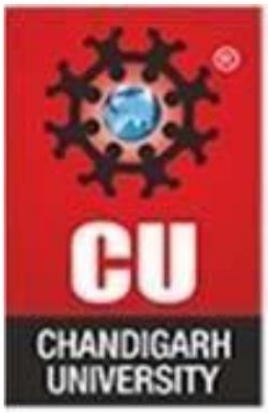


CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

AP ASSIGNMENT 3

ALIZA ASIF

22BCS50175

22BCS_FL_IOT_601_A

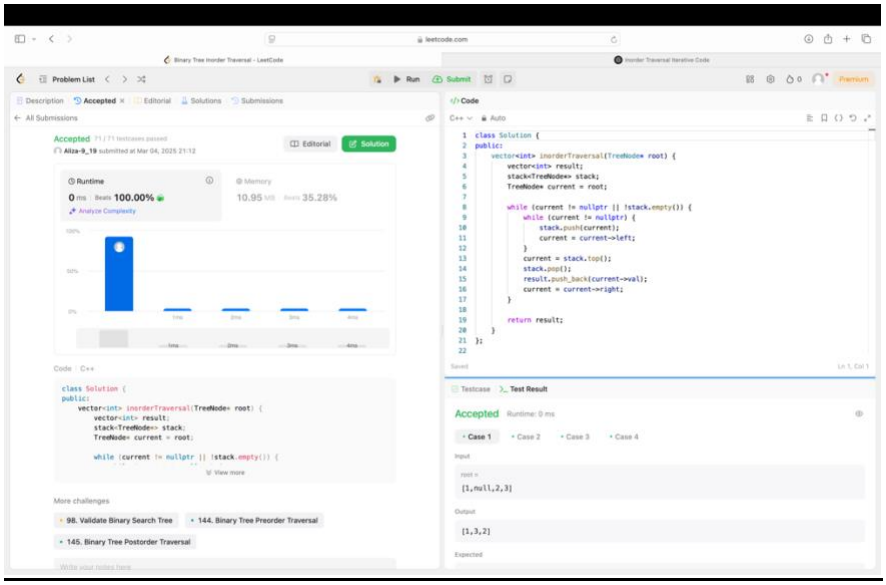
AP ASSIGNMENT 3

Q1. Binary Tree Inorder Traversal (94)

Implementation Code:

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        stack<TreeNode*> stack;
        TreeNode* current = root;

        while (current != nullptr || !stack.empty()) {
            while (current != nullptr) {
                stack.push(current);
                current = current->left;
            }
            current = stack.top();
            stack.pop();
            result.push_back(current->val);
            current = current->right;
        }

        return result;
    }
};
```

Output:



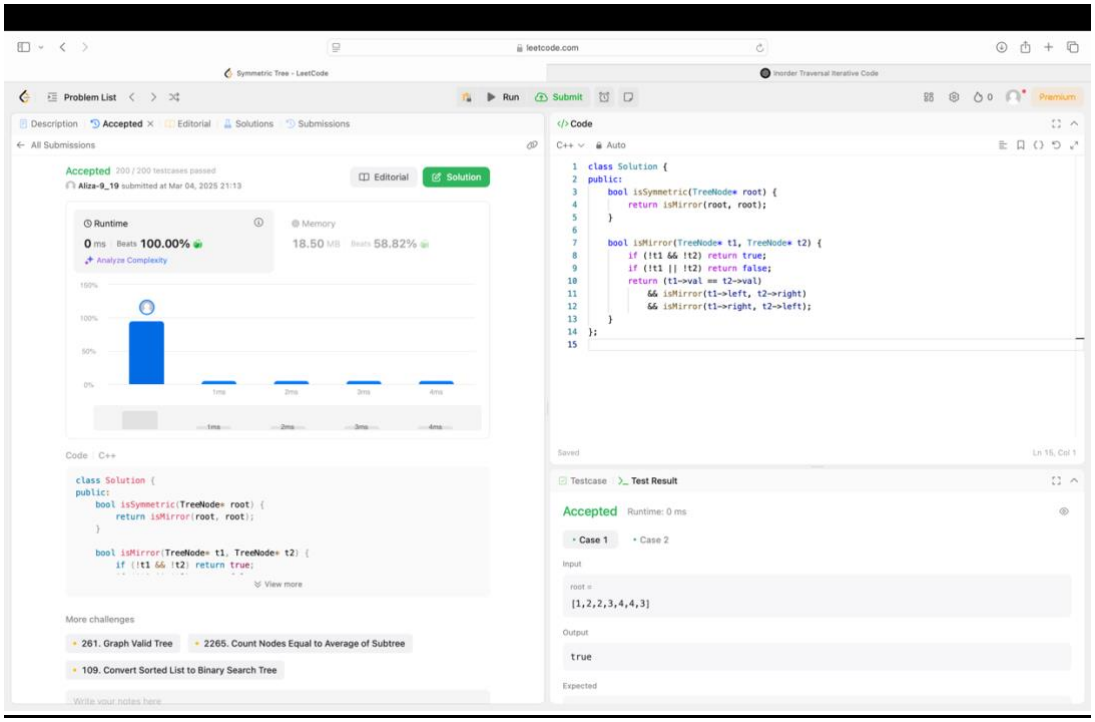
Q2. Symmetric Tree (101)

Implementation Code:

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return isMirror(root, root);
    }

    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);
    }
```

};

Output:



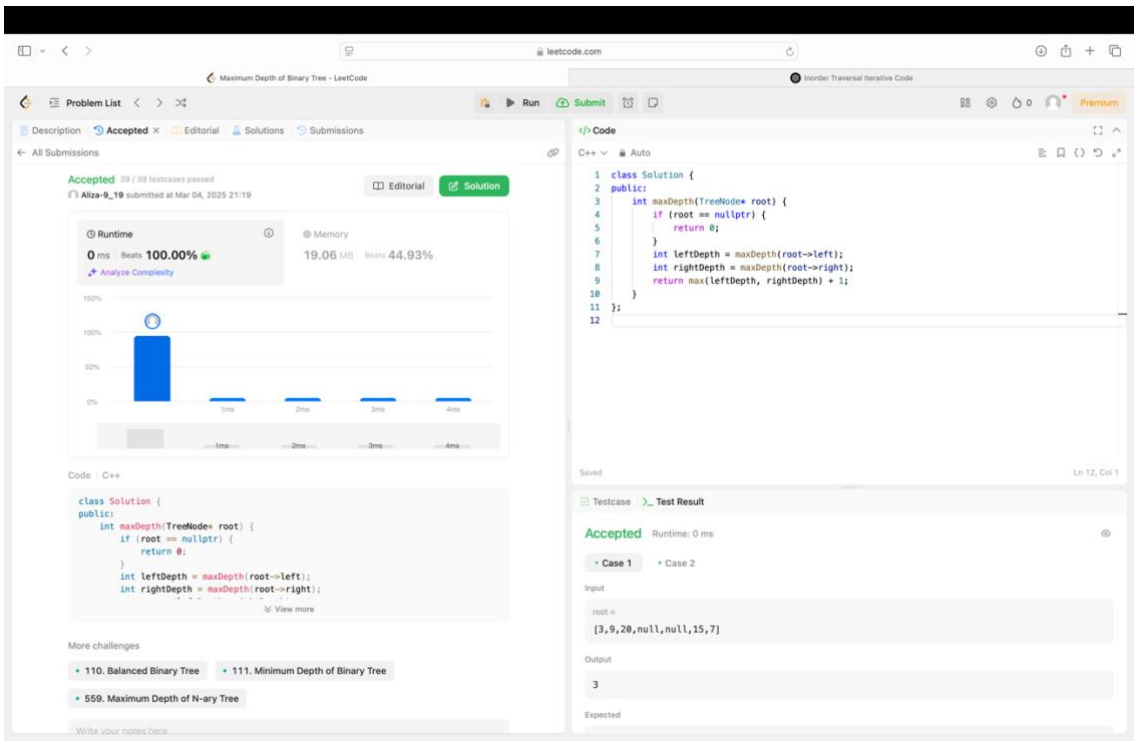
Q3. Maximum Depth of Binary Tree (104)

Implementation Code:

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

```
        return max(leftDepth, rightDepth) + 1;
    }
};
```

Output:



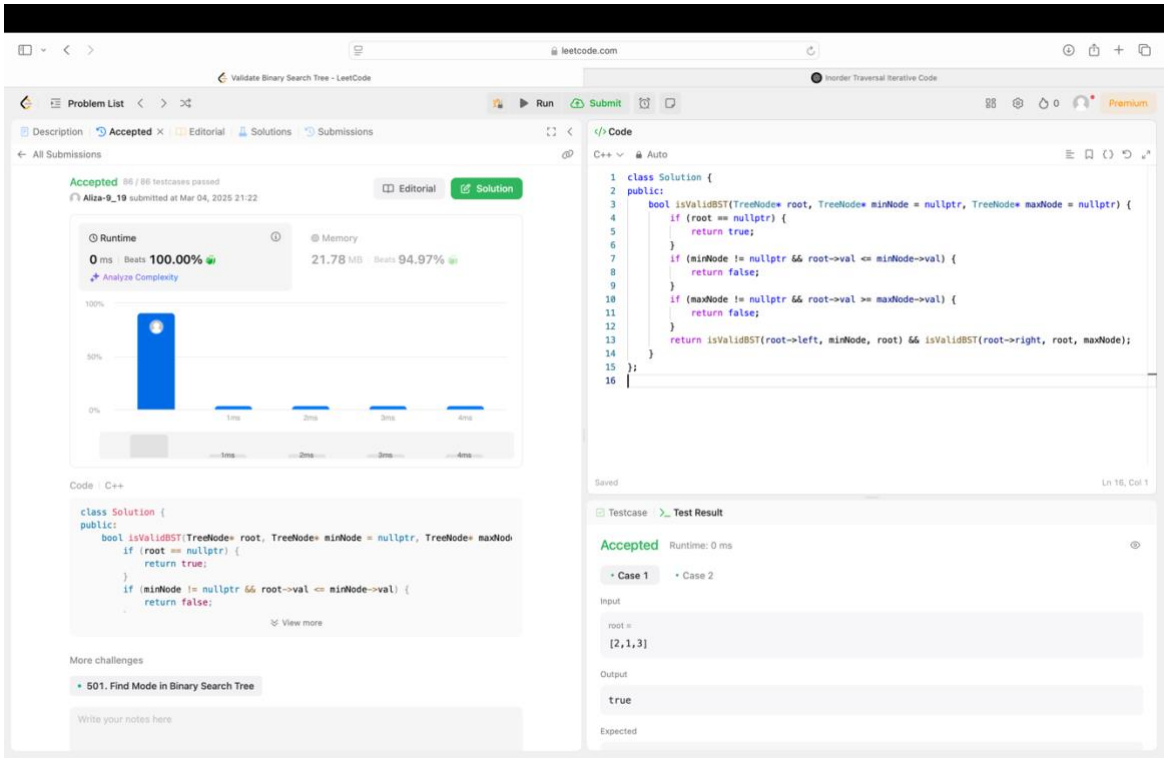
Q4. Validate Binary Search Tree (98)

Implementation Code:

```
class Solution {
public:
    bool isValidBST(TreeNode* root, TreeNode* minNode = nullptr, TreeNode* maxNode =
nullptr) {
        if (root == nullptr) {
            return true;
        }
```

```
        if (minNode != nullptr && root->val <= minNode->val) {
            return false;
        }
        if (maxNode != nullptr && root->val >= maxNode->val) {
            return false;
        }
        return isValidBST(root->left, minNode, root) && isValidBST(root->right, root, maxNode);
    }
};
```

Output:



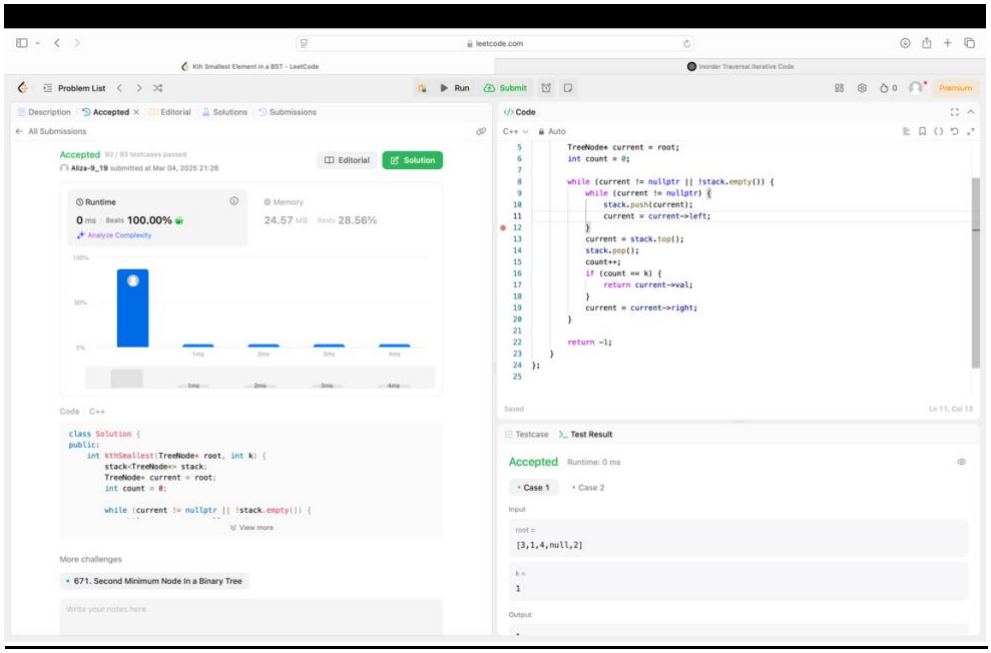
Q5. Kth Smallest Element in a BST (230)

Implementation Code:

```
class Solution {
```

```
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> stack;
        TreeNode* current = root;
        int count = 0;

        while (current != nullptr || !stack.empty()) {
            while (current != nullptr) {
                stack.push(current);
                current = current->left;
            }
            current = stack.top();
            stack.pop();
            count++;
            if (count == k) {
                return current->val;
            }
            current = current->right;
        }

        return -1;
    }
};
```

Output:



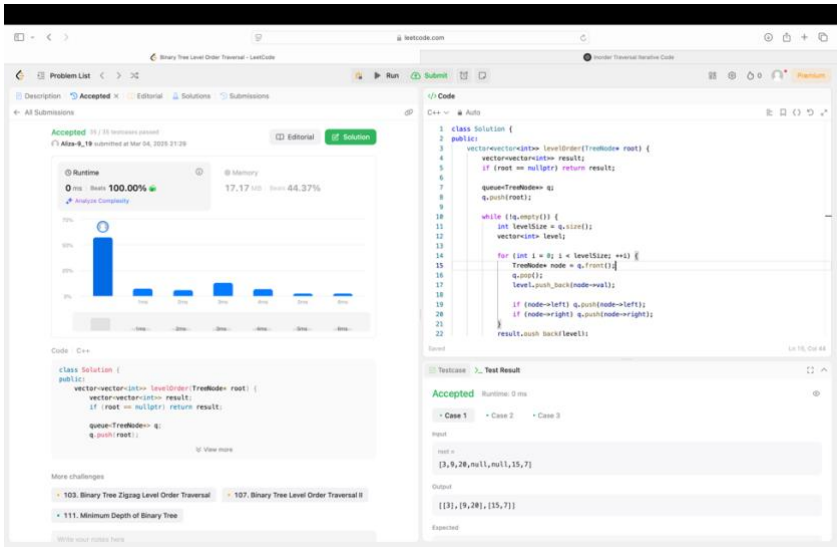
Q6. Binary Tree Level Order Traversal (102)

Implementation Code:

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (root == nullptr) 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;
    }
};
```

Output:



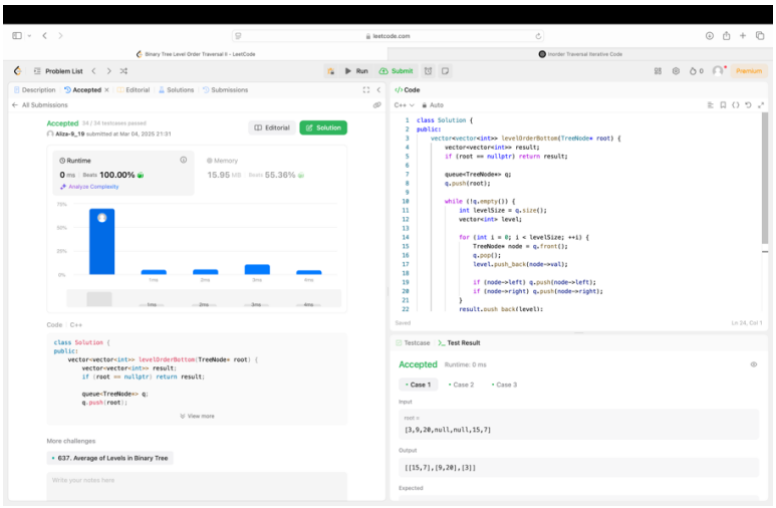
Q7. Binary Tree Level Order Traversal II (107)

Implementation Code:

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> result;
        if (root == nullptr) 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);
        }

        reverse(result.begin(), result.end());
        return result;
    }
};
```

Output:



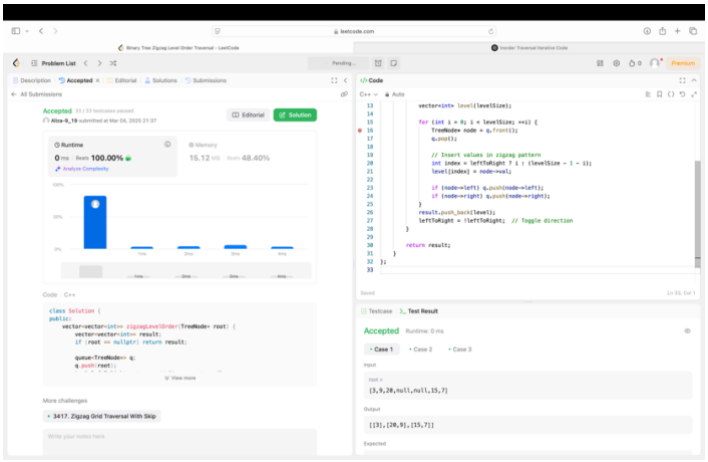
Q8. Binary Tree Zigzag Level Order Traversal (103)

Implementation Code:

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (root == nullptr) return result;
```

```
        queue<TreeNode*> q;
        q.push(root);
        bool leftToRight = true;

        while (!q.empty()) {
            int levelSize = q.size();
            vector<int> level(levelSize);

            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop();
                int index = leftToRight ? i : (levelSize - 1 - i);
                level[index] = node->val;

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(level);
            leftToRight = !leftToRight;
        }

        return result;
    }
};
```

Output:

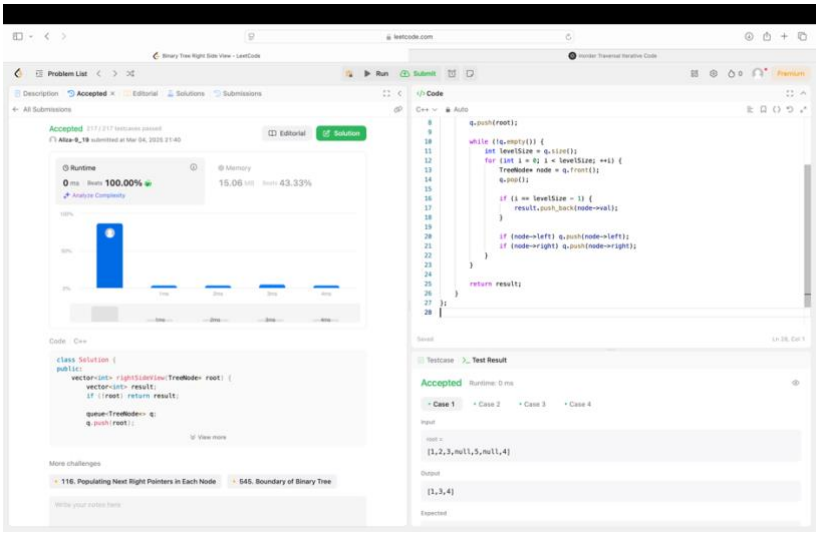


Q9. Binary Tree Right Side View (199)

Implementation Code:

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> result;
        if (!root) return result;

        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            int levelSize = q.size();
            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop();

                if (i == levelSize - 1) {
                    result.push_back(node->val);
                }

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

        return result;
    }
};
```

Output:



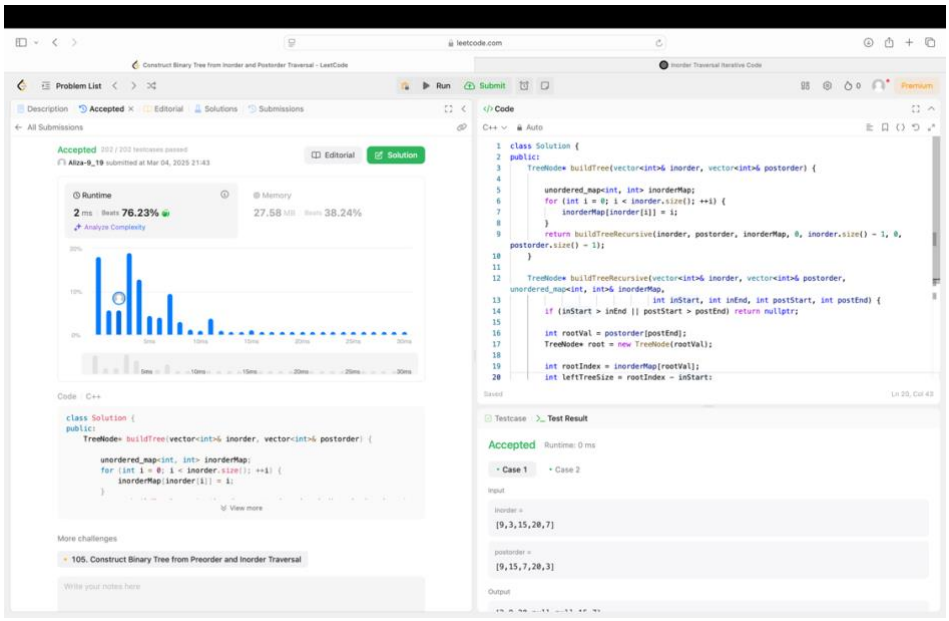
Q10. Construct Binary Tree from Inorder and Postorder Traversal (106)

Implementation Code:

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {

        unordered_map<int, int> inorderMap;
        for (int i = 0; i < inorder.size(); ++i) {
            inorderMap[inorder[i]] = i;
        }
        return buildTreeRecursive(inorder, postorder, inorderMap, 0, inorder.size() - 1, 0, postorder.size()
- 1);
    }

    TreeNode* buildTreeRecursive(vector<int>& inorder, vector<int>& postorder, unordered_map<int,
int>& inorderMap,
                                 int inStart, int inEnd, int postStart, int postEnd) {
        if (inStart > inEnd || postStart > postEnd) return nullptr;

        int rootVal = postorder[postEnd];
        TreeNode* root = new TreeNode(rootVal);

        int rootIndex = inorderMap[rootVal];
        int leftTreeSize = rootIndex - inStart;
```

```
        root->left = buildTreeRecursive(inorder, postorder, inorderMap, inStart, rootIndex - 1, postStart,
postStart + leftTreeSize - 1);
        root->right = buildTreeRecursive(inorder, postorder, inorderMap, rootIndex + 1, inEnd, postStart
+ leftTreeSize, postEnd - 1);

        return root;
    }
};
```

Output:



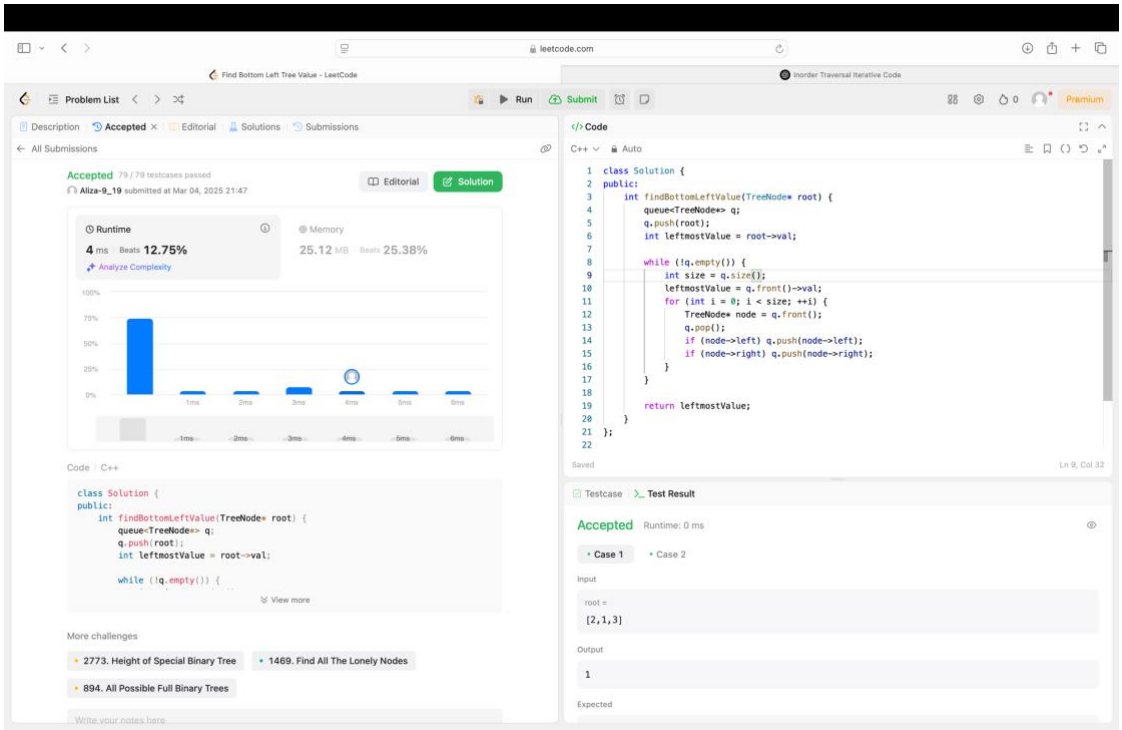
Q11. Find Bottom Left Tree Value (513)

Implementation Code:

```
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        queue<TreeNode*> q;
        q.push(root);
        int leftmostValue = root->val;

        while (!q.empty()) {
            int size = q.size();
            leftmostValue = q.front()->val;
```

```
            for (int i = 0; i < size; ++i) {
                TreeNode* node = q.front();
                q.pop();
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }

        return leftmostValue;
    }
};
```

Output:



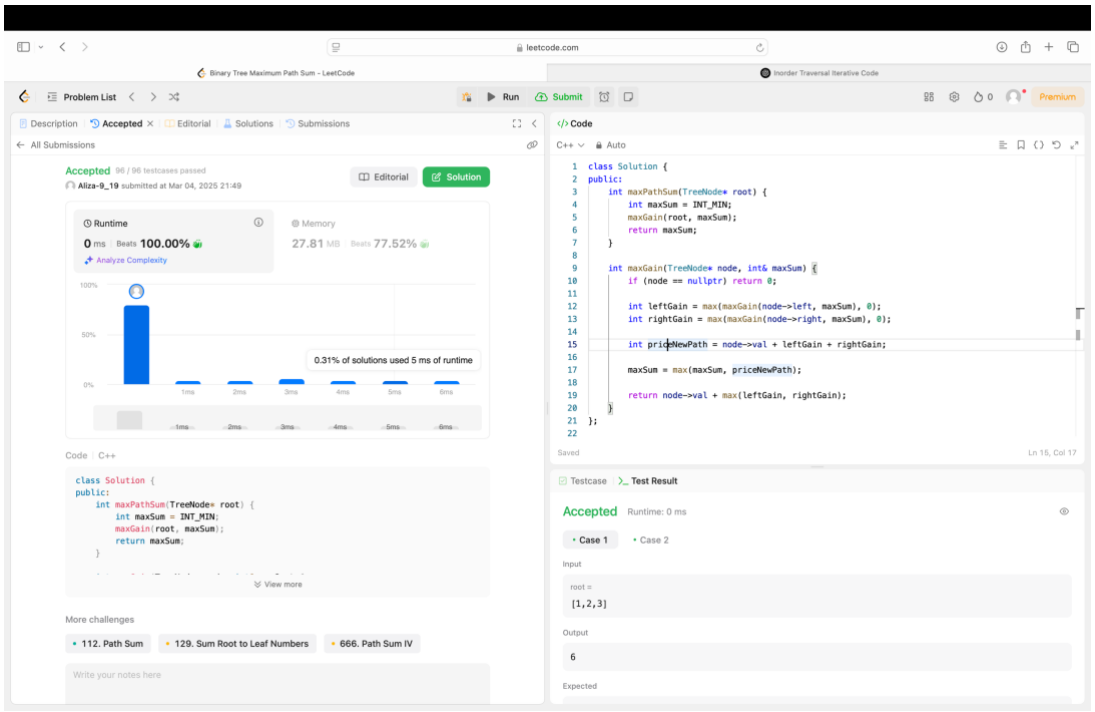
Q12. Binary Tree Maximum Path Sum (214)

Implementation Code:

```
class Solution {
public:
    int maxPathSum(TreeNode* root) {
        int maxSum = INT_MIN;
        maxGain(root, maxSum);
```

```
        return maxSum;
    }

    int maxGain(TreeNode* node, int& maxSum) {
        if (node == nullptr) return 0;

        int leftGain = max(maxGain(node->left, maxSum), 0);
        int rightGain = max(maxGain(node->right, maxSum), 0);

        int priceNewPath = node->val + leftGain + rightGain;

        maxSum = max(maxSum, priceNewPath);

        return node->val + max(leftGain, rightGain);
    }
};
```

Output:

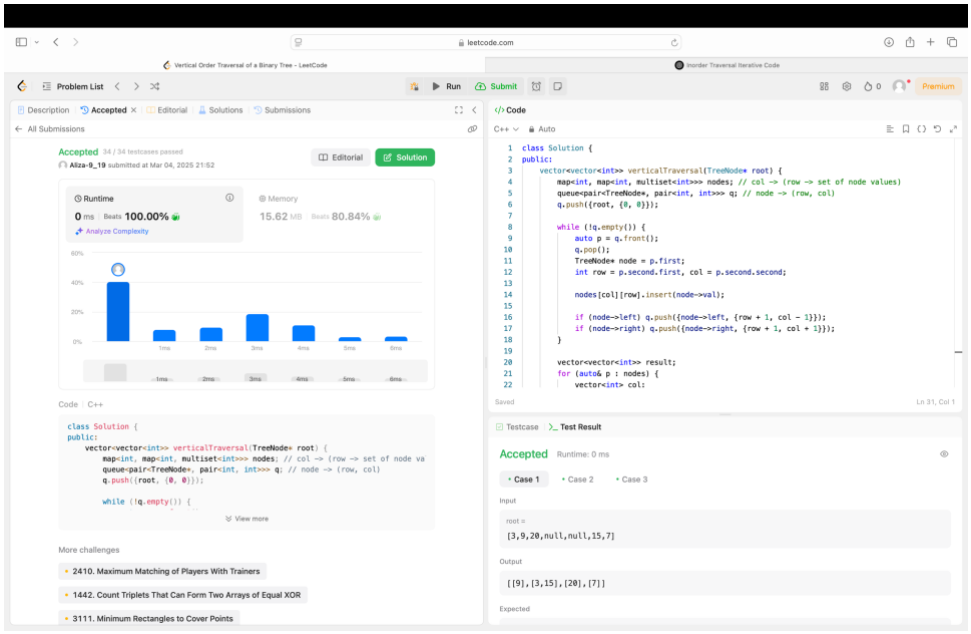


Q13. Vertical Order Traversal of a Binary Tree (987)

Implementation Code:

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        map<int, map<int, multiset<int>>> nodes; // col -> (row -> set of node values)
        queue<pair<TreeNode*, pair<int, int>>> q; // node -> (row, col)
        q.push({root, {0, 0}});

        while (!q.empty()) {
            auto p = q.front();
            q.pop();
            TreeNode* node = p.first;
            int row = p.second.first, col = p.second.second;

            nodes[col][row].insert(node->val);

            if (node->left) q.push({node->left, {row + 1, col - 1}});
            if (node->right) q.push({node->right, {row + 1, col + 1}});
        }

        vector<vector<int>> result;
        for (auto& p : nodes) {
            vector<int> col;
            for (auto& q : p.second) {
                col.insert(col.end(), q.second.begin(), q.second.end());
            }
            result.push_back(col);
        }
        return result;
    }
};
```

Output:


leetcode.com
Vertical Order Traversal of a Binary Tree - LeetCode
Inorder Traversal Iterative Code
Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 34 / 34 testcases passed
Aliza-9_19 submitted at Mar 04, 2025 21:52
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
15.62 MB | Beats 80.84%
Code | C++
class Solution {
public:
vector<vector<int>> verticalTraversal(TreeNode* root) {
map<int, map<int, multiset<int>>> nodes; // col -> (row -> set of node va
queue<pair<TreeNode*, pair<int, int>>> q; // node -> (row, col)
q.push({root, {0, 0}});
while (!q.empty()) {
View more
More challenges
2410. Maximum Matching of Players With Trainers
1442. Count Triplets That Can Form Two Arrays of Equal XOR
3111. Minimum Rectangles to Cover Points
Code
C++
Auto
class Solution {
public:
vector<vector<int>> verticalTraversal(TreeNode* root) {
map<int, map<int, multiset<int>>> nodes; // col -> (row -> set of node values)
queue<pair<TreeNode*, pair<int, int>>> q; // node -> (row, col)
q.push({root, {0, 0}});
while (!q.empty()) {
auto p = q.front();
q.pop();
TreeNode* node = p.first;
int row = p.second.first, col = p.second.second;
nodes[col][row].insert(node->val);
if (node->left) q.push({node->left, {row + 1, col - 1}});
if (node->right) q.push({node->right, {row + 1, col + 1}});
}
vector<vector<int>> result;
for (auto& p : nodes) {
vector<int> col;
Saved
Ln 31, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
root =
[3,9,20,null,null,15,7]
Output
[[9],[3,15],[20],[7]]
Expected