Experiment 7

Advanced Programming Lab – II

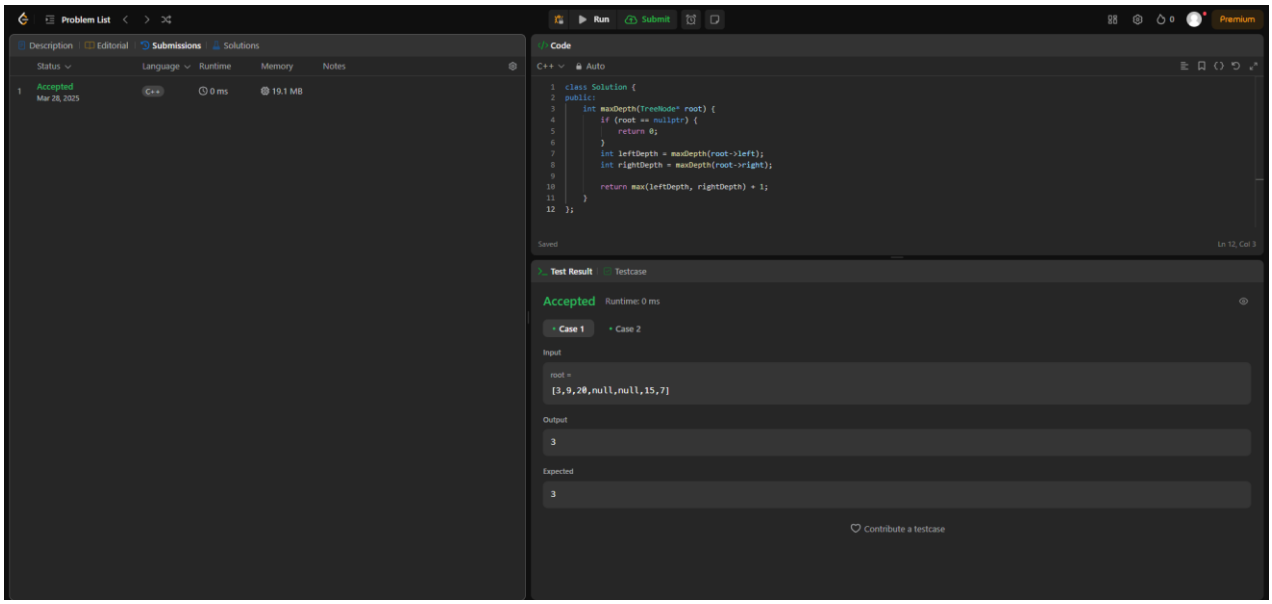
Submitted By – Ratnesh Tiwari

UID – 22BCS17201

Section – 22BCS_IOT – 609 (B)

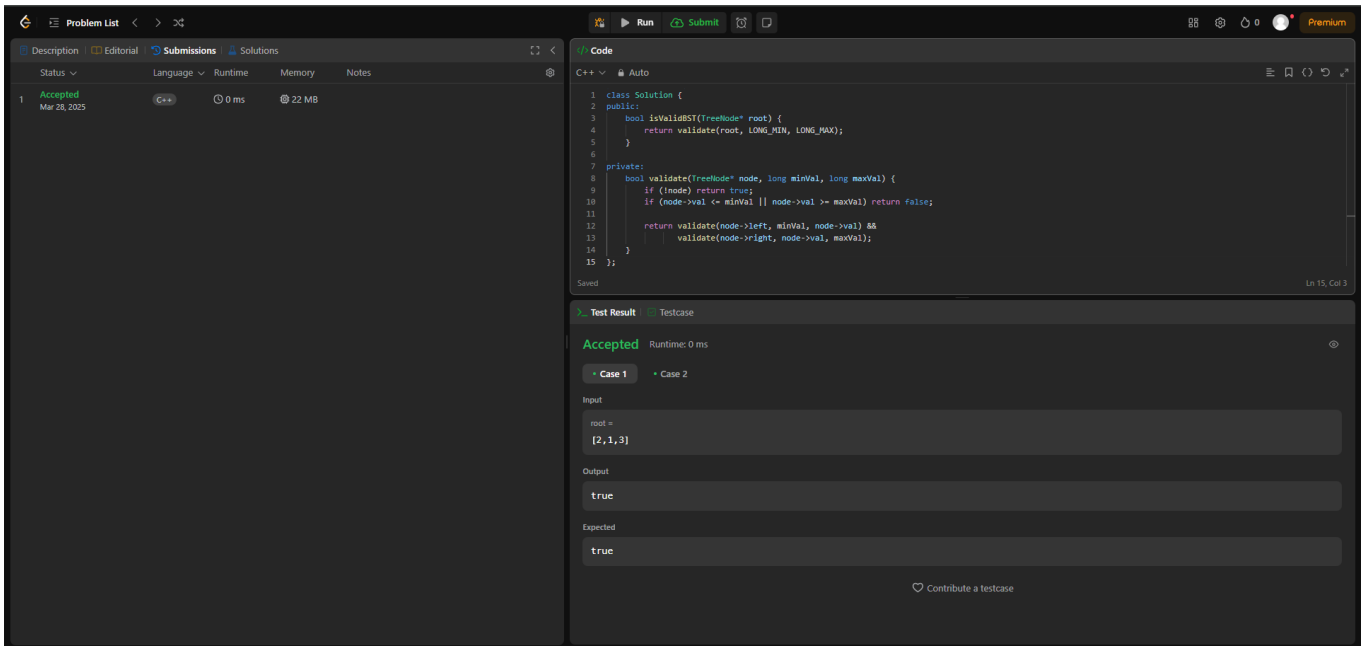
1. Maximum Depth of Binary Tree: https://leetcode.com/problems/maximum-depth-of-binary-tree/

```
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;
    }
};
```



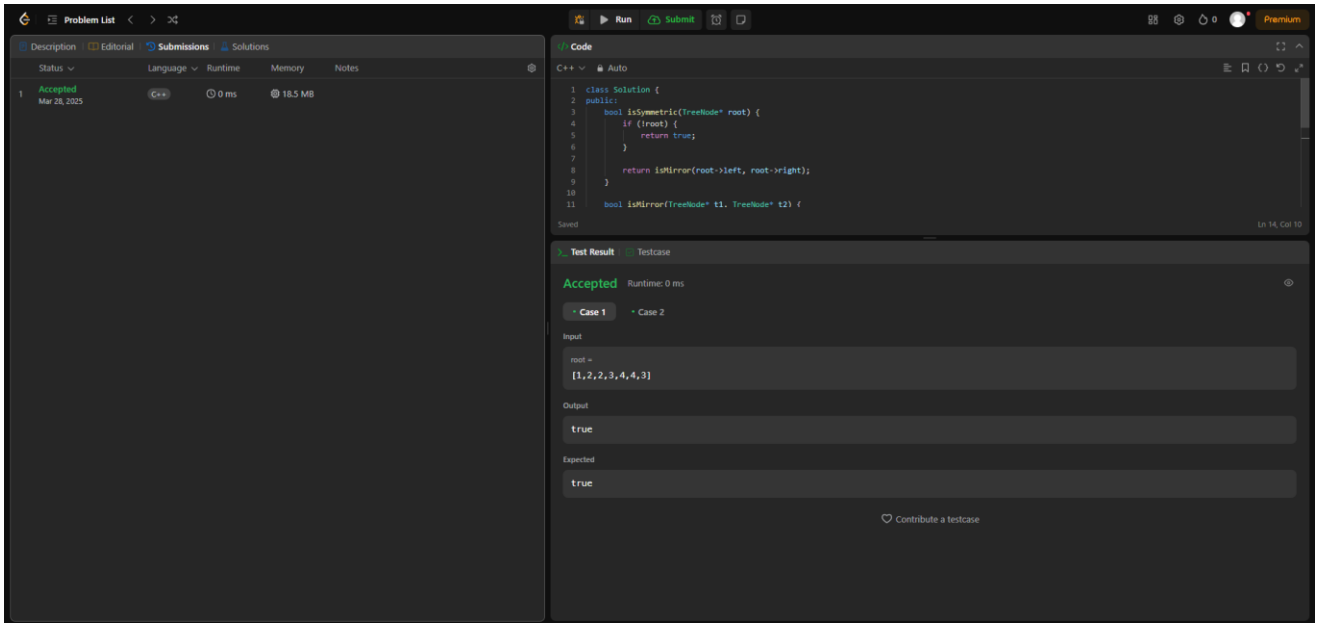
2. Validate Binary Search Tree: https://leetcode.com/problems/validate-binary-search-tree/

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return validate(root, LONG_MIN, LONG_MAX);
    }

private:
    bool validate(TreeNode* node, long minVal, long maxVal) {
        if (!node) return true;
        if (node->val <= minVal || node->val >= maxVal) return false;

        return validate(node->left, minVal, node->val) &&
               validate(node->right, node->val, maxVal);
    }
};
```



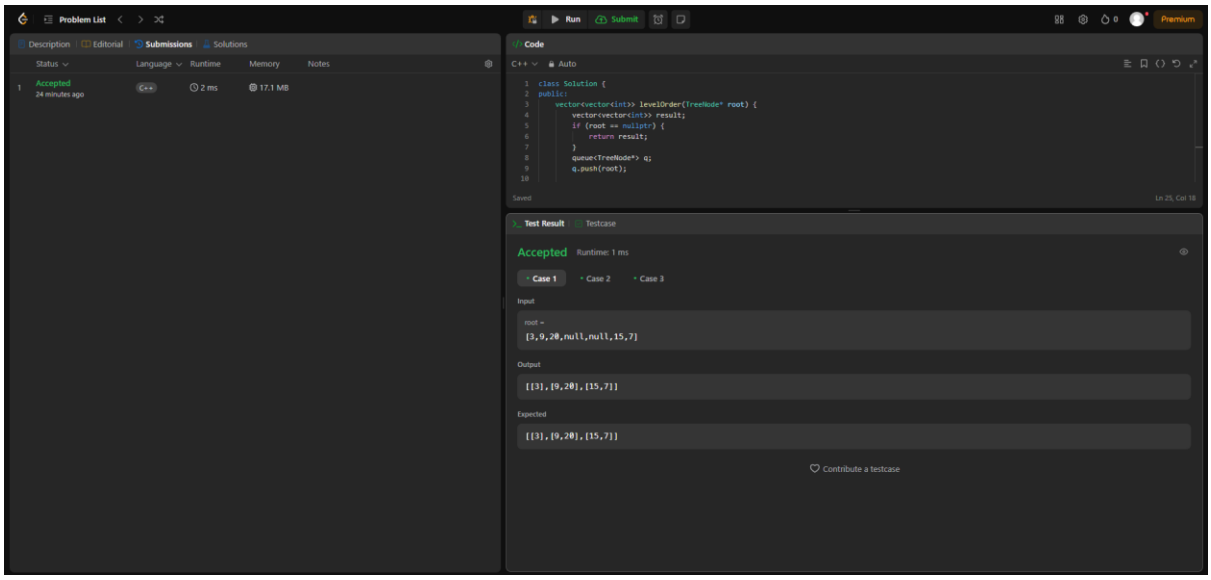
3. Symmetric Tree: https://leetcode.com/problems/symmetric-tree/

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (!root) {
            return true;
        }

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

    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);
    }
};
```



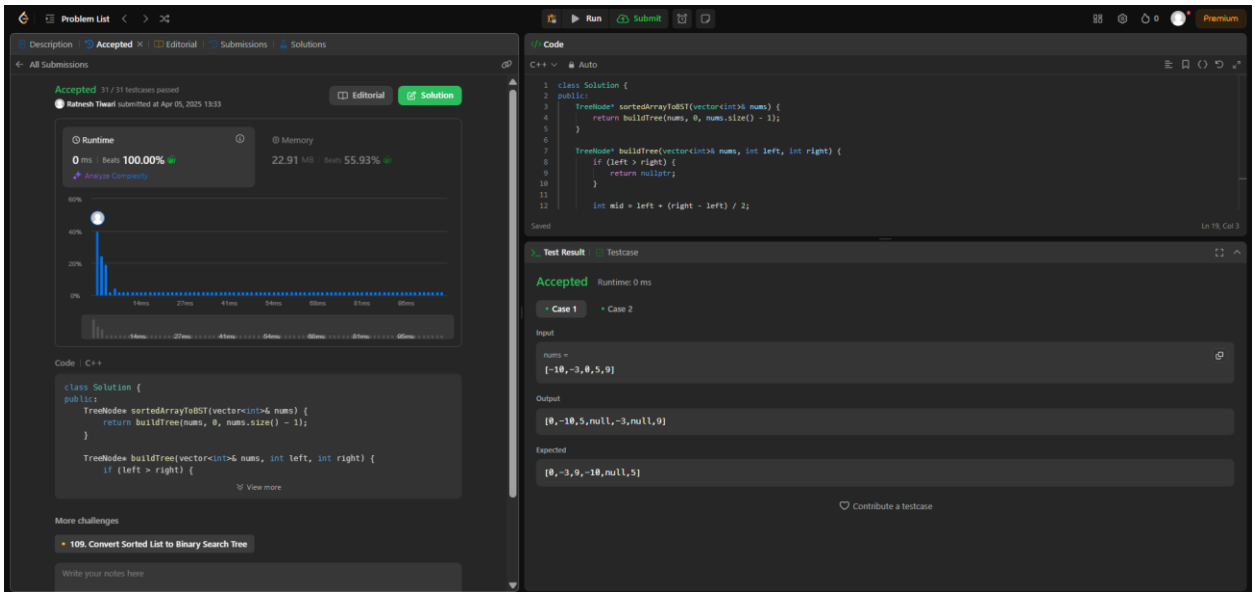
4. Binary Tree Level Order Traversal: https://leetcode.com/problems/binary-tree-level-order-traversal/

```
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> currentLevel;

            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop();
                currentLevel.push_back(node->val);

                if (node->left) {
                    q.push(node->left);
                }
                if (node->right) {
                    q.push(node->right);
                }
            }
            result.push_back(currentLevel);
        }
        return result;
    }
};
```



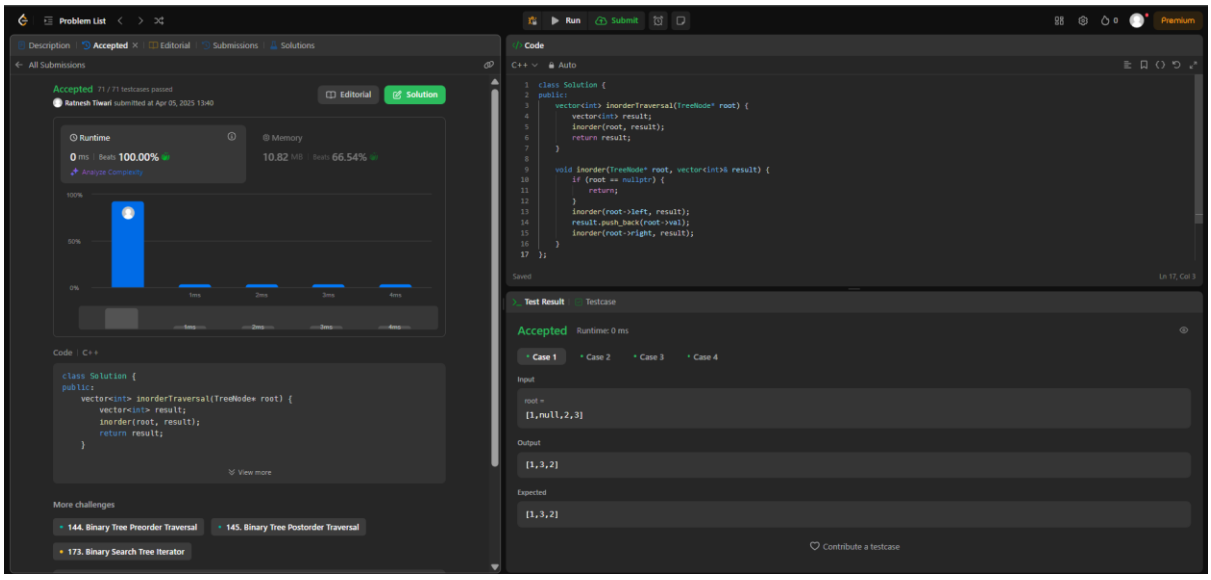
5. Convert Sorted Array to Binary Search Tree: https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return buildTree(nums, 0, nums.size() - 1);
    }

    TreeNode* buildTree(vector<int>& nums, int left, int right) {
        if (left > right) {
            return nullptr;
        }

        int mid = left + (right - left) / 2;
        TreeNode* root = new TreeNode(nums[mid]);
        root->left = buildTree(nums, left, mid - 1);
        root->right = buildTree(nums, mid + 1, right);

        return root;
    }
};
```



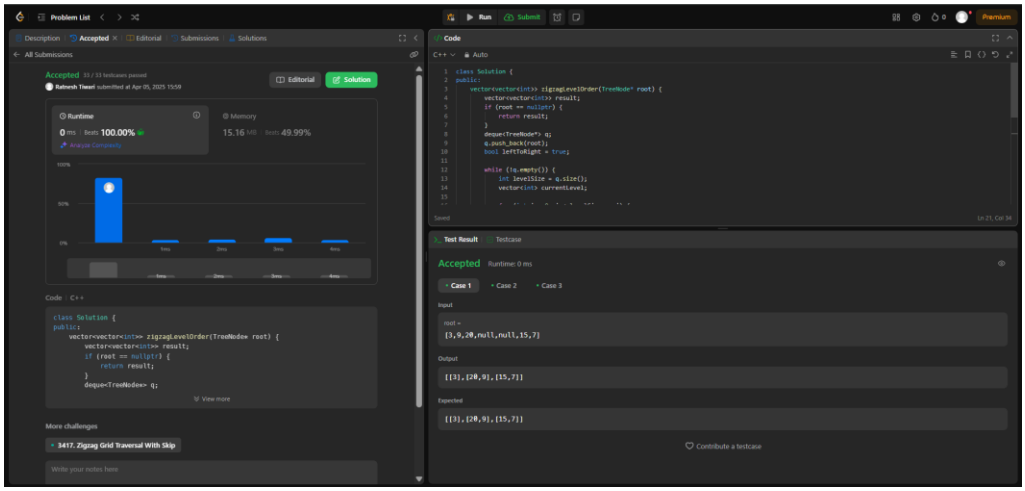
6. Binary Tree Inorder Traversal: https://leetcode.com/problems/binary-tree-inorder-traversal/

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorder(root, result);
        return result;
    }

    void inorder(TreeNode* root, vector<int>& result) {
        if (root == nullptr) {
            return;
        }
        inorder(root->left, result);
        result.push_back(root->val);
        inorder(root->right, result);
    }
};
```



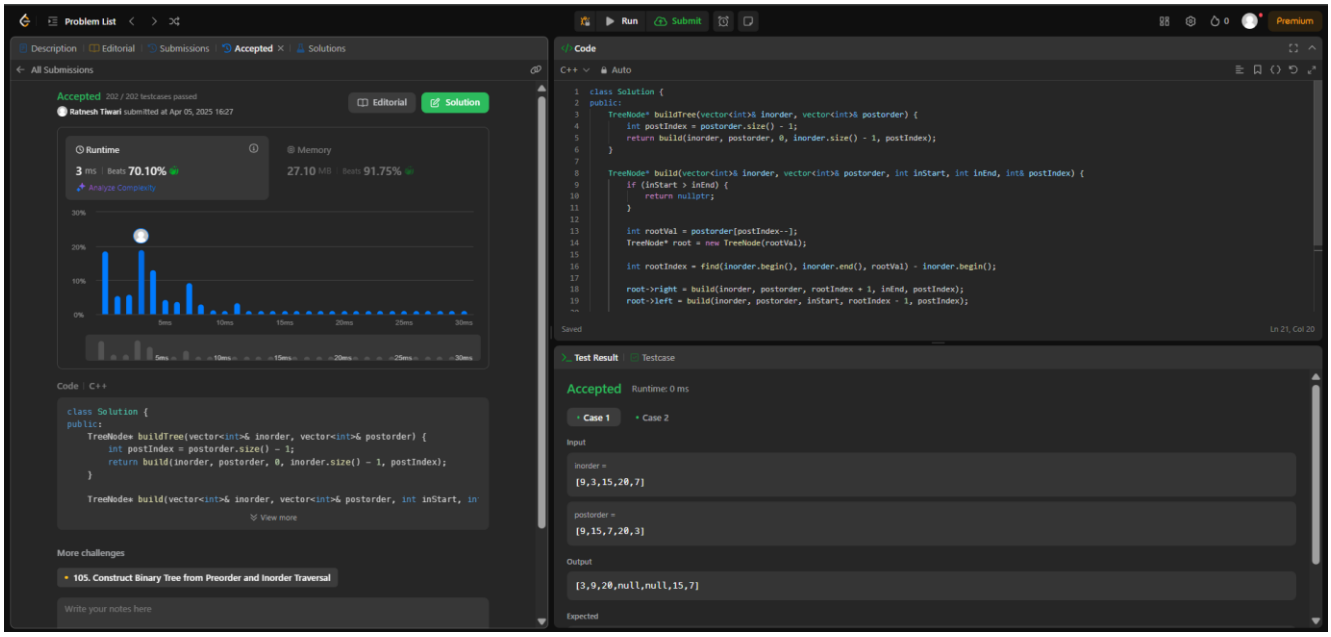
7. Binary Tree Zigzag Level Order Traversal: https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (root == nullptr) {
            return result;
        }
        deque<TreeNode*> q;
        q.push_back(root);
        bool leftToRight = true;

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

            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop_front();
                currentLevel.push_back(node->val);

                if (node->left) {
                    q.push_back(node->left);
                }
                if (node->right) {
                    q.push_back(node->right);
                }
            }
            if (!leftToRight) {
                reverse(currentLevel.begin(), currentLevel.end());
            }
            result.push_back(currentLevel);
            leftToRight = !leftToRight;
        }
        return result;
    }
};
```



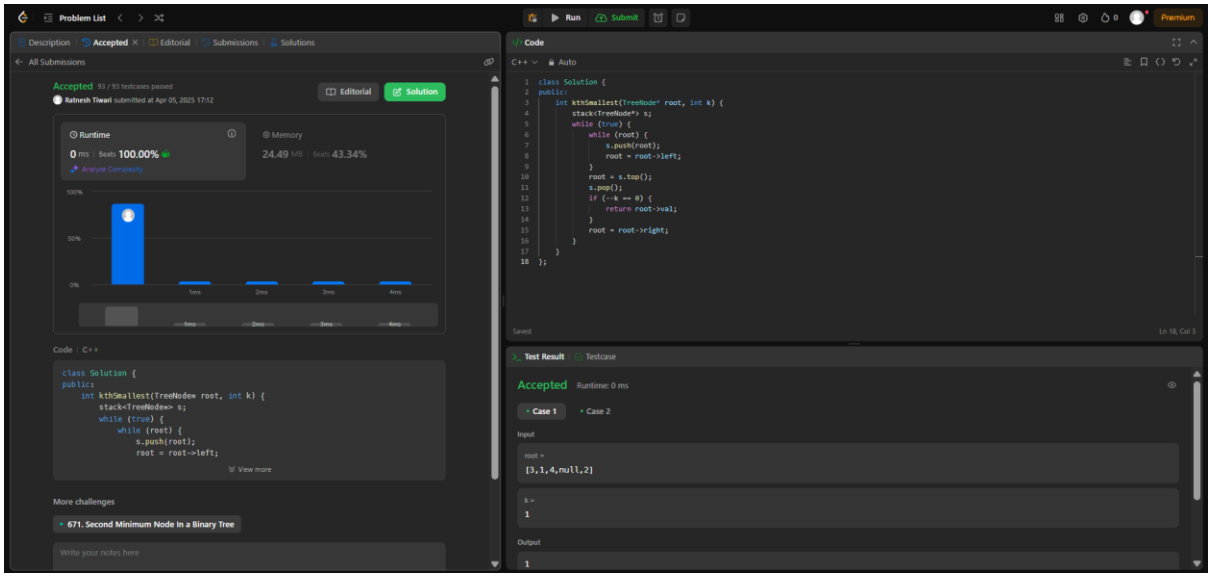
8. Construct Binary Tree from Inorder and Postorder Traversal:
https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        int postIndex = postorder.size() - 1;
        return build(inorder, postorder, 0, inorder.size() - 1, postIndex);
    }

    TreeNode* build(vector<int>& inorder, vector<int>& postorder, int inStart, int inEnd, int& postIndex) {
        if (inStart > inEnd) {
            return nullptr;
        }

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

        int rootIndex = find(inorder.begin(), inorder.end(), rootVal) - inorder.begin();

        root->right = build(inorder, postorder, rootIndex + 1, inEnd, postIndex);
        root->left = build(inorder, postorder, inStart, rootIndex - 1, postIndex);

        return root;
    }
};
```



9. Kth Smallest element in a BST: https://leetcode.com/problems/kth-smallest-element-in-a-bst/

```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> s;
        while (true) {
            while (root) {
                s.push(root);
                root = root->left;
            }
            root = s.top();
            s.pop();
            if (--k == 0) {
                return root->val;
            }
            root = root->right;
        }
    }
};
```



10. Populating Next Right Pointers in Each Node:
    https://leetcode.com/problems/populating-next-right-pointers-in-each-node/

```
class Solution {
public:
    Node* connect(Node* root) {
        if (!root) {
            return nullptr;
        }
        Node* levelStart = root;

        while (levelStart->left) {
            Node* current = levelStart;
            while (current) {
                current->left->next = current->right;
                if (current->next) {
                    current->right->next = current->next->left;
                }
                current = current->next;
            }
            levelStart = levelStart->left;
        }
        return root;
    }
};
```