UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE-6th Sem)

Subject Name: Advanced Programming Lab-2

Subject Code: 22CSP-351

Assignment - 03

Submitted to:

Vishal Kumar

Submitted by:

Name: **JEET BHARTI**

UID: **22BCS14804**

Section: **FL_IOT-602**

Group:

ASSIGNMENT – 3

Student Name: JEET BHARTI **UID: 22BCS14804**
Branch: BE-CSE **Section: FL_IOT-602 'A'**
Semester: 6th **Date of Performance: 14/02/25**
Subject Name: AP LAB-II **Subject Code: 22CSP-351**

1. Aim:

To solve the following problems on Leetcode, with the goal of optimizing solutions in terms of time complexity and space efficiency:

1) Binary Tree Inorder Traversal (94)
Aim: To perform inorder traversal (left, root, right) of a binary tree and return the sequence of visited nodes in a list. The traversal should be implemented using recursion or an iterative approach with a stack.

2) Symmetric Tree (101)
Aim: To determine whether a given binary tree is symmetric around its center, meaning it is a mirror reflection of itself when split from the root. The solution should explore both recursive and iterative approaches using queues.

3) Maximum Depth of Binary Tree (104)
Aim: To compute the maximum depth (height) of a binary tree, which is the longest path from the root to a leaf node. The depth is measured by the number of nodes along this path.

4) Validate Binary Search Tree (98)
Aim: To determine whether a given binary tree is a valid Binary Search Tree (BST), ensuring that for every node, the left subtree contains values less than the node, and the right subtree contains values greater than the node. The solution should consider edge cases, such as duplicate values and large depth constraints.

5) Kth Smallest Element in a BST (230)
Aim: To find the kth smallest element in a given Binary Search Tree (BST). The problem leverages the BST property, where an inorder traversal yields elements in sorted order, allowing an efficient solution.

6) Binary Tree Level Order Traversal (102)
Aim: To traverse a binary tree level by level (Breadth-First Search) and return a list of node values grouped by each level from top to bottom. The traversal should be implemented using a queue-based approach.

7) Binary Tree Level Order Traversal II (107)
Aim: To perform level order traversal on a binary tree and return the values of nodes at each level in bottom-up order, meaning the lowest level appears first in the output.

8) Binary Tree Zigzag Level Order Traversal (103)
Aim: To traverse a binary tree in a zigzag manner, where nodes at each level alternate between left-to-right and right-to-left traversal order. The traversal should be implemented using a queue or deque to facilitate the directional switching.

9) Binary Tree Right Side View (199)
Aim: To return a list of node values that are visible when looking at a binary tree from the right side, meaning only the rightmost node at each level should be included in the output.

10) Construct Binary Tree from Inorder and Postorder Traversal (106)
Aim: To construct a binary tree given its inorder and postorder traversal sequences. The solution should reconstruct the tree by identifying the root from the postorder list and partitioning the inorder list accordingly.

11) Find Bottom Left Tree Value (513)
Aim: To find the leftmost node value in the last row (deepest level) of a given binary tree. The solution should ensure the value returned belongs to the lowest possible depth and is the leftmost node at that depth.

12) Binary Tree Maximum Path Sum (124)
Aim: To find the maximum path sum in a binary tree, where a path is defined as any sequence of connected nodes, and the path sum is the sum of all node values in that sequence. The solution should account for paths that may or may not pass through the root.

13) Vertical Order Traversal of a Binary Tree (987)
Aim: To perform a vertical order traversal of a binary tree, where nodes are grouped by their vertical position and sorted based on horizontal position and value. The traversal follows a top-down, left-to-right approach for sorting nodes within the same vertical level.

2. Objective:

The objective of these problems is to develop a deep understanding of various tree traversal techniques, binary tree properties, and Binary Search Tree (BST) operations. These problems cover fundamental and advanced concepts, including:

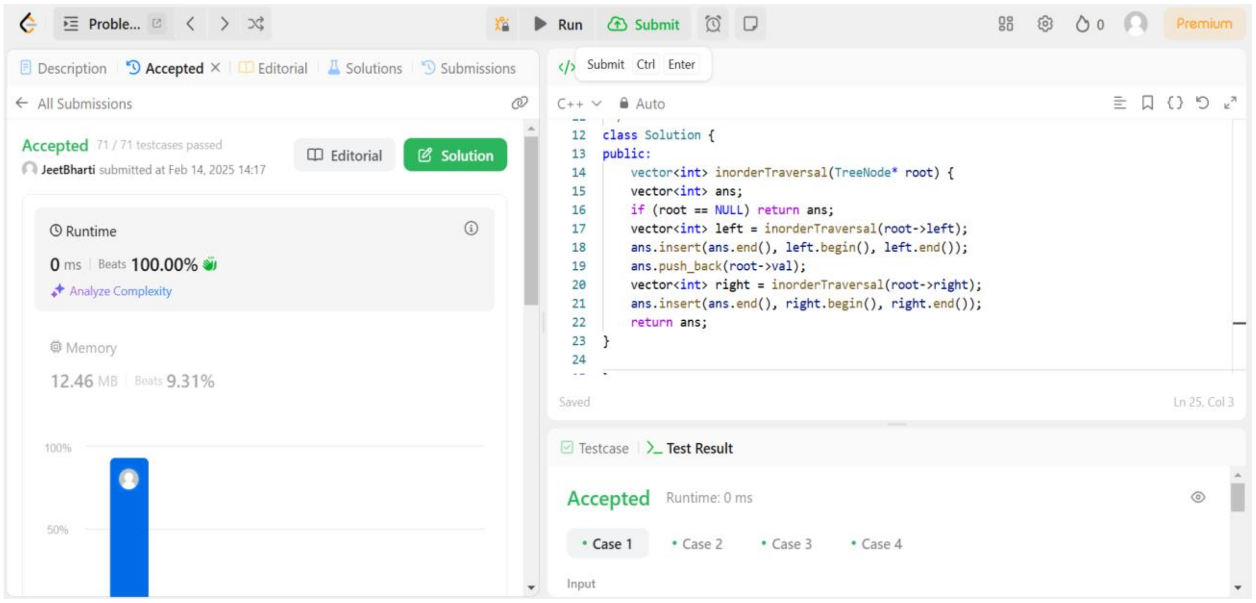
- Tree Traversal: Inorder, level order, zigzag, vertical, and postorder traversals.
- Tree Properties: Checking symmetry, depth calculation, and validation of BSTs.
- Tree Construction: Reconstructing a tree from traversal sequences.
- Path-Based Computations: Finding maximum path sums, right-side views, and bottom-left values.
- Search and Retrieval in BSTs: Finding the kth smallest element efficiently.

3. Implementation/Code:

Problem No. 94: Binary Tree Inorder Traversal

Code:

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

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



Accepted 71 / 71 testcases passed
JeetBharti submitted at Feb 14, 2025 14:17

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
12.46 MB | Beats 9.31%

100%
50%

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
    vector<int> ans;
    if (root == NULL) return ans;
    vector<int> left = inorderTraversal(root->left);
    ans.insert(ans.end(), left.begin(), left.end());
    ans.push_back(root->val);
    vector<int> right = inorderTraversal(root->right);
    ans.insert(ans.end(), right.begin(), right.end());
    return ans;
}
```

Saved   Ln 25, Col 3

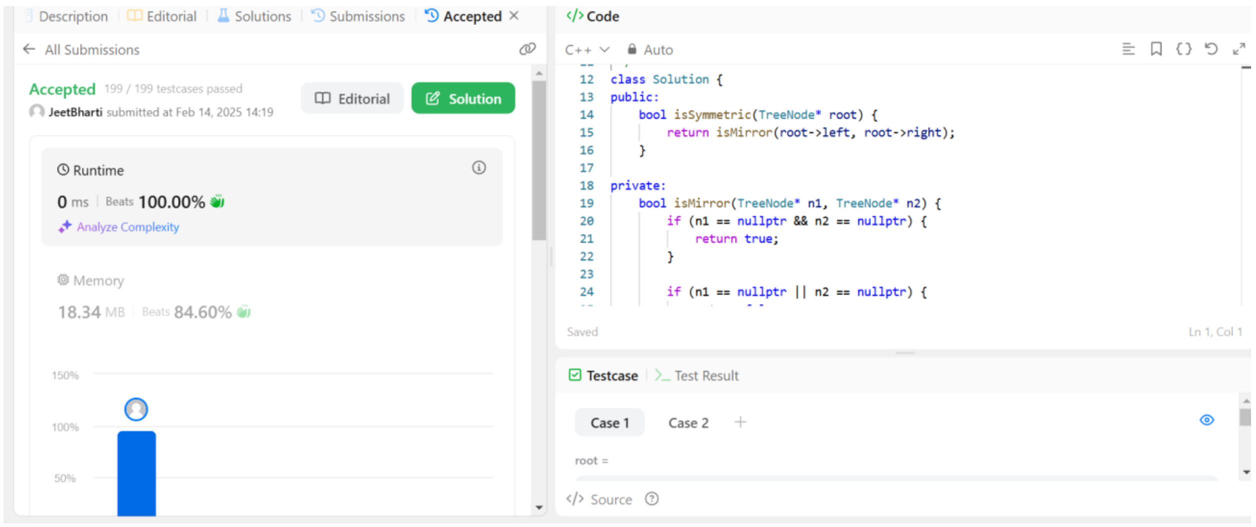
Accepted   Runtime: 0 ms

Case 1   Case 2   Case 3   Case 4

Input

Problem No. 101: Symmetric Tree
Code:

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        return isMirror(root->left, root->right);
    }

    bool isMirror(TreeNode* left, TreeNode* right) {
        if (!left && !right) return true;
        if (!left || !right) return false;
        if (left->val != right->val) return false;
        return isMirror(left->left, right->right) && isMirror(left->right, right->left);
    }
};
```


Description | Editorial | Solutions | Submissions | Accepted ×

All Submissions

Accepted 199 / 199 testcases passed
JeetBharti submitted at Feb 14, 2025 14:19

Editorial Solution

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
18.34 MB | Beats 84.60%

Code
C++ Auto

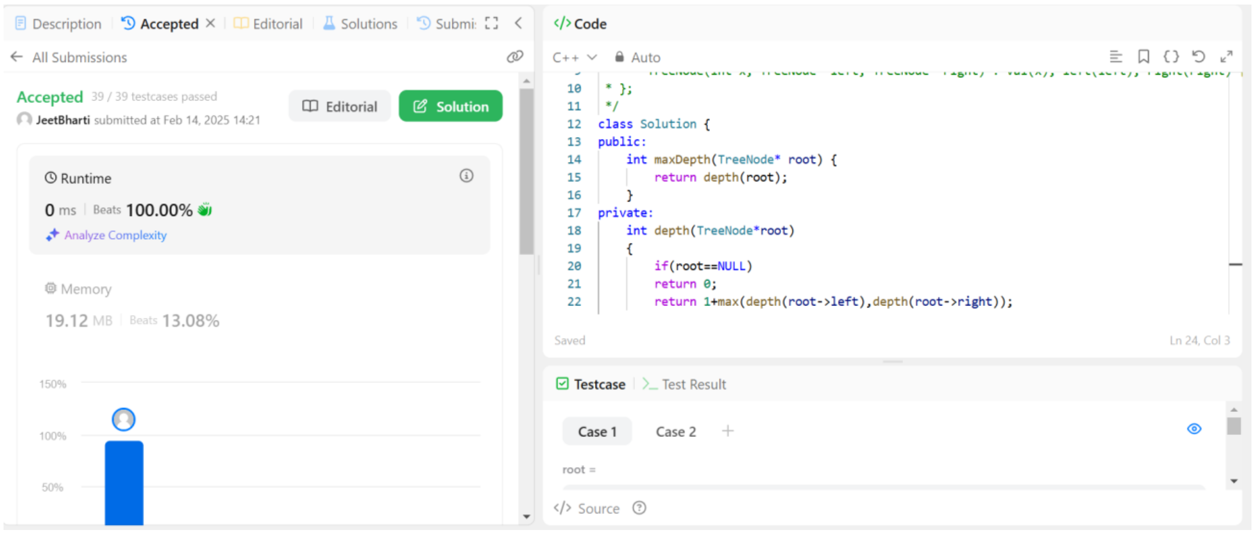
```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return isMirror(root->left, root->right);
    }

private:
    bool isMirror(TreeNode* n1, TreeNode* n2) {
        if (n1 == nullptr && n2 == nullptr) {
            return true;
        }

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

Saved Ln 1, Col 1

Testcase | Test Result
Case 1 Case 2
root =
Source

Problem No. 104: Maximum Depth of Binary Tree
Code:

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (!root) return 0;
        int h1 = maxDepth(root->left);
        int h2 = maxDepth(root->right);
        return max(h1, h2) + 1;
    }
};
```


Description | Accepted × | Editorial | Solutions | Submi:

All Submissions

Accepted 39 / 39 testcases passed
JeetBharti submitted at Feb 14, 2025 14:21

Editorial Solution

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
19.12 MB | Beats 13.08%

Code
C++ Auto

```
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        return depth(root);
    }
private:
    int depth(TreeNode*root)
    {
        if(root==NULL)
        return 0;
        return 1+max(depth(root->left),depth(root->right));
```

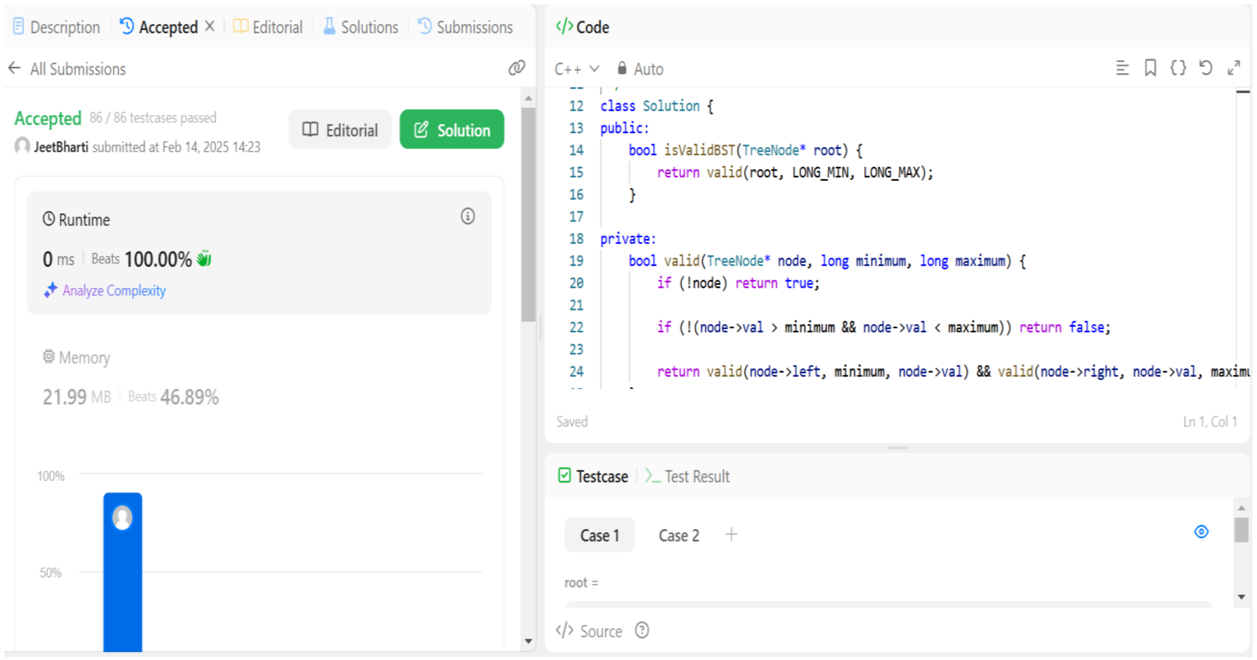
Saved Ln 24, Col 3

Testcase | Test Result
Case 1 Case 2
root =
Source

Problem No. 98: Validate Binary Search Tree

Code:

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```



Accepted 86 / 86 testcases passed
JeetBharti submitted at Feb 14, 2025 14:23

Runtime
0 ms | Beats 100.00%

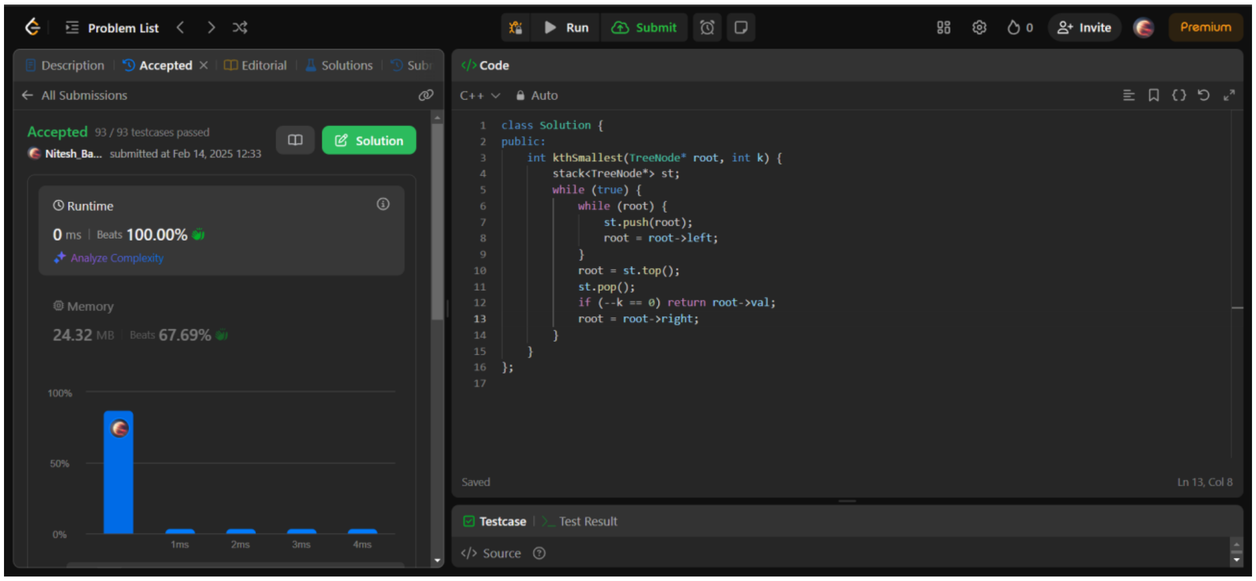
Memory
21.99 MB | Beats 46.89%

```
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, maximu
```

Problem No. 230: Kth Smallest Element in a BST

Code:

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



Accepted 93 / 93 testcases passed
Nitesh_Ba... submitted at Feb 14, 2025 12:33

Runtime
0 ms | Beats 100.00%

Memory
24.32 MB | Beats 67.69%

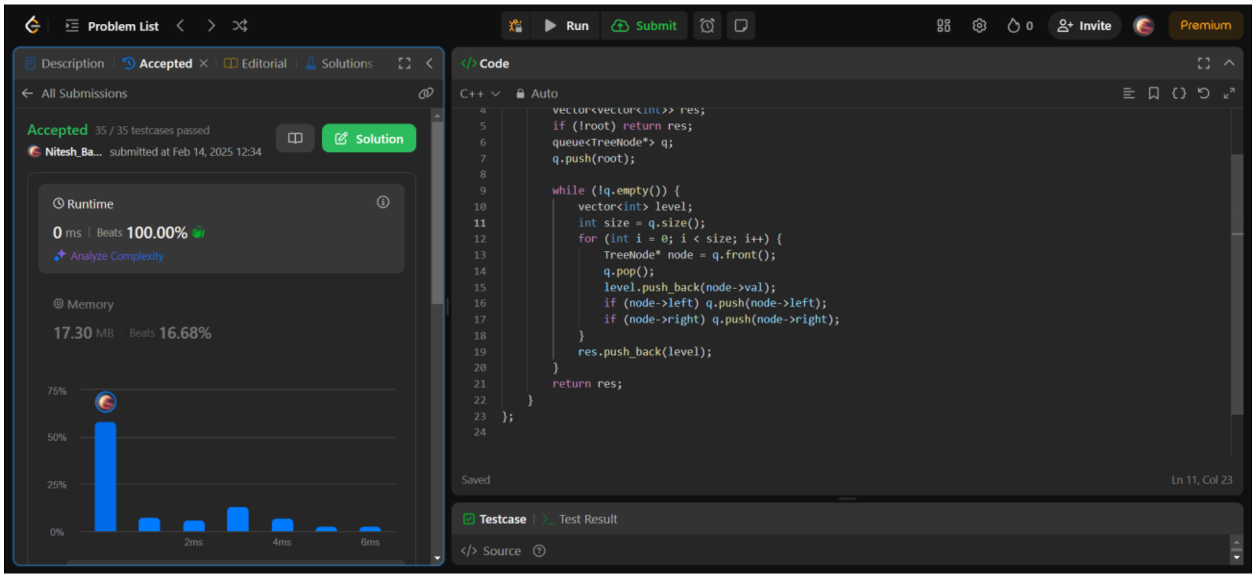
```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        stack<TreeNode*> st;
        while (true) {
            while (root) {
                st.push(root);
                root = root->left;
            }
            root = st.top();
            st.pop();
            if (--k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

Saved
Ln 13, Col 8

Problem No. 102: Binary Tree Level Order Traversal

Code:

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            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);
            }
            res.push_back(level);
        }
        return res;
    }
};
```



Accepted 35 / 35 testcases passed
Nitesh_Ba... submitted at Feb 14, 2025 12:34

Runtime
0 ms | Beats 100.00%

Memory
17.30 MB | Beats 16.68%

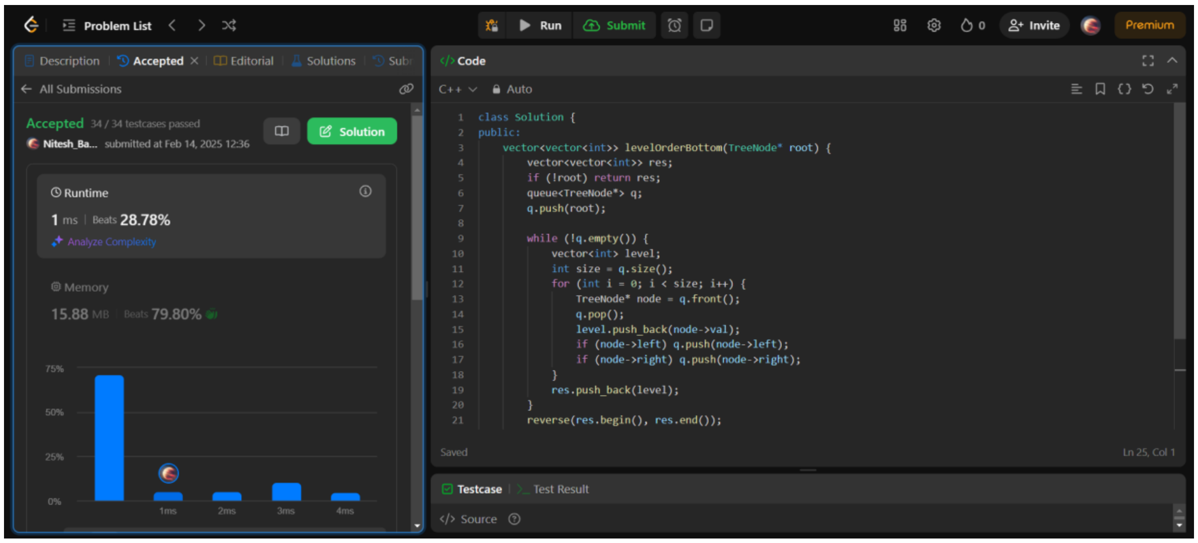
```
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            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);
            }
            res.push_back(level);
        }
        return res;
    }
};
```

Saved
Ln 11, Col 23

Problem No. 107: Binary Tree Level Order Traversal II

Code:

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            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);
            }
            res.push_back(level);
        }
        reverse(res.begin(), res.end());
        return res;
    }
};
```



Accepted 34 / 34 testcases passed
Nitesh_Ba... submitted at Feb 14, 2025 12:36

Runtime
1 ms | Beats 28.78%

Memory
15.88 MB | Beats 79.80%

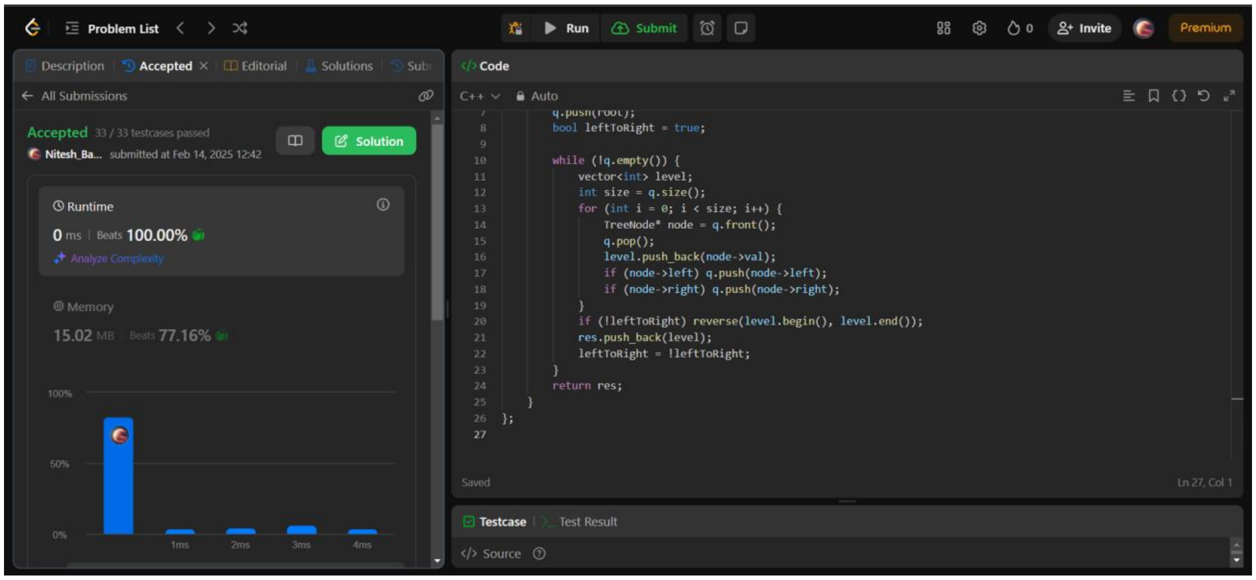
```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            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);
            }
            res.push_back(level);
        }
        reverse(res.begin(), res.end());
```

Saved
Ln 25, Col 1

Problem No. 103: Binary Tree Zigzag Level Order Traversal

Code:

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);
        bool leftToRight = true;

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            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);
            }
            if (!leftToRight) reverse(level.begin(), level.end());
            res.push_back(level);
            leftToRight = !leftToRight;
        }
        return res;
    }
};
```



Accepted 33 / 33 testcases passed
Nitesh_Ba... submitted at Feb 14, 2025 12:42

0 ms | Beats 100.00%

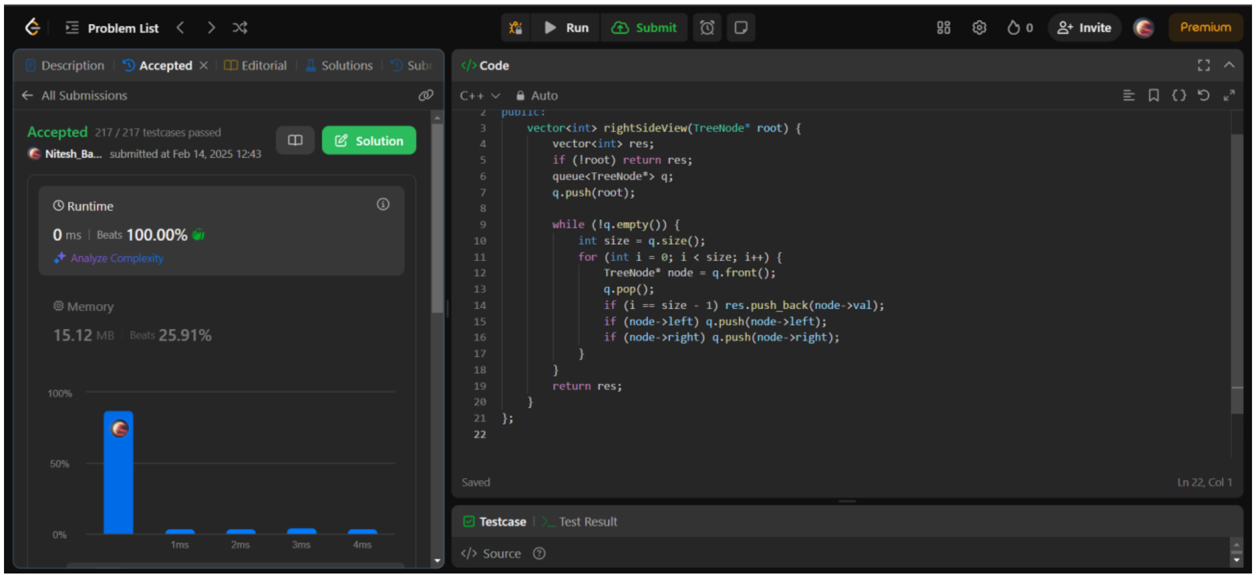
15.02 MB | Beats 77.16%

```
        q.push(root);
        bool leftToRight = true;

        while (!q.empty()) {
            vector<int> level;
            int size = q.size();
            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);
            }
            if (!leftToRight) reverse(level.begin(), level.end());
            res.push_back(level);
            leftToRight = !leftToRight;
        }
        return res;
    }
};
```

Problem No. 199: Binary Tree Right Side View

Code:

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                if (i == size - 1) res.push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return res;
    }
};
```



Accepted 217 / 217 testcases passed
Nitesh_Ba... submitted at Feb 14, 2025 12:43

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
15.12 MB | Beats 25.91%

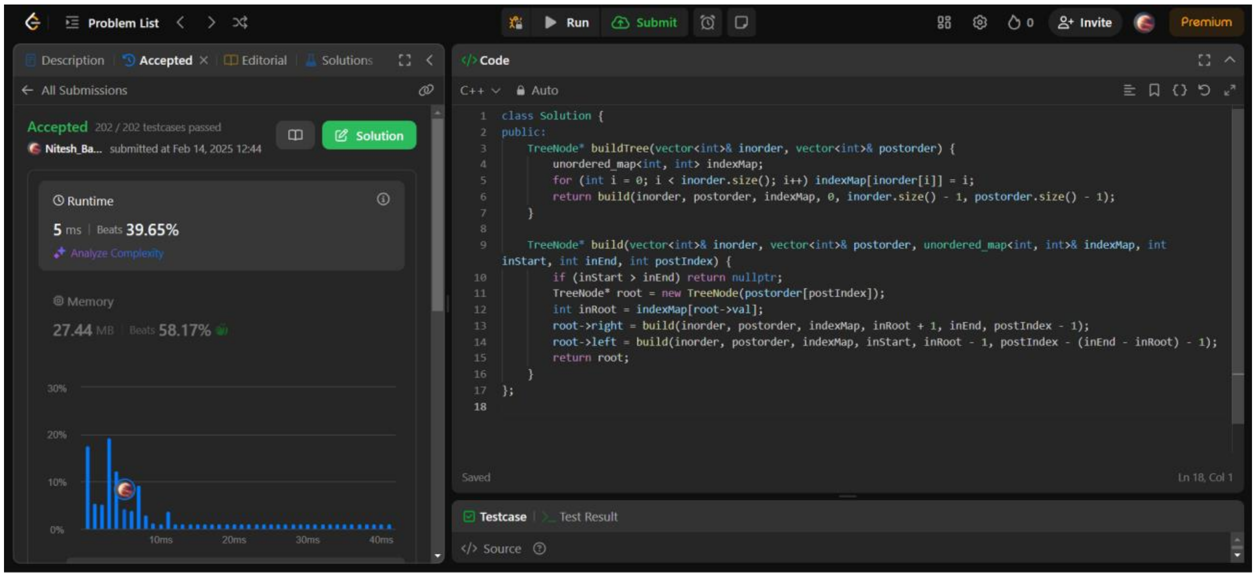
```
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);

        while (!q.empty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                if (i == size - 1) res.push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return res;
    }
};
```

Saved
Ln 22, Col 1

Problem No. 106: Construct Binary Tree from Inorder and Postorder Traversal

Code:

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> indexMap;
        for (int i = 0; i < inorder.size(); i++) indexMap[inorder[i]] = i;

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

    }


    TreeNode* build(vector<int>& inorder, vector<int>& postorder, unordered_map<int, int>&
indexMap, int inStart, int inEnd, int postIndex) {

        if (inStart > inEnd) return nullptr;

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

        int inRoot = indexMap[root->val];

        root->right = build(inorder, postorder, indexMap, inRoot + 1, inEnd, postIndex - 1);

        root->left = build(inorder, postorder, indexMap, inStart, inRoot - 1, postIndex - (inEnd -
inRoot) - 1);

        return root;

    }

};
```



Accepted 202 / 202 testcases passed
Nitesh_Ba... submitted at Feb 14, 2025 12:44

Runtime
5 ms | Beats 39.65%
Analyze Complexity

Memory
27.44 MB | Beats 58.17%

C++ Auto

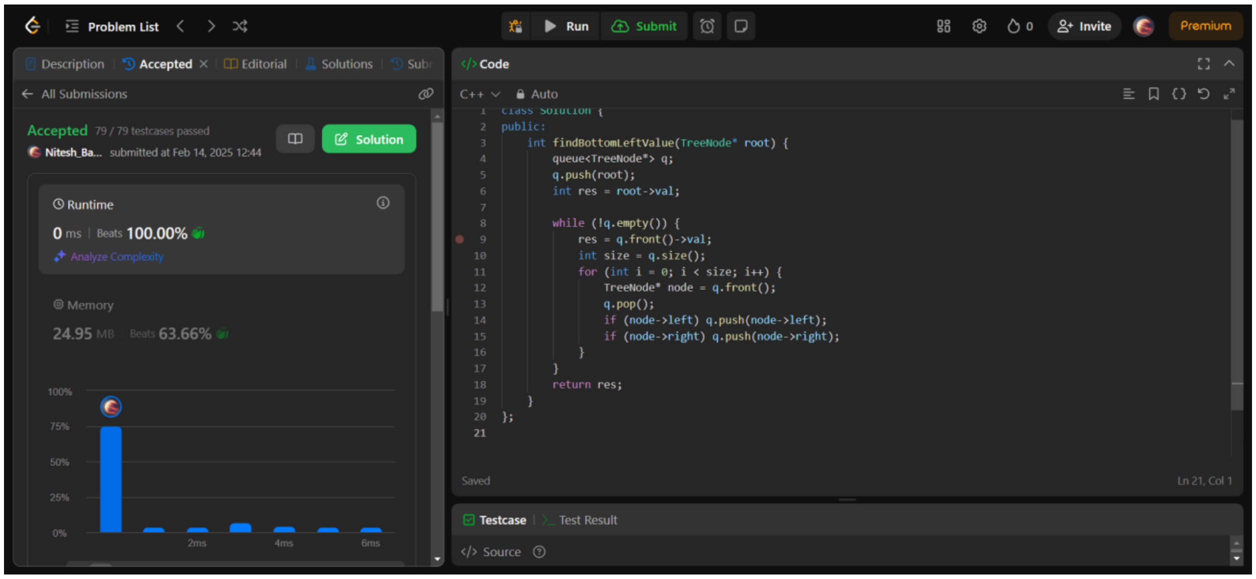
```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> indexMap;
        for (int i = 0; i < inorder.size(); i++) indexMap[inorder[i]] = i;
        return build(inorder, postorder, indexMap, 0, inorder.size() - 1, postorder.size() - 1);
    }

    TreeNode* build(vector<int>& inorder, vector<int>& postorder, unordered_map<int, int>& indexMap, int
inStart, int inEnd, int postIndex) {
        if (inStart > inEnd) return nullptr;
        TreeNode* root = new TreeNode(postorder[postIndex]);
        int inRoot = indexMap[root->val];
        root->right = build(inorder, postorder, indexMap, inRoot + 1, inEnd, postIndex - 1);
        root->left = build(inorder, postorder, indexMap, inStart, inRoot - 1, postIndex - (inEnd - inRoot) - 1);
        return root;
    }
};
```

Saved
Ln 18, Col 1

Problem No. 513: Find Bottom Left Tree Value

Code:

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

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



Accepted 79 / 79 testcases passed

Nitesh_Ba... submitted at Feb 14, 2025 12:44

Runtime
0 ms | Beats 100.00%

Memory
24.95 MB | Beats 63.66%

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

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

Saved   Ln 21, Col 1