

ASSIGNMENT 3

Name- Badal Yadav

UID-22BCS1092

SEC-601 'A'

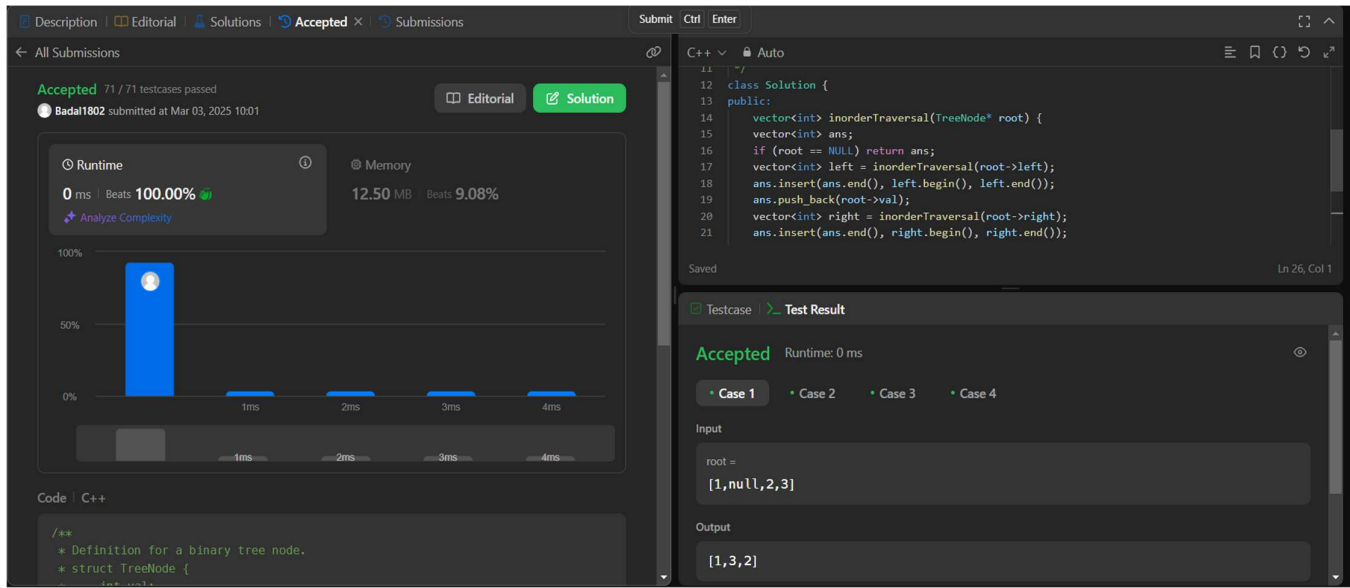
94. Binary Tree Inorder Traversal

CODE:

```
/**
 * 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> 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;
    }
};
```

```
}
};
```

OUTPUT:



101. Symmetric Tree

CODE:

```
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);
    }
};
```

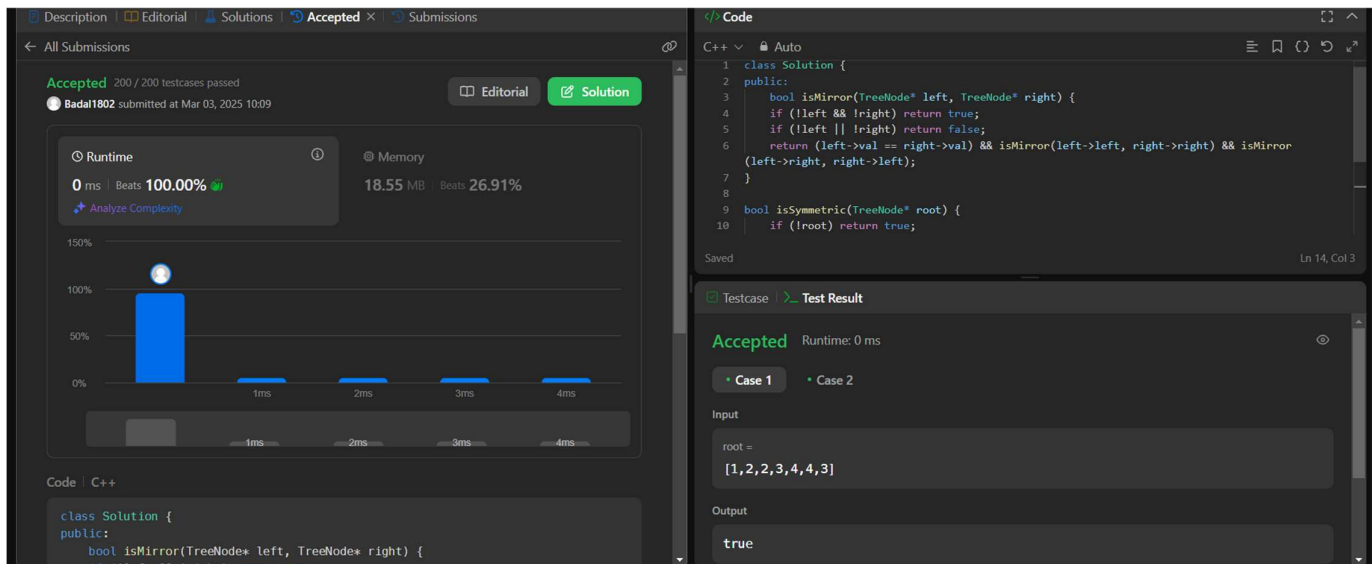
```

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

};

```

OUTPUT:



104. [Maximum Depth of Binary Tree](#)

CODE:

```

/**
 * 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:

    int maxDepth(TreeNode* root) {

        if(root==nullptr){

            return 0;

        }

        int leftdepth=maxDepth(root->left);

        int rightdepth=maxDepth(root->right);


        return 1+max(leftdepth,rightdepth);

    }

};

```

OUTPUT:

The screenshot displays the LeetCode submission interface for the problem "Maximum Depth of Binary Tree". The interface is divided into several sections:

- Submission Status:** Shows "Accepted" with 39/39 testcases passed. The submission was made by "Badal1802" on Mar 03, 2025 at 10:13.
- Performance Metrics:**
 - Runtime:** 0 ms, Beats 100.00%.
 - Memory:** 19.06 MB, Beats 44.95%.
- Runtime Graph:** A bar chart showing the runtime performance relative to other submissions. The user's submission is represented by a blue bar at the top, indicating it is the fastest.
- Code Editor:** Displays the C++ code for the solution. The code defines a `TreeNode` struct and a `Solution` class with a `maxDepth` method that recursively calculates the maximum depth of the binary tree.
- Test Case Results:** Shows the results for "Case 1". The input is `root = [3,9,20,null,null,15,7]` and the output is `3`.

98. Validate Binary Search Tree

CODE:

```
class Solution {  
  
public:  
  
    bool check(TreeNode* root, long long mini, long long maxi){  
  
        if(root==NULL)return true;  
  
  
        if(root->val <=mini || root->val >= maxi)return false;  
  
        return check(root->left,mini,root->val) && check(root->right, root->val, maxi);  
    }  
  
    bool isValidBST(TreeNode* root) {  
  
        return check(root, LLONG_MIN, LLONG_MAX);  
    }  
};
```

OUTPUT:

The screenshot shows a coding platform interface with the following components:

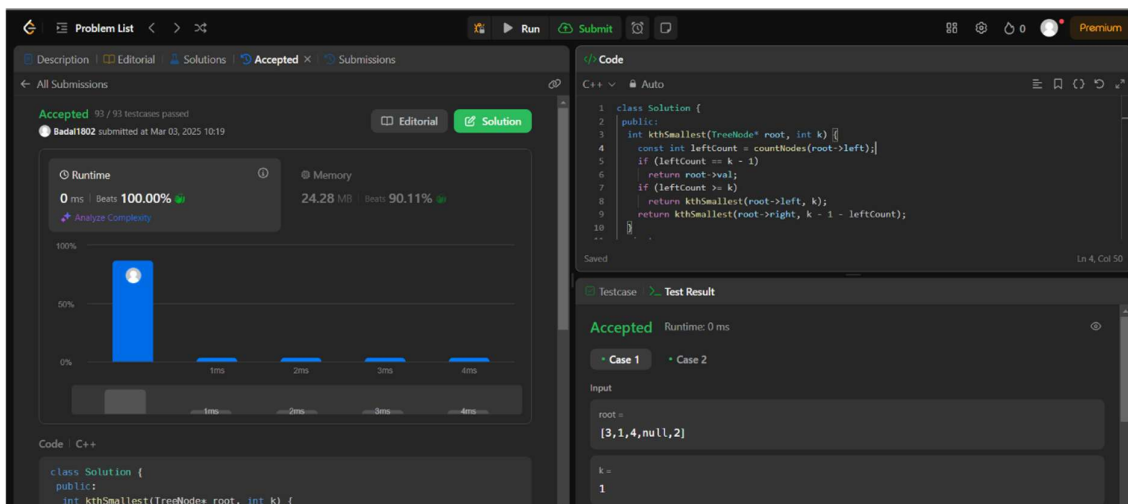
- Problem List:** Shows the problem name and status (Accepted).
- Submissions:** Shows the submission status (Accepted) and the user's name (Badal1802) with the submission time (Mar 03, 2025 10:16).
- Runtime and Memory:** Shows the runtime (0 ms) and memory (21.95 MB) for the submission. The runtime is 100.00% and the memory is 48.36%.
- Code Editor:** Displays the C++ code for the solution, which is the same as the code provided in the previous block.
- Testcase and Test Result:** Shows the test result for the submission, which is Accepted. The runtime is 0 ms. The input is [2,1,3] and the output is true.

230. Kth Smallest Element in a BST

CODE:

```
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        const int leftCount = countNodes(root->left);
        if (leftCount == k - 1)
            return root->val;
        if (leftCount >= k)
            return kthSmallest(root->left, k);
        return kthSmallest(root->right, k - 1 - leftCount);
    }
private:
    int countNodes(TreeNode* root) {
        if (root == nullptr)
            return 0;
        return 1 + countNodes(root->left) + countNodes(root->right);
    }
};
```

OUTPUT:



102. Binary Tree Level Order Traversal

CODE:

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

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

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

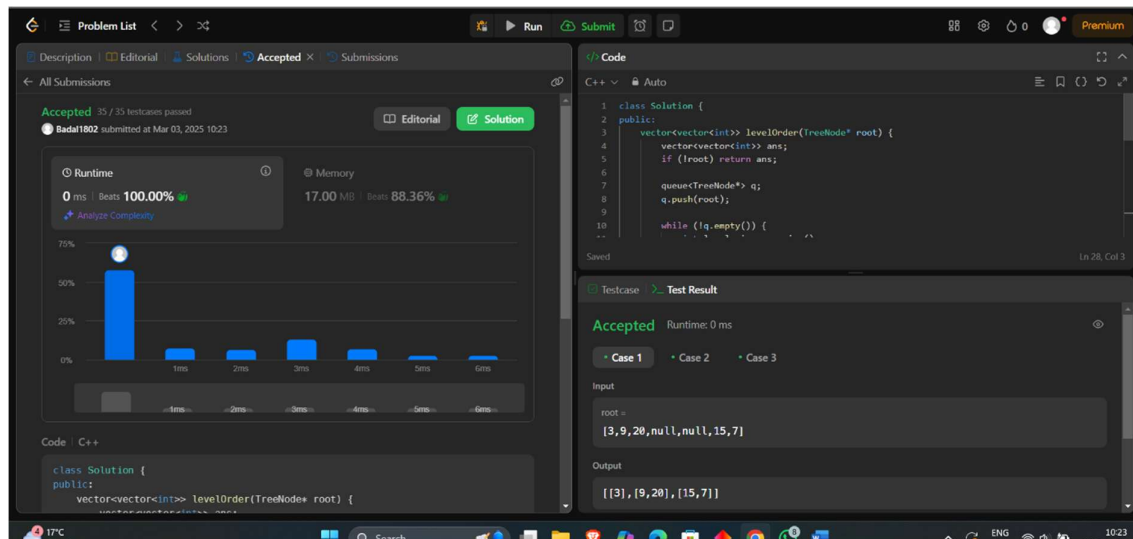
            for (int i = 0; i < level_size; ++i) {
                TreeNode* node = q.front();
                q.pop();
                level.push_back(node->val);

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

            ans.push_back(level);
        }

        return ans;
    };
};
```

OUTPUT:



107. Binary Tree Level Order Traversal II

CODE:

```
class Solution {
public:
    vector<vector<int>>ans;
    void helper(TreeNode * root,int level)
    {
        if(root==NULL)
            return ;
        if(ans.size()<=level)
        {
            ans.resize(level+1);
        }
        ans[level].push_back(root->val);
        if(root->left)
            helper(root->left,level+1);
    }
};
```



```

        if(root->right)
            helper(root->right,level+1);

        return ;

    }

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

        helper(root,0);

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

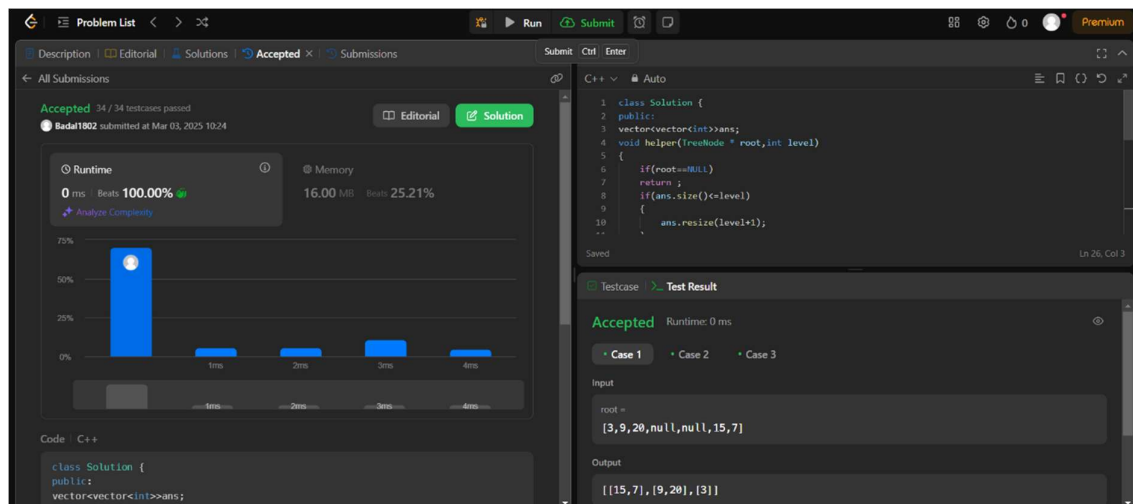
        return ans;

    }

};

```

OUTPUT:



103. Binary Tree Zigzag Level Order Traversal

CODE:

```

class Solution {

```

public:

```
void solve(vector<vector<int>>& ans, TreeNode* temp, int level) {  
    if (temp == NULL) return;  
    if (ans.size() <= level) ans.push_back({});  
    if (level % 2 == 0) ans[level].push_back(temp->val);  
    else ans[level].insert(ans[level].begin(), temp->val);  
    solve(ans, temp->left, level + 1);  
    solve(ans, temp->right, level + 1);  
}
```

```
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {  
    vector<vector<int>> ans;  
    solve(ans, root, 0);  
    return ans;  
}  
};
```

OUTPUT:

The screenshot displays a coding platform interface with the following components:

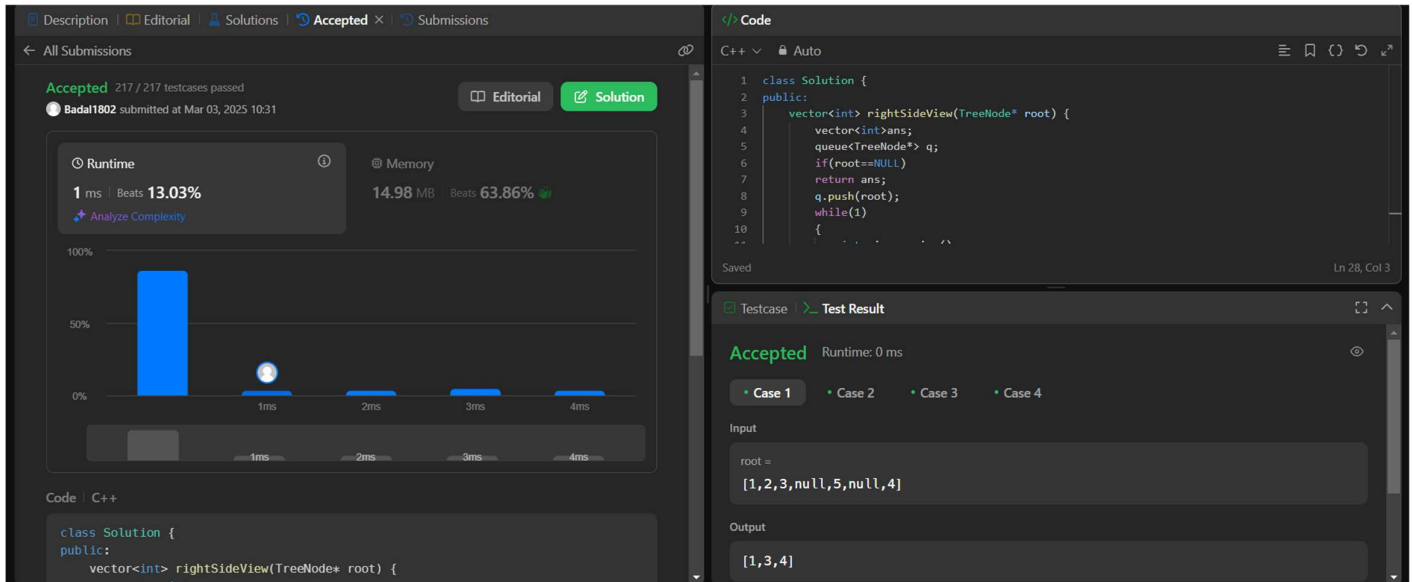
- Submission Header:** Shows "Accepted" status, "33 / 33 testcases passed", and the user "Badal1802" submitted at "Mar 03, 2025 10:27".
- Performance Metrics:**
 - Runtime:** 0 ms, Beats 100.00%.
 - Memory:** 15.17 MB, Beats 48.49%.
- Bar Chart:** A chart showing the distribution of runtime performance across various test cases, with a single bar at 0ms.
- Code Editor:** Displays the C++ code for the zigzag level order traversal solution.
- Test Results:**
 - Accepted:** Runtime: 0 ms.
 - Case 1:** Input: "root = [3,9,20,null,null,15,7]", Output: "[[3],[20,9],[15,7]]".

199. [Binary Tree Right Side View](#)

CODE:

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> ans;
        queue<TreeNode*> q;
        if(root==NULL)
            return ans;
        q.push(root);
        while(1)
        {
            int size=q.size();
            if(size==0)
                return ans;
            vector<int> data;
            while(size--)
            {
                TreeNode* temp=q.front();
                q.pop();
                data.push_back(temp->val);
                if(temp->left!=NULL)
                    q.push(temp->left);
                if(temp->right!=NULL)
                    q.push(temp->right);
            }
            ans.push_back(data.back());
        }
    }
};
```

OUTPUT:



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

    TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int inorderStart, int
inorderEnd, int postorderStart, int postorderEnd, unordered_map<int, int>& index) {
        if (inorderStart > inorderEnd || postorderStart > postorderEnd) {
            return nullptr;
        }
        int rootVal = postorder[postorderStart];
        int rootIndex = index[rootVal];
        TreeNode* root = new TreeNode(rootVal);
        root->left = buildTreeHelper(inorder, postorder, inorderStart, rootIndex, postorderStart,
postorderStart + rootIndex - inorderStart, index);
        root->right = buildTreeHelper(inorder, postorder, rootIndex + 1, inorderEnd, postorderStart +
rootIndex - inorderStart + 1, postorderEnd, index);
        return root;
    }
};
```

```

        return nullptr;
    }

    int rootVal = postorder[postorderEnd];

    TreeNode* root = new TreeNode(rootVal);

    int inorderRootIndex = index[rootVal];

    int leftSubtreeSize = inorderRootIndex - inorderStart;

    root->left = buildTreeHelper(inorder, postorder, inorderStart, inorderRootIndex - 1,
    postorderStart, postorderStart + leftSubtreeSize - 1, index);

    root->right = buildTreeHelper(inorder, postorder, inorderRootIndex + 1, inorderEnd,
    postorderStart + leftSubtreeSize, postorderEnd - 1, index);

    return root;
}

};

```

OUTPUT:

The screenshot displays a LeetCode submission for the problem "Construct Binary Tree from Inorder and Postorder Traversal". The submission is accepted, with a runtime of 4 ms (beating 51.65% of solutions) and a memory usage of 27.56 MB (beating 37.92%). The code is written in C++ and defines a `Solution` class with a `buildTree` method. The test case shows an inorder array `[9, 3, 15, 20, 7]` and a postorder array `[9, 15, 7, 20, 3]`.

```

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

```

Runtime: 4 ms | Beats 51.65%
Memory: 27.56 MB | Beats 37.92%

Accepted 202 / 202 testcases passed
Badal1802 submitted at Mar 03, 2025 10:37

Testcase 1: Accepted
Runtime: 0 ms

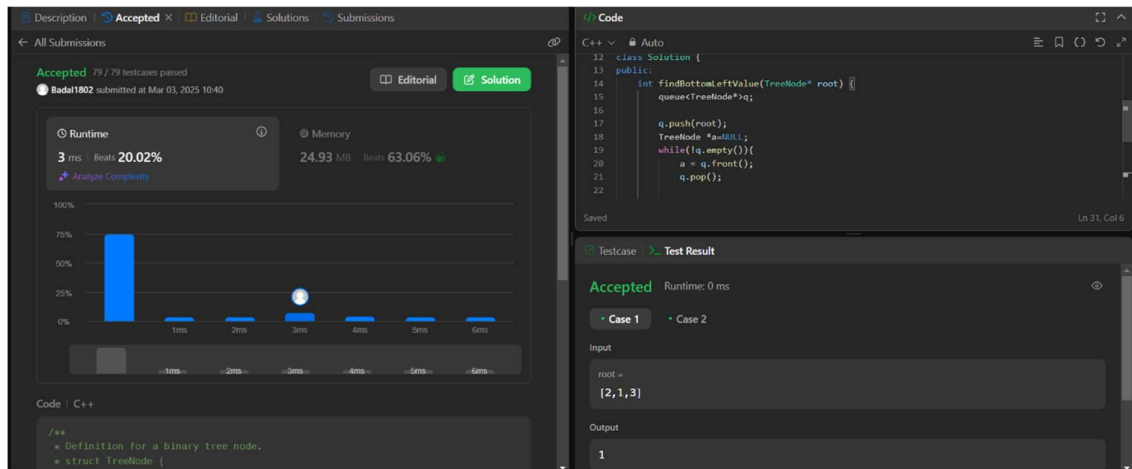
Case 1
Input:
inorder = [9, 3, 15, 20, 7]
postorder = [9, 15, 7, 20, 3]

513. Find Bottom Left Tree Value

CODE:

```
class Solution {  
public:  
    int findBottomLeftValue(TreeNode* root) {  
        queue<TreeNode*>q;  
        q.push(root);  
        TreeNode *a=NULL;  
        while(!q.empty()){  
            a = q.front();  
            q.pop();  
            if(a->right){  
                q.push(a->right);  
            }  
            if(a->left){  
                q.push(a->left);  
            }  
        }  
        return a->val;  
    }  
};
```

OUTPUT:



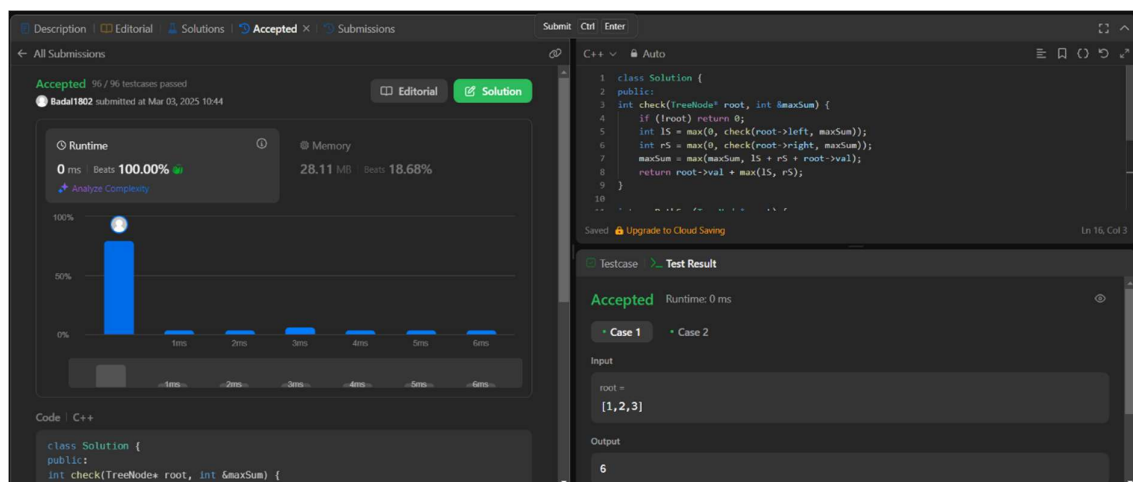
124. Binary Tree Maximum Path Sum

CODE:

```
class Solution {
public:
    int check(TreeNode* root, int &maxSum) {
        if (!root) return 0;
        int lS = max(0, check(root->left, maxSum));
        int rS = max(0, check(root->right, maxSum));
        maxSum = max(maxSum, lS + rS + root->val);
        return root->val + max(lS, rS);
    }

    int maxPathSum(TreeNode* root) {
        int maxSum = INT_MIN;
        check(root, maxSum);
        return maxSum;
    }
};
```

OUTPUT:



987. [Vertical Order Traversal of a Binary Tree](#)

CODE:

```
class Solution {
public:
    void inorder(TreeNode* root, int row, int col, multiset<pair<pair<int, int>, int>>& st)
    {
        if(root == nullptr) return;
        inorder(root->left, row+1, col-1, st);
        st.insert({{col, row}, root->val});
        inorder(root->right, row+1, col+1, st);
    }

    vector<vector<int>> verticalTraversal(TreeNode* root) {
        vector<vector<int>>ans;
        if(root == nullptr) return ans;
        multiset<pair<pair<int, int>, int>>st;
        inorder(root, 0, 0, st);

        auto it = st.begin();
        while(it != st.end())
        {
            vector<int>vec;
            int col = it->first.first;
            while(it != st.end() && it->first.first == col)
            {
                vec.push_back(it->second);
                it++;
            }
            ans.push_back(vec);
        }
        return ans;
    }
};
```



```
}  
};
```

OUTPUT:

