Priyanshu
22BCS16931
BCS_FL_IOT-603 (A)
Assignment-3

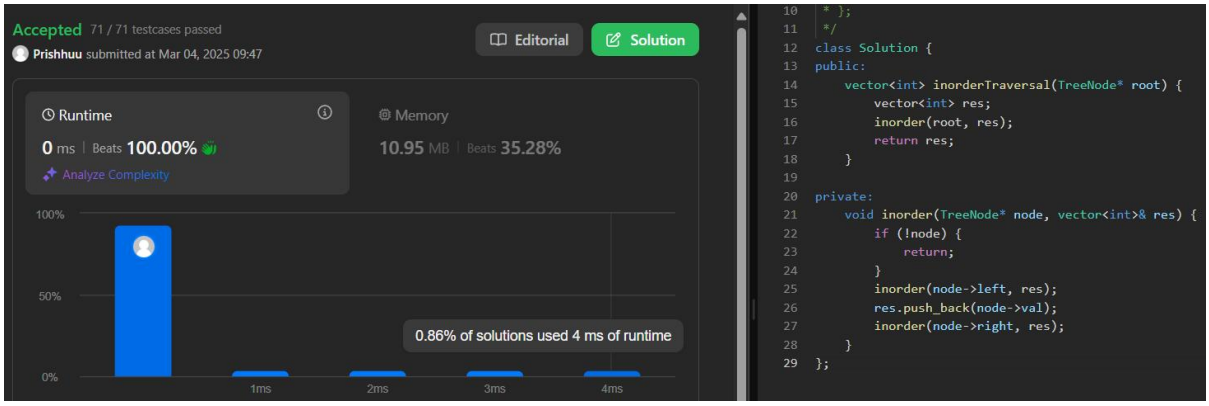
1. 94.Binary Tree Inorder Traversal

- **Solution:**

```cpp
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);
    }
};
```

- **Screenshot:**



2. 101.Symmetric Tree

- **Solution:**

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

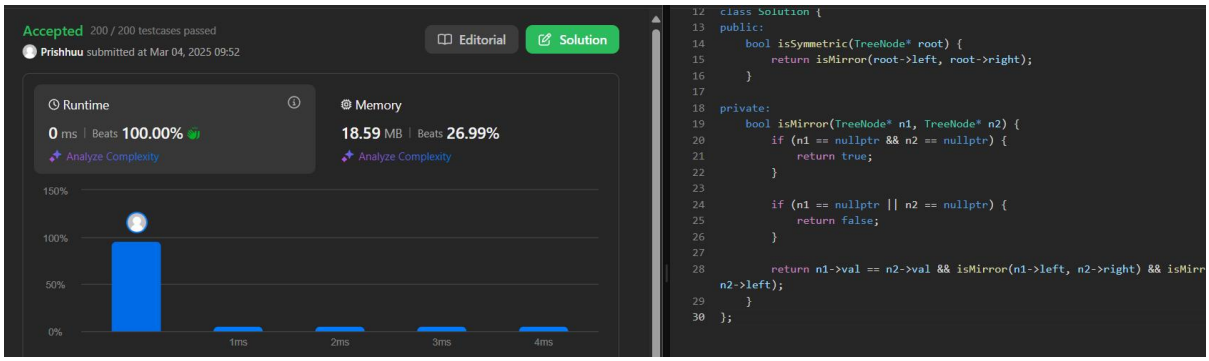
```
    }

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

        if (n1 == nullptr || n2 == nullptr) {
            return false;
        }

        return n1->val == n2->val && isMirror(n1->left, n2->right) && isMirror(n1->right, n2->left);
    }
};
```

- **Screenshot:**



3. 104.Maximum Depth of Binary Tree

- **Solution:**

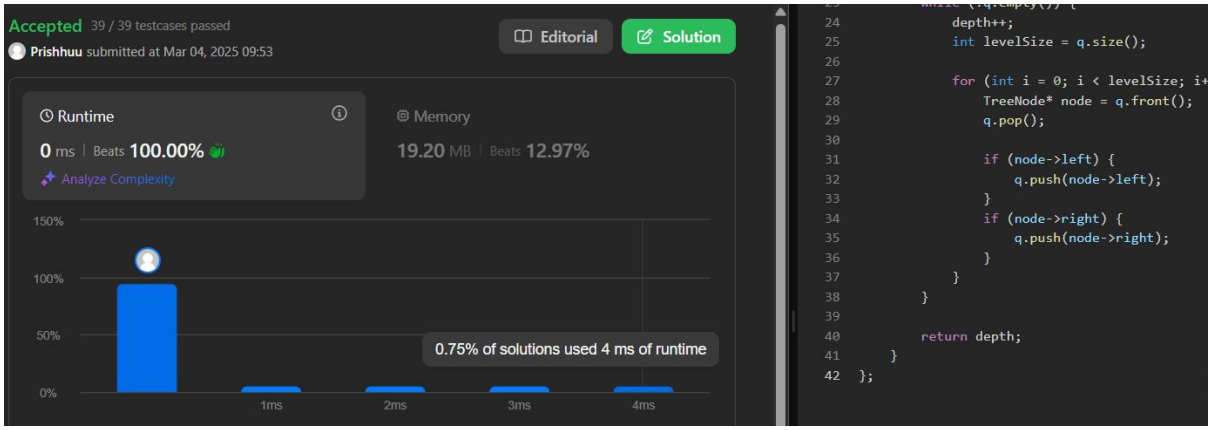
```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (!root) {
            return 0;
        }

        queue<TreeNode*> q;
        q.push(root);
        int depth = 0;
        while (!q.empty()) {
            depth++;
            int levelSize = q.size();
            for (int i = 0; i < levelSize; i++) {
                TreeNode* node = q.front();
                q.pop();
                if (node->left) {
                    q.push(node->left);
                }
                if (node->right) {
                    q.push(node->right);
                }
            }
```

```
        }

        return depth;
    }
};
```

- **Screenshot:**



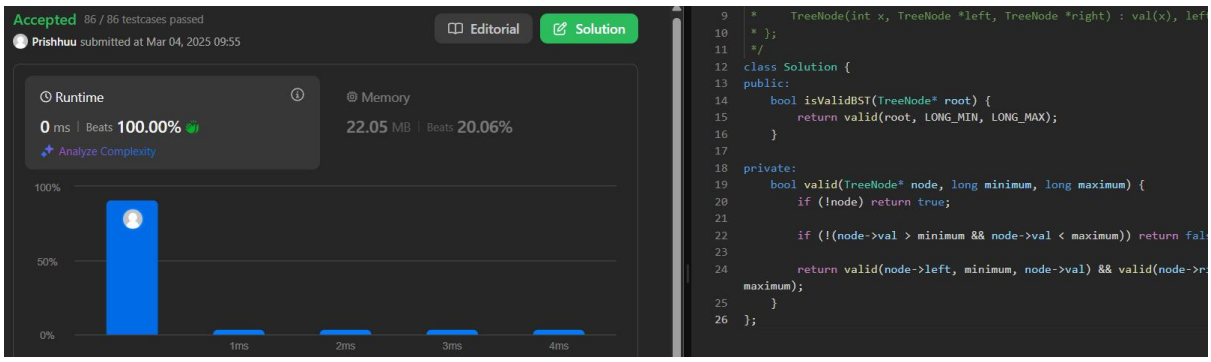
4. *98.Validate Binary Search Tree*

- **Solution:**

```
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);
    }
};
```

- **Screenshot:**



5. *230.Kth Smallest Element in a BST*

- **Solution:**

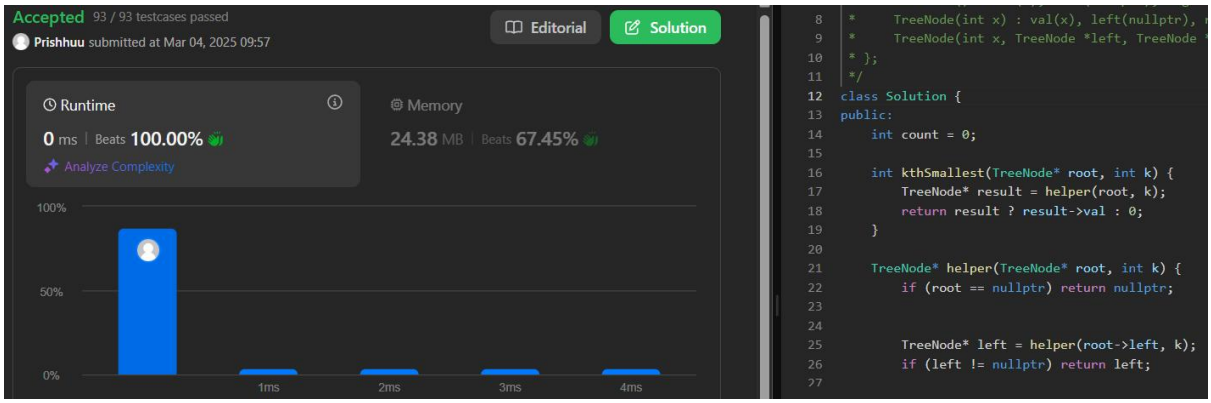
```cpp
class Solution {
public:
    int count = 0;

    int kthSmallest(TreeNode* root, int k) {
        TreeNode* result = helper(root, k);
        return result ? result->val : 0;
    }
    TreeNode* helper(TreeNode* root, int k) {
        if (root == nullptr) return nullptr;

        TreeNode* left = helper(root->left, k);
        if (left != nullptr) return left;
        count++;
        if (count == k) return root;

        return helper(root->right, k);
    }
};
```

- **Screenshot:**



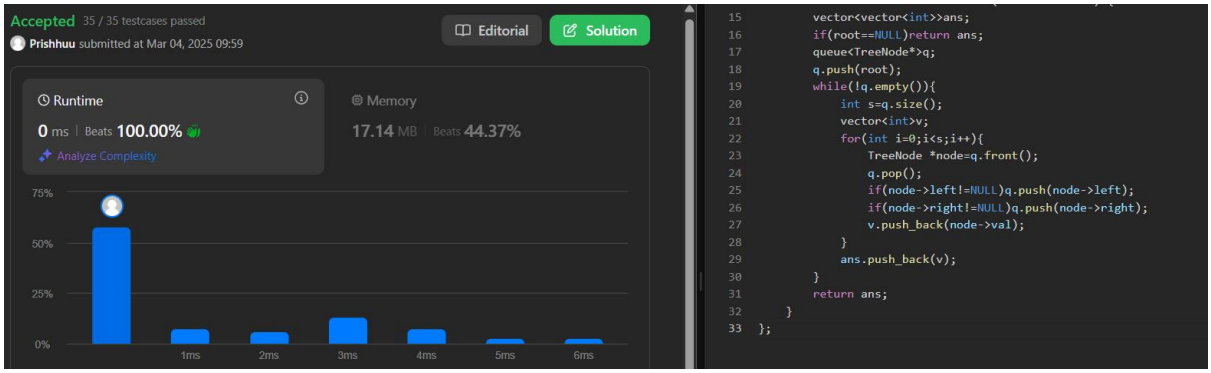
6. *102. Binary Tree Level Order Traversal*

- **Solution:**

```cpp
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL)return ans;
        queue<TreeNode*>q;
        q.push(root);
        while(!q.empty()){
            int s=q.size();
            vector<int>v;
            for(int i=0;i<s;i++){
                TreeNode *node=q.front();
```

```
                q.pop();
                if(node->left!=NULL)q.push(node->left);
                if(node->right!=NULL)q.push(node->right);
                v.push_back(node->val);
            }
            ans.push_back(v);
        }
        return ans;
    }
};
```

- **Screenshot:**



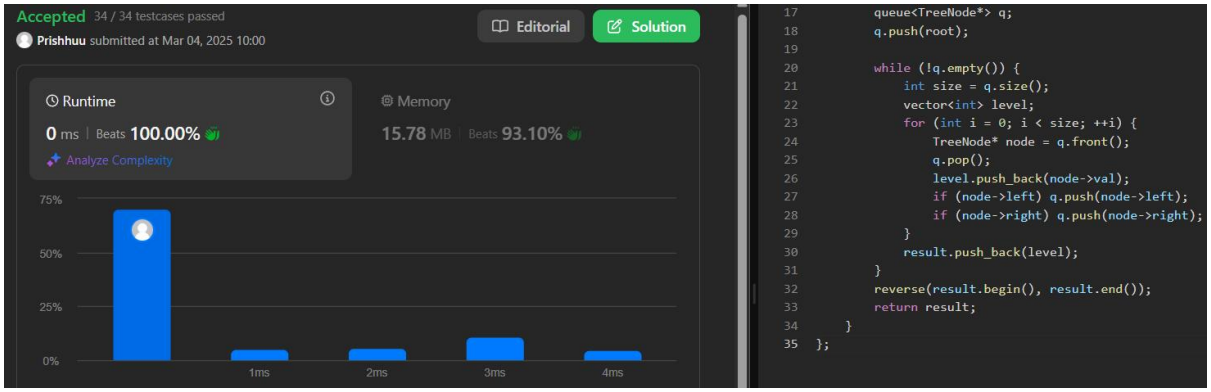
7. 107.Binary Tree Level Order Traversal II

- **Solution:**

```
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;
    }
};
```

- **Screenshot:**



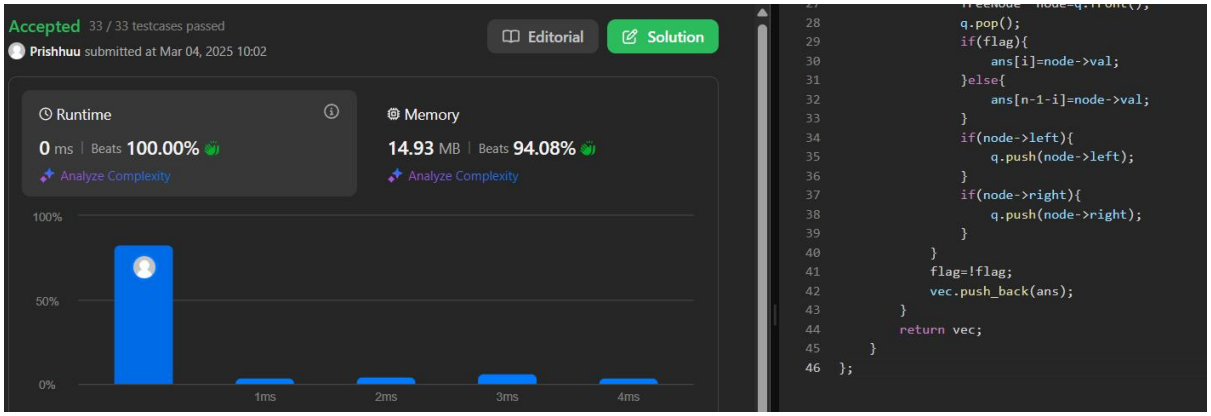
8. *103.Binary Tree Zigzag Level Order Traversal*

- **Solution:**

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>>vec;
        if(!root){
            return {};
        }
        queue<TreeNode*>q;
        q.push(root);
        bool flag=1;

        while(!q.empty()){
            int n=q.size();
            vector<int>ans(n);
            for(int i=0;i<n;i++){
                TreeNode* node=q.front();
                q.pop();
                if(flag){
                    ans[i]=node->val;
                }else{
                    ans[n-1-i]=node->val;
                }
                if(node->left){
                    q.push(node->left);
                }
                if(node->right){
                    q.push(node->right);
                }
            }
            flag=!flag;
            vec.push_back(ans);
        }
        return vec;
    }
};
```

- **Screenshot:**

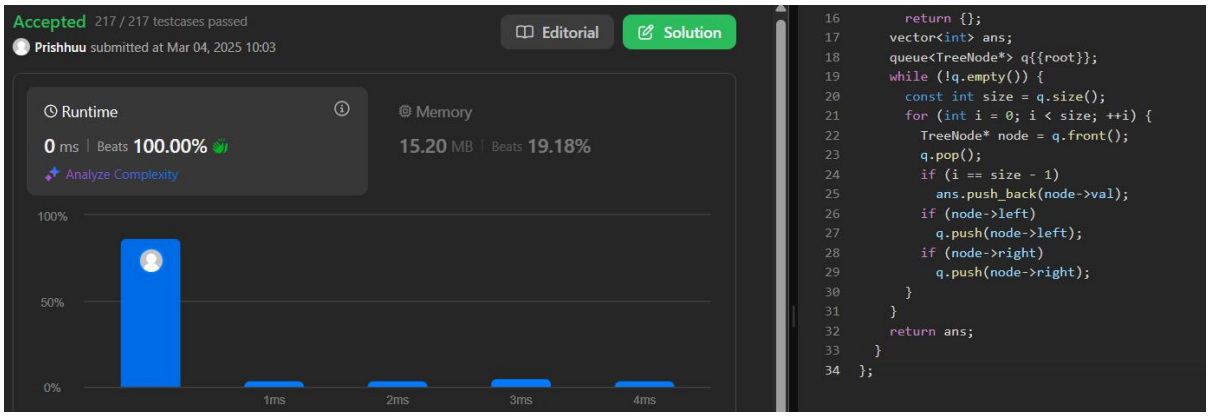


9. 199.Binary Tree Right Side View

- **Solution:**

```
class Solution {
 public:
  vector<int> rightSideView(TreeNode* root) {
    if (root == nullptr)
      return {};
    vector<int> ans;
    queue<TreeNode*> q{{root}};
    while (!q.empty()) {
      const int size = q.size();
      for (int i = 0; i < size; ++i) {
        TreeNode* node = q.front();
        q.pop();
        if (i == size - 1)
          ans.push_back(node->val);
        if (node->left)
          q.push(node->left);
        if (node->right)
          q.push(node->right);
      }
    }
    return ans;
  }
};
```

- **Screenshot:**



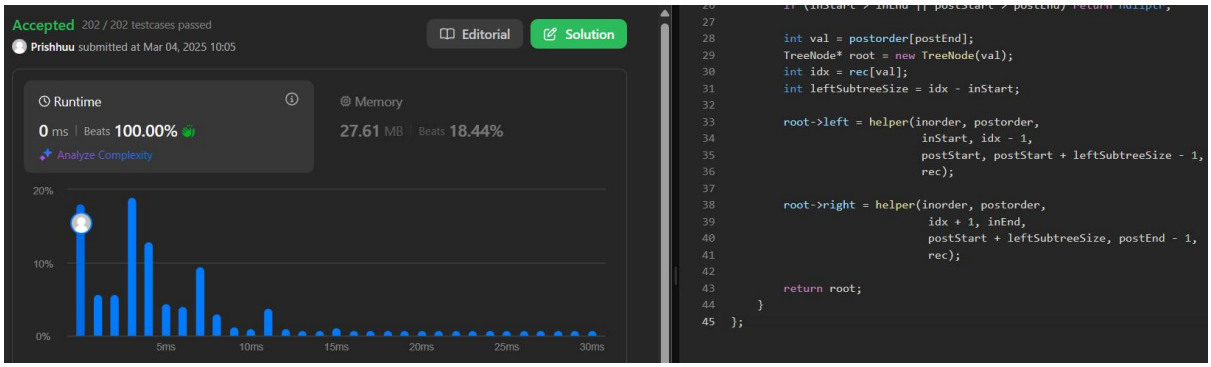
10. 106.Construct Binary Tree from Inorder and Postorder Traversal

- **Solution:**

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

    TreeNode* helper(vector<int>& inorder, vector<int>& postorder,
                     int inStart, int inEnd,
                     int postStart, int postEnd,
                     unordered_map<int, int>& rec) {
        if (inStart > inEnd || postStart > postEnd) return nullptr;
        int val = postorder[postEnd];
        TreeNode* root = new TreeNode(val);
        int idx = rec[val];
        int leftSubtreeSize = idx - inStart;
        root->left = helper(inorder, postorder,
                            inStart, idx - 1,
                            postStart, postStart + leftSubtreeSize - 1,
                            rec);
        root->right = helper(inorder, postorder,
                             idx + 1, inEnd,
                             postStart + leftSubtreeSize, postEnd - 1,
                             rec);
        return root;
    }
};
```

- **Screenshot:**



11. 513.Find Bottom Left Tree Value

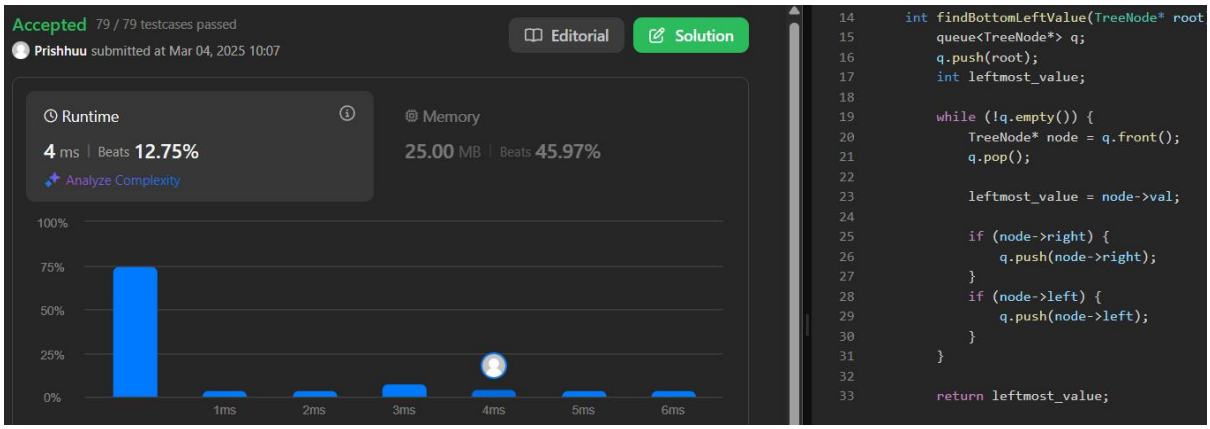
- **Solution:**

```cpp
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;
    }
};
```

- **Screenshot:**



12. 124. Binary Tree Maximum Path Sum

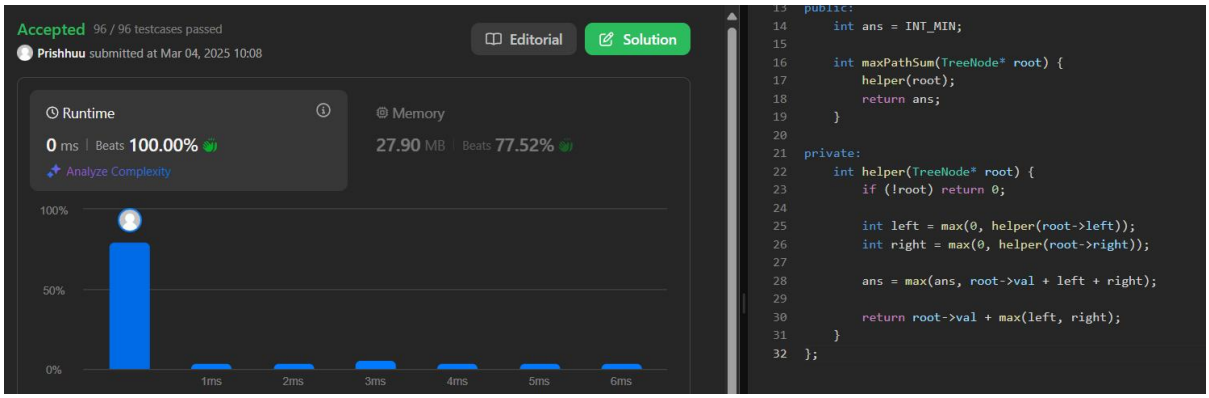
- **Solution:**

```
class Solution {
public:
    int ans = INT_MIN;

    int maxPathSum(TreeNode* root) {
        helper(root);
        return ans;
    }
private:
    int helper(TreeNode* root) {
        if (!root) return 0;

        int left = max(0, helper(root->left));
        int right = max(0, helper(root->right));
        ans = max(ans, root->val + left + right);
        return root->val + max(left, right);
    }
};
```

- **Screenshot:**




```
public:
    int ans = INT_MIN;

    int maxPathSum(TreeNode* root) {
        helper(root);
        return ans;
    }

private:
    int helper(TreeNode* root) {
        if (!root) return 0;

        int left = max(0, helper(root->left));
        int right = max(0, helper(root->right));

        ans = max(ans, root->val + left + right);

        return root->val + max(left, right);
    }
};
```