Assingment -2

Student Name: Subhojit Ghosh **UID: 22BCS15368**

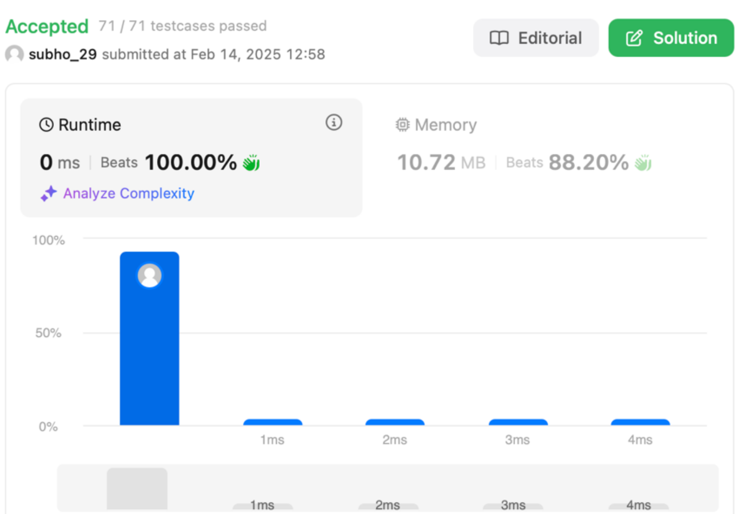
Branch: CSE **Section/Group:FL-602-A**

Semester: 6 **Date of Performance:14..2025**

Subject Name: Advanced Programming **Subject Code: 22CSH-359**

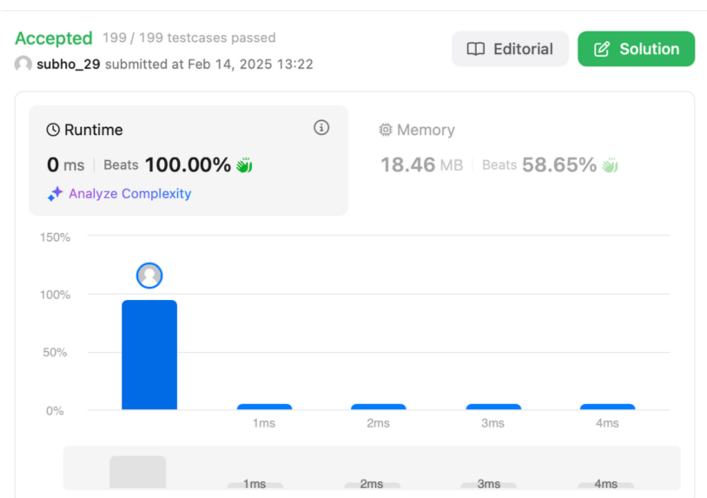
1. .Binary Tree Inorder Traversal

```
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);
    }
};
```



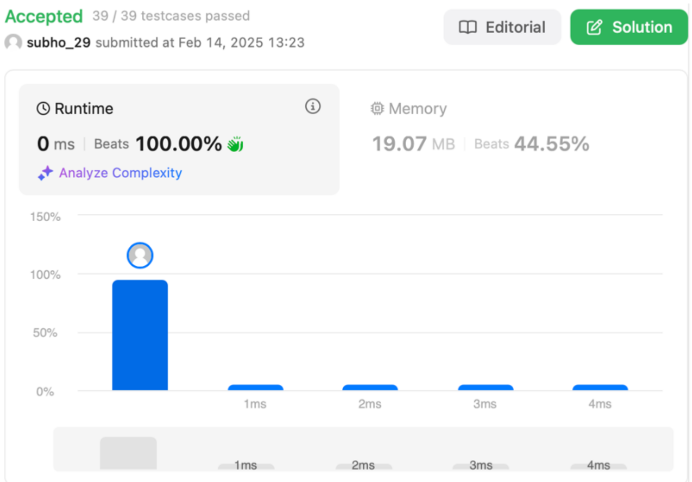
2. .[Symmetric Tree]

```
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);
}

};
```



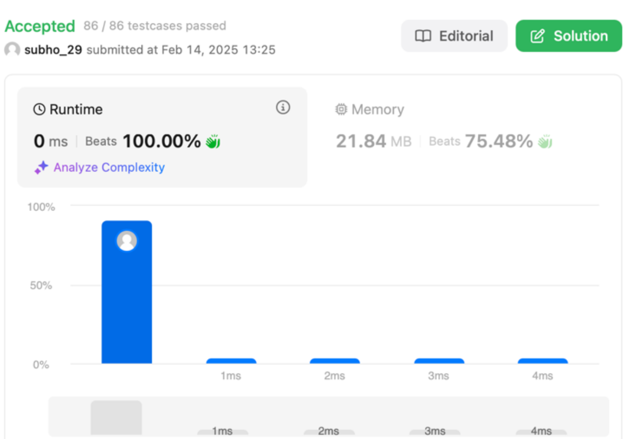
3. .[Maximum Depth of Binary Tree]

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



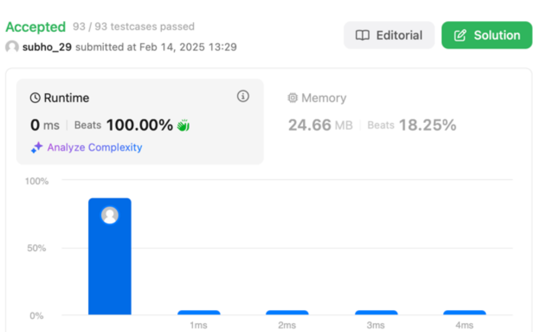
4. [Validate Binary Search Tree]

```
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);
    }
};
```



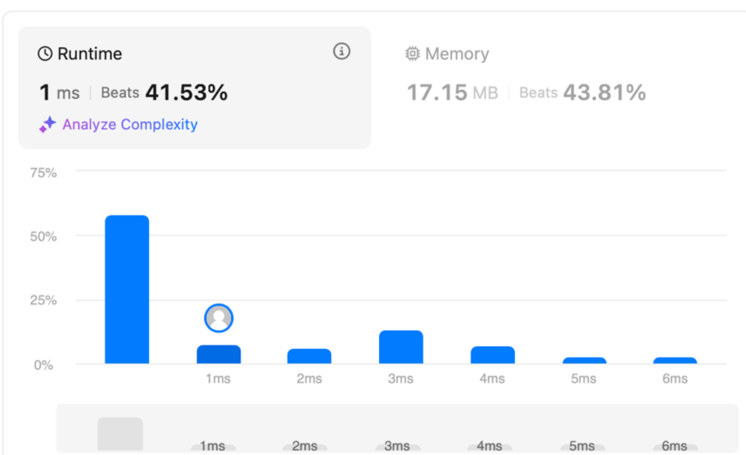
5. [Kth Smallest Element in a BST]

```cpp
#include <bits/stdc++.h>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    int count = 0; // Counter for visited nodes

    int kthSmallest(TreeNode* root, int k) {
        TreeNode* result = helper(root, k);
        return result ? result->val : 0; // Return value or 0 if not found
    }

    TreeNode* helper(TreeNode* root, int k) {
        if (root == nullptr) return nullptr;

        // Traverse left subtree
        TreeNode* left = helper(root->left, k);
        if (left != nullptr) return left; // If found in left subtree

        count++; // Increment count for current node
        if (count == k) return root; // Found k-th smallest

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



6. [Binary Tree Level Order Traversal]

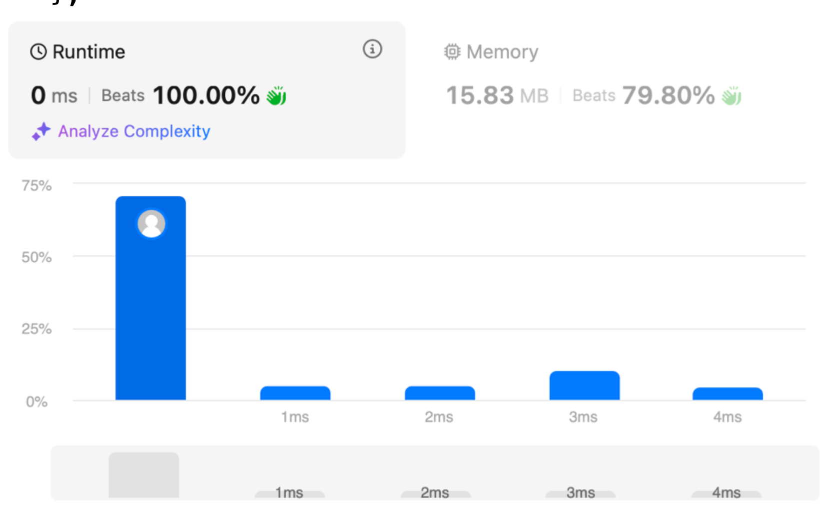
```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL)return ans;
        queue<TreeNode*>q;
        q.push(root);
        while(!q.empty()){
            int s=q.size();
            vector<int>v;
            for(int i=0;i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



7. [Binary Tree Level Order Traversal II]

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

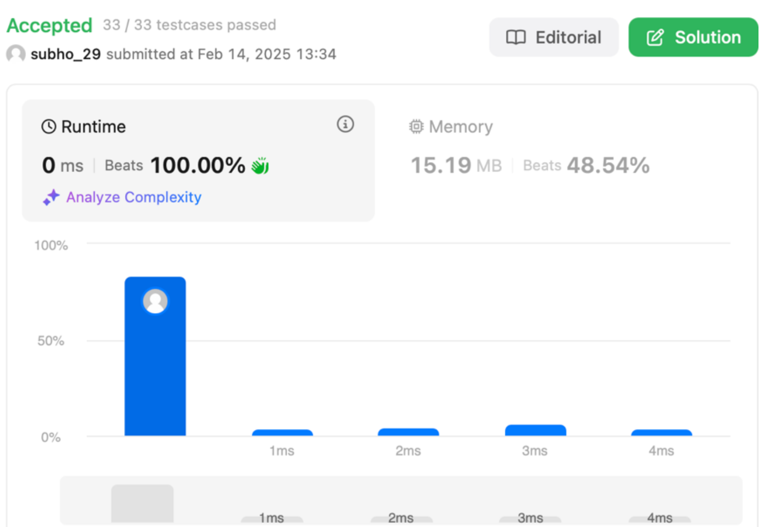
```
            queue<TreeNode*> q;
            q.push(root);

            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;
        }
    };
```



8. [Binary Tree Zigzag Level Order Traversal]()

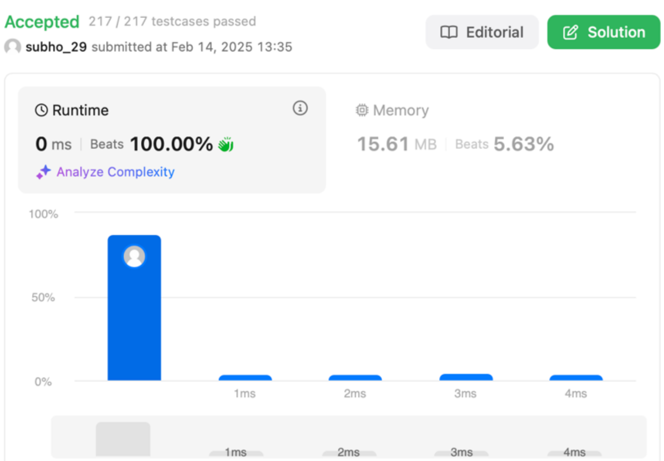
```
class Solution {
 public:
  vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
    if (root == nullptr)
      return {};
    vector<vector<int>> ans;
    deque<TreeNode*> dq{{root}};
    bool isLeftToRight = true;
    while (!dq.empty()) {
```

```
      vector<int> currLevel;
      for (int sz = dq.size(); sz > 0; --sz)
        if (isLeftToRight) {
          TreeNode* node = dq.front();
          dq.pop_front();
          currLevel.push_back(node->val);
          if (node->left)
            dq.push_back(node->left);
          if (node->right)
            dq.push_back(node->right);
        } else {
          TreeNode* node = dq.back();
          dq.pop_back();
          currLevel.push_back(node->val);
          if (node->right)
            dq.push_front(node->right);
          if (node->left)
            dq.push_front(node->left);
        }
      ans.push_back(currLevel);
      isLeftToRight = !isLeftToRight;
    }
    return ans;
  }
};
```



9. [Binary Tree Right Side View]

```
class Solution {
public:
    vector<int> res;
    unordered_map<int,int> mp;
    void check(TreeNode* root,int n){
        if(!root){
            return;
        }
        if(!(mp.find(n) != mp.end())){
            res.push_back(root->val);
            mp[n]++;
        }
        check(root->right,n+1);
        check(root->left,n+1);
    }
    vector<int> rightSideView(TreeNode* root) {
      check(root,0);
      return res;
    }
};
```



10. [Construct Binary Tree from Inorder and Postorder Traversal]

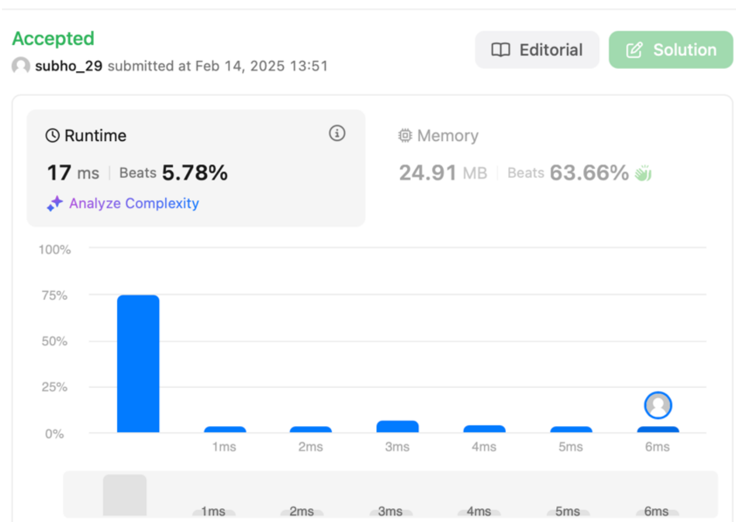
```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> index;
        for (int i = 0; i < inorder.size(); i++) {
            index[inorder[i]] = i;
        }
        return buildTreeHelper(inorder, postorder, 0, inorder.size()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

11. [Find Bottom Left Tree Value]()

```
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();
```

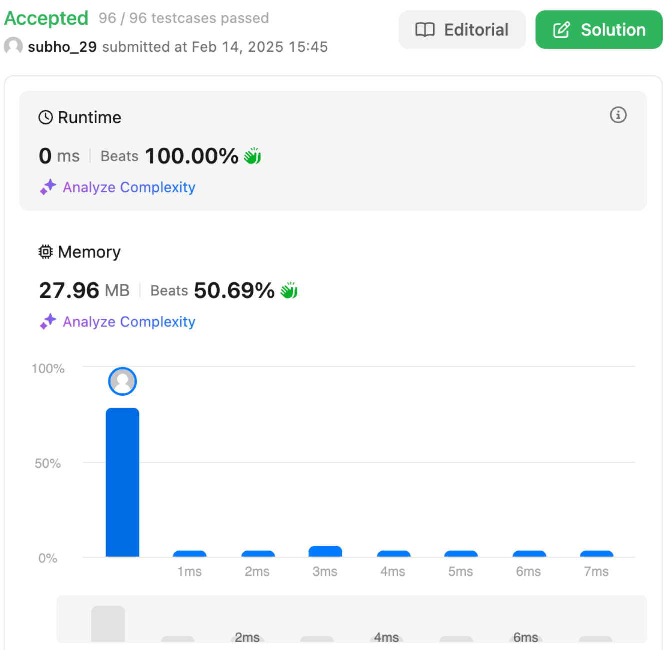
```
                leftmost_value = node->val;

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

            return leftmost_value;
        }
    };
```



12. [Binary Tree Maximum Path Sum]()

```
class Solution {
 public:
  int maxPathSum(TreeNode* root) {
    int ans = INT_MIN;
    maxPathSumDownFrom(root, ans);
    return ans;
  }

 private:
  int maxPathSumDownFrom(TreeNode* root, int& ans) {
```

```
        if (root == nullptr)
          return 0;
        const int l = max(0, maxPathSumDownFrom(root->left, ans));
        const int r = max(0, maxPathSumDownFrom(root->right, ans));
        ans = max(ans, root->val + l + r);
        return root->val + max(l, r);
      }
    };
```



13. [Vertical Order Traversal of a Binary Tree]

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        map<int,map<int,multiset<int>>>nodes;
        queue<pair<TreeNode*,pair<int,int>>>q;
        q.push({root,{0,0}});
        while(!q.empty()){
            auto t = q.front();
            q.pop();
            TreeNode* a = t.first;
            int x =t.second.first, y = t.second.second;
            nodes[x][y].insert(a->val);
            if(a->left){
```

```
                q.push({a->left,{x-1,y+1}});
            }
            if(a->right){
                q.push({a->right,{x+1,y+1}});
            }

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

Accepted 34 / 34 testcases passed

subho_29 submitted at Feb 14, 2025 15:47

Editorial Solution

Runtime

1 ms | Beats 60.15%

Analyze Complexity

Memory

16.31 MB | Beats 46.72%