Assignment -2

Student Name: Anirudh Gautam **UID: 22BCS12994**

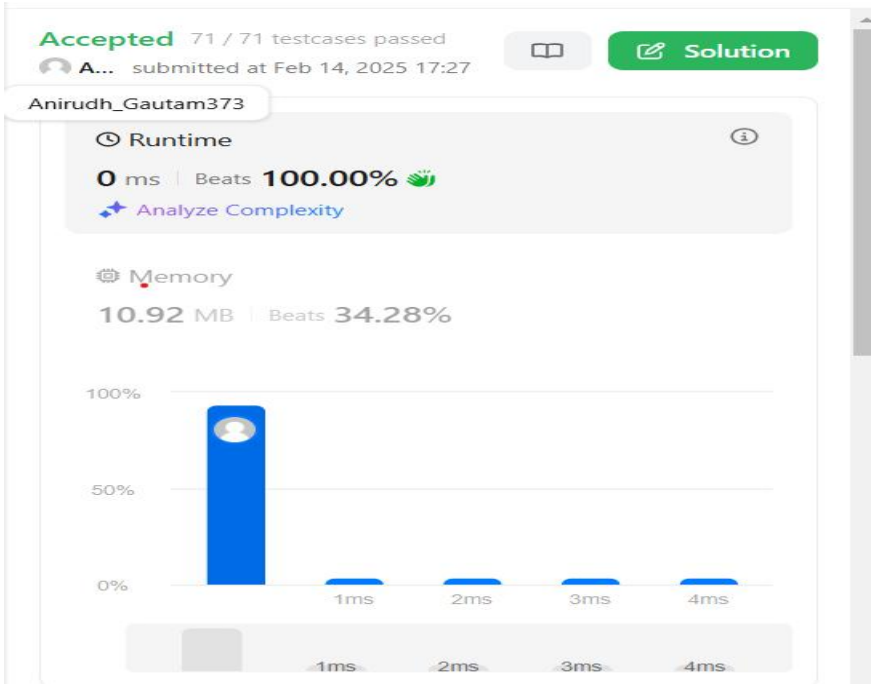
Branch: CSE **Section/Group:FL-602-A**

Semester: 6 **Date of Performance:14.02.2025**

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

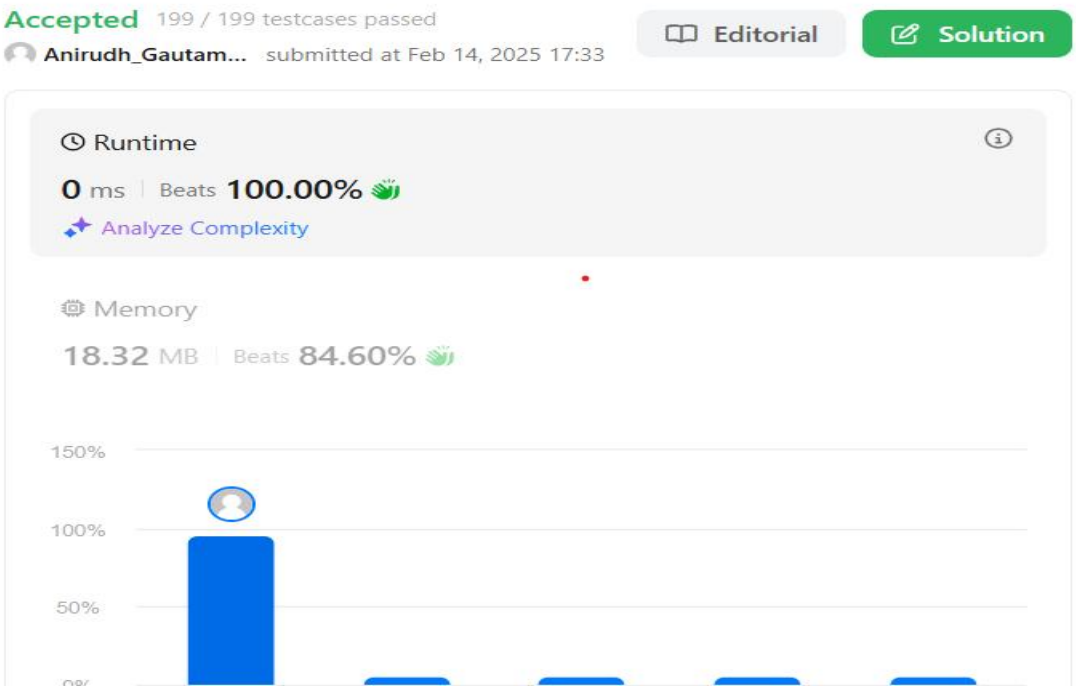
1..Binary Tree Inorder Traversal

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int>res;
        DFS(root,res);
        return res;
    }
private:
    void DFS(TreeNode* r, vector<int>& res)
    {
        if(r==NULL)
        return;
        DFS(r->left,res);
        res.push_back(r->val);
        DFS(r->right,res);
    }
};
```



.Symmetric Tree

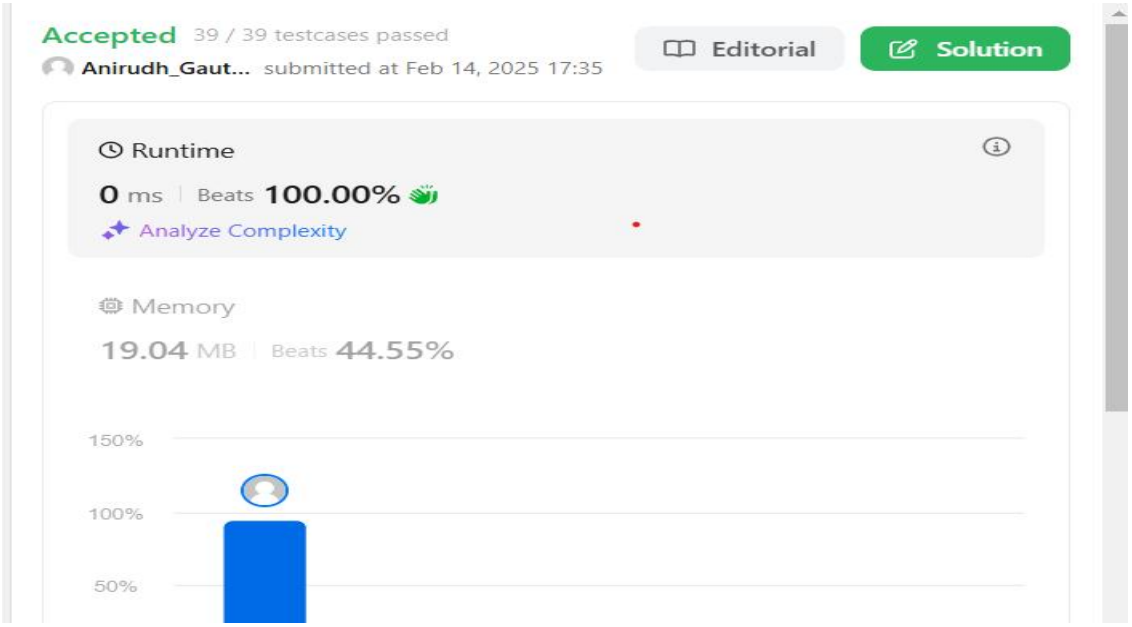
```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return same(root->left,root->right);
    }
private:
    bool same(TreeNode*l,TreeNode*r)
    {
        if(l==nullptr && r==nullptr)
        return true;
        if(l==nullptr || r==nullptr)
        return false;
        if(l->val != r->val)
        return false;
        return same(l->left,r->right) && same(l->right,r->left);
    }
};
```



2..Maximum Depth of Binary Tree

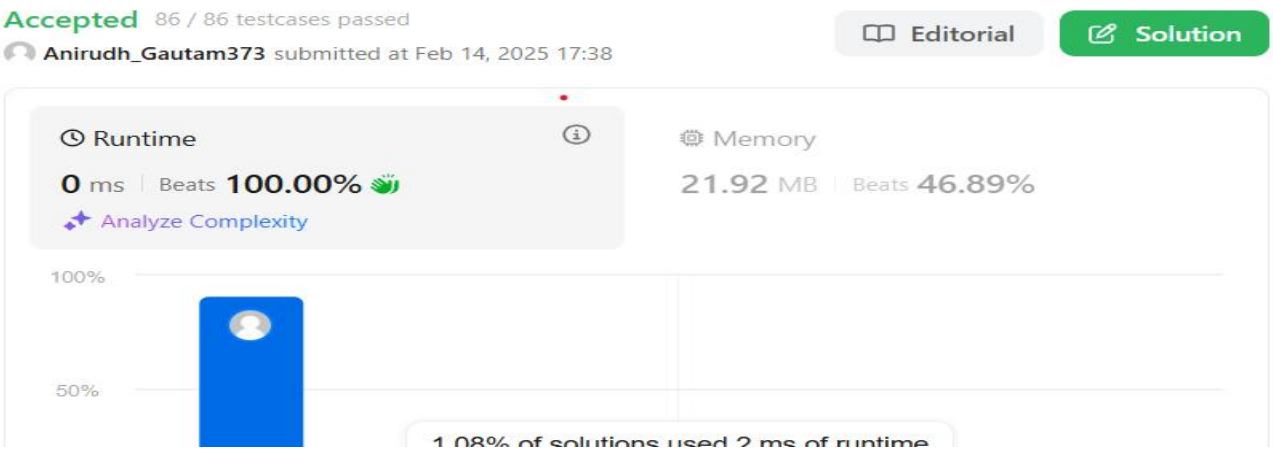
```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        return solve(root);
    }
    int solve(TreeNode *root){
        if(root ==NULL){
            return 0;
```

```
        }
        return 1+ max(solve(root->left),solve(root->right));
    }
};
```



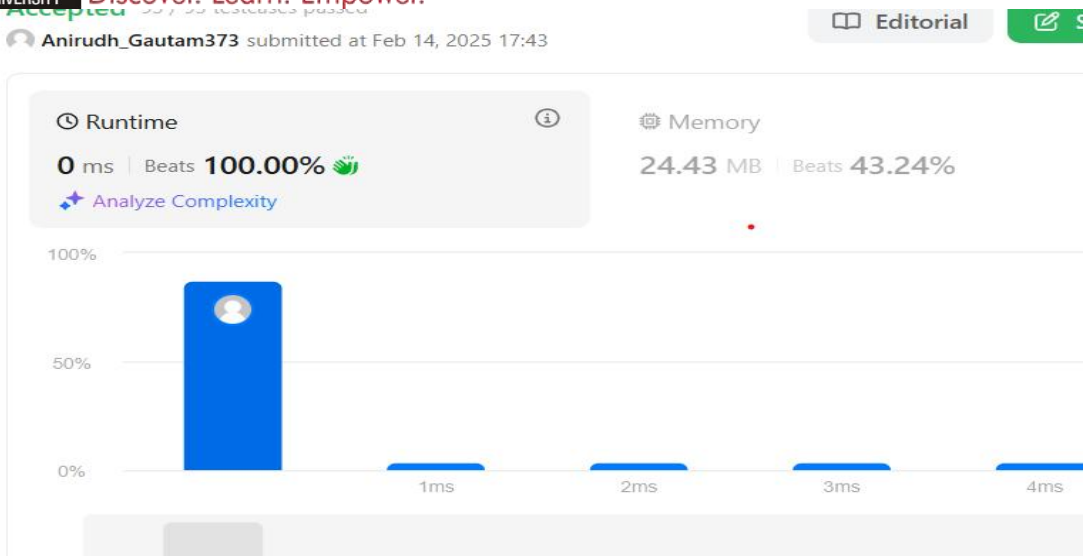
3. Validate Binary Search Tree

```
class Solution {

bool isPossible(TreeNode* root, long long l, long long r){
    if(root == nullptr)  return true;
    if(root->val < r and root->val > l)
        return isPossible(root->left, l, root->val) and
                          isPossible(root->right, root->val, r);
    else return false;
}

public:
    bool isValidBST(TreeNode* root) {
        long long int min = -1000000000000, max = 1000000000000;
        return isPossible(root, min, max);
    }
};
```



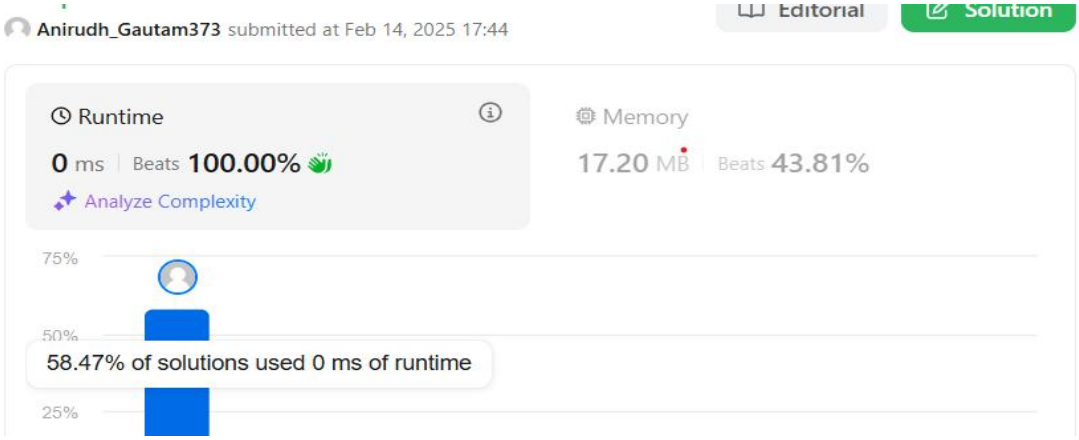
4. Kth Smallest Element in a BST

```
class Solution {
public:
    pair<int,int> kthSmall(TreeNode* root,int k){
        if(root == NULL)    return make_pair(-1,0);
        pair<int,int> pr = kthSmall(root->left,k);
        if(pr.first!=-1)    return pr;
        else k -= pr.second;
        if(k==1)    {
            pr.first = root->val;
            return pr;
        }
        pair<int,int> p2 = kthSmall(root->right,k-1);
        p2.second += 1 + pr.second;
        return p2;
    }
    int kthSmallest(TreeNode* root, int k) {
        return kthSmall(root,k).first;
    }
};
```



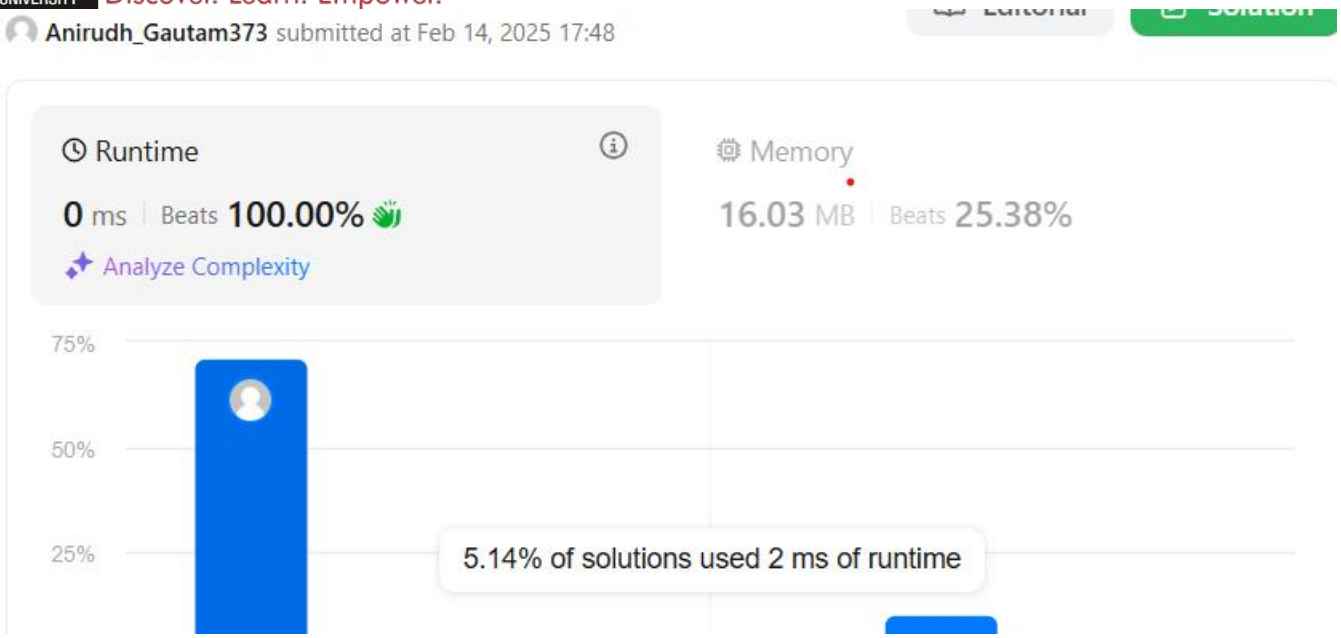
5. Binary Tree Level Order Traversal

```
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;
}
};
```



6. Binary Tree Level Order Traversal II

```
  class Solution {
public:
   vector<vector<int>> levelOrderBottom(TreeNode* root) {
      vector<vector<int>> ans;

      if(root == NULL){
         return ans;
      }

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

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

         for(int i=0;i<size;i++){
            TreeNode* curr = Q.front();
            Q.pop();

            level.push_back(curr->val);

            if(curr->left != NULL){
               Q.push(curr->left);
            }

            if(curr->right != NULL){
               Q.push(curr->right);
            }
         }
         ans.push_back(level);
      }
      reverse(ans.begin(), ans.end());
      return ans;
   }
};
```



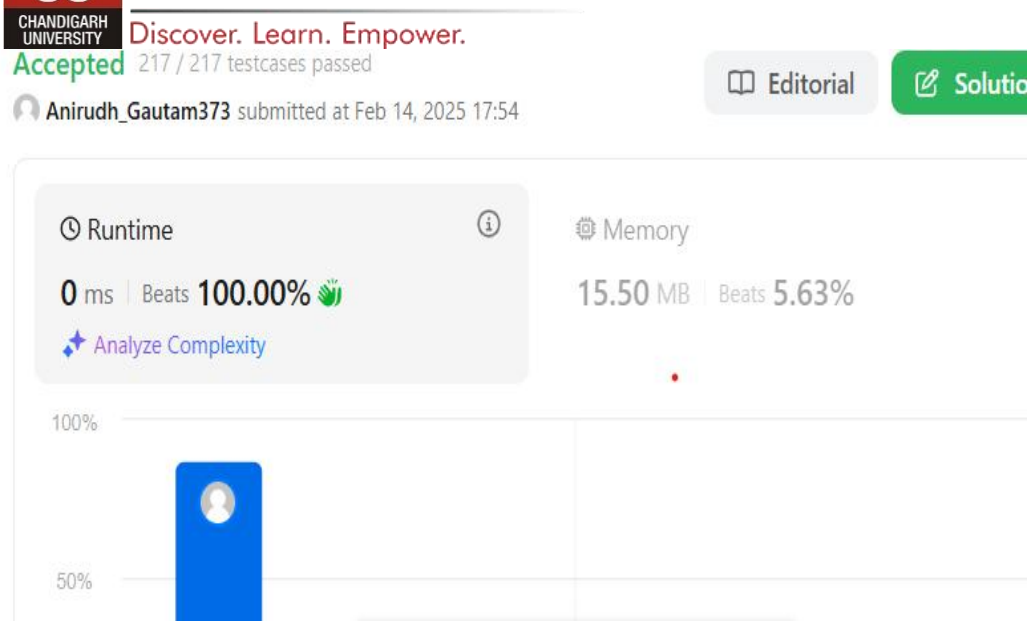
7. Binary Tree Zigzag Level Order Traversal

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

        while (!q.empty()) {
            int levelSize = q.size();
            vector<int> level(levelSize);
            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop();
                int index = leftToRight ? i : (levelSize - 1 - i);
                level[index] = node->val;

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            leftToRight = !leftToRight;
```

```
            result.push_back(level);
        }

        return result;
    }
};
```

8. Binary Tree Right Side View

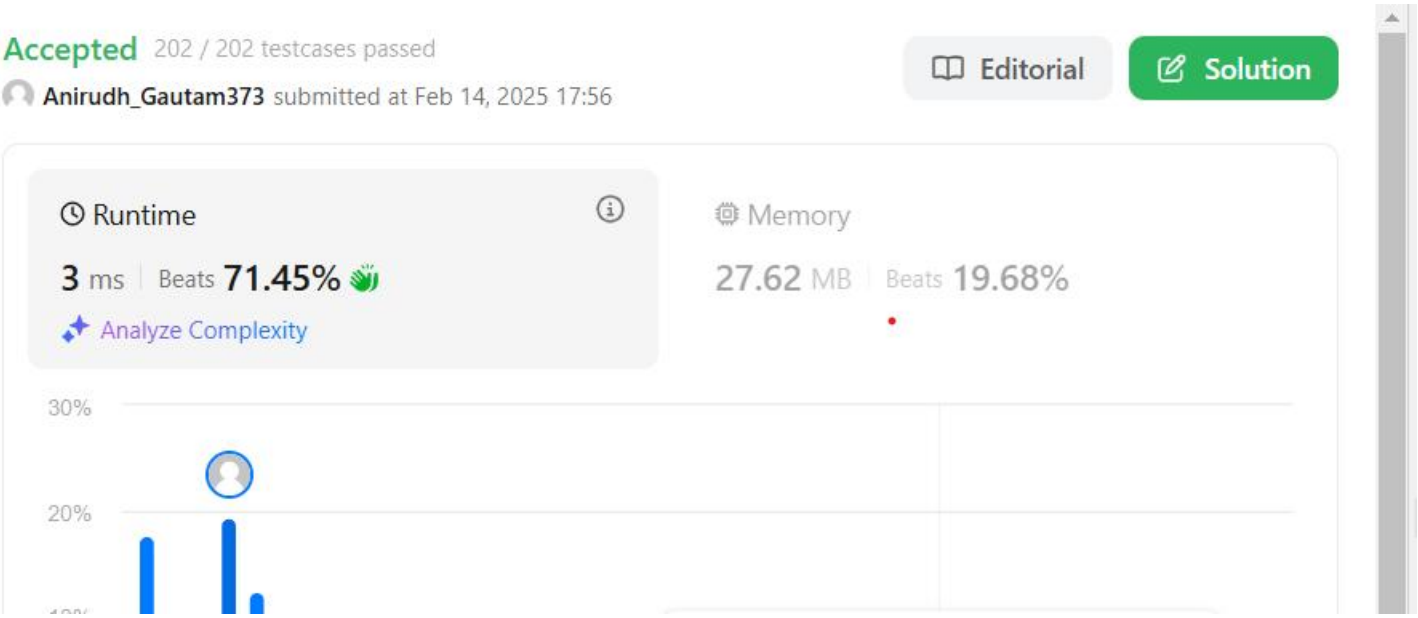
```
class Solution

{ 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;
}
};
```



9. Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {
public:
    void fillMap(unordered_map<int, int> &mapp, vector<int> &inorder, int size) {
        for (int i = 0; i < size; i++) {
            mapp[inorder[i]] = i;
        }
    }

    TreeNode* solve(vector<int>& inorder, vector<int>& postorder, int &postorderIndex, int inorderStart, int
      inorderEnd, int size, unordered_map<int, int> &mapp) {
        if (postorderIndex < 0 || inorderStart > inorderEnd) {
            return nullptr;
        }

        //node
        int element = postorder[postorderIndex];
        TreeNode* root = new TreeNode(element);
        postorderIndex--;

        int inorderIndex = mapp[element];

        // right
        root->right = solve(inorder, postorder, postorderIndex, inorderIndex + 1, inorderEnd, size, mapp);

        // left
        root->left = solve(inorder, postorder, postorderIndex, inorderStart, inorderIndex - 1, size, mapp);

        return root;
    }
```

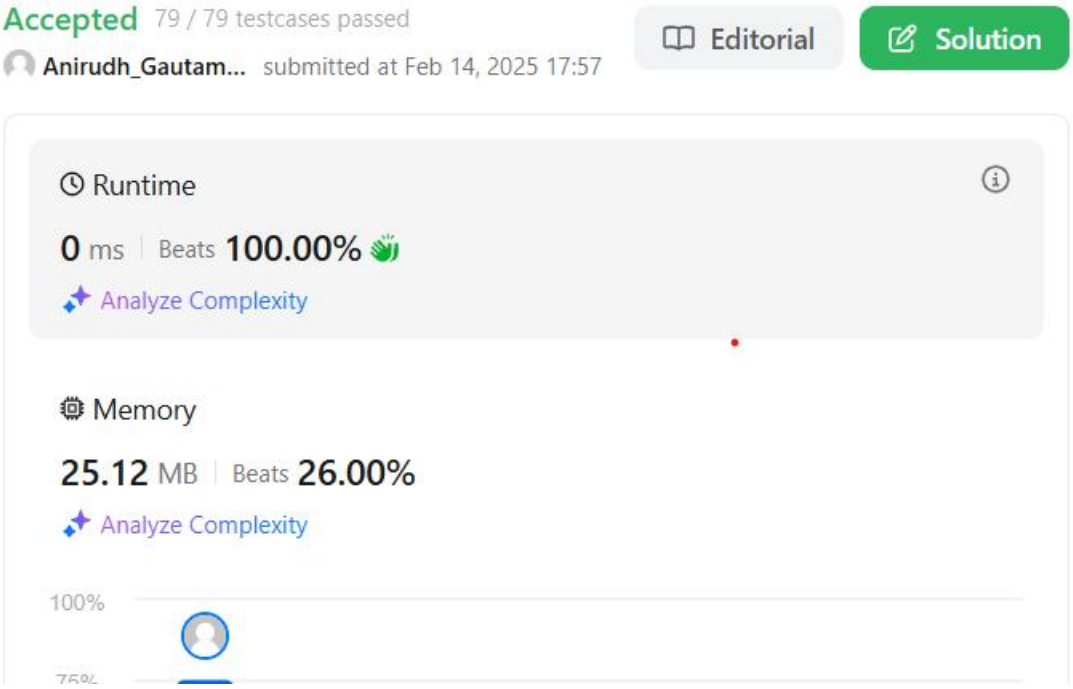
```
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        int size = inorder.size();
        int postorderIndex = size - 1;
        int inorderStart = 0;
        int inorderEnd = size - 1;
        unordered_map<int, int> mapp;
        fillMap(mapp, inorder, size);

        TreeNode* ans = solve(inorder, postorder, postorderIndex, inorderStart, inorderEnd, size, mapp);
        return ans;
    }
};
```



10. 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();
```

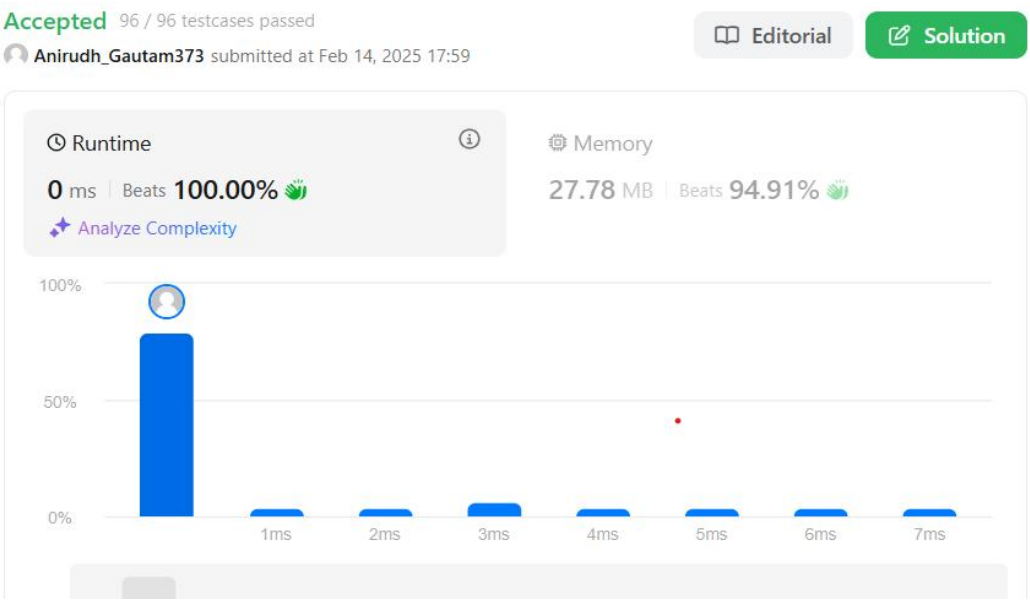
```
leftmost_value = node->val; if (node->right) {
q.push(node->right);
}
if (node->left) { q.push(node->left);
}
}

return leftmost_value;
}
};
```



11. 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);
}
};
```



12. 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;
}
};
```