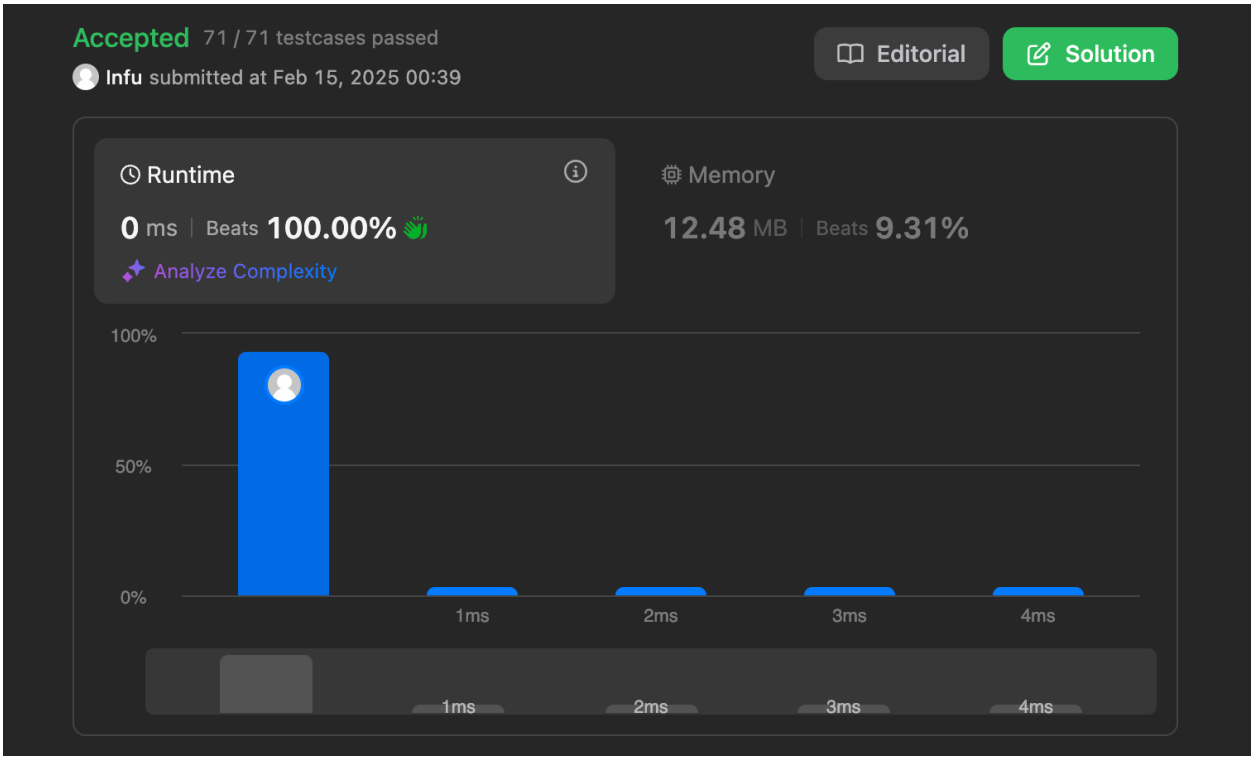
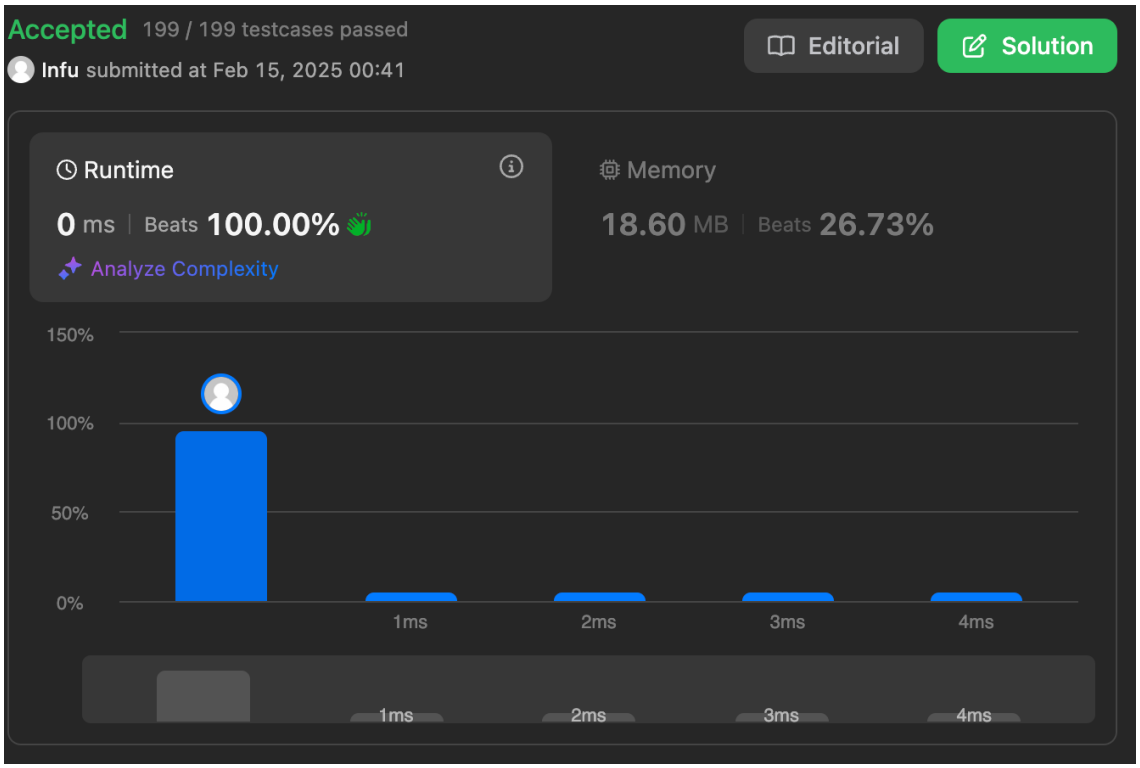
94. Binary Tree Inorder Traversal

```
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;
     }
};
```



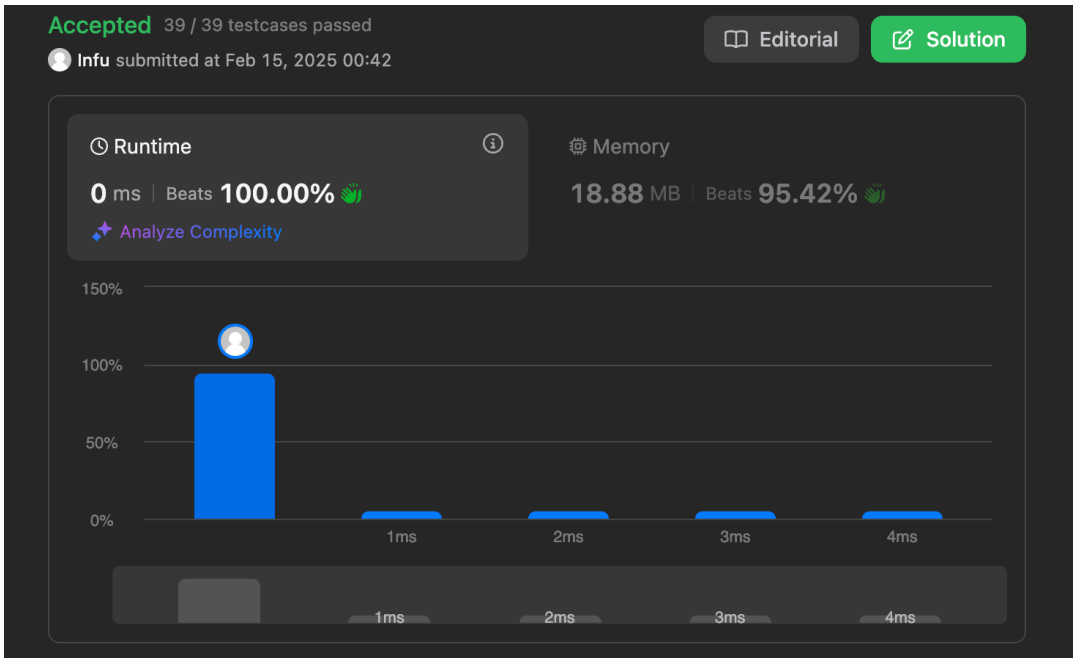
101. Symmetric Tree

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {

        if(root==NULL) return true; //Tree is empty

        return isSymmetricTest(root->left,root->right);
    }

    bool isSymmetricTest(TreeNode* p , TreeNode* q){
        if(p == NULL && q == NULL) //left & right node is NULL
            return true;

        else if(p == NULL || q == NULL) //one of them is Not NULL
            return false;

        else if(p->val!=q->val)
            return false;

        return isSymmetricTest(p->left,q->right) && isSymmetricTest(p->right,q->left);
    }
};
```



104. Maximum Depth of Binary Tree

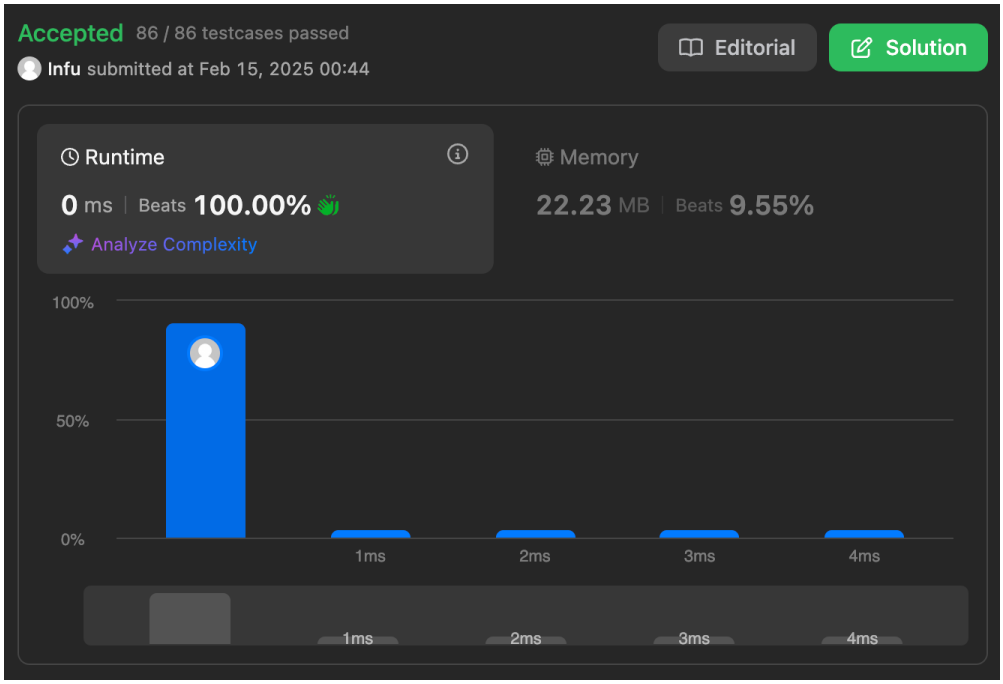
```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(!root) return 0;
        return 1 + max(maxDepth(root->left), maxDepth(root->right));
    }
};
```



98. Validate Binary Search Tree

```
class Solution {
public:
    void inorder(TreeNode *root,vector<int>&ans)
    {
        if (root == NULL)
            return;
        inorder(root->left,ans);
        ans.push_back(root->val);
        inorder(root->right,ans);
    }
    bool isValidBST(TreeNode* root)
    {
        vector<int>ans;
        inorder(root,ans);
        for(int i=1;i<ans.size();i++)
```

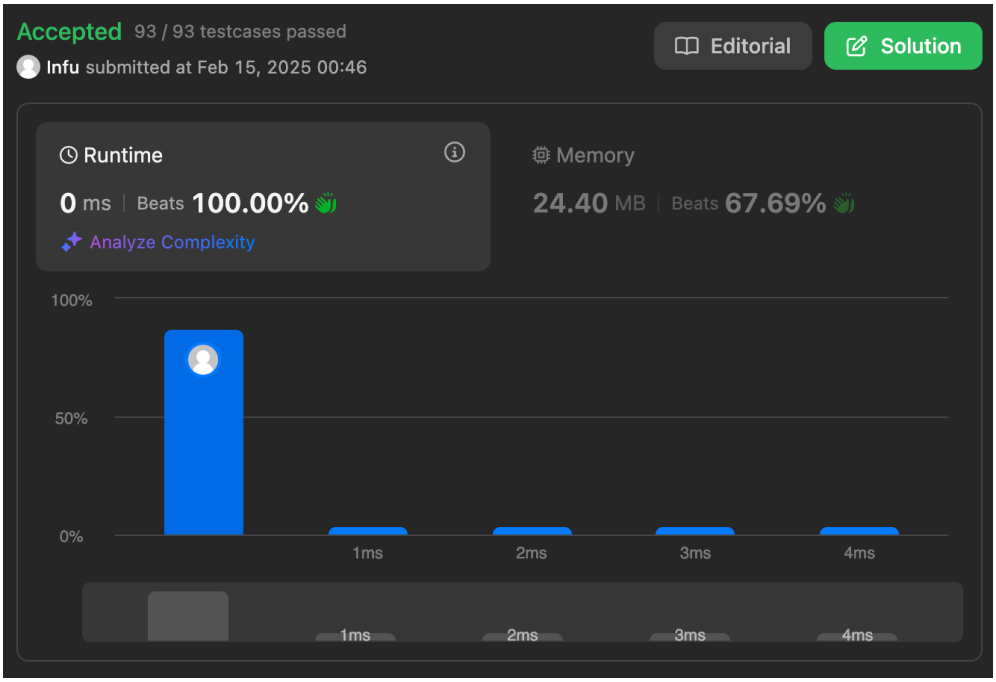
```
        {
            if(ans[i]<=ans[i-1])
            {
                return false;
            }
        }
        return true;
    }
};
```



230. Kth Smallest Element in a BST

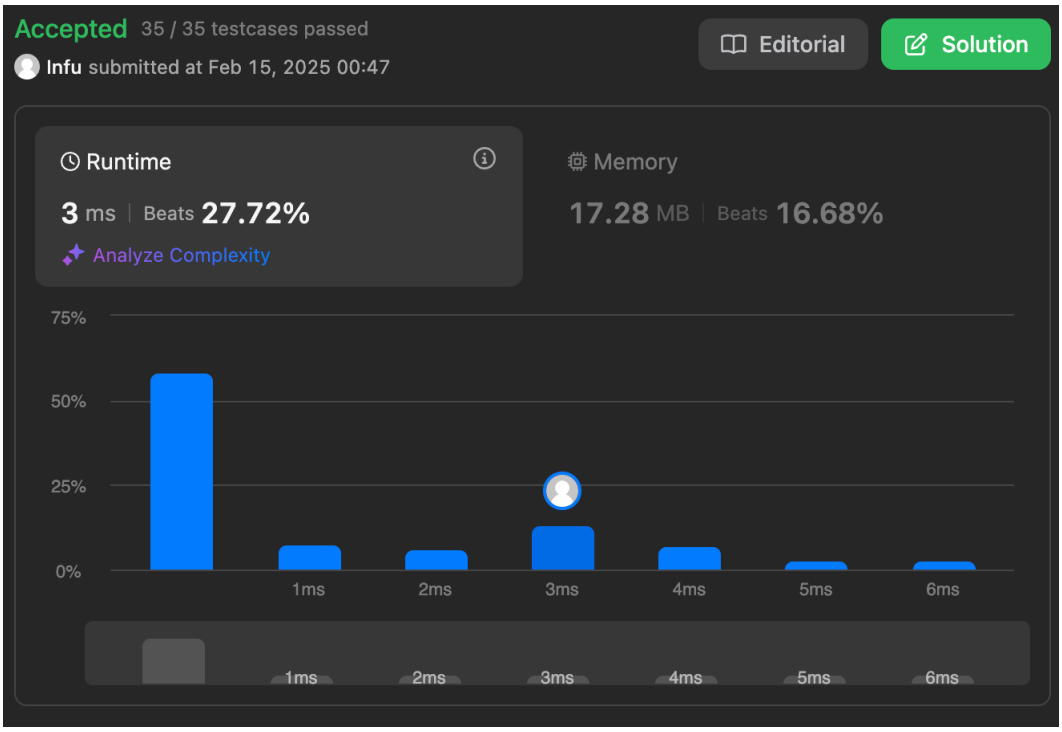
```
class Solution {
public:
    void solve(TreeNode* root, int &cnt, int &ans, int k){
        if(root == NULL)    return;
        //left, root, right
        solve(root->left, cnt, ans, k);
```

```
        cnt++;
        if(cnt == k){
            ans = root->val;
            return;
        }
        solve(root->right, cnt, ans, k);
    }
    int kthSmallest(TreeNode* root, int k) {

        int cnt = 0;
        int ans;
        solve(root, cnt, ans, k);
        return ans;
    }
};
```



102. Binary Tree Level Order Traversal

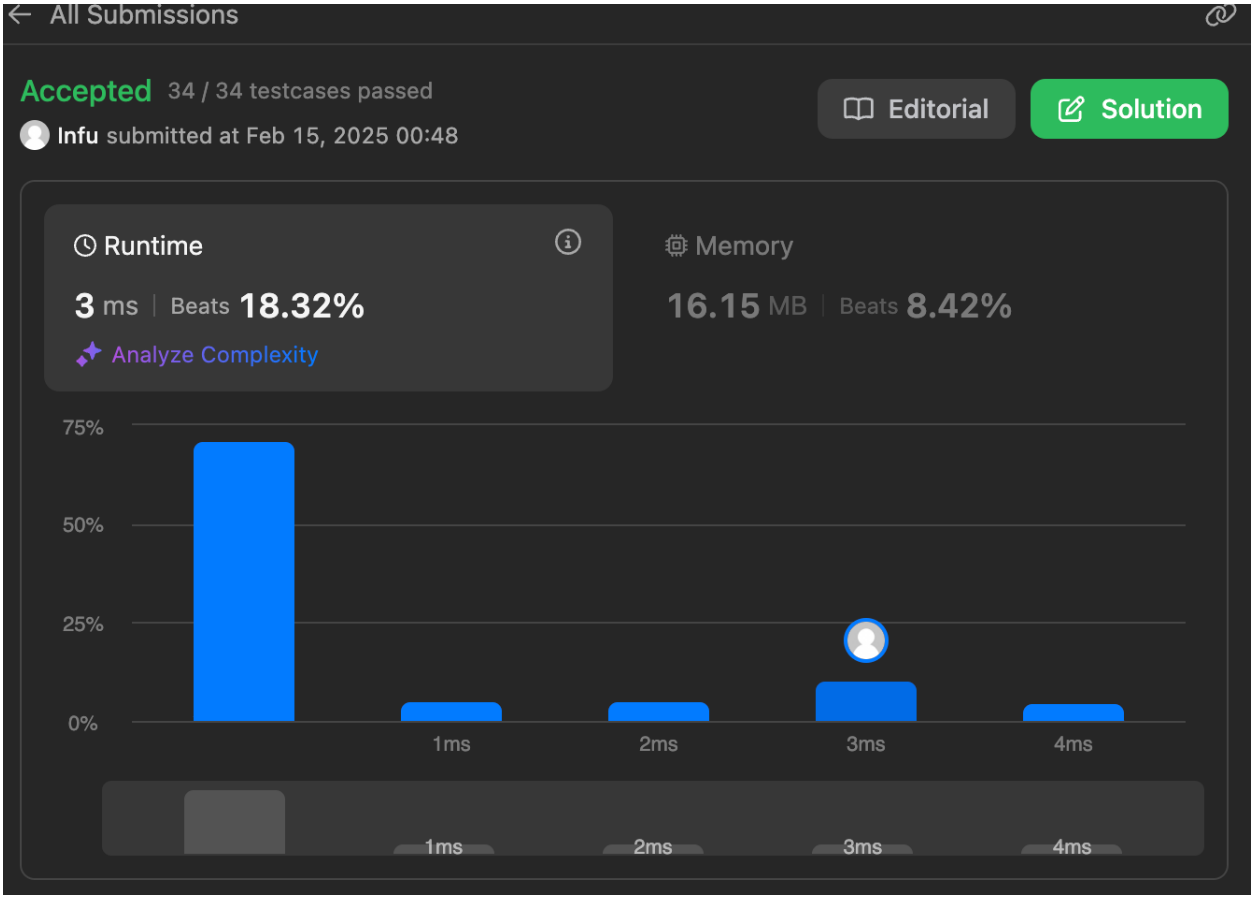
```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        if(root==NULL)return ans;
        queue<TreeNode*>q;
        q.push(root);
        while(!q.empty()){
            int s=q.size();
            vector<int>v;
            for(int i=0;i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



107. Binary Tree Level Order Traversal II

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        if(!root) return {};
        queue<TreeNode*> q;
        vector<vector<int>> ans;
        q.push(root);
        while(!q.empty()){
            int s = q.size();
            vector<int> sol;
            for(int i = 0; i < s; i++){
                auto t = q.front();
                q.pop();
                sol.push_back(t->val);
```

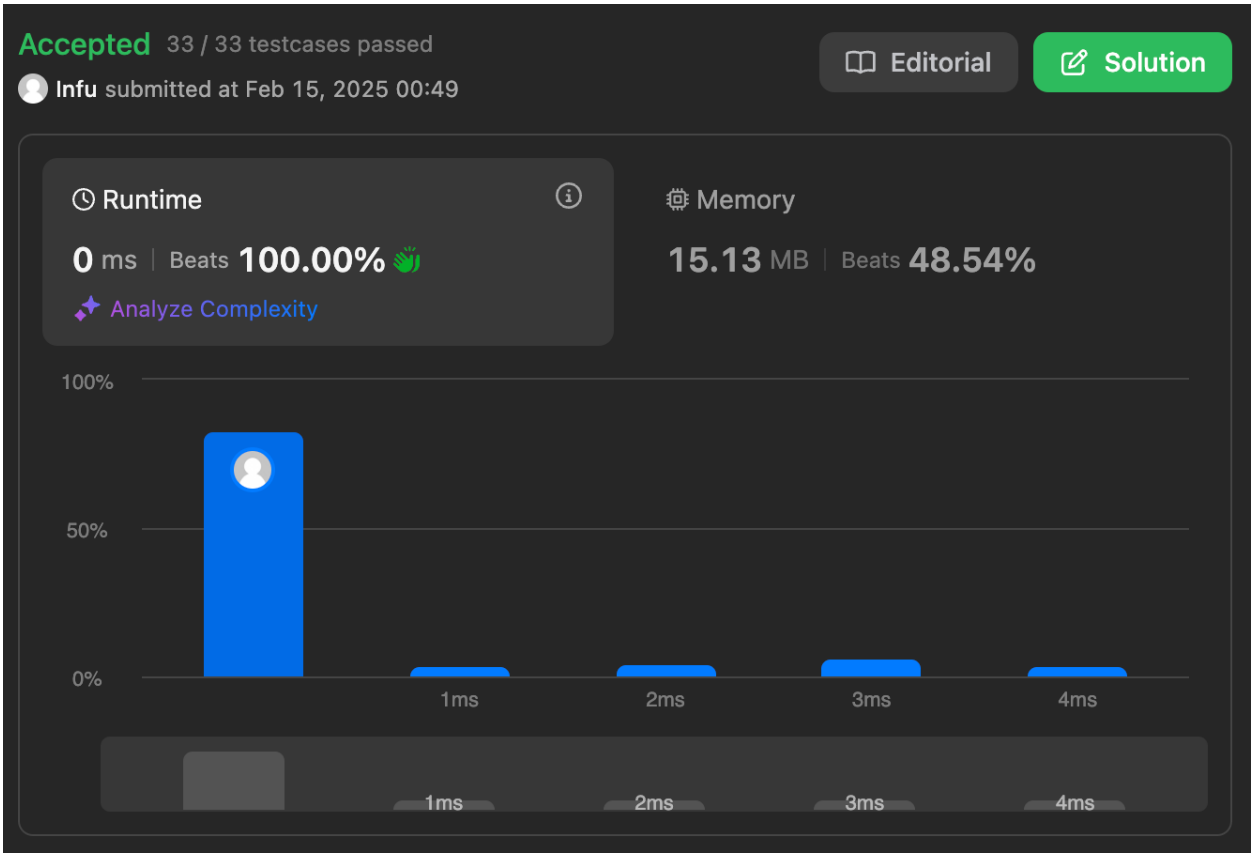
```
                if(t->left) q.push(t->left);
                if(t->right) q.push(t->right);
            }
            ans.insert(ans.begin(), sol);
        }
        return ans;
    }
};
```



103. Binary Tree Zigzag Level Order Traversal

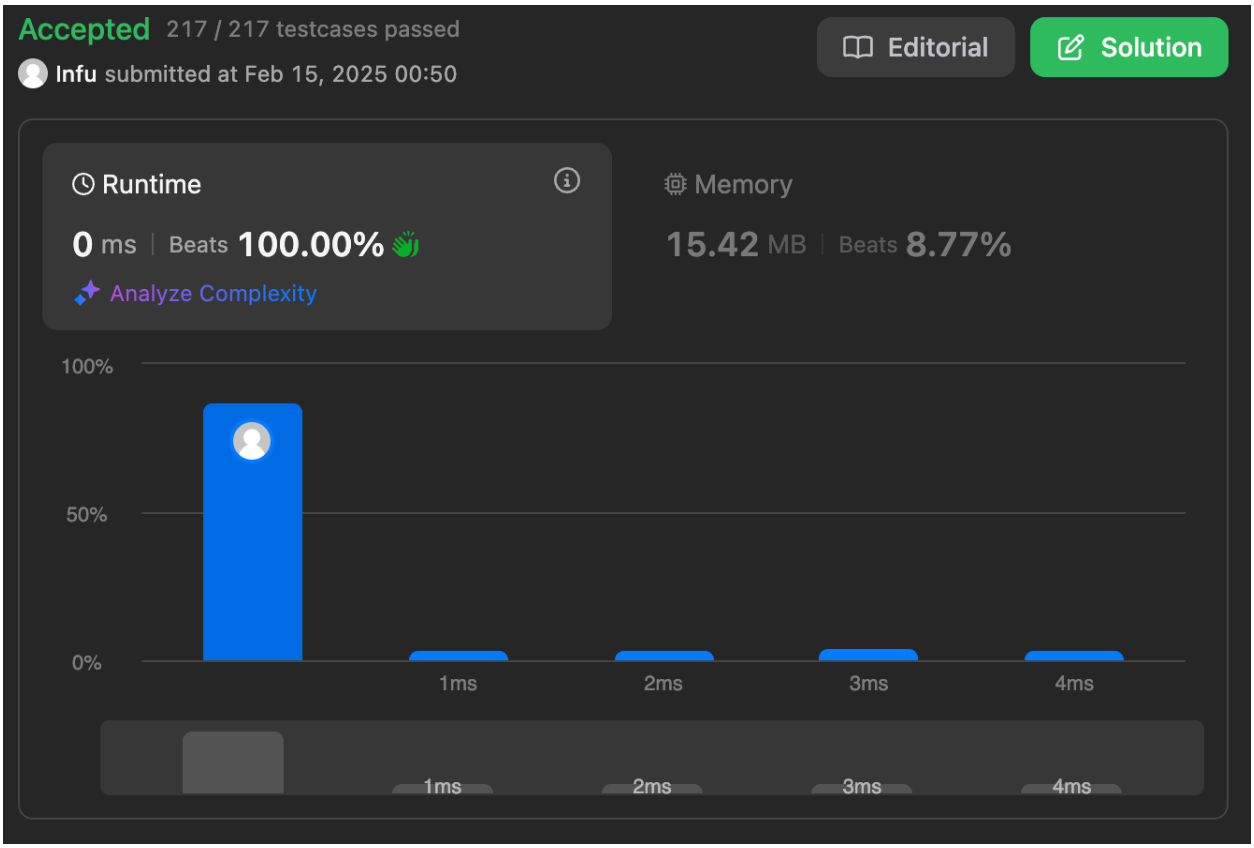
```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if(root == NULL) return result;
        queue<TreeNode*> q;
        q.push(root);
        bool r = false;
        while(!q.empty()){
            vector<int> sol;
            int count = 0;
            int size = q.size();
            while(count != size){
                TreeNode* temp = q.front();
                q.pop();
                sol.push_back(temp->val);
                if(temp->left != NULL) q.push(temp->left);
                if(temp->right != NULL) q.push(temp->right);
                count += 1;
                }
            if(r == true){
                reverse(sol.begin(), sol.end());
                result.push_back(sol);
                r = !r;
            }else{
```

```
                result.push_back(sol);
                r = !r;
            }
        }
        return result;
    }
};
```



199. Binary Tree Right Side View

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> ans;
        if(root == NULL) return ans;
        queue<pair<TreeNode*, int>> q;
        map<int, int> values;
        q.push({root,0});
        while(!q.empty()){
            auto temp = q.front();
            q.pop();
            values[temp.second] = temp.first->val;
            if(temp.first->left != NULL) q.push({temp.first->left,
temp.second + 1});
            if(temp.first->right != NULL) q.push({temp.first->right,
temp.second + 1});
        }
        for(auto value: values){
            ans.push_back(value.second);
        }
        return ans;
    }
};
```

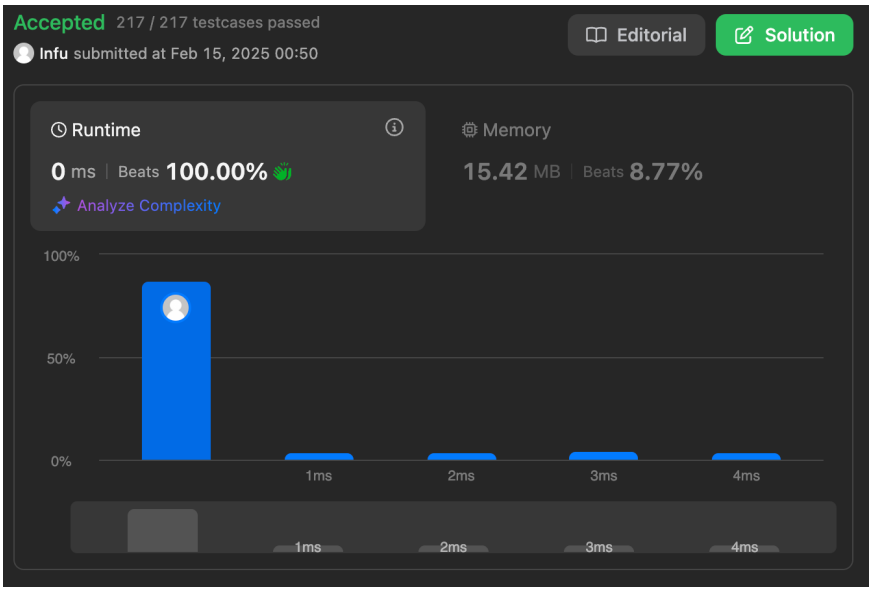


106. Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {
```

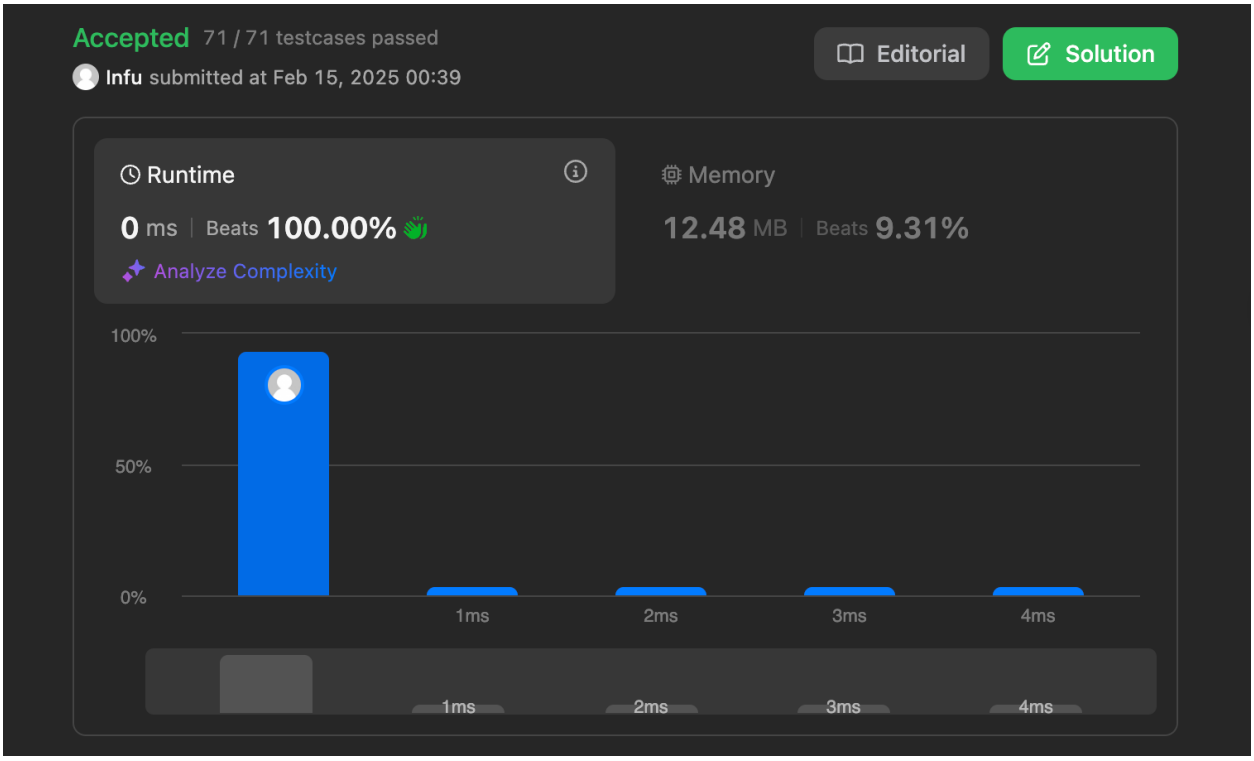
```
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder)
{
        unordered_map<int, int> index;
        for (int i = 0; i < inorder.size(); i++) {
            index[inorder[i]] = i;
        }
        return buildTreeHelper(inorder, postorder, 0, inorder.size()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



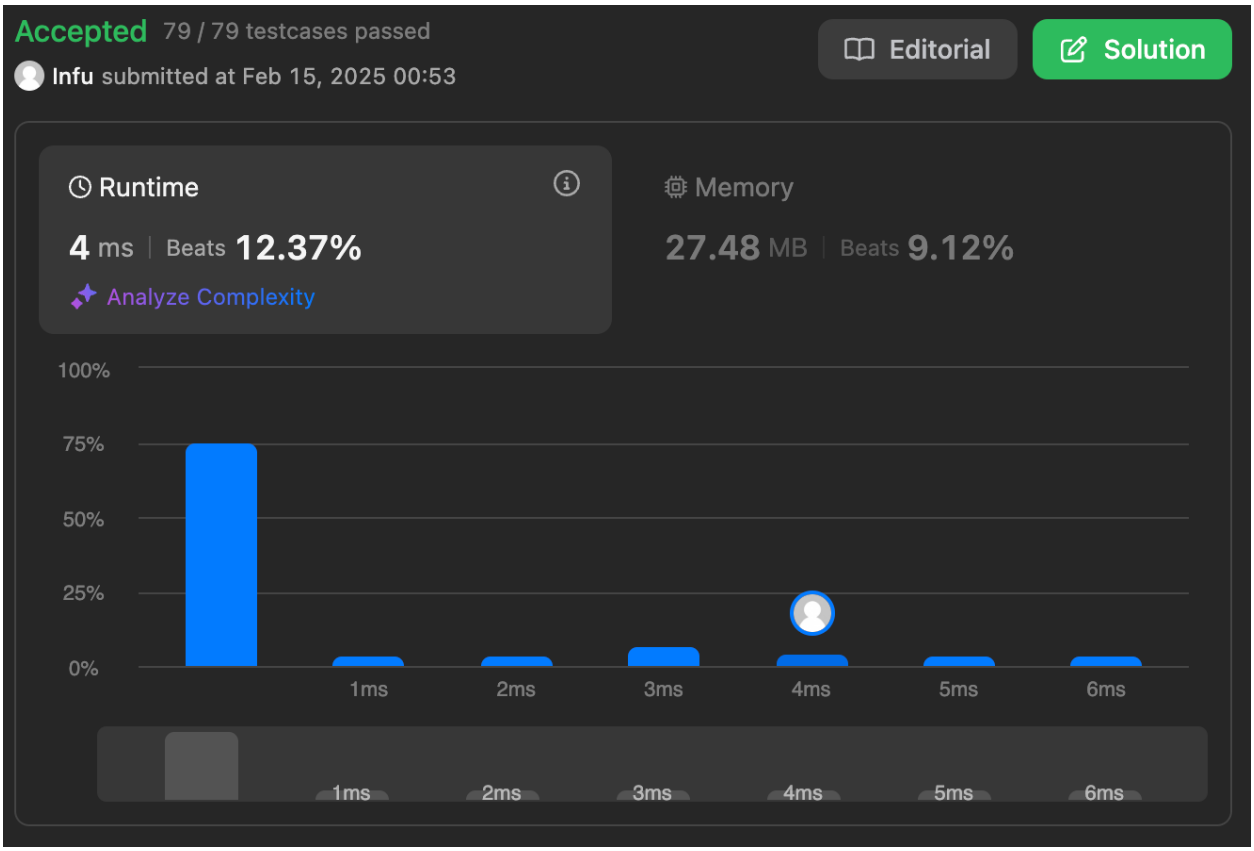
94. Binary Tree Inorder Traversal

```
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;
     }
};
```



513. Find Bottom Left Tree Value

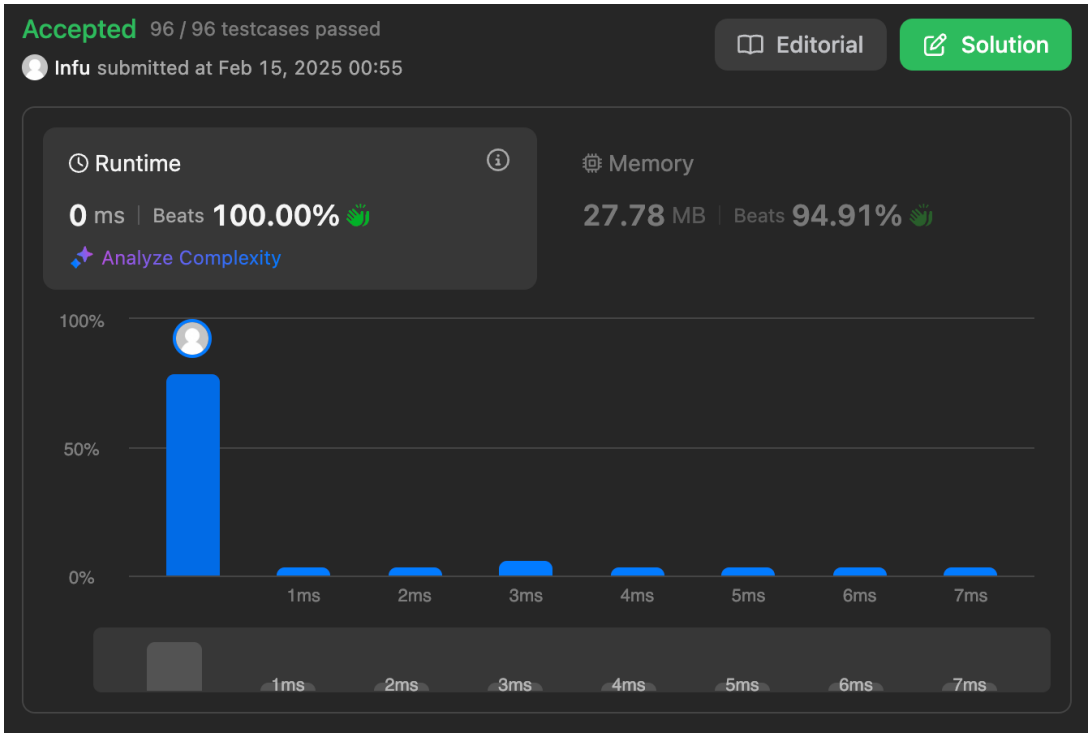
```
class Solution {
public:
    void tt(TreeNode* root, int level, vector<vector<int>>&nums){
        if(root==NULL){
            return;
        }
        if(nums.size()<=level){
            nums.push_back({});
        }

        nums[level].push_back(root->val);

        tt(root->right,level+1,nums);
        tt(root->left,level+1,nums);

    }
    int findBottomLeftValue(TreeNode* root) {
       vector<vector<int>>nums;
        tt(root,0,nums);
        return nums.back().back();
    }
};
```



124. Binary Tree Maximum Path Sum

```
class Solution {
public:
    int maxPathFinder(TreeNode* root, int& maxx){
        if(root == NULL){
            return 0;
        }
        int lh = max(0, maxPathFinder(root->left, maxx));
        int rh = max(0, maxPathFinder(root->right, maxx));

        maxx = max(maxx, lh + rh + root->val);
        return max(lh, rh) + root->val;
    }
```

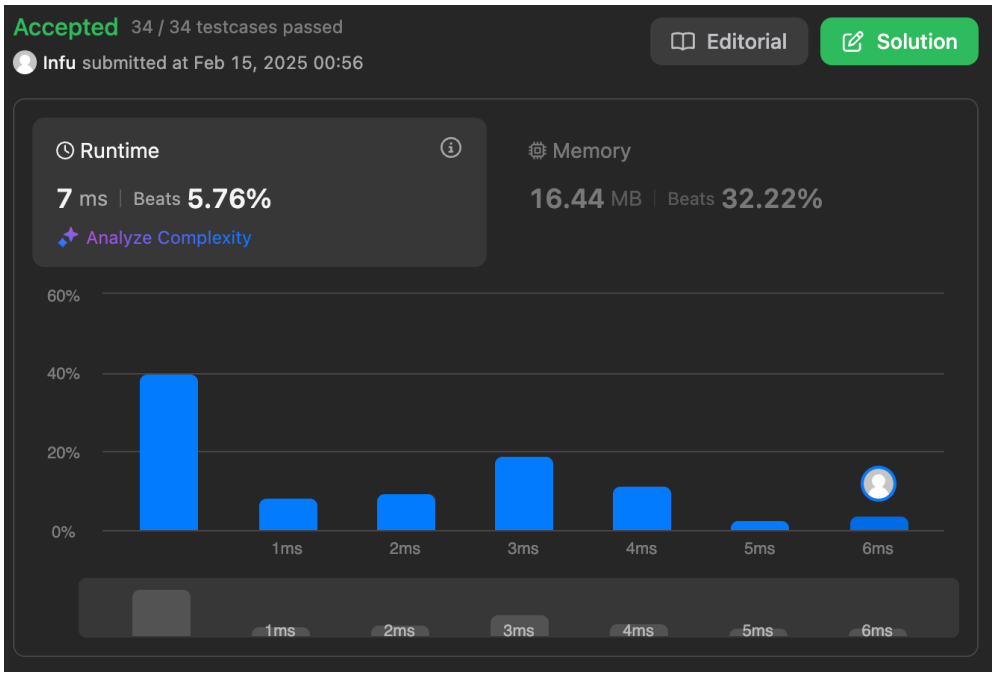
```
int maxPathSum(TreeNode* root) {
    int maxx = INT_MIN;
    maxPathFinder(root, maxx);
    return maxx;
}
```



987. 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>>> todo;
        todo.push({root, {0,0}});
        while(!todo.empty()){
            auto temp = todo.front();
            todo.pop();
```

```
            int value = temp.first->val;

nodes[temp.second.first][temp.second.second].insert(value);
            if(temp.first->left != NULL) todo.push({temp.first->left,
{temp.second.first-1, temp.second.second + 1}});
            if(temp.first->right != NULL) todo.push({temp.first-
>right, {temp.second.first+1, temp.second.second + 1}});
        }
        vector<vector<int>> result;
        for(auto node: nodes){
            vector<int> sol;
            for(auto value: node.second){
                sol.insert(sol.end(), value.second.begin(),
value.second.end());
            }
            result.push_back(sol);
        }
        return result;
    }
};
```


Accepted 34 / 34 testcases passed
Infu submitted at Feb 15, 2025 00:56
Editorial
Solution
Runtime
7 ms | Beats 5.76%
Analyze Complexity
Memory
16.44 MB | Beats 32.22%
60%
40%
20%
0%
1ms
2ms
3ms
4ms
5ms
6ms