# NAME- Abir Chakraborty| UID- 22BCS14321 | SECTION- 601/A

**1** [**BINARY**](https://leetcode.com/problems/longest-nice-substring/description/) **TREE INORDER TRAVERSAL**

class Solution { public:

 void inOrder(TreeNode\* root , vector<int> &arr){

        if(root==nullptr) return;

        inOrder(root->left,arr);

        arr.push\_back(root->val);

        inOrder(root->right,arr);

    }

    vector<int> inorderTraversal(TreeNode\* root) {

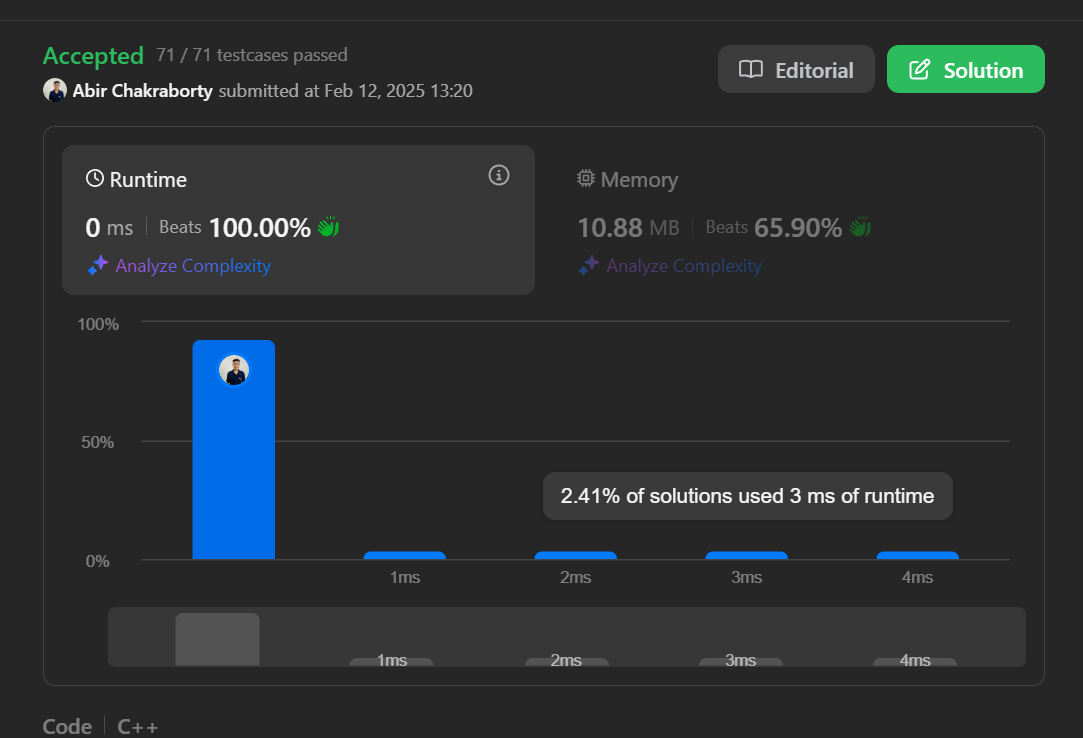
        vector<int> ans;

        inOrder(root, ans);

        return ans;

}

};



# 2. SYMMETRIC TREE

class Solution { public:

bool isSymmetric(TreeNode\* root) {

        if (!root) {

            return true;

        }

        return isSymmetricUtil(root->left, root->right);

    }

    bool isSymmetricUtil(TreeNode\* root1, TreeNode\* root2) {

        if (root1 == NULL || root2 == NULL) {

            return root1 == root2;

        }

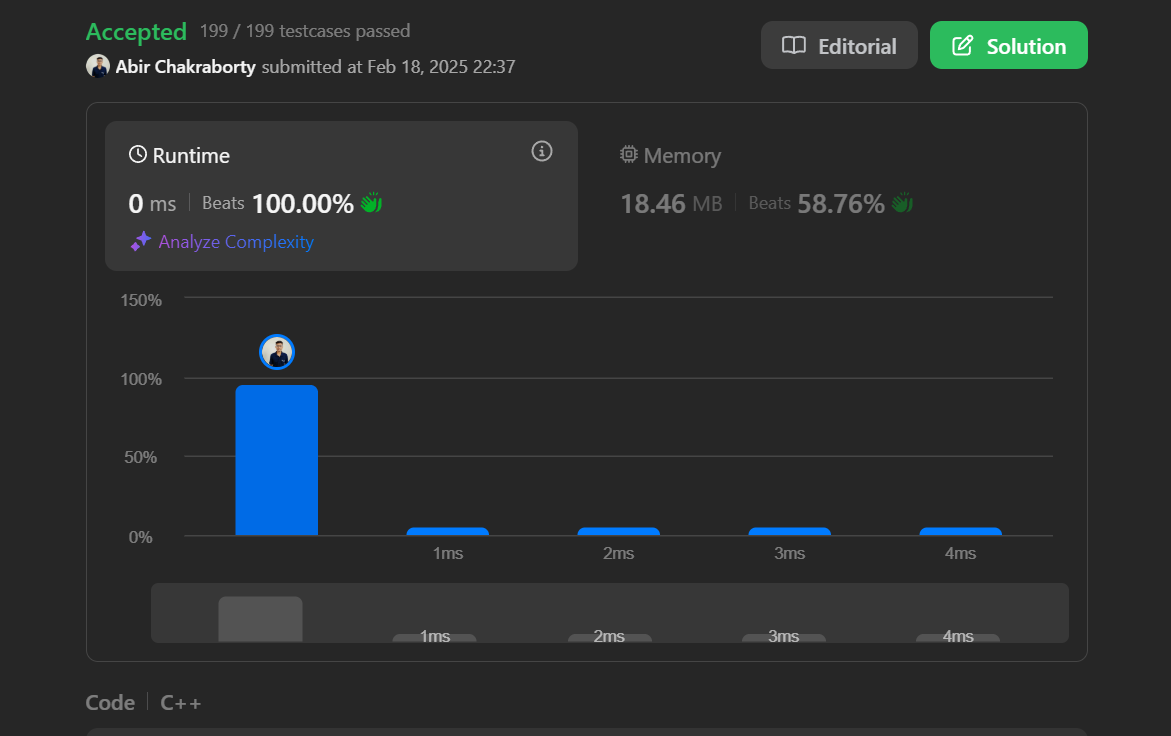
        return (root1->val == root2->val) &&

               isSymmetricUtil(root1->left, root2->right) &&

               isSymmetricUtil(root1->right, root2->left);

}

};



# 3 MAXIMUM DEPTH OF BINARY TREE

# class Solution {

# public:

# int maxDepth(TreeNode\* root) {

# if(root==nullptr){

# return 0;

# }

# int lh=maxDepth(root->left);

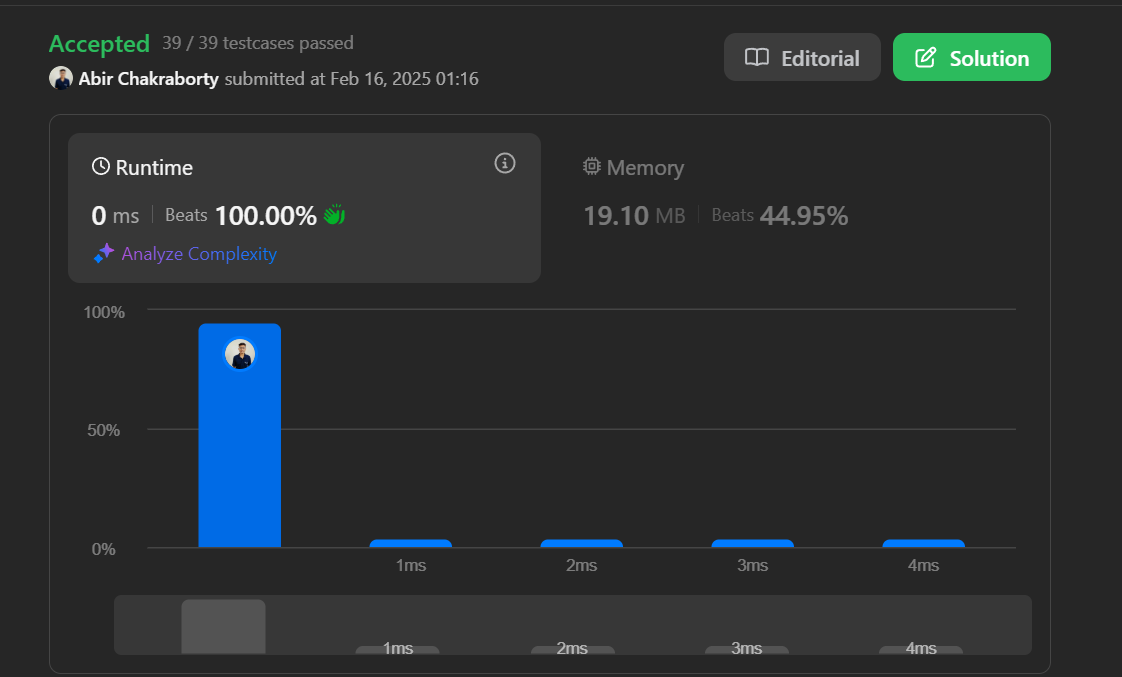
# int rh=maxDepth(root->right);

# return  1+ max(lh,rh);

# 

# }

# };



# VALIDATE BINARY SEARCH TREE

class Solution { public:

 bool isValid(TreeNode\* root, long minVal, long maxVal) {

        if (!root)

            return true;

        if (root->val <= minVal || root->val >= maxVal)

            return false;

        return isValid(root->left, minVal, root->val) &&

               isValid(root->right, root->val, maxVal);

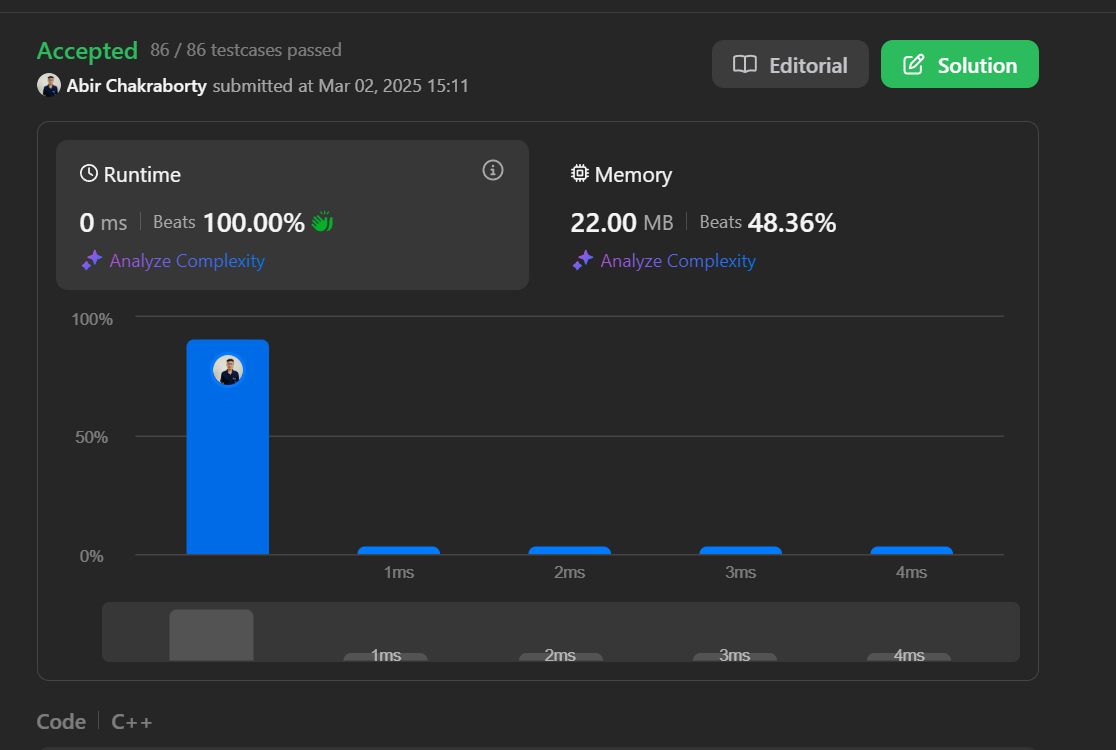
    }

    bool isValidBST(TreeNode\* root) {

        return isValid(root, LONG\_MIN, LONG\_MAX);

}

};



# K th SMALLEST ELEMENT IN A BINARY TREE

class Solution {

   void inorder(TreeNode\* root, int& counter, int k, int& kSmallest) {

        if (!root || counter >= k) return;

        inorder(root->left, counter, k, kSmallest);

        if (++counter == k) {

            kSmallest = root->val;

            return;

        }

        inorder(root->right, counter, k, kSmallest);

    }

public:

    int kthSmallest(TreeNode\* root, int k) {

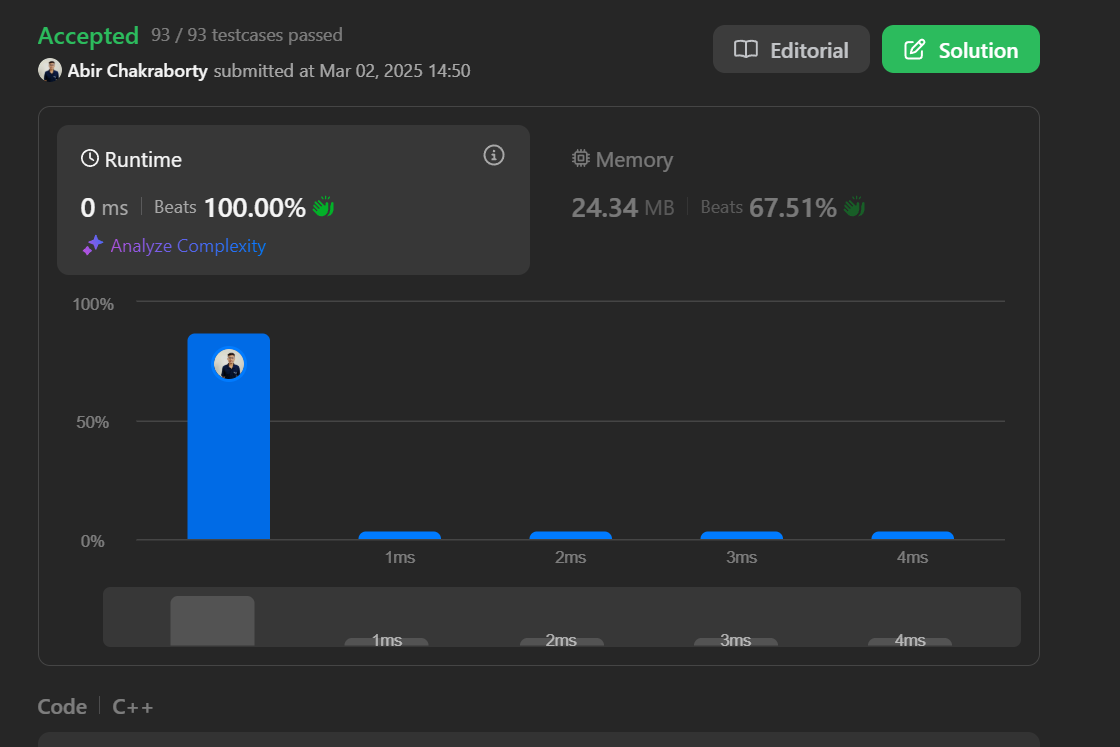
        int kSmallest = INT\_MIN, counter = 0;

        inorder(root, counter, k, kSmallest);

        return kSmallest;

    }

};



# BINARY TREE LEVEL ORDER TRAVERSAL

class Solution { public:

 vector<vector<int>> levelOrder(TreeNode\* root) {

        vector<vector<int>> res;

        if(root==nullptr){

            return res;

        }

        queue<TreeNode\*> q;

        q.push(root);

        while(!q.empty()){

            vector<int> ans;

            int size=q.size();

            for(int i=0;i<size;i++){

                TreeNode\* node=q.front();

                q.pop();

                ans.push\_back(node->val);

                if(node->left){

                    q.push(node->left);

                }

                if(node->right){

                    q.push(node->right);

                }

            }

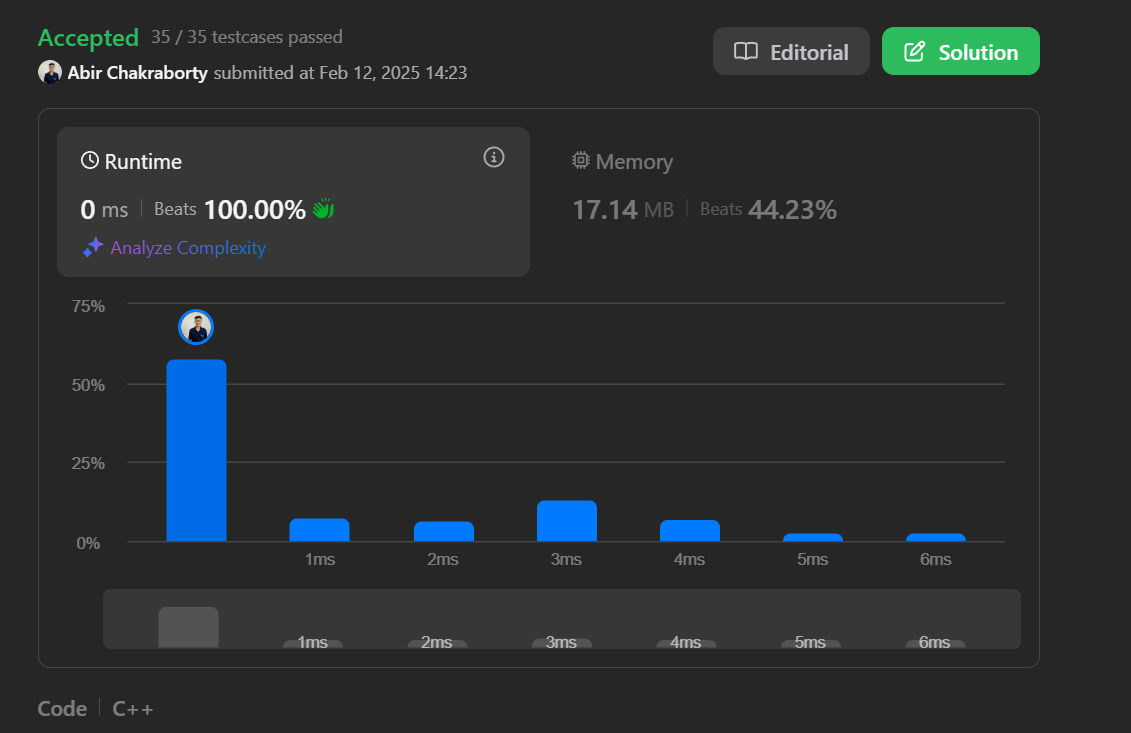
            res.push\_back(ans);

        }

        return res;

}

};



1. **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);

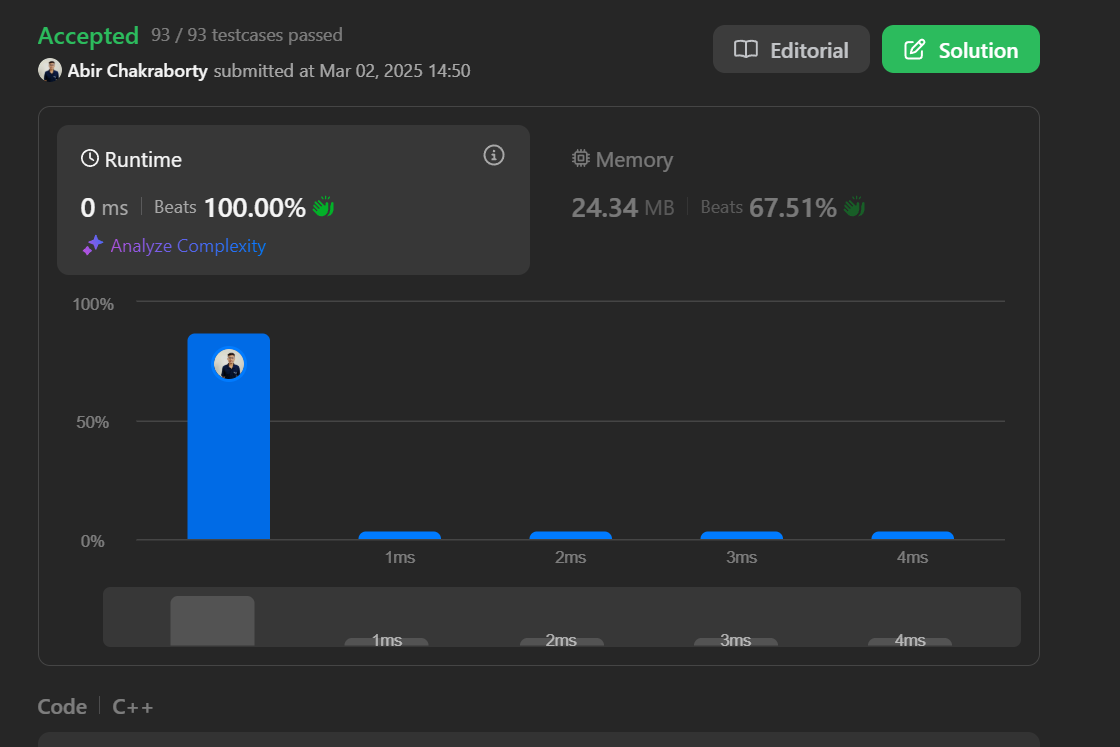
}

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

return result;

}

};



1. **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\*> nodesQueue;

        nodesQueue.push(root);

        bool leftToRight = true;

        while(!nodesQueue.empty()){

            int size = nodesQueue.size();

            vector<int> row(size);

            for(int i = 0; i < size; i++){

                TreeNode\* node = nodesQueue.front();

                nodesQueue.pop();

                int index = leftToRight ? i : (size - 1 - i);

                row[index] = node->val;

                if(node->left){

                    nodesQueue.push(node->left);

                }

                if(node->right){

                    nodesQueue.push(node->right);

                }

            }

            leftToRight = !leftToRight;

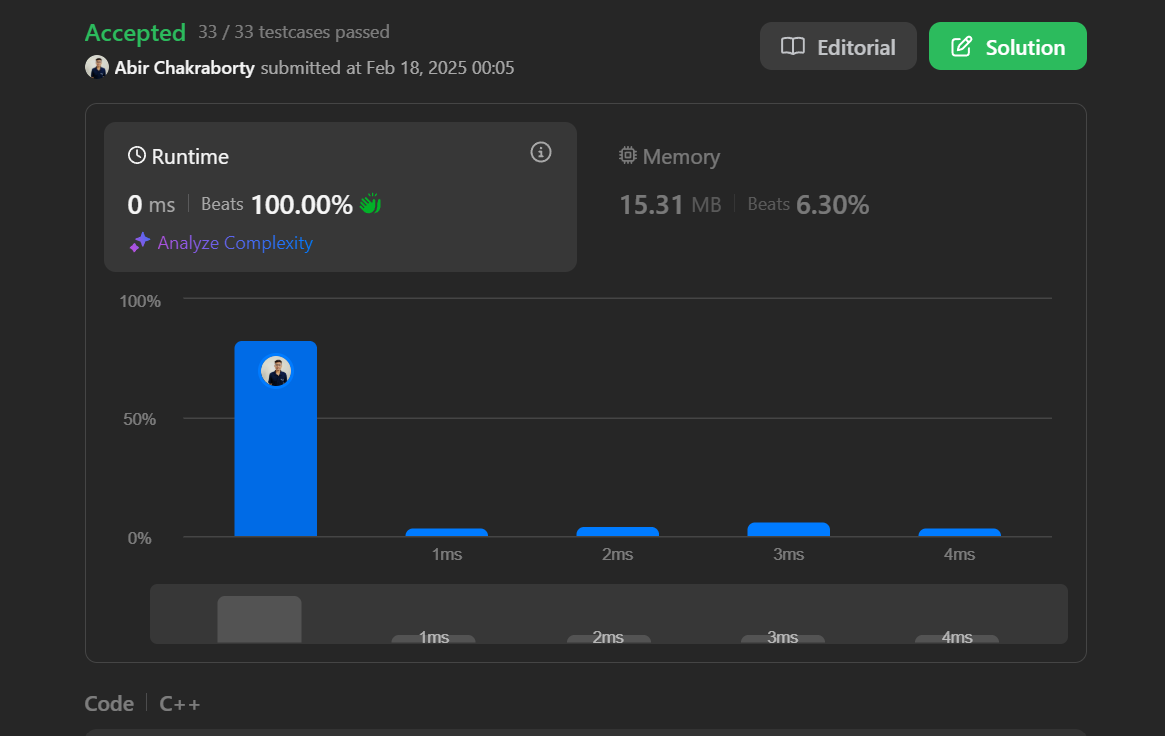
            result.push\_back(row);

        }

        return result;

    }

};



1. **BINARY TREE RIGHT SIDE VIEW**

class Solution {

public:

    vector<int> rightSideView(TreeNode\* root) {

        vector<int> res;

        recursionRight(root, 0, res);

        return res;

    }

    void recursionRight(TreeNode\* root, int level, vector<int>& res) {

        if (root == NULL) {

            return;

        }

        if (res.size() == level) {

            res.push\_back(root->val);

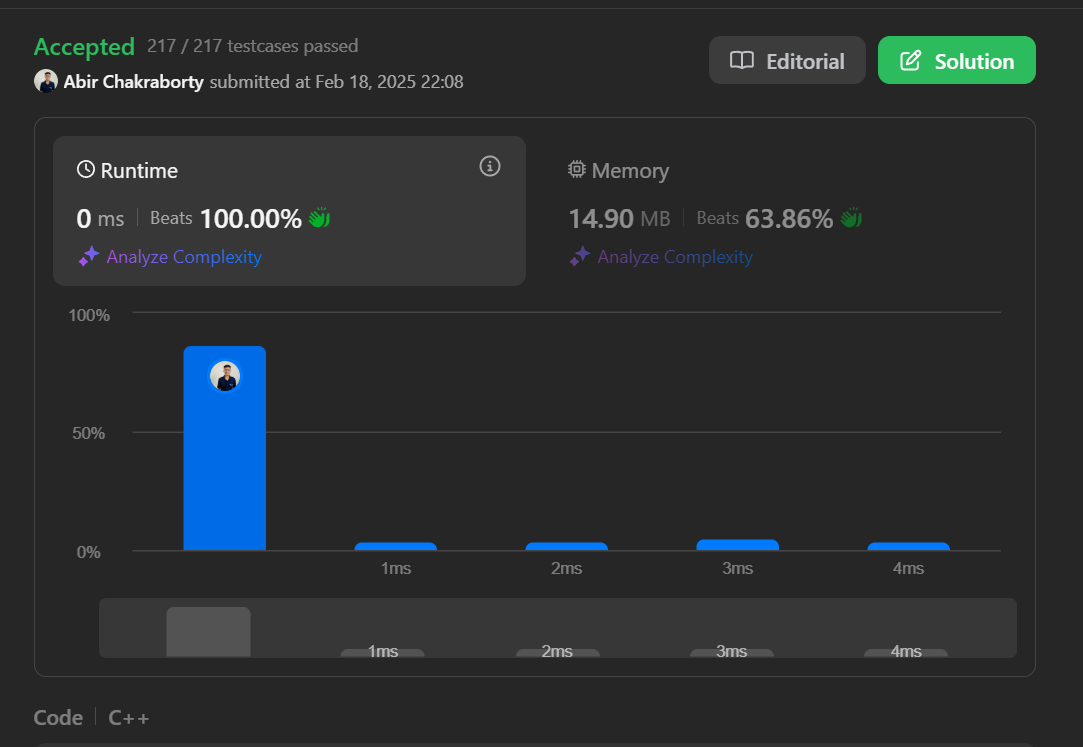
        }

        recursionRight(root->right, level + 1, res);

        recursionRight(root->left, level + 1, res);

    }

};



1. **CONSTRUCT BINARY TREE FROM INORDER AND POSTORDER**

class Solution {

public:

    TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

        if (inorder.size() != postorder.size()) {

            return NULL;

        }

        map<int, int> hm;

        for (int i = 0; i < inorder.size(); i++) {

            hm[inorder[i]] = i;

        }

        return buildTreePostIn(inorder, 0, inorder.size() - 1, postorder, 0,

                               postorder.size() - 1, hm);

    }

    TreeNode\* buildTreePostIn(vector<int>& inorder, int is, int ie,

                              vector<int>& postorder, int ps, int pe,

                              map<int, int>& hm) {

        if (ps > pe || is > ie) {

            return NULL;

        }

        TreeNode\* root = new TreeNode(postorder[pe]);

        int inRoot = hm[postorder[pe]];

        int numsLeft = inRoot - is;

        root->left = buildTreePostIn(inorder, is, inRoot - 1, postorder, ps,

                                     ps + numsLeft - 1, hm);

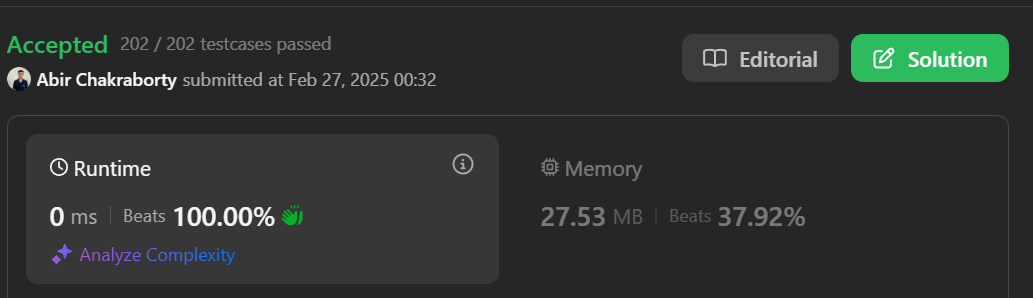
        root->right = buildTreePostIn(inorder, inRoot + 1, ie, postorder,

                                      ps + numsLeft, pe - 1, hm);

        return root;

    }

};



1. **FIND BOTTOM LEFT TREE VALUE**

class Solution {

public:

    int findBottomLeftValue(TreeNode\* root) {

        int last=0;

        queue<TreeNode\*> q;

        q.push(root);

        while(!q.empty())

        {

            int count=q.size();

            for(int i=0;i<count;i++)

            {

                TreeNode\* curr=q.front();

                q.pop();

                if(i==0)

                    last=curr->val;

                if(curr->left)

                    q.push(curr->left);

                if(curr->right)

                    q.push(curr->right);

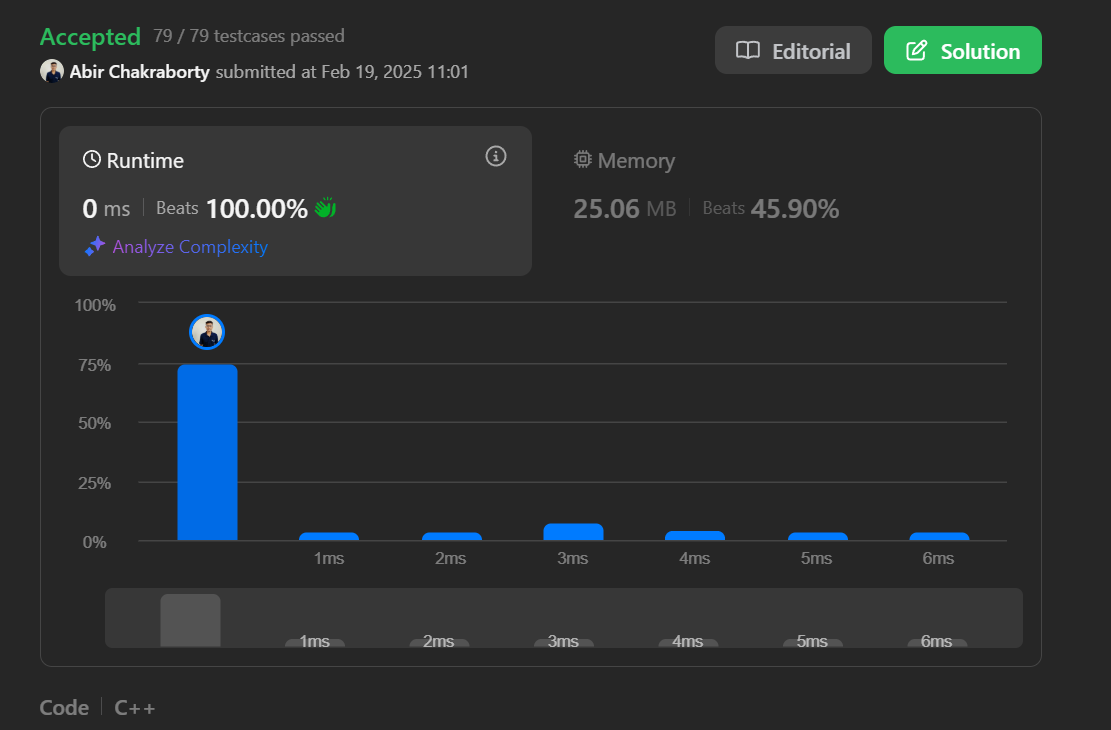
            }

        }

        return last;

    }

};



1. **BINARY TREE MAXIMUM PATH SUM**

class Solution {

public:

    int findMaxPathSum(TreeNode\* root, int &maxi) {

        if (root == nullptr) {

            return 0;

        }

        int leftMaxPath = max(0, findMaxPathSum(root->left, maxi));

        int rightMaxPath = max(0, findMaxPathSum(root->right, maxi));

        maxi = max(maxi, leftMaxPath + rightMaxPath + root->val);

        return max(leftMaxPath, rightMaxPath) + root->val;

    }

    int maxPathSum(TreeNode\* root) {

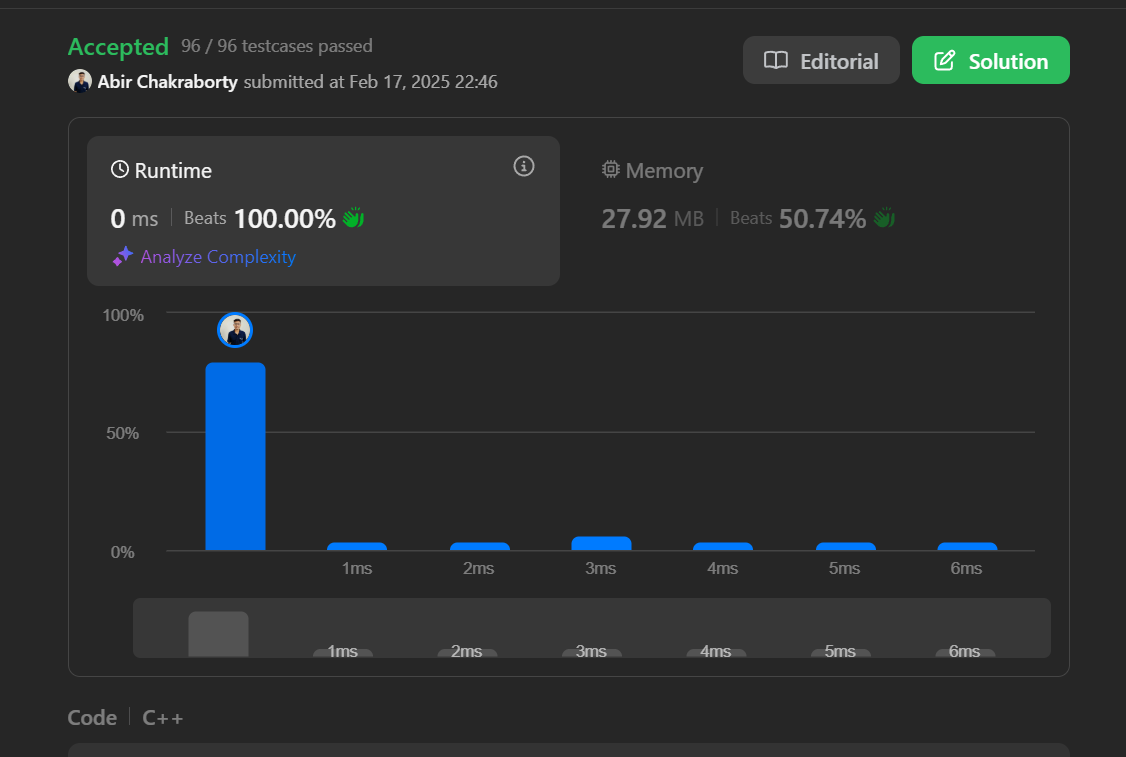
        int maxi = INT\_MIN;

        findMaxPathSum(root, maxi);

        return maxi;

    }

};



1. **VERTICAL ORDER TRAVERSAL OF 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 p = todo.front();

            todo.pop();

            TreeNode\* temp = p.first;

            int x = p.second.first;

            int y = p.second.second;

            nodes[x][y].insert(temp->val);

            if(temp->left){

                todo.push({temp->left, {x-1, y+1}});

            }

            if(temp->right){

                todo.push({temp->right, {x+1, y+1}});

            }

        }

        vector<vector<int>> ans;

        for(auto p: nodes){

            vector<int> col;

            for(auto q: p.second){

                col.insert(col.end(), q.second.begin(), q.second.end());

            }

            ans.push\_back(col);

        }

        return ans;

    }

};

