**ADVANCED PROGRAMMING**

**ASSIGNMENT 3**

1. **94.Binary Tree Inorder Traversal**

**Code :**

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);

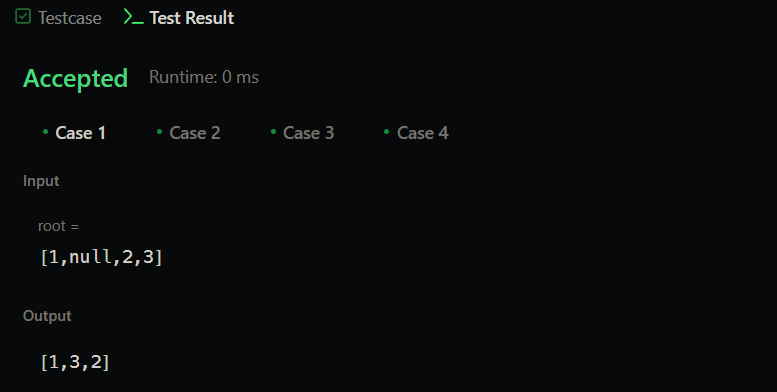
        res.push\_back(r->val);

        DFS(r->right,res);

    }

};

**Output :**

****

1. **101.Symmetric Tree**

**Code :**

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) {

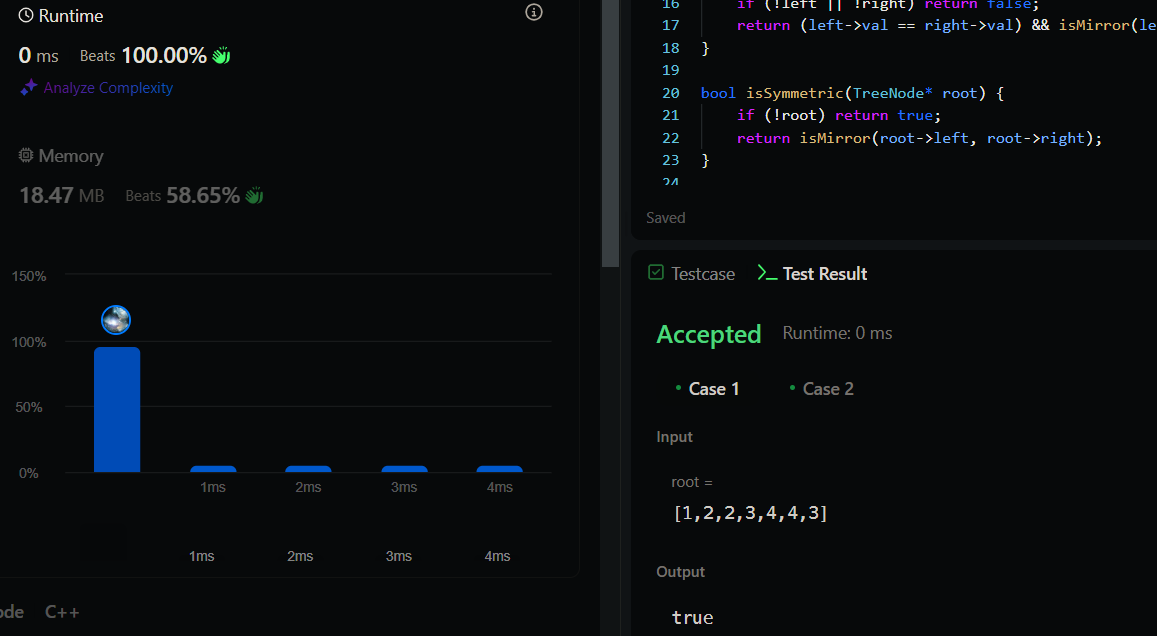
    if (!root) return true;

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

}

};

**Output :**

****

1. **104.Maximum Depth of Binary Tree**

**Code :**

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        return depth(root);

    }

private:

    int depth(TreeNode\*root)

    {

        if(root==NULL)

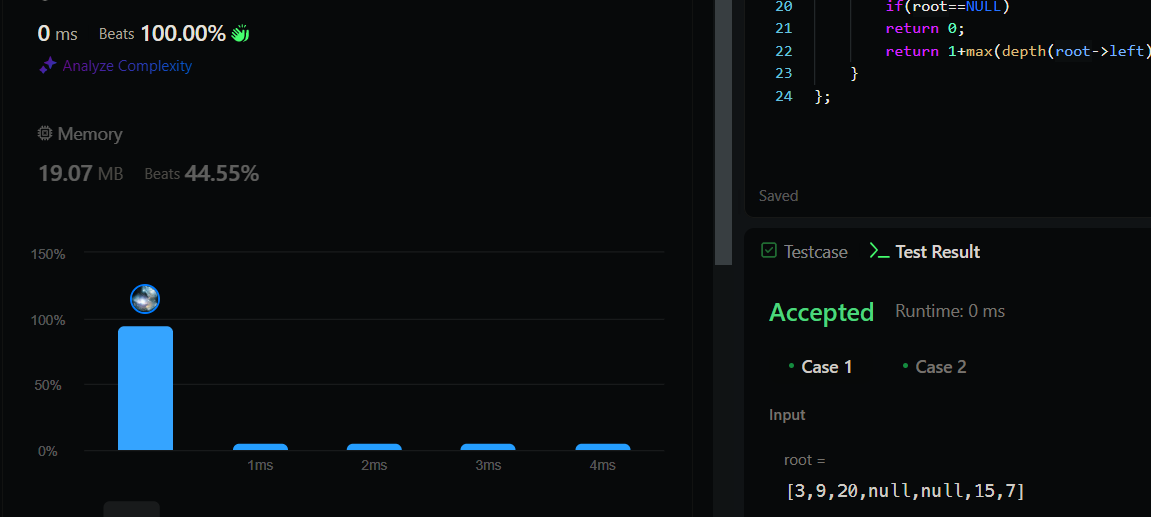
        return 0;

        return 1+max(depth(root->left),depth(root->right));

    }

};

**Output :**

****

1. **98.Validate Binary Search Tree**

**Code :**

class Solution {

public:

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

    {

        if(root == NULL) return true;

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

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

    }

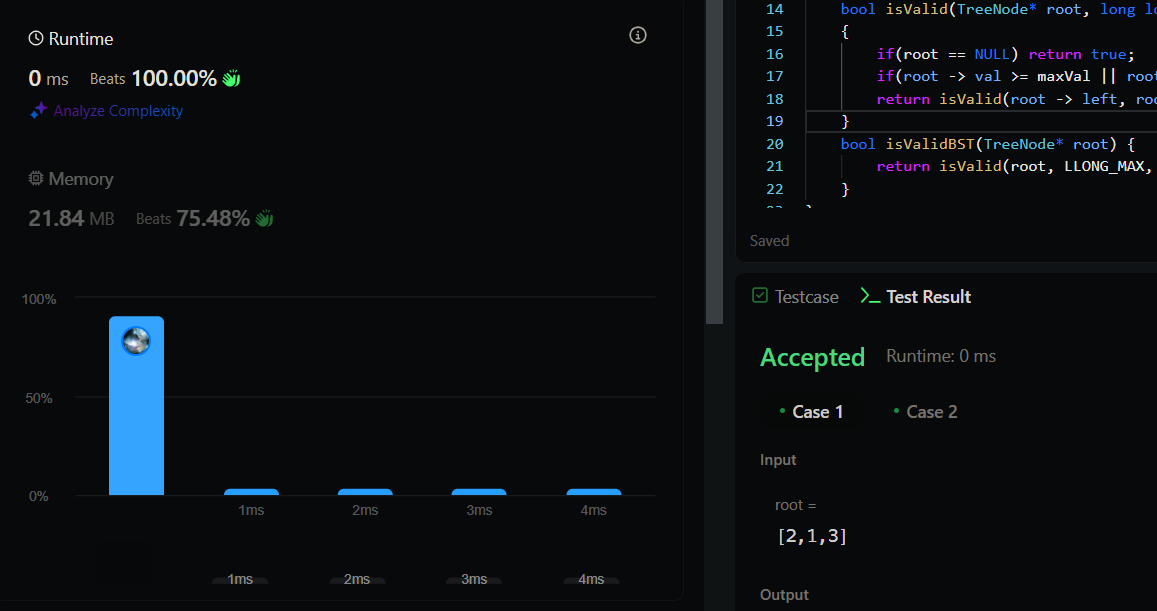
    bool isValidBST(TreeNode\* root) {

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

    }

};

**Output :**

****

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

**Code :**

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;

    }

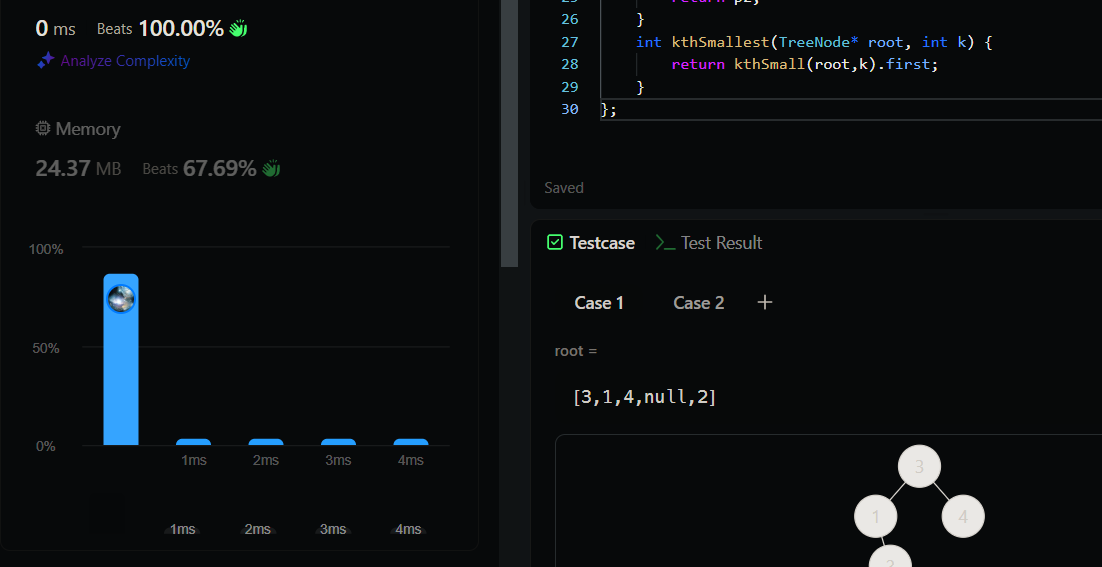
    int kthSmallest(TreeNode\* root, int k) {

        return kthSmall(root,k).first;

    }

};

**Output :**

****

1. **102. Binary Tree Level Order Traversal**

**Code :**

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);

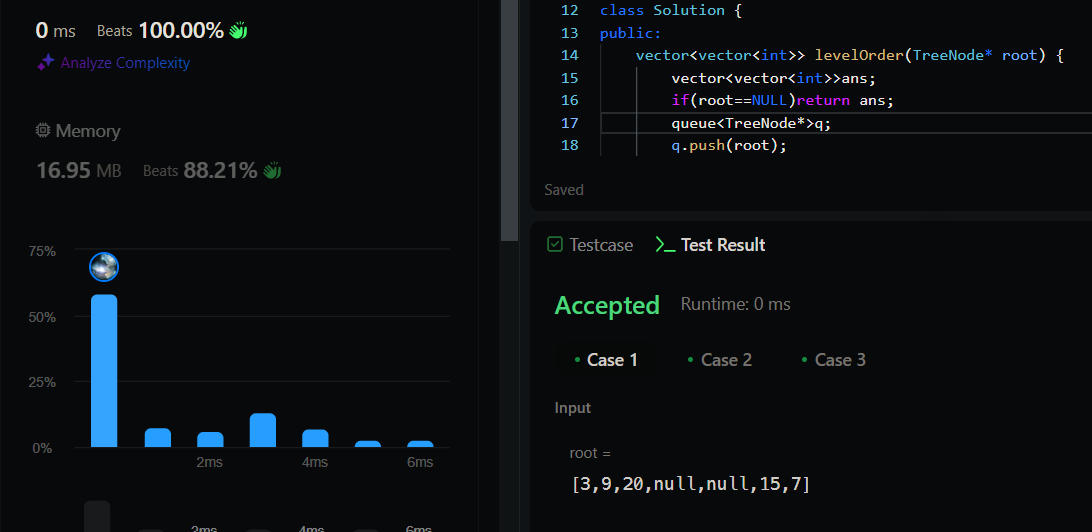
        }

        return ans;

    }

};

**Output :**

****

1. **107.Binary Tree Level Order Traversal II**

**Code :**

class Solution {

public:

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

        vector<vector<int>> levels;

        if (!root) return levels;

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            int n = q.size();

            vector<int> level;

            for (int i = 0; i < n; ++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);

            }

            levels.insert(levels.begin(), level);

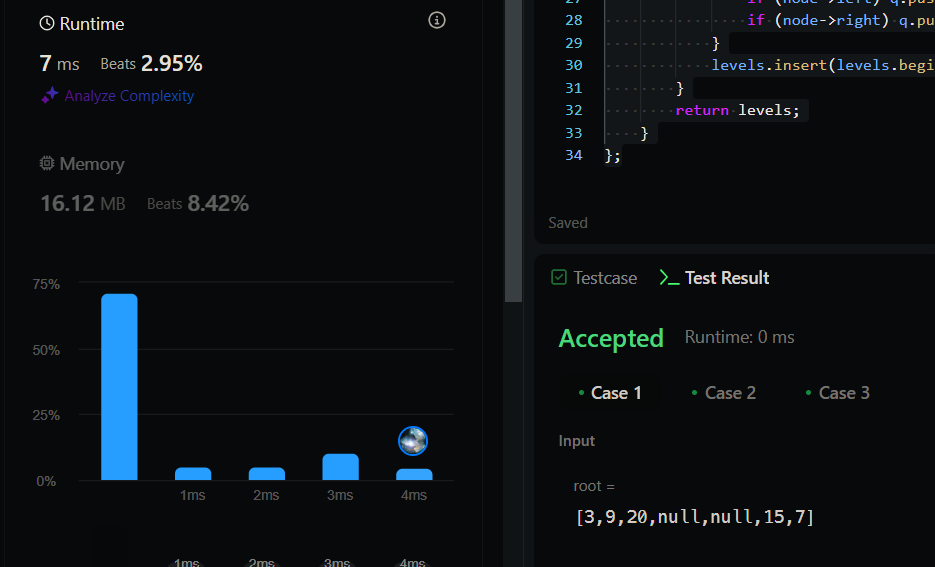
        }

        return levels;

    }

};

**Output :**

****

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

**Code :**

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);

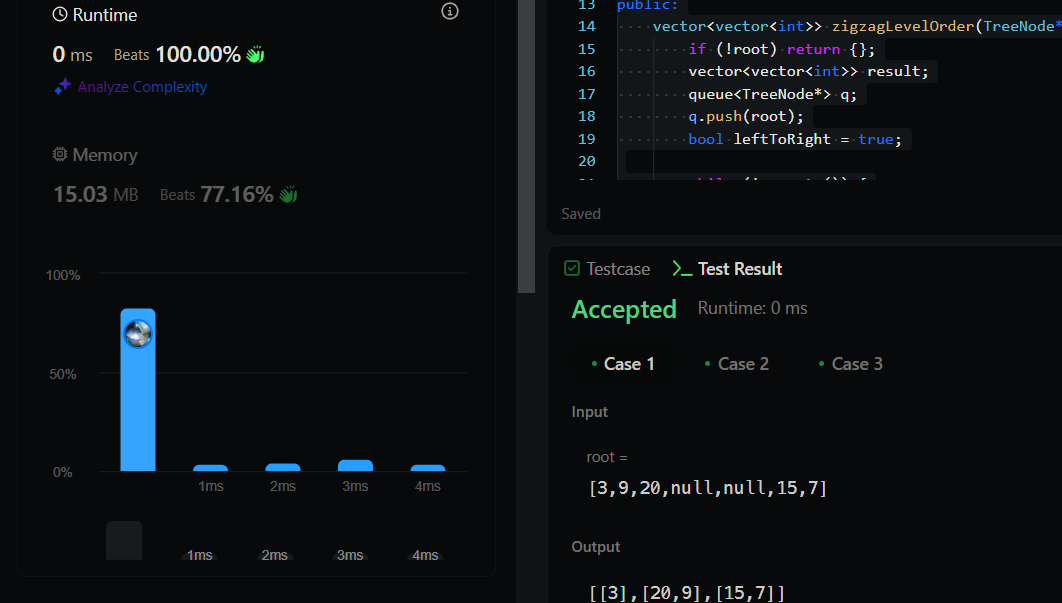
        }

        return result;

    }

};

**Output :**

****

1. **199.Binary Tree Right Side View**

**Code :**

class Solution {

public:

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

        vector<int> result;

        if (!root) return result;

        queue<TreeNode\*> q;

        q.push(root);

        while (!q.empty()) {

            int levelSize = q.size();

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

                TreeNode\* node = q.front();

                q.pop();

                if (i == levelSize - 1) result.push\_back(node->val);

                if (node->left) q.push(node->left);

                if (node->right) q.push(node->right);

            }

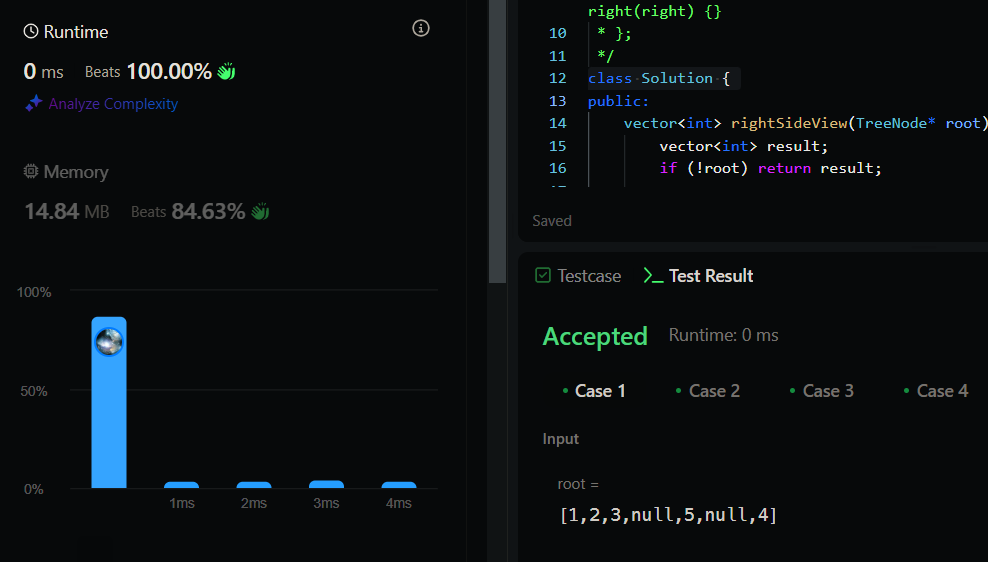
        }

        return result;

    }

};

**Output :**

****

1. **106.Construct Binary Tree from Inorder and Postorder Traversal**

**Code :**

class Solution {

public:

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

        int n = size(inorder), postIdx = n-1;

        return build(inorder, postorder, 0, n-1, postIdx);

    }

    TreeNode\* build(vector<int>& in, vector<int>& post, int inStart, int inEnd, int& postIdx) {

        if(inStart > inEnd) return nullptr;

        TreeNode\* root = new TreeNode(post[postIdx--]);

        int inIdx = find(begin(in), end(in), root -> val) - begin(in);

        root -> right = build(in, post, inIdx+1, inEnd, postIdx);

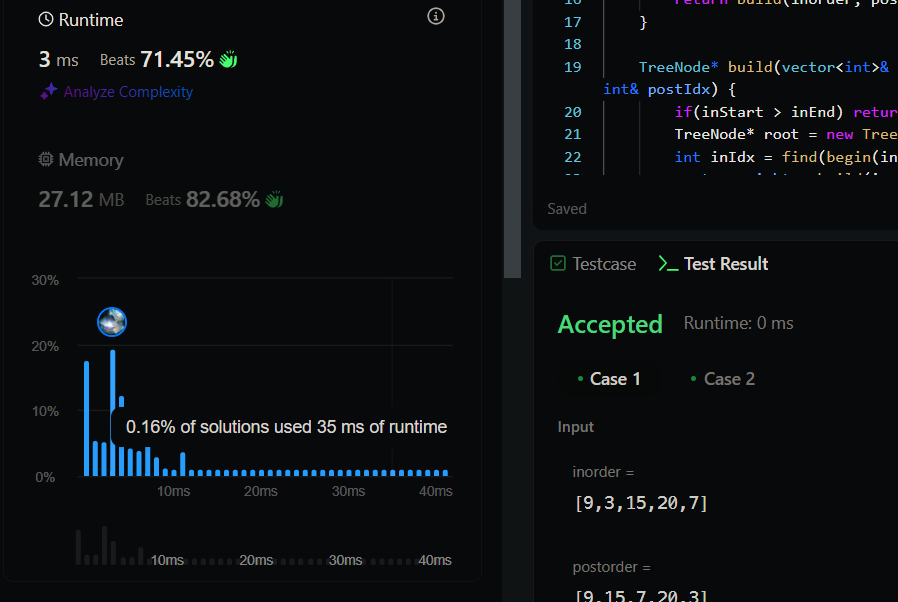
        root -> left  = build(in, post, inStart, inIdx-1, postIdx);

        return root;

    }

};

**Output :**

****

1. **513.Find Bottom Left Tree Value**

**Code :**

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;

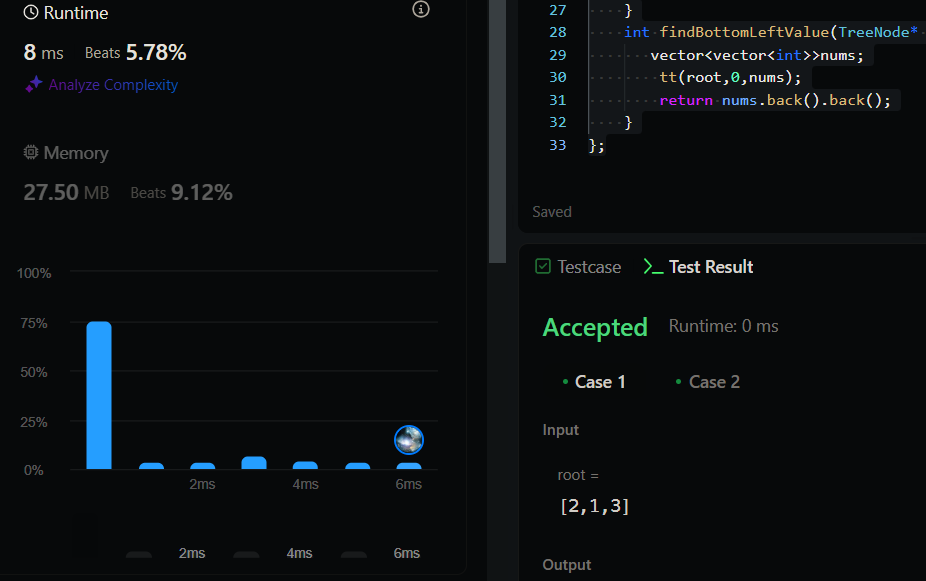
        tt(root,0,nums);

        return nums.back().back();

    }

};

**Output :**

****

1. **124. Binary Tree Maximum Path Sum**

**Code :**

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));

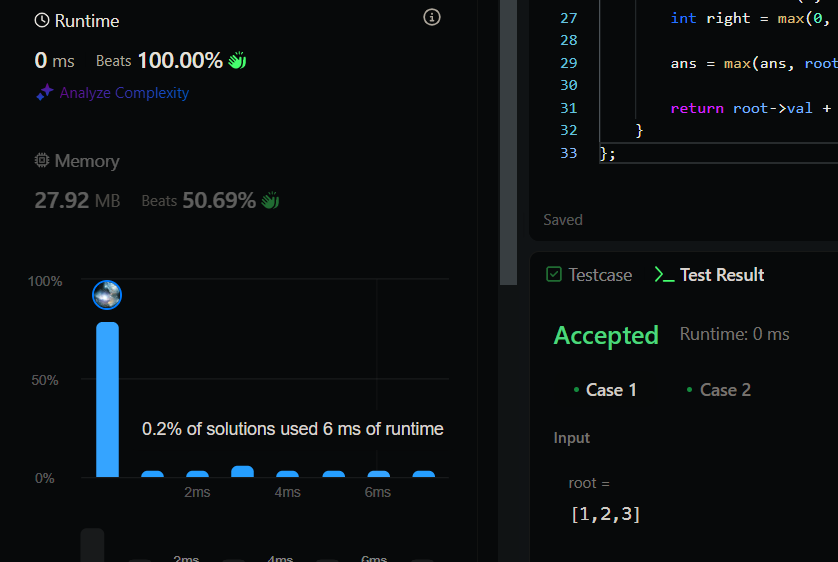
        ans = max(ans, root->val + left + right);

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

    }

};

**Output :**

****

1. **987.Vertical Order Traversal of a Binary Tree**

**Code :**

class Solution {

public:

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

        vector<vector<int>> out;

        map<int, vector<int>> final\_mp;

        queue<pair<int, TreeNode\*>> q;

        q.push({0, root});

        while (!q.empty()) {

            int n = q.size();

            map<int, vector<int>> mp;

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

                auto it = q.front();

                int index = it.first;

                TreeNode\* node = it.second;

                mp[index].push\_back(node->val);

                q.pop();

                if (node->left != NULL)

                    q.push({index - 1, node->left});

                if (node->right != NULL)

                    q.push({index + 1, node->right});

            }

            for (auto it : mp) {

                sort(it.second.begin(), it.second.end());

                vector<int> temp = final\_mp[it.first];

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

                    temp.push\_back(it.second[i]);

                }

                final\_mp[it.first] = temp;

            }

        }

        for (auto it : final\_mp) {

            out.push\_back(it.second);

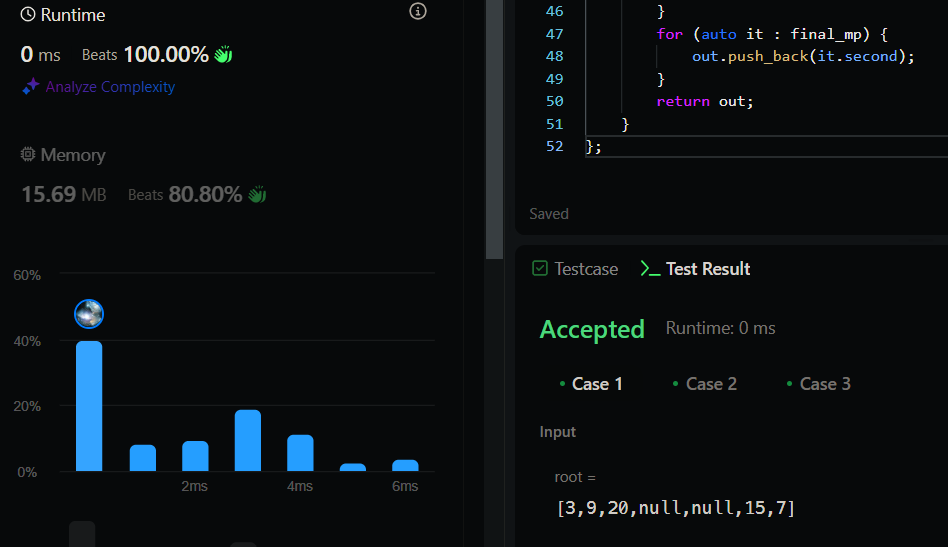
        }

        return out;

    }

};

**Output :**

****