**Name:Pawan Singh**

**Uid:22BCS14871**

**SEC/GRP:FL\_IOT\_602/A**

Experiment – 3(AP)

94.[Binary Tree Inorder Traversal](https://leetcode.com/problems/binary-tree-inorder-traversal/)

class Solution {

public:

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

        if(root==nullptr)

        {

            return {};

        }

        vector<int>result;

        vector<int>lt=inorderTraversal(root->left);

        result.insert(result.end(),lt.begin(),lt.end());

        result.push\_back(root->val);

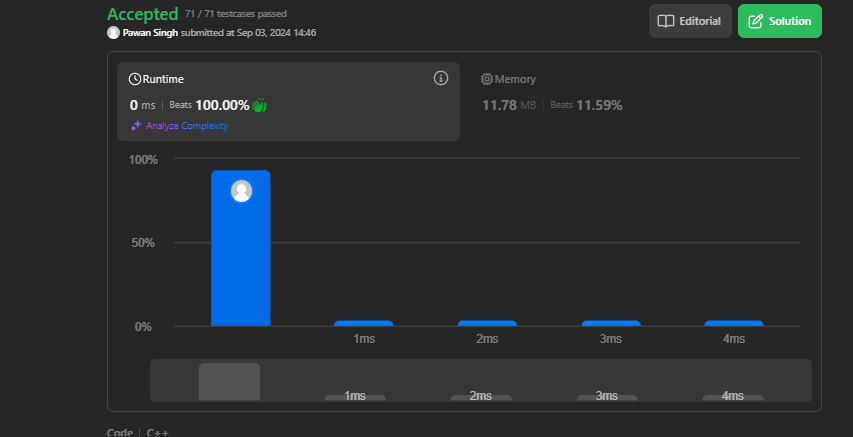
        vector<int>rt=inorderTraversal(root->right);

        result.insert(result.end(),rt.begin(),rt.end());

        return result;

    }

};



101.[Symmetric Tree](http://leetcode.com/problems/symmetric-tree/description/)

class Solution {

public:

bool ismirror(TreeNode\* r,TreeNode\* l)

{

    if(r==nullptr && l==nullptr)

    {

        return true;

    }

    if(r==nullptr || l==nullptr)

    {

        return false;

    }

    return (r->val==l->val) && ismirror(r->right,l->left) && ismirror(r->left,l->right);

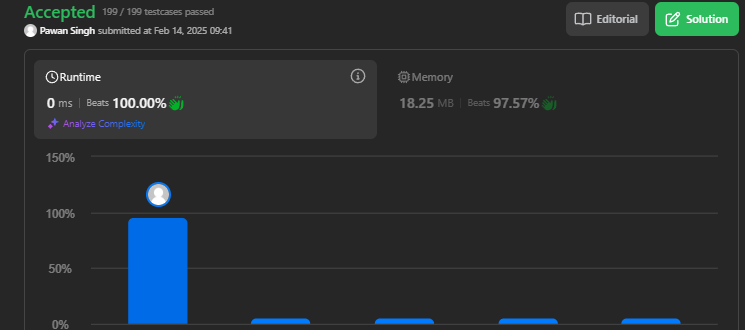
}

    bool isSymmetric(TreeNode\* root) {

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

    }

};



104.[Maximum Depth of Binary Tree](https://leetcode.com/problems/maximum-depth-of-binary-tree/description/)

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if(root==nullptr)

        {

            return 0;

        }

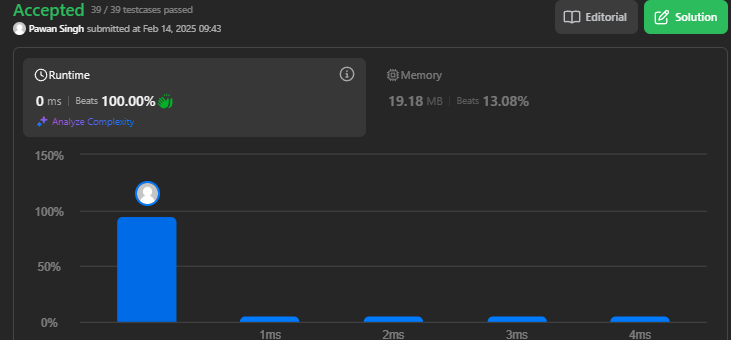
        int lh=maxDepth(root->left);

        int rh=maxDepth(root->right);

        return 1+max(lh,rh);

    }

};



98.[Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree/description/)

class Solution {

public:

    bool helper(TreeNode\* node,TreeNode\* minnode,TreeNode\* maxnode)

    {

        if(!node) return true;

        if((minnode && node->val <= minnode->val) || maxnode && node->val >= maxnode->val)

        {

            return false;

        }

        return helper(node->left,minnode,node) && helper(node->right,node,maxnode);

    }

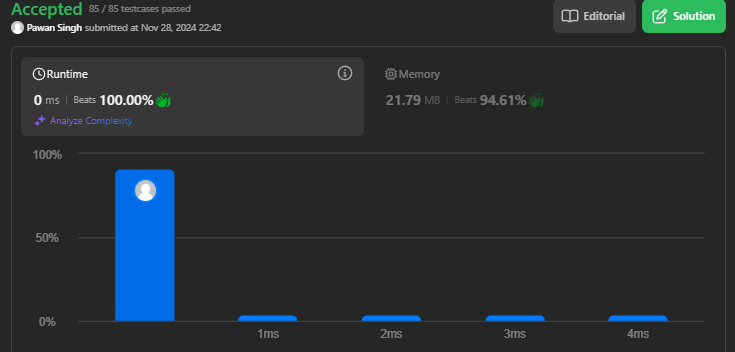
    bool isValidBST(TreeNode\* root)

    {

       return helper(root,nullptr,nullptr);

    }

};



230.[Kth Smallest Element in a BST](https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/)

class Solution {

public:

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

        if(root==nullptr)

        {

            return INT\_MAX;

        }

        int leftans=kthSmallest(root->left,k);

        if(k==0)

        {

            return leftans;

        }

        if(--k==0)

        {

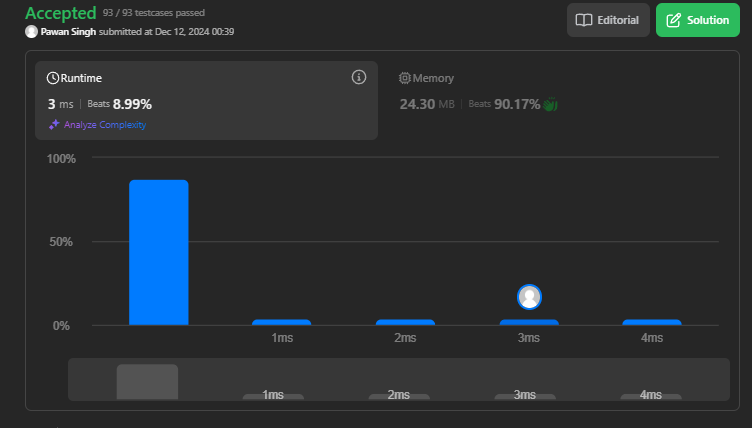
            return root->val;

        }

        return kthSmallest(root->right,k);

    }

};



102.[Binary Tree Level Order Traversal](https://leetcode.com/problems/binary-tree-level-order-traversal/description/)

class Solution {

public:

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

    vector<vector<int>>ans;

    if(root==nullptr)

    { return ans;   }

    queue<TreeNode\*>q;

    q.push(root);

    while(!q.empty())

    {

        vector<int>level;

        int n=q.size();

        for(int i=0;i<n;i++)

        {

            TreeNode\* node=q.front();

            q.pop();

            if(node->left != nullptr)

            {

                q.push(node->left);

            }

            if(node->right != nullptr)

            {

                q.push(node->right);

            }

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

        }

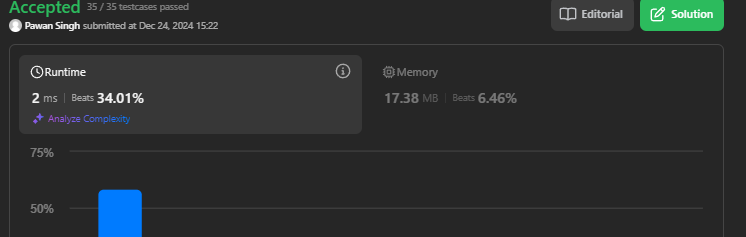
        ans.push\_back(level);

    }

return ans;

    }

};



107.[Binary Tree Level Order Traversal II](https://leetcode.com/problems/binary-tree-level-order-traversal-ii/description/)

class Solution {

public:

vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

vector<vector<int>> ans;

multiset<int> pq{0};

vector<pair<int, int>> points;

for(auto b: buildings){

points.push\_back({b[0], -b[2]});

points.push\_back({b[1], b[2]});

}

sort(points.begin(), points.end());

int ongoingHeight = 0;

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

int currentPoint = points[i].first;

int heightAtCurrentPoint = points[i].second;

if(heightAtCurrentPoint < 0){

pq.insert(-heightAtCurrentPoint);

} else {

pq.erase(pq.find(heightAtCurrentPoint));

}

auto pqTop = \*pq.rbegin();

if(ongoingHeight != pqTop){

ongoingHeight = pqTop;

ans.push\_back({currentPoint, ongoingHeight});

}

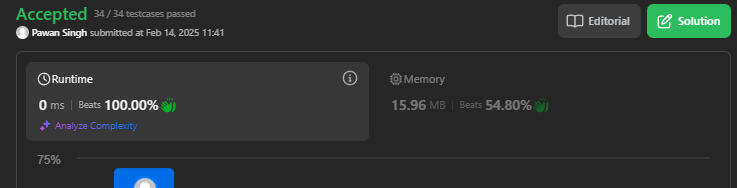
}

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

return ans;

}

};



103.[Binary Tree Zigzag Level Order Traversal](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/)

class Solution {

public:

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

       vector<vector<int>>res;

       if(!root)

       {return res;}

       queue<TreeNode\*>q;

       q.push(root);

       while(!q.empty())

       {

        vector<int>ans;

        int  n=q.size();

        int c=0;

        for(int i=0;i<n;i++)

        {

            TreeNode\* temp=q.front();

            q.pop();

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

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

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

        }

        if(c%2!=0)

        {

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

        }

         res.push\_back(ans);

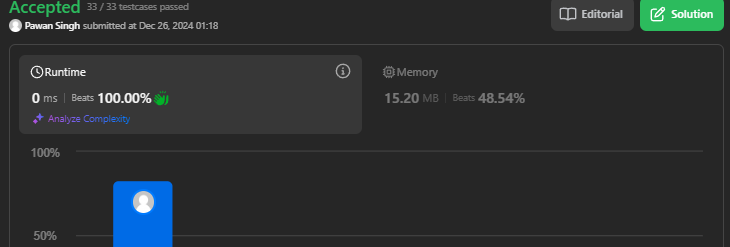
            c++;

       }

       return res;

    }

};



199.[Binary Tree Right Side View](https://leetcode.com/problems/binary-tree-right-side-view/description/)

class Solution {

public:

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

      if (!root) return {};

        vector<int> 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);

                }

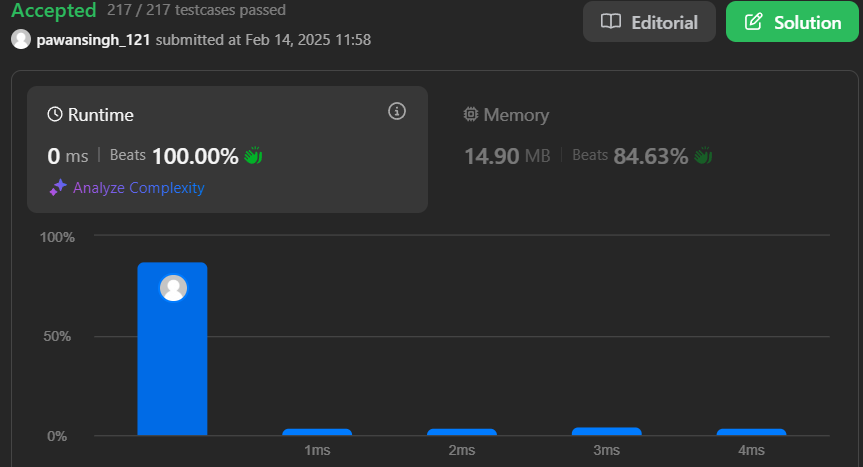
            }

        }

        return result;

    }

};



106.[Construct Binary Tree from Inorder and Postorder Traversal](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/description/)

class Solution {

public:

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

return buildTreeHelper(inorder, postorder, 0, inorder.size() - 1, postorder.size() - 1);

}

private:

TreeNode\* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int inStart, int inEnd, int postIndex) {

if (inStart > inEnd) return nullptr;

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

int inIndex = 0;

for (int i = inStart; i <= inEnd; ++i) {

if (inorder[i] == root->val) {

inIndex = i;

break;

}

}

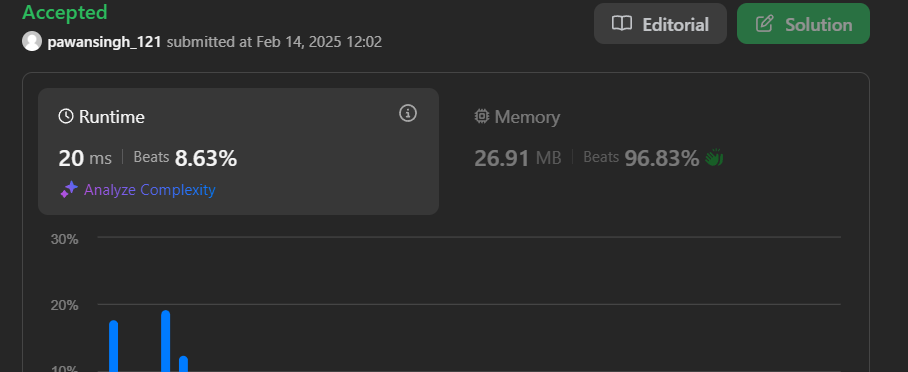
root->right = buildTreeHelper(inorder, postorder, inIndex + 1, inEnd, postIndex - 1);

root->left = buildTreeHelper(inorder, postorder, inStart, inIndex - 1, postIndex - (inEnd - inIndex) - 1);

return root;

}

};



513.[Find Bottom Left Tree Value](https://leetcode.com/problems/find-bottom-left-tree-value/description/)

class Solution {

public:

    int findBottomLeftValue(TreeNode\* root) {

        if (!root) return -1;

        queue<TreeNode\*> q;

        q.push(root);

        int result = -1;

        while (!q.empty()) {

            int size = q.size();

            result = q.front()->val;

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

                TreeNode\* node = q.front();

                q.pop();

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

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

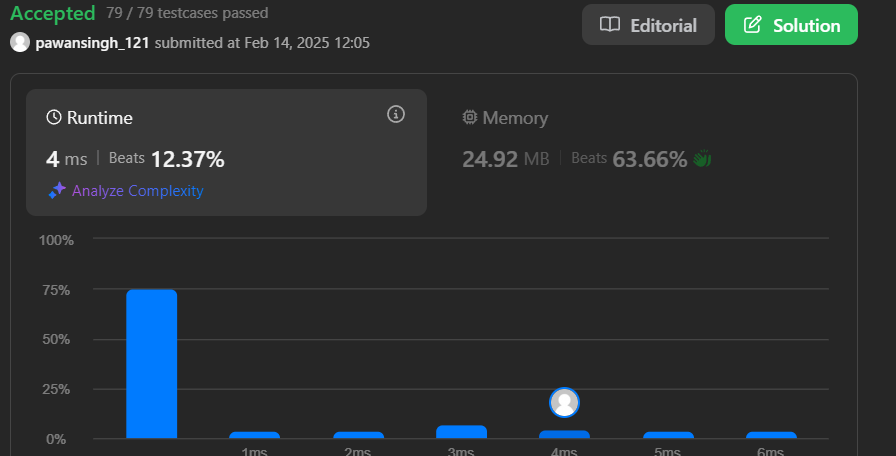
            }

        }

        return result;

    }

};



124.[Binary Tree Maximum Path Sum](https://leetcode.com/problems/binary-tree-maximum-path-sum/description/)

class Solution {

public:

    int maxPathSum(TreeNode\* root) {

        int maxSum = INT\_MIN;

        maxGain(root, maxSum);

        return maxSum;

    }

private:

    int maxGain(TreeNode\* node, int& maxSum) {

        if (!node) return 0;

        int leftGain = max(maxGain(node->left, maxSum), 0);

        int rightGain = max(maxGain(node->right, maxSum), 0);

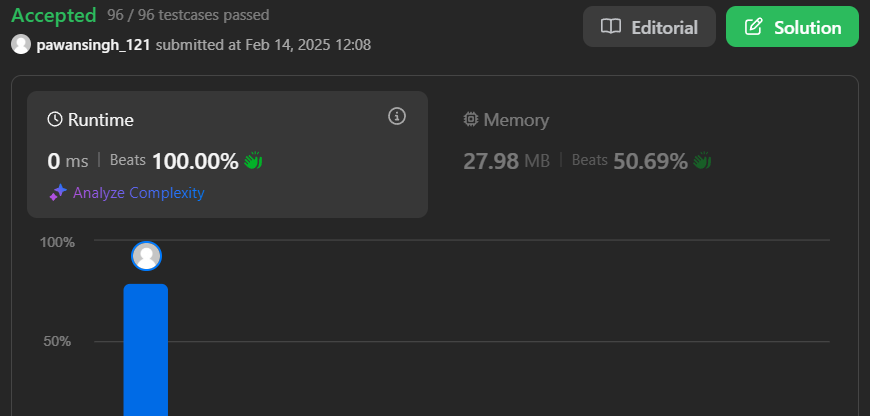
        int priceNewpath = node->val + leftGain + rightGain;

        maxSum = max(maxSum, priceNewpath);

        return node->val + max(leftGain, rightGain);

    }

};



987.[Vertical Order Traversal of a Binary Tree](https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/description/)

class Solution {

public:

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

if (!root) return {};

map<int, map<int, multiset<int>>> nodes;

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

q.push({root, {0, 0}});

while (!q.empty()) {

auto p = q.front();

q.pop();

TreeNode\* node = p.first;

int x = p.second.first, y = p.second.second;

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

if (node->left) {

q.push({node->left, {x - 1, y + 1}});

}

if (node->right) {

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

}

}

vector<vector<int>> result;

for (auto& p : nodes) {

vector<int> col;

for (auto& q : p.second) {

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

}

result.push\_back(col);

}

return result;

}

};

