**Assignment 3**

**Name: Aditi Singh UID: 22BCS10670**

**Section: FL\_IOT\_602 Group: A**

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

**Sol:** class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

List<Integer> res = new ArrayList<>();

inorder(root, res);

return res;

}

private void inorder(TreeNode node, List<Integer> res) {

if (node == null) {

return;

}

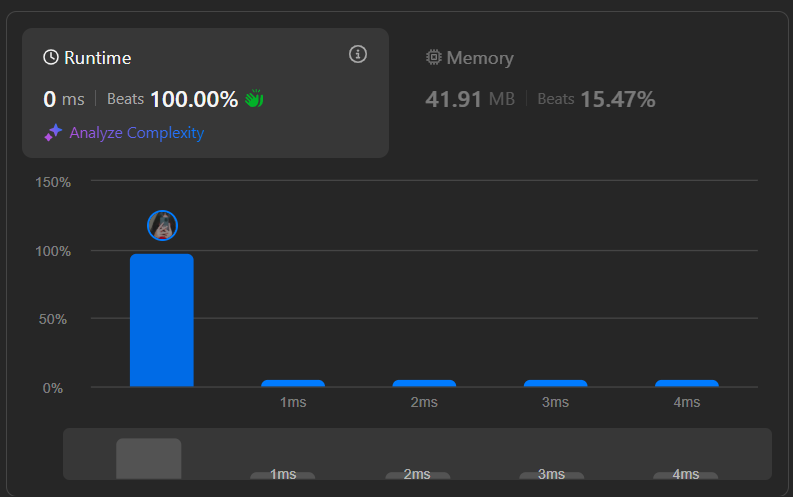
inorder(node.left, res);

res.add(node.val);

inorder(node.right, res);

}

}**Output:**



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

**Sol:** public class Solution {

    public boolean isSymmetric(TreeNode root) {

        if (root == null) return true;

        return isTrue(root.left, root.right);

    }

    private boolean isTrue(TreeNode t1, TreeNode t2) {

        if (t1 == null && t2 == null) return true;

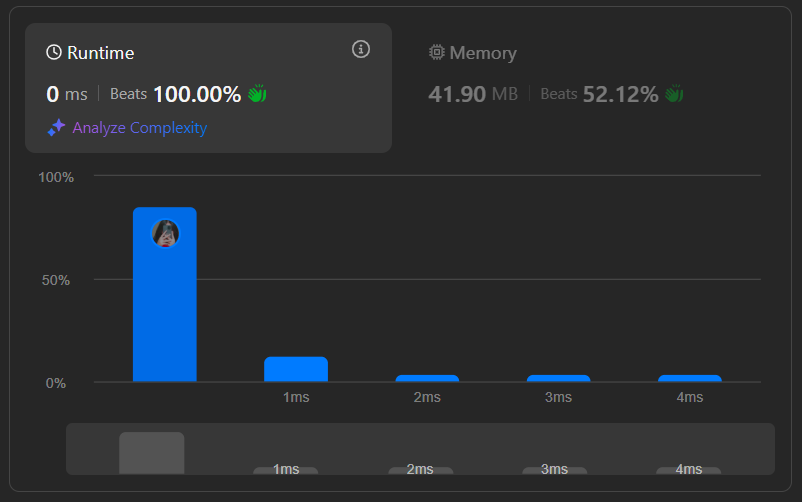
        if (t1 == null || t2 == null) return false;

        return t1.val == t2.val && isTrue(t1.left, t2.right) && isTrue(t1.right, t2.left);

    }

}

**Output:**

****

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

**Sol:** class Solution {

    public int maxDepth(TreeNode root) {

        if (root == null) {

            return 0;

        }

        int leftDepth = maxDepth(root.left);

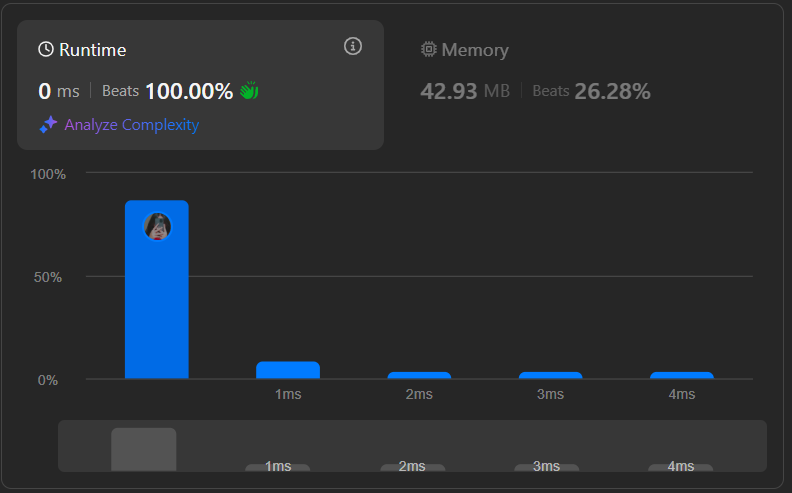
        int rightDepth = maxDepth(root.right);

        return Math.max(leftDepth, rightDepth) + 1;

    }

}

**Output:**

****

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

**Sol:** class Solution {

public boolean isValidBST(TreeNode root) {

return valid(root, Long.MIN\_VALUE, Long.MAX\_VALUE);

}

private boolean valid(TreeNode node, long minimum, long maximum) {

if (node == null) return true;

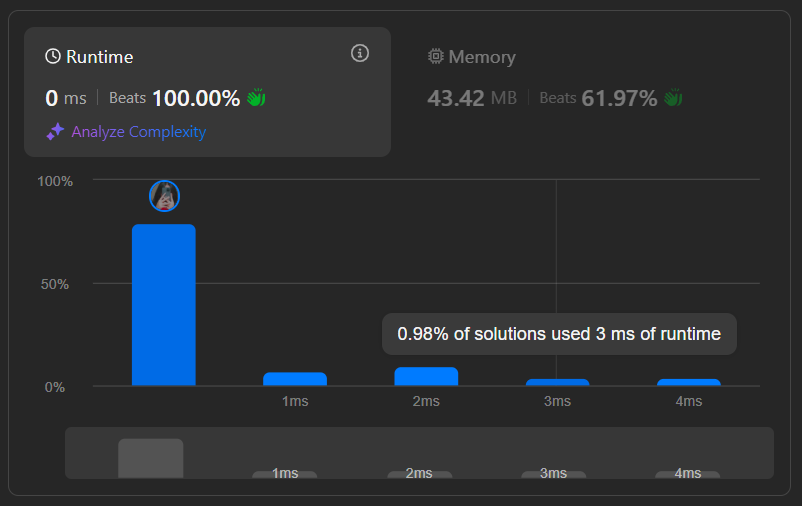
if (!(node.val > minimum && node.val < maximum)) return false;

return valid(node.left, minimum, node.val) && valid(node.right, node.val, maximum);

}

}

**Output:**

****

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

**Sol:** class Solution {

    private int count = 0;

    public int kthSmallest(TreeNode root, int k) {

        TreeNode result = helper(root, k);

        return result != null ? result.val : 0;

    }

    private TreeNode helper(TreeNode root, int k) {

        if (root == null) return null;

        TreeNode left = helper(root.left, k);

        if (left != null) return left;

        count++;

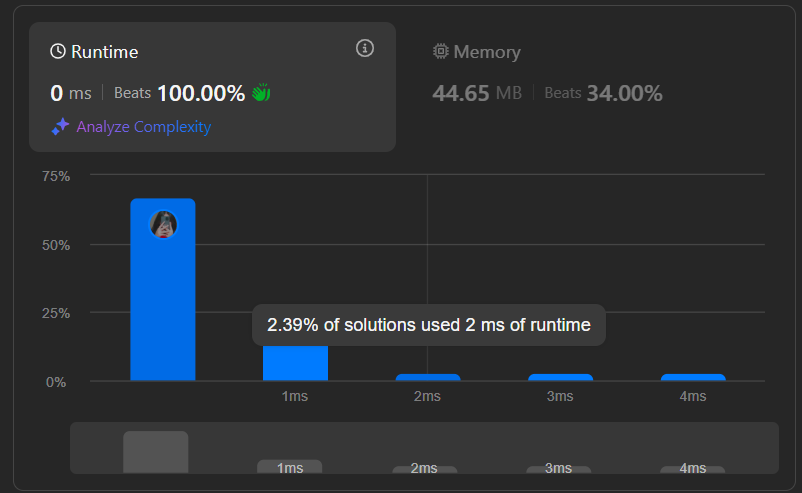
        if (count == k) return root;

        return helper(root.right, k);

    }

}

**Output:**

****

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

**Sol:**

class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

List<List<Integer>> result = new ArrayList<>();

if (root == null)return result;

Queue<TreeNode> queue = new LinkedList<>();

queue.offer(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

List<Integer> level = new ArrayList<>();

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

TreeNode current = queue.poll();

level.add(current.val);

if (current.left != null)queue.offer(current.left);

if (current.right != null)queue.offer(current.right);

}

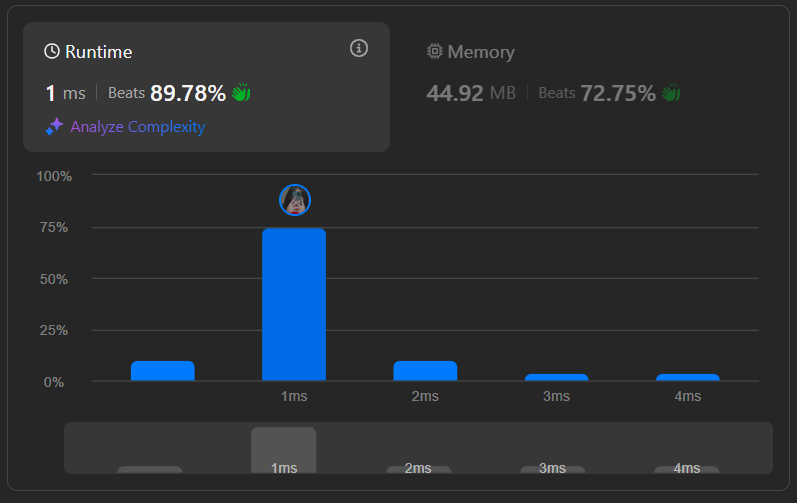
result.add(level);

}

return result;

}

}

**Output:**  

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

**Sol:** class Solution {

    public List<List<Integer>> levelOrderBottom(TreeNode root) {

        List<List<Integer>> levels = new ArrayList<>();

        if (root == null) return levels;

        Queue<TreeNode> queue = new LinkedList<>();

        queue.offer(root);

        while (!queue.isEmpty()) {

            int n = queue.size();

            List<Integer> level = new ArrayList<>();

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

                TreeNode node = queue.poll();

                level.add(node.val);

                if (node.left != null) queue.offer(node.left);

                if (node.right != null) queue.offer(node.right);

            }

            levels.add(0, level);

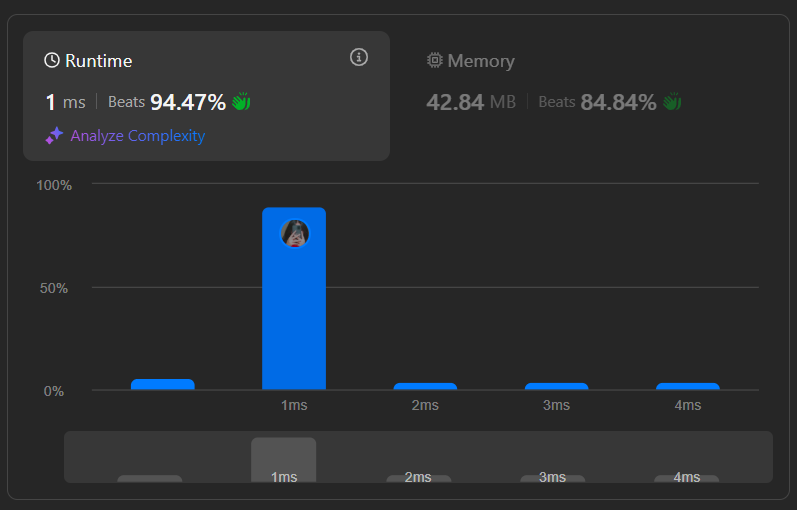
        }

        return levels;

    }

}

**Output:**



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

**Sol:** class Solution {

    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

        List<List<Integer>> res = new ArrayList<>();

        if (root == null) return res;

        Queue<TreeNode> q = new LinkedList<>();

        q.offer(root);

        boolean left = true;

        while (!q.isEmpty()) {

            int size = q.size();

            LinkedList<Integer> lvl = new LinkedList<>();

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

                TreeNode node = q.poll();

                if (left) lvl.addLast(node.val);

                else lvl.addFirst(node.val);

                if (node.left != null) q.offer(node.left);

                if (node.right != null) q.offer(node.right);

            }

            res.add(lvl);

            left = !left;

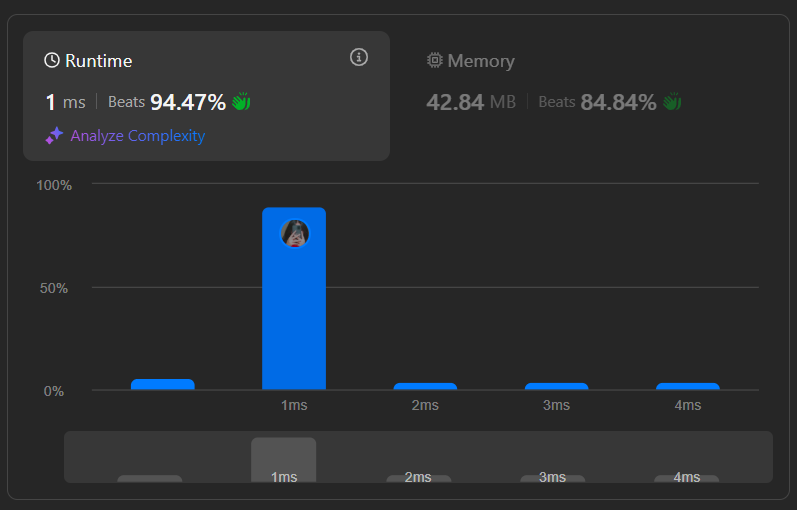
        }

        return res;

    }

}

**Output:**



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

**Sol:** class Solution {

List<Integer> result = new ArrayList<>();

public List<Integer> rightSideView(TreeNode root) {

dfs(root, 0);

return result;

}

private void dfs(TreeNode node, int depth){

if (node == null) return;

if (result.size() == depth){

result.add(node.val);

}

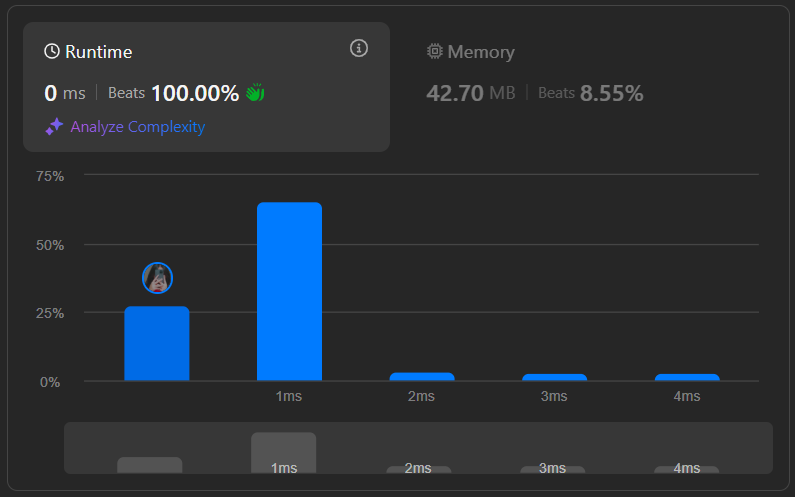
dfs(node.right, depth + 1);

dfs(node.left, depth + 1);

}

}

**Output:**



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

**Sol:** class Solution {

int i, o;

public TreeNode buildTree(int[] in, int[] po) {

i = o = po.length - 1;

return dfs(in, po, 3001);

}

private TreeNode dfs(int[] in, int[] po, int leftBoundary) {

if (o == -1 || in[i] == leftBoundary) return null;

TreeNode node = new TreeNode(po[o--]);

node.right = dfs(in, po, node.val);

i--;

node.left = dfs(in, po, leftBoundary);

return node;

}

}

}

**Output:**



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

**Sol:** class Solution {

    int depth=0;

    int b=0;

    public int findBottomLeftValue(TreeNode root) {

        bot(root,1);

        return b;

    }

    void bot(TreeNode root,int d)

    {

        if(root==null) return ;

        if(d>depth)

        {

            depth=d;

            b=root.val;

        }

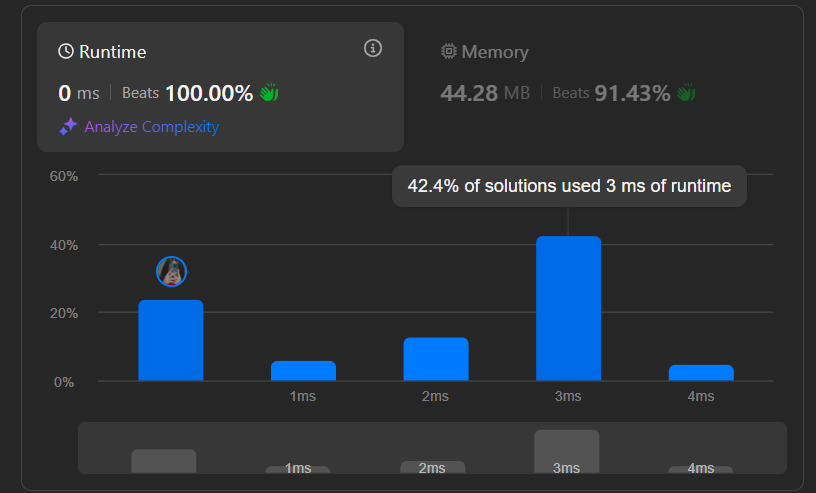
        bot(root.left,d+1);

        bot(root.right,d+1);

    }

}

**Output:**



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

**Sol:** class Solution {

     int MaxSum = Integer.MIN\_VALUE;

     int maxPathSum(TreeNode root) {

        dfs(root);

        return MaxSum;

    }

     int dfs(TreeNode node) {

        if (node == null) {

            return 0;

        }

        int leftMax = Math.max((dfs(node.left)), 0);

        int rightMax = Math.max((dfs(node.right)), 0);

        int PathSum = node.val + leftMax + rightMax;

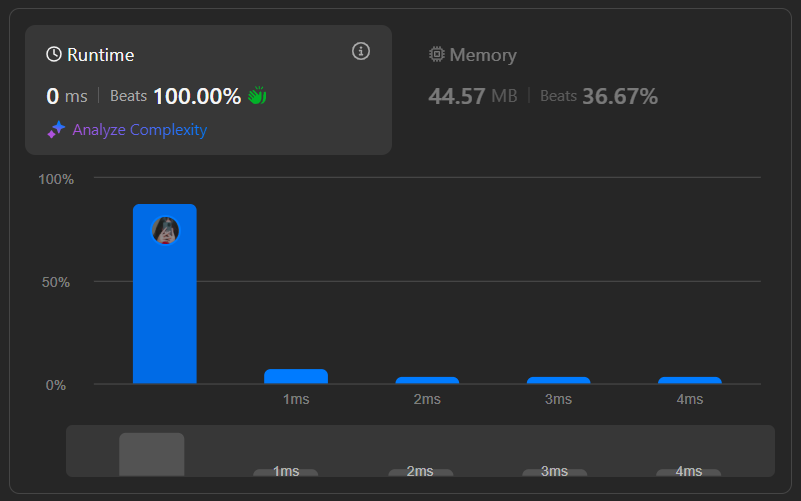
        MaxSum = Math.max(MaxSum, PathSum);

        return node.val + Math.max(leftMax, rightMax);

    }

}

**Output:**



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

**Sol:** class Solution {

Map<Integer, ArrayList<int[]>> map = new TreeMap<>();

public List<List<Integer>> verticalTraversal(TreeNode root) {

dfs(root, 0, 0);

List<List<Integer>> result = new ArrayList<>();

for(ArrayList<int[]> list: map.values()) {

Collections.sort(list, (a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) : Integer.compare(a[0], b[0]));

ArrayList<Integer> current = new ArrayList<>();

for(int[] num : list) {

current.add(num[1]);

}

result.add(current);

}

return result;

}

void dfs(TreeNode root, int index, int dept) {

if(root == null) {

return;

}

map.putIfAbsent(index, new ArrayList<>());

map.get(index).add(new int[]{dept, root.val});

dfs(root.left, index - 1, dept + 1);

dfs(root.right, index + 1, dept + 1);

}

}

**Output:**

