DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Assignment 5

Student Name: Aakriti Singh
Branch: CSE
Semester: 6th
Subject Name: Advance prog. Lab

UID: 22BCS11071
Section/Group: 22BCS_IOT_612
Date of Performance:04/03/25
Subject Code: 22CSP-351

Q1)Valid Parenthesis

- **Code:**

```
 class Solution {
public int maxDepth(TreeNode root) {
    if(root==null){
        return 0;
    }
    int leftht=maxDepth(root.left);
    int rightht=maxDepth(root.right);
    return Math.max(leftht,rightht)+1;
}
}
```

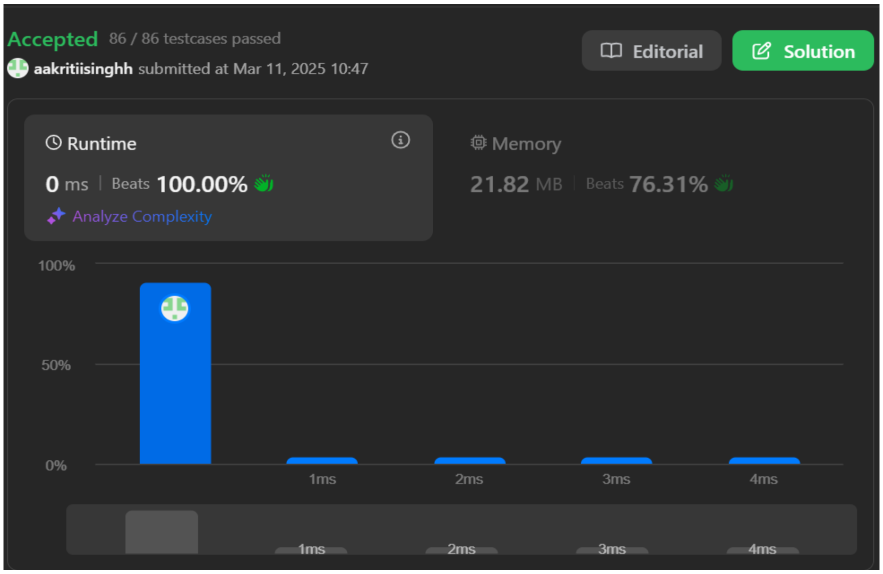
- **Screenshot:**

Q2) Validate Binary Search Tree

- **Code:**

```
    class Solution {
 public:
bool isValidBST(TreeNode* root) {
    return valid(root, LONG_MIN, LONG_MAX);
}

private:
    bool valid(TreeNode* node, long minimum, long maximum) {
        if (!node) return true;

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

        return valid(node->left, minimum, node->val) && valid(node->right, node->val, maximum);
    }
};
```

- **Screenshot:**



Accepted 86 / 86 testcases passed
aakritiisinghh submitted at Mar 11, 2025 10:47

Runtime: 0 ms | Beats 100.00%

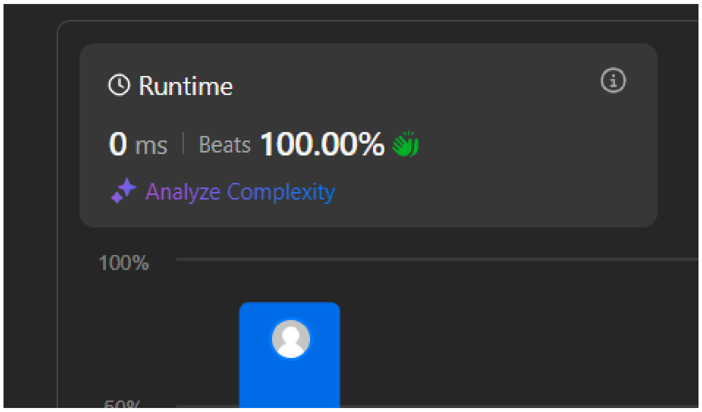
Memory: 21.82 MB | Beats 76.31%

Q3) <u>Symmetric Tree</u>

- **Code:**

```
class Solution {
    public boolean isMirror(TreeNode t1,TreeNode t2){
        if(t1==null && t2==null){
            return true;
        }
        if(t1==null || t2==null){
            return false;
        }
        return (t1.val == t2.val) && isMirror(t1.left, t2.right) && isMirror(t1.right, t2.left);
    }
    public boolean isSymmetric(TreeNode root) {
        if (root == null) {
            return true;
        }
        return isMirror(root.left, root.right);
    }
}
```

- **Screenshot:**



Runtime

0 ms | Beats 100.00%

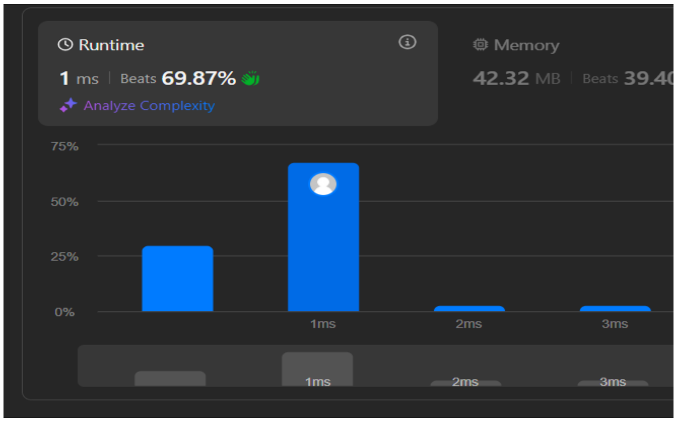
Analyze Complexity

Q4) Binary Tree Zigzag Level Order Traversal

- **Code:**

```
          class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>>result=new ArrayList<>();
        if (root == null) return result;
        Queue<TreeNode>q=new LinkedList<>();
        q.add(root);
        int level = 0;
        while(!q.isEmpty()){
            int size=q.size();
            List<Integer>ls=new LinkedList<>();
            for(int i=0;i<size;i++){
                TreeNode curr=q.poll();
                ls.add(curr.val);

                if (curr.left != null) q.add(curr.left);
                if (curr.right != null) q.add(curr.right);
            }
            if (level % 2 == 1) {
                Collections.reverse(ls);
            }
            result.add(new ArrayList<>(ls));
            level++;
        }
        return result;
    }
}
```

- **Screenshot:**

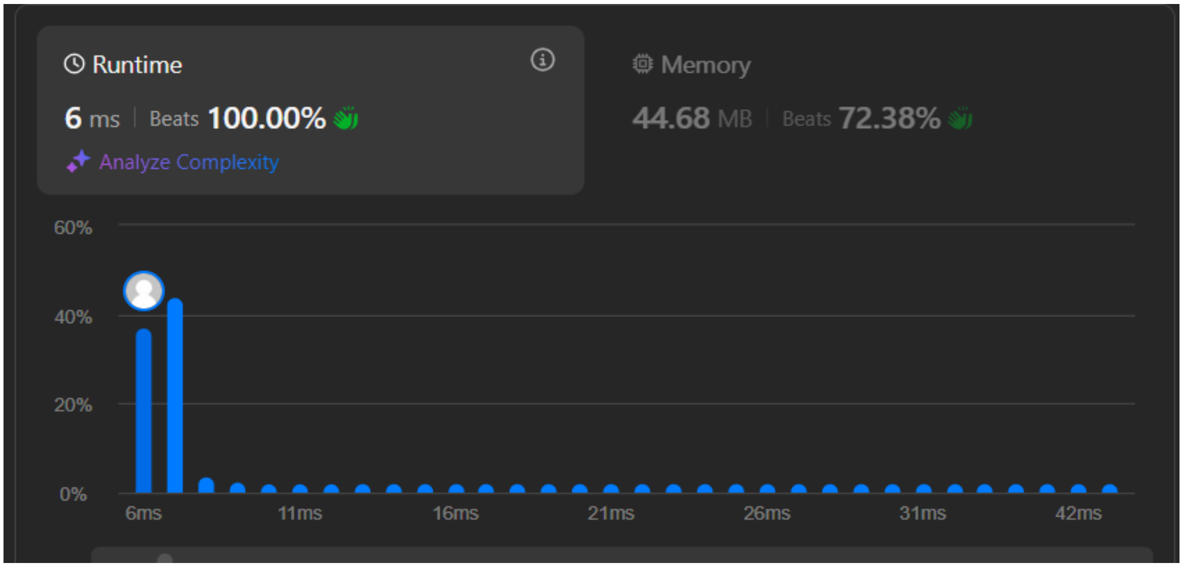


Q5) Lowest Common Ancestor of a Binary Tree

- **Code:**

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        // Base case: null node
        if (root == null) return null;

        // If the current node is either p or q, return it
        if (root == p || root == q) return root;

        // Recur for left and right children
        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);

        // If both left and right return a non-null value, current node is LCA
        if (left != null && right != null) return root;

        // Otherwise, return the non-null child (or null if both are null)
        return left != null ? left : right;
    }
}
```

- **Screenshot:**

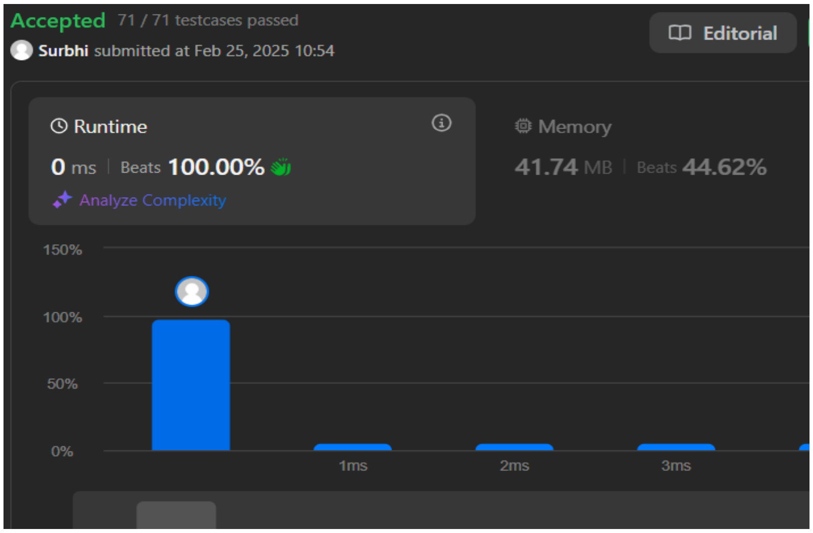


Q6) Binary Tree Inorder Traversal

- **Code:**

```
    class Solution {
  public void Traversal(TreeNode root,List<Integer>ans){
    if(root==null) return;
    Traversal(root.left,ans);
    ans.add(root.val);
    Traversal(root.right,ans);
  }
  public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer>ans=new ArrayList<>();
    Traversal(root,ans);
    return ans;
  }
}
```

- **Screenshot:**



Accepted 71 / 71 testcases passed

Surbhi submitted at Feb 25, 2025 10:54

Runtime

0 ms | Beats 100.00%

Memory

41.74 MB | Beats 44.62%

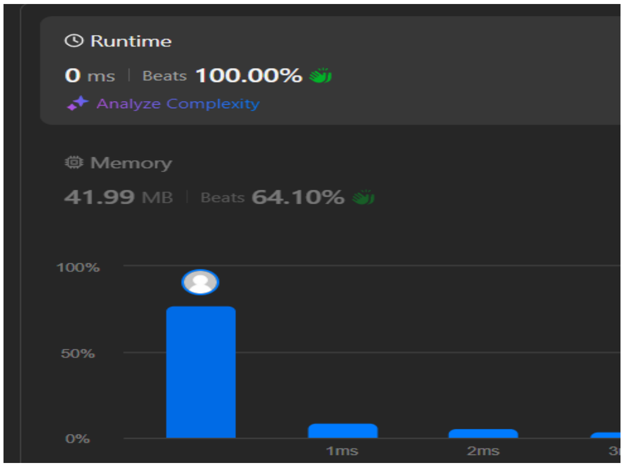
Q7) Binary Tree Level Order Traversal

- **Code:**

```
class Solution {
  public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> ans = new ArrayList<>();
    if (root == null) return ans;

    Queue<TreeNode> queue = new LinkedList<>();
    queue.add(root);

    while (!queue.isEmpty()) {
      int levelSize = queue.size();
      List<Integer> level = new ArrayList<>();

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

```
                TreeNode node = queue.poll();
                level.add(node.val);

                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right);
            }
            ans.add(level);
        }
        return ans;
    }
}
```

- **Screenshot:**



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

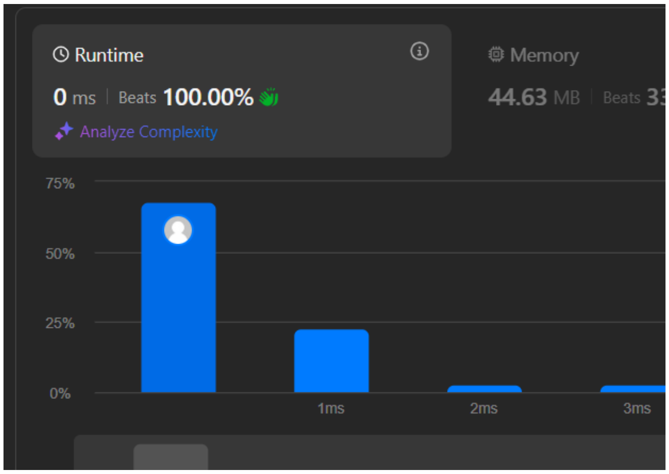
41.99 MB | Beats 64.10%

Q8) <u>Kth smallest element in a BST</u>

- **Code:**

```
class Solution {
    private int count = 0;
    private int result = 0;

    public void inOrder(TreeNode root,int k){
        if(root==null){
            return;
        }
        inOrder(root.left,k);
        count++;
        if (count == k) {
            result = root.val;
            return;
        }
        inOrder(root.right,k);
    }
    public int kthSmallest(TreeNode root, int k) {
        inOrder(root,k);
        return result;
    }
}
```

- **Screenshot:**



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

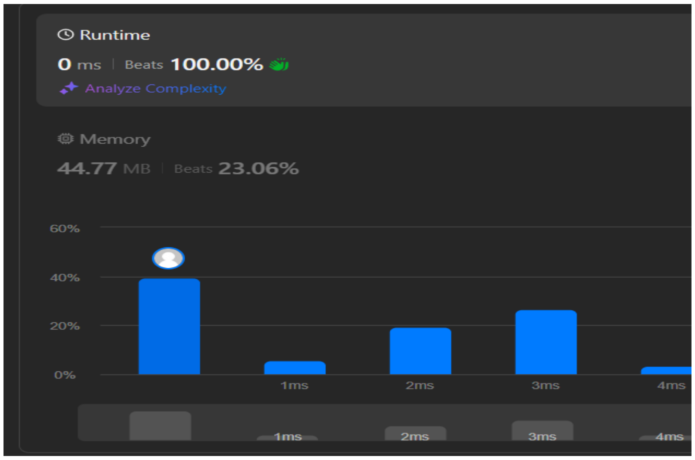
44.63 MB | Beats 33

Q9) Populating Next Right Pointers in Each Node

- **Code:**

```
class Solution {
 public Node connect(Node root) {
    if (root == null) return null;
    if (root.left != null) root.left.next = root.right;

    if (root.right != null && root.next != null) root.right.next = root.next.left;
    connect(root.left);
    connect(root.right);
    return root;
 }
}
```

- **Screenshot:**



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

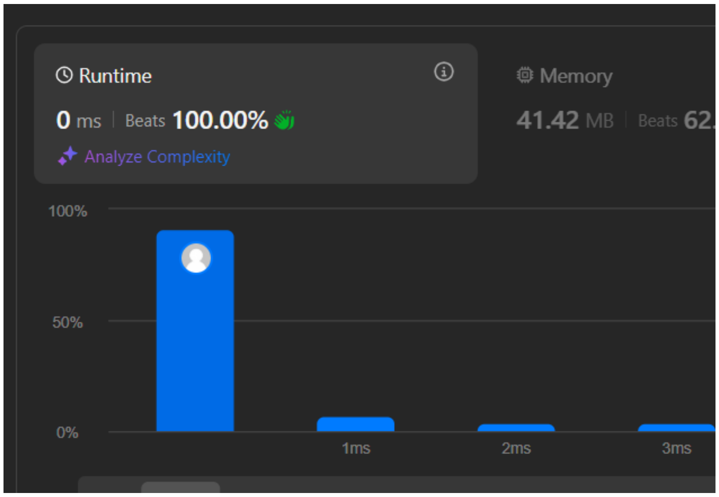
44.77 MB | Beats 23.06%

Q10) Sum of Left Leaves

- **Code:**

```
class Solution {
   public int calSum(TreeNode node){
      if (node == null) {
         return 0;
      }
      int sum = 0;
      if(node.left!= null && node.left.left==null && node.left.right==null){
            sum+=node.left.val;
        }
        sum += calSum(node.left);
        sum += calSum(node.right);
        return sum;
    }
    public int sumOfLeftLeaves(TreeNode root) {
        return calSum(root);
    }
}
```

- **Screenshot:**



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

41.42 MB | Beats 62.