

NAME: Shashank Kumar uid: 22BCS13407 Sec- 605(B)

## 1. Binary Tree Inorder Traversal

The screenshot shows a LeetCode submission for the problem "Binary Tree Inorder Traversal". The submission is accepted, with 71/71 testcases passed. The runtime is 0 ms, beating 100.00% of submissions, and the memory usage is 41.43 MB, beating 91.28%. The code is written in Java and implements an inorder traversal of a binary tree, returning the values in a list.

**Runtime:** 0 ms | Beats 100.00%  
**Memory:** 41.43 MB | Beats 91.28%

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> list = new ArrayList<>();
        inorder(root, list);
        return list;
    }
    private void inorder(TreeNode root, List<Integer> list) {
        if (root == null) return;
        inorder(root.left, list);
        list.add(root.val);
        inorder(root.right, list);
    }
}
```

## 2. Symmetric Tree

The screenshot shows a LeetCode submission for the problem "Symmetric Tree". The submission is accepted, with a runtime of 0 ms. The code is written in Java and implements a recursive function to check if a binary tree is symmetric.

**Accepted** Runtime: 0 ms

**Case 1** | Case 2

Input: root = [1,2,2,3,4,4,3]

Output: true

Expected: true

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        return isMirror(root.left, root.right);
    }
    private 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);
    }
}
```

### 3. Maximum Depth of binary tree

The screenshot shows a LeetCode submission for the problem "Maximum Depth of binary tree". The submission is accepted, with 39/39 test cases passed. The runtime is 0 ms, beating 100.00% of submissions, and the memory is 42.54 MB, beating 76.42% of submissions. A bar chart shows the runtime distribution, with the majority of submissions falling between 0 and 1 ms. The code is written in Java and defines a `TreeNode` class and a `maxDepth` method in the `Solution` class.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {
        //if root is null, then there is no depth return 0
        if(root == null)
            return 0;

        // stores the length of left subtree
        int left = maxDepth(root.left);
        //stores the length of right subtree
        int right = maxDepth(root.right);

        //return the maximum of two
        return Math.max(left, right) + 1;
    }
}
```

### 4. Validate binary search tree

The screenshot shows a LeetCode submission for the problem "Validate binary search tree". The submission is accepted, with 86/86 test cases passed. The runtime is 1 ms, beating 20.90% of submissions, and the memory is 44.43 MB, beating 23.70% of submissions. A bar chart shows the runtime distribution, with the majority of submissions falling between 0 and 1 ms. The code is written in Java and defines a `TreeNode` class and a `isValidBST` method in the `Solution` class.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 * }
 */
class Solution {
    public void inorder(List<Integer> res, TreeNode root)
    {
        if(root == null)
        {
            return;
        }
        inorder(res, root.left);
        res.add(root.val);
        inorder(res, root.right);
    }

    public boolean isValidBST(TreeNode root) {
        ArrayList<Integer> res = new ArrayList<Integer>();
        inorder(res, root);
        for(int i = 0; i < res.size() - 1; i++)
        {
            if(res.get(i) >= res.get(i + 1))
            {
                return false;
            }
        }
        return true;
    }
}
```

## 5. Kth Smallest Element in a BST

The screenshot shows the submission interface for the problem "Kth Smallest Element in a BST". The submission is accepted, with 93/93 testcases passed. The runtime is 0ms, which beats 100.00% of other submissions. The memory usage is 44.93 MB, which beats 8.98% of other submissions. A bar chart shows the distribution of runtimes, with the majority being 0ms. The code is written in Java and implements a recursive approach to find the kth smallest element in a Binary Search Tree (BST).

```
int res = -1;
while(curr != null){
    TreeNode left = curr.left;
    if(left == null){
        k--;
        if(k == 0){
            res = curr.val;
            break;
        }
        curr = curr.right;
    }else{
        TreeNode leftRightMost = getLeftRightMost(curr.left, curr);
        if(leftRightMost.right == null){
            leftRightMost.right = curr;
            curr = curr.left;
        }else{
            leftRightMost.right = null;
            k--;
            if(k == 0){
                res = curr.val;
                break;
            }
            curr = curr.right;
        }
    }
}
```

## 6. Binary Tree Level Order Traversal

The screenshot shows the submission interface for the problem "Binary Tree Level Order Traversal". The submission is accepted, with 35/35 testcases passed. The runtime is 1ms, which beats 89.92% of other submissions. The memory usage is 45.18 MB, which beats 34.91% of other submissions. A bar chart shows the distribution of runtimes, with the majority being 1ms. The code is written in Java and implements a breadth-first search (BFS) approach to traverse the binary tree level by level.

```
public class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        List<List<Integer>> wrapList = new LinkedList<List<Integer>>();

        if(root == null) return wrapList;
        queue.offer(root);
        while(!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new LinkedList<Integer>();
            for(int i=0; i<levelNum; i++){
                if(queue.peek().left != null) queue.offer(queue.peek().left);
                if(queue.peek().right != null) queue.offer(queue.peek().right);
                subList.add(queue.poll().val);
            }
            wrapList.add(subList);
        }
        return wrapList;
    }
}
```

## 7. Binary Tree Level Order Traversal II

Accepted 34 / 34 testcases passed  
Shanky2811 submitted at Feb 19, 2025 11:36

Runtime: 1 ms | Beats: 94.57%  
Memory: 43.08 MB | Beats: 50.02%

Code | Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;

```

```
12 *         this.right = right;
13 *     }
14 * }
15 */
16
17 class Solution {
18     public List<List<Integer>> levelOrderBottom(TreeNode root) {
19         List<List<Integer>> arr = new ArrayList<>();
20         if (root == null) return arr;
21         Queue<TreeNode> queue = new LinkedList<>();
22         queue.offer(root);
23         while (!queue.isEmpty()) {
24             int size = queue.size();
25             ArrayList<Integer> sub = new ArrayList<>();
26             while (size > 0) {
27                 TreeNode node = queue.poll();
28                 if (node.left != null) {
29                     queue.offer(node.left);
30                 }
31                 if (node.right != null) {
32                     queue.offer(node.right);
33                 }
34                 sub.add(node.val);
35                 size--;
36             }
37             arr.add(sub);
38         }
39         Collections.reverse(arr);
40         return arr;
41     }

```

Ln 41, Col 2 Saved Run Submit

## 8. Binary tree zigzag level order traversal

Accepted 33 / 33 testcases passed  
Shanky2811 submitted at Feb 19, 2025 11:36

Runtime: 0 ms | Beats: 100.00%  
Memory: 42.18 MB | Beats: 76.62%

Code | Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;

```

```
22 }
23
24 private void leftzigzag(TreeNode root, int level, List<List<Integer>> l){
25     if (root == null) {
26         return;
27     }
28     if (l.size() < level) {
29         l.add(new ArrayList<>());
30     }
31     if (level % 2 == 0) {
32         l.get(level-1).add(root.val);
33     }
34     leftzigzag(root.right, level+1, l);
35     leftzigzag(root.left, level+1, l);
36     return;
37 }
38
39 private void rightzigzag(TreeNode root, int level, List<List<Integer>> l){
40     if (root == null) {
41         return;
42     }
43     if (l.size() < level) {
44         l.add(new ArrayList<>());
45     }
46     if (level % 2 != 0) {
47         l.get(level-1).add(root.val);
48     }
49     rightzigzag(root.left, level+1, l);
50     rightzigzag(root.right, level+1, l);
51     return;

```

Ln 51, Col 2 Saved Run Submit

## 9. Binary Tree Right Side View

The screenshot shows a LeetCode submission for the "Binary Tree Right Side View" problem. The submission is accepted, with 217 test cases passed. The runtime is 1 ms, which is 72.61% faster than the average, and the memory usage is 42.04 MB, which is 78.73% less than the average. The code is in Java and implements a recursive solution to find the right side view of a binary tree.

**Runtime:** 1 ms | Beats 72.61%  
**Memory:** 42.04 MB | Beats 78.73%

**Code (Java):**

```
8 * TreeNode(int val) { this.val = val; }
9 * TreeNode(int val, TreeNode left, TreeNode right) {
10 *     this.val = val;
11 *     this.left = left;
12 *     this.right = right;
13 * }
14 *
15 */
16 class Solution {
17
18     List<Integer> ls = new ArrayList<>();
19     Set<Integer> set = new HashSet<>();
20
21     public List<Integer> rightSideView(TreeNode root) {
22         helper(root, 0);
23         return ls;
24     }
25
26     public void helper(TreeNode root, int level) {
27         if (root == null) return;
28
29         if (set.add(level)) {
30             // This ensures we only add the first node at each level
31             ls.add(root.val);
32         }
33
34         helper(root.right, level + 1);
35         helper(root.left, level + 1);
36     }
37 }
```

## 10. Construct binary tree from inorder and post order traversal

Problem List < > > >

Description Editorial Solutions Accepted Submissions Testcase Test Result

All Submissions

Accepted 202 / 202 testcases passed  
Shanky2811 submitted at Feb 19, 2025 11:38

Editorial Solution

Runtime 0 ms | Beats 100.00%  
Memory 44.06 MB | Beats 91.96%

Analyze Complexity

Code | Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;

```

```

8  *   TreeNode(int val) { this.val = val; }
9  *   TreeNode(int val, TreeNode left, TreeNode right) {
10 *       this.val = val;
11 *       this.left = left;
12 *       this.right = right;
13 *   }
14 * }
15 */
16 class Solution {
17     int i=0;
18     int j=0;
19     public TreeNode buildTree(int[] inorder, int[] postorder) {
20         i=inorder.length-1;
21         j=i;
22         return buildTree(inorder,postorder,-3001);
23     }
24     public TreeNode buildTree(int[] inorder, int[] postorder,int x) {
25         if(i<0)
26             return null;
27         if(inorder[j]==x){
28             j--;
29             return null;
30         }
31
32         TreeNode cur = new TreeNode(postorder[i--]);
33         cur.right = buildTree(inorder,postorder,cur.val);
34         cur.left = buildTree(inorder,postorder,x);
35         return cur;
36     }
37 }

```

Ln 37, Col 2 Saved Run Submit

## 11. Find Bottom Left Tree Value

Problem List < > > >

Description Editorial Solutions Accepted Submissions Testcase Test Result

All Submissions

Accepted 79 / 79 testcases passed  
Shanky2811 submitted at Feb 19, 2025 11:38

Editorial Solution

Runtime 3 ms | Beats 57.24%  
Memory 44.54 MB | Beats 71.22%

Analyze Complexity

Code | Java

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;

```

```

27     int levelSize=q.size();
28
29     for(int i=0;i<levelSize;i++){
30         TreeNode curr=q.remove();
31
32
33         if(i==0){
34             leftBottomValue=curr.val; // it will add very first Node at every
35             level to it,and end up adding the leftMost value of bottom level
36         }
37
38         if(curr.left!=null){
39             q.add(curr.left);
40         }
41         if(curr.right!=null){
42             q.add(curr.right);
43         }
44
45     }
46     return leftBottomValue;
47 }
48
49 }
50
51
52 /**
53
54 if ques is for bottom right then simple check conditions with
55 if(i==size-1) */

```

Ln 55, Col 17 Saved Run Submit

## 12. Binary Tree Maximum Path Sum

The screenshot shows a LeetCode problem page for "Binary Tree Maximum Path Sum". The solution is accepted, with 96/96 testcases passed. The runtime is 0 ms, beating 100.00% of solutions. The memory is 44.60 MB, beating 26.75% of solutions. A bar chart shows the runtime distribution. The code is in Java, using a recursive approach to find the maximum path sum.

**Accepted** 96 / 96 testcases passed  
Shanky2811 submitted at Feb 19, 2025 11:39

**Runtime** 0 ms | Beats 100.00%  
**Memory** 44.60 MB | Beats 26.75%

Code | Java

```
public class Solution {
    private int maxSum = Integer.MIN_VALUE;

    public int maxPathSum(TreeNode root) {
        maxGain(root);
        return maxSum;
    }

    private int maxGain(TreeNode node) {
        if (node == null) return 0;

        int leftGain = Math.max(maxGain(node.left), 0);
        int rightGain = Math.max(maxGain(node.right), 0);

        int priceNewPath = node.val + leftGain + rightGain;
        maxSum = Math.max(maxSum, priceNewPath);
        return node.val + Math.max(leftGain, rightGain);
    }
}
```

## 13. Vertical Order Traversal of a Binary Tree

The screenshot shows a LeetCode problem page for "Vertical Order Traversal of a Binary Tree". The solution is accepted, with 34/34 testcases passed. The runtime is 2 ms, beating 99.53% of solutions. The memory is 42.68 MB, beating 49.91% of solutions. A bar chart shows the runtime distribution. The code is in Java, using a BFS approach to traverse the tree vertically.

**Accepted** 34 / 34 testcases passed  
Shanky2811 submitted at Feb 19, 2025 11:40

**Runtime** 2 ms | Beats 99.53%  
**Memory** 42.68 MB | Beats 49.91%

Code | Java

```
class Solution {
    static int leastCol;
    static int mostCol;

    static List<List<Integer>> ans;

    public List<List<Integer>> verticalTraversal(TreeNode root) {
        ans = new ArrayList<>();
        bfs(root, 0, 0);
        return ans;
    }

    private void bfs(TreeNode node, int row, int col) {
        if (node == null) return;
        pq.add(new Pair(row, col));
        if (node.left != null) {
            q.add(new Pair(row+1, col-1));
            leastCol = Math.min(leastCol, col-1);
        }
        if (node.right != null) {
            q.add(new Pair(row+1, col+1));
            mostCol = Math.max(mostCol, col+1);
        }
    }

    static class Pair implements Comparable<Pair> {
        int row;
        int col;
        Pair(int r, int c, int n) {
            row = r;
            col = c;
            node = n;
        }
        public int compareTo(Pair p) {
            if (col < p.col) return -1;
            if (col > p.col) return 1;
            return 0;
        }
    }
}
```