Name : Kumar Devashish **UID** : 22BCS10248
section/grp.: 22BCS_FL_IOT-602-A **Subject** : AP Lab
Branch : CSE **Semester** : 6th
Date : 14/02/2025 **Subject Code :** 22CSP-351

Q.1 Binary Tree Inorder Traversal

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        TreeNode current = root;

        while (current != null) {
            if (current.left == null) {
                result.add(current.val);
                current = current.right;
            } else {
                TreeNode predecessor = current.left;
                while (predecessor.right != null && predecessor.right != current) {
                    predecessor = predecessor.right;
                }
                if (predecessor.right == null) {
                    predecessor.right = current;
                    current = current.left;
                } else {
                    predecessor.right = null;
                    result.add(current.val);
                    current = current.right;
                }
            }
        }

        return result;
    }
}
```

Q.2 Symmetric Tree

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) {
            return true;
        }
        return sym(root.left, root.right);
    }

    private boolean sym(TreeNode t1, TreeNode t2) {
```

```
        if (t1 == null && t2 == null) {
            return true;
        }

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

        return (t1.val == t2.val) && sym(t1.left, t2.right) && sym(t1.right, t2.left);
    }
}
```

Q. 3 Maximum Depth of Binary Tree

```
class Solution {
    public int maxDepth(TreeNode root) {

        if (root == null) {
            return 0;
        }
          int ld = maxDepth(root.left);
        int rd = maxDepth(root.right);


        return Math.max(ld, rd) + 1;
    }
}
```

Q. 4 Validate Binary Search Tree

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        Stack<TreeNode> stack = new Stack<>();
        TreeNode current = root;
        long prev = Long.MIN_VALUE;

        while (current != null || !stack.isEmpty()) {
            while (current != null) {
                stack.push(current);
                current = current.left;
            }
            current = stack.pop();

            if (current.val <= prev) return false;
            prev = current.val;
            current = current.right;
        }
        return true;
    }
}
```

Q.5 Kth Smallest Element in a BST

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

    public int kthSmallest(TreeNode root, int k) {
        inorder(root, k);
        return result;
    }

    private void inorder(TreeNode node, int k) {
        if (node == null) return;

        inorder(node.left, k);

        count++;
        if (count == k) {
            result = node.val;
            return;
        }

        inorder(node.right, k);
    }
}
```

Q.6 Binary Tree Level Order Traversal

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

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

Q.7 Binary Tree Level Order Traversal II

```
class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        List<List<Integer>> result = new LinkedList<>();
        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 node = queue.poll();
                level.add(node.val);

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

            result.addFirst(level);
        }

        return result;
    }
}
```

Q.8 Binary Tree Zigzag Level Order Traversal

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

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

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

            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();

                if (leftToRight) {
                    level.add(node.val);
                } else {
                    level.add(0, node.val);
                }
```

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

        result.add(level);
        leftToRight = !leftToRight;
    }

    return result;
  }
}
```

Q.9 Binary Tree Right Side View

```
class Solution {
  public List<Integer> rightSideView(TreeNode root) {
    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();
      int rightMostValue = 0;

      for (int i = 0; i < levelSize; i++) {
        TreeNode node = queue.poll();
        rightMostValue = node.val;

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

      result.add(rightMostValue);
    }

    return result;
  }
}
```

Q.10 Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {
  private Map<Integer, Integer> inorderIndexMap;
  private int postIndex;

  public TreeNode buildTree(int[] inorder, int[] postorder) {
    if (inorder == null || postorder == null || inorder.length != postorder.length) return null;

    inorderIndexMap = new HashMap<>();
```

```
        postIndex = postorder.length - 1;


        for (int i = 0; i < inorder.length; i++) {
            inorderIndexMap.put(inorder[i], i);
        }

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

    private TreeNode buildTreeHelper(int[] inorder, int[] postorder, int inLeft, int inRight) {
        if (inLeft > inRight) return null;


        int rootValue = postorder[postIndex--];
        TreeNode root = new TreeNode(rootValue);
        int inIndex = inorderIndexMap.get(rootValue);
        root.right = buildTreeHelper(inorder, postorder, inIndex + 1, inRight);
        root.left = buildTreeHelper(inorder, postorder, inLeft, inIndex - 1);

        return root;
    }
}
```

Q.11 Find Bottom Left Tree Value

```
class Solution {
    public int findBottomLeftValue(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        int bottomLeftValue = root.val;

        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            bottomLeftValue = node.val; // Update latest leftmost value

            if (node.right != null) queue.offer(node.right);  // Right is enqueued first
            if (node.left != null) queue.offer(node.left);    // Left is enqueued last
        }

        return bottomLeftValue;
    }
}
```

Q.12 Binary Tree Maximum Path Sum

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

    public int maxPathSum(TreeNode root) {
        dfs(root);
        return maxSum;
```

```
    }

    private int dfs(TreeNode node) {
        if (node == null) return 0;
        int left = Math.max(0, dfs(node.left));
        int right = Math.max(0, dfs(node.right));
        int localMax = node.val + left + right;
        maxSum = Math.max(maxSum, localMax);
        return node.val + Math.max(left, right);
    }
}
```

Q.13 Vertical Order Traversal of a Binary Tree

```
class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {

        TreeMap<Integer, List<int[]>> columnMap = new TreeMap<>();
        Queue<Tuple> queue = new LinkedList<>();
        queue.offer(new Tuple(root, 0, 0));
    while (!queue.isEmpty()) {
            Tuple t = queue.poll();
            TreeNode node = t.node;
            int col = t.col, row = t.row;

            columnMap.putIfAbsent(col, new ArrayList<>());
            columnMap.get(col).add(new int[]{row, node.val});

            if (node.left != null) queue.offer(new Tuple(node.left, col - 1, row + 1));
            if (node.right != null) queue.offer(new Tuple(node.right, col + 1, row + 1));
        }
        List<List<Integer>> result = new ArrayList<>();
        for (List<int[]> nodeList : columnMap.values()) {
            nodeList.sort((a, b) -> a[0] == b[0] ? a[1] - b[1] : a[0] - b[0]);
            List<Integer> sortedValues = new ArrayList<>();
            for (int[] pair : nodeList) sortedValues.add(pair[1]);
            result.add(sortedValues);
        }
        return result;
    }
    private static class Tuple {
        TreeNode node;
        int col, row;
        Tuple(TreeNode node, int col, int row) {
            this.node = node;
            this.col = col;
            this.row = row;
        }
    }
}
```