# ASSIGNMENT 18TH SEPTEMBER TREE

**Q1. Given the root of a binary tree, return the spiral level order traversal of its nodes' values.**

```java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Stack;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

public List<List<Integer>> spiralOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    if (root == null) {
        return result;
    }

    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    boolean reverseLevel = false;

    while (!queue.isEmpty()) {
        int levelSize = queue.size();
        List<Integer> levelValues = new ArrayList<>();
        Stack<TreeNode> levelStack = new Stack<>();

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

```
            TreeNode node = queue.poll();

            if (reverseLevel) {
                levelStack.push(node);
            } else {
                levelValues.add(node.val);
            }

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

        if (reverseLevel) {
            while (!levelStack.isEmpty()) {
                levelValues.add(levelStack.pop().val);
            }
        }

        result.add(levelValues);
        reverseLevel = !reverseLevel;
    }

    return result;
}
```

**Q2. Given the root of a binary tree, check if it is a complete binary tree or not.**

```java
import java.util.LinkedList;
import java.util.Queue;
```

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

public boolean isCompleteTree(TreeNode root) {
    if (root == null) {
        return true;
    }

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

    while (!queue.isEmpty()) {
        TreeNode node = queue.poll();

        if (node == null) {
            while (!queue.isEmpty() && queue.peek() == null) {
                queue.poll();
            }
            return queue.isEmpty();
        }

        queue.offer(node.left);
        queue.offer(node.right);
    }

    return true;
}
```

**Q3. Given the root of a binary tree, return the reverse level order traversal of its nodes' values.**

```java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.Stack;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

public List<List<Integer>> reverseLevelOrder(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> levelValues = new ArrayList<>();

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

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

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

        result.add(0, levelValues); // Insert at the beginning to reverse the order
    }

    return result;
}
```

**Q4. Given the root of a binary tree, return the left view of its nodes' values.**

```java
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

public List<Integer> leftView(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();
        for (int i = 0; i < levelSize; i++) {
            TreeNode node = queue.poll();

            if (i == 0) { // First node in the current level (leftmost)
                result.add(node.val);
            }

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

    return result;
}
```

**Q5. Given the root of a binary tree, convert the binary tree into its mirror and print its pre-order traversal.**

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

public void mirrorBinaryTree(TreeNode root) {
    if (root == null) {
```

```
        return;
    }

    // Swap left and right subtrees
    TreeNode temp = root.left;
    root.left = root.right;
    root.right = temp;

    // Recursively mirror the left and right subtrees
    mirrorBinaryTree(root.left);
    mirrorBinaryTree(root.right);
}

public void preOrderTraversal(TreeNode root) {
    if (root == null) {
        return;
    }

    System.out.print(root.val + " ");
    preOrderTraversal(root.left);
    preOrderTraversal(root.right);
}

// To use the functions:
// TreeNode root = ... // Initialize your binary tree
// mirrorBinaryTree(root);
// preOrderTraversal(root);
```