CSE ASSIGNMENT 4:

Reg no:22BCE7067

NAME:CH.SAITHARUN

1. Write a program to implement Representation of Binary tree with linked list.

PROGRAME:

```
class TreeNode {
   int data;
   TreeNode left;
   TreeNode right;

   public TreeNode(int data) {
      this.data = data;
      this.left = null;
      this.right = null;
   }
}

public class BinaryTree {
   TreeNode root;

   public BinaryTree(int data) {
      root = new TreeNode(data);
   }
```

```
    public BinaryTree() {
        root = null;
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree(1);
        tree.root.left = new TreeNode(2);
        tree.root.right = new TreeNode(3);
        tree.root.left.left = new TreeNode(4);
        tree.root.left.right = new TreeNode(5);

        System.out.println("Binary Tree created successfully.");
    }
}
```

OUTPUT:

```
Binary Tree created successfully.
```

2. Write a java program to find the largest value in each level of Binary Tree.

PROGRAME:

```
import java.util.*;

class TreeNode {
    int data;
```

```
    TreeNode left;
    TreeNode right;

    public TreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTree {
    TreeNode root;

    public static List<Integer> findLargestValuesInEachLevel(TreeNode root) {
        List<Integer> largestValues = new ArrayList<>();
        if (root == null)
            return largestValues;

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

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
```

```
        int maxVal = Integer.MIN_VALUE;

        for (int i = 0; i < levelSize; i++) {
            TreeNode node = queue.poll();
            maxVal = Math.max(maxVal, node.data);

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

        largestValues.add(maxVal);
    }

    return largestValues;
}

public static void main(String[] args) {
    BinaryTree tree = new BinaryTree(1);
    tree.root.left = new TreeNode(3);
    tree.root.right = new TreeNode(2);
    tree.root.left.left = new TreeNode(5);
```

```
        tree.root.left.right = new TreeNode(9);
        tree.root.right.right = new TreeNode(6);

        List<Integer> largestValues = findLargestValuesInEachLevel(tree.root);

        System.out.println("Largest values in each level: " + largestValues);
    }
}
```

OUTPUT:

```
Largest values in each level: [1, 3, 9, 6]
```

3. Write a java Program to Determine if given Two Trees are Identical or not.

PROGRAME:

```
class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;

    public TreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

```
public class BinaryTree {
    TreeNode root;

    public static boolean areIdentical(TreeNode root1, TreeNode root2) {
        if (root1 == null && root2 == null)
            return true;

        if (root1 != null && root2 != null) {
            return (root1.data == root2.data
                    && areIdentical(root1.left, root2.left)
                    && areIdentical(root1.right, root2.right));
        }

        return false;
    }

    public static void main(String[] args) {
        BinaryTree tree1 = new BinaryTree(1);
        tree1.root.left = new TreeNode(2);
        tree1.root.right = new TreeNode(3);

        BinaryTree tree2 = new BinaryTree(1);
```

```
        tree2.root.left = new TreeNode(2);
        tree2.root.right = new TreeNode(3);

        boolean identical = areIdentical(tree1.root, tree2.root);

        System.out.println("Are the two trees identical? " + identical);
    }
}
```

OUTPUT:

```
Are the two trees identical? true
```

4. Write a program to implement Binary tree traversals- In-order, Pre-order, Post-order using recursion.

PROGRAME:

```
class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;

    public TreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
```

```
    }
}

public class BinaryTree {
    TreeNode root;

    public void inorderTraversal(TreeNode node) {
        if (node == null)
            return;
        inorderTraversal(node.left);
        System.out.print(node.data + " ");
        inorderTraversal(node.right);
    }

    public void preorderTraversal(TreeNode node) {
        if (node == null)
            return;
        System.out.print(node.data + " ");
        preorderTraversal(node.left);
        preorderTraversal(node.right);
    }

    public void postorderTraversal(TreeNode node) {
```

```
    if (node == null)
        return;
    postorderTraversal(node.left);
    postorderTraversal(node.right);
    System.out.print(node.data + " ");
}

public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
    tree.root = new TreeNode(1);
    tree.root.left = new TreeNode(2);
    tree.root.right = new TreeNode(3);
    tree.root.left.left = new TreeNode(4);
    tree.root.left.right = new TreeNode(5);

    System.out.print("Inorder traversal: ");
    tree.inorderTraversal(tree.root);
    System.out.println();

    System.out.print("Preorder traversal: ");
    tree.preorderTraversal(tree.root);
    System.out.println();
```

```
        System.out.print("Postorder traversal: ");
        tree.postorderTraversal(tree.root);
        System.out.println();
    }
}
```

OUTPUT:

```
Inorder traversal: 4 2 5 1 3
Preorder traversal: 1 2 4 5 3
Postorder traversal: 4 5 2 3 1
```

5. Write a program to implement an AVL Tree. Insert and delete a node from an AVL Tree.

PROGRAME:

```
class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;
    int height;

    public TreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
        this.height = 1;
    }
```

```
}

public class AVLTree {
    TreeNode root;

    public int height(TreeNode node) {
        if (node == null)
            return 0;
        return node.height;
    }

    public int getBalanceFactor(TreeNode node) {
        if (node == null)
            return 0;
        return height(node.left) - height(node.right);
    }

    public TreeNode rightRotate(TreeNode y) {
        TreeNode x = y.left;
        TreeNode T2 = x.right;

        x.right = y;
        y.left = T2;
```

```
    y.height = Math.max(height(y.left), height(y.right)) + 1;
    x.height = Math.max(height(x.left), height(x.right)) + 1;

    return x;
}

public TreeNode leftRotate(TreeNode x) {
    TreeNode y = x.right;
    TreeNode T2 = y.left;

    y.left = x;
    x.right = T2;

    x.height = Math.max(height(x.left), height(x.right)) + 1;
    y.height = Math.max(height(y.left), height(y.right)) + 1;

    return y;
}

public TreeNode insert(TreeNode root, int data) {
    if (root == null)
        return new TreeNode(data);
```

```
if (data < root.data)
    root.left = insert(root.left, data);
else if (data > root.data)
    root.right = insert(root.right, data);
else
    return root; // Duplicate data not allowed

root.height = 1 + Math.max(height(root.left), height(root.right));

int balance = getBalanceFactor(root);

// Left-Left Case
if (balance > 1 && data < root.left.data)
    return rightRotate(root);

// Right-Right Case
if (balance < -1 && data > root.right.data)
    return leftRotate(root);

// Left-Right Case
if (balance > 1 && data > root.left.data) {
    root.left = leftRotate(root.left);
```

```
        return rightRotate(root);
    }

    // Right-Left Case
    if (balance < -1 && data < root.right.data) {
        root.right = rightRotate(root.right);
        return leftRotate(root);
    }

    return root;
}

public void insert(int data) {
    root = insert(root, data);
}

public static void main(String[] args) {
    AVLTree tree = new AVLTree();
    tree.insert(3);
    tree.insert(2);
    tree.insert(1);
    tree.insert(4);
    tree.insert(5);
```

```
        tree.insert(6);
        tree.insert(7);
        tree.insert(8);
        tree.insert(9);

        System.out.println("AVL Tree created successfully.");
    }
}
```

OUTPUT:

```
AVL Tree created successfully.
```

6. Create AVL Tree (Balanced BST) for the following sequence 3, 2, 1, 4, 5, 6, 7, 8, 9

PROGRAME.

```
AVLTree tree = new AVLTree();
tree.insert(3);
tree.insert(2);
tree.insert(1);
tree.insert(4);
tree.insert(5);
tree.insert(6);
tree.insert(7);
tree.insert(8);
```

tree.insert(9);

System.out.println("AVL Tree created successfully.");

OUTPUT:

```
AVL Tree created successfully.
```