Binary Trees

1. Binary Tree is Univalued or Not

```
public class BTisUnivaluedorNot {
    static class Node{
        int data;
        Node left;
        Node right;
    }
    static Node newNode(int data){
        Node temp = new Node();
        temp.data = data;
        temp.left = temp.right = null;
        return (temp);
    }
    static boolean isUnivalTree(Node root){
        if (root == null) {
            return true;
        }
        if (root.left != null && root.data != root.left.data) {
            return false;
        }
        if (root.right != null && root.data != root.right.data) {
            return false;
        }
        return isUnivalTree(root.left) && isUnivalTree(root.right);
    }
    public static void main(String[] args) {
         /*     1
              /    \
             1     1
            / \   / \
           1  1  1  1
          */
        Node root = newNode(1);
        root.left = newNode(1);
        root.right = newNode(1);
        root.left.left = newNode(1);
        root.left.right = newNode(1);
        root.right.left = newNode(1);
        root.right.right = newNode(1);
        if (isUnivalTree(root)) {
            System.out.println("YES, Given Tree is Univalued.");
        } else {
            System.out.println("NO, Given Tree is not Univalued.");
        }
    }
}
```

2. Build Tree Preorder

```java
public class BuildTreePreorder {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static class BinaryTree{
        static int idx = -1;
        public static Node buildTree(int nodes[]) {
            idx++;
            if (nodes[idx] == -1) {
                return null;
            }
            Node newNode = new Node(nodes[idx]);
            newNode.left = buildTree(nodes);
            newNode.right = buildTree(nodes);
            return newNode;
        }
    }
    public static void main(String[] args) {
        int nodes[] = {1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1};   // -1 indicates null
        BinaryTree tree = new BinaryTree();
        Node root = tree.buildTree(nodes);
        System.out.println("Root of Binary Tree: " + root.data);
    }
}
```

3. Count no. of Nodes

```java
public class CountOfNodes {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static int count(Node root){
        if (root == null) {
            return 0;
        }
        int leftCount = count(root.left);
        int rightCount = count(root.right);
        return leftCount + rightCount + 1;
    }
    public static void main(String[] args) {
        /*     1
```

```
            /    \
           2      3
          / \    / \
         4 5  6  7
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.println("Number of Total Nodes: " + count(root));
    }
}
```

4. Delete Leaf with Values X

```
import java.util.*;

public class DeleteLeafWithValuesX {
    static class Node {
        int data;
        Node left, right;
    }

    // Method to create a new node
    static Node newNode(int data) {
        Node newNode = new Node();
        newNode.data = data;
        newNode.left = null;
        newNode.right = null;
        return newNode;
    }

    // In-order traversal of the tree
    static void inOrder(Node node) {
        if (node == null) {
            return;
        }
        inOrder(node.left);
        System.out.print(node.data + " ");
        inOrder(node.right);
    }

    // Method to delete leaves with value x
    static Node deleteLeaves(Node root, int x) {
        if (root == null) {
            return null;
        }

        // Recursively call for left and right subtree
        root.left = deleteLeaves(root.left, x);
        root.right = deleteLeaves(root.right, x);

        // If the current node is a leaf and its value is equal to x
```

```
        if (root.left == null && root.right == null && root.data == x) {
            return null; // delete the leaf
        }
        return root;
    }

    public static void main(String[] args) {
        Node root = newNode(10);
        root.left = newNode(3);
        root.right = newNode(10);
        root.left.left = newNode(3);
        root.left.right = newNode(1);
        root.right.right = newNode(3);
        root.right.right.left = newNode(3);
        root.right.right.right = newNode(3);

        System.out.println("In-order traversal before deleting leaves:");
        inOrder(root);

        int x = 3;
        root = deleteLeaves(root, x);

        System.out.println("\nIn-order traversal after deleting leaves with value " + x + ":");
        inOrder(root);
    }
}
```

5. Diameter of Tree 1st Approach

```
public class DiameterOfTreeA1 {
    public static class Node{
        int data;           // Approach 1: O(n^2) - We use two methods i.e. height and diameter.
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static int height(Node root){
        if (root == null) {
            return 0;
        }
        int lh = height(root.left);
        int rh = height(root.right);
        return Math.max(lh, rh) + 1;
    }
    public static int diameter(Node root){
        if (root == null) {
            return 0;
        }
        int leftDiam = diameter(root.left);
        int leftHt = height(root.left);
        int rightDiam = diameter(root.right);
        int rightHt = height(root.right);
```

```
        int selfDiam = leftHt + rightHt + 1;
        return Math.max(selfDiam, Math.max(leftDiam, rightDiam));
    }
    public static void main(String[] args) {
        /*     1
              /    \
             2     3
            / \   / \
           4 5  6  7
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.println("Diameter of a Binary Tree: " + diameter(root));
    }
}
```

6. Diameter of Tree 2nd Approach

```
public class DiameterOfTreeA2 {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static class Info{
        int diam;               // Approach 2: O(n) - We will only one method for height and
diameter.
        int ht;
        public Info(int diam, int ht){
            this.diam = diam;
            this.ht = ht;
        }
    }
    public static Info diameter(Node root){
        if (root == null) {
            return new Info(0, 0);
        }
        Info leftInfo = diameter(root.left);
        Info rightInfo = diameter(root.right);
        int diam = Math.max(Math.max(leftInfo.diam, rightInfo.diam), leftInfo.ht + rightInfo.ht +
1);
        int ht = Math.max(leftInfo.ht, rightInfo.ht) + 1;
        return new Info(diam, ht);
    }
    public static void main(String[] args) {
        /*     1
```

```
            /    \
           2      3
          / \    / \
          4 5  6  7
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        // Get the diameter information
        Info result = diameter(root);
        System.out.println("Diameter of a Binary Tree: " + result.diam);
    }
}
```

7. Find Duplicates Subtree

```
import java.util.HashMap;

public class FindDuplicatesSubtree {
    static HashMap<String, Integer> m;

    static class Node {
        int data;
        Node left;
        Node right;

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

    static String inorder(Node node) {
        if (node == null) {
            return "";
        }

        String str = "(";
        str += inorder(node.left);
        str += Integer.toString(node.data);
        str += inorder(node.right);
        str += ")";

        if (m.get(str) != null && m.get(str) == 1) {
            System.out.print(node.data + " ");
        }

        if (m.containsKey(str)) {
            m.put(str, m.get(str) + 1);
        } else {
            m.put(str, 1);
```

```
        }

        return str;
    }

    static void printAllDuplicates(Node root) {
        m = new HashMap<>();
        inorder(root);
    }

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

        System.out.print("Duplicates Nodes in BT are: ");
        printAllDuplicates(root);
    }
}
```

8. Height of Binary Tree

```
public class HeightOfTree {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static int height(Node root){
        if (root == null) {
            return 0;
        }
        int lh = height(root.left);
        int rh = height(root.right);
        return Math.max(lh, rh) + 1;
    }
    public static void main(String[] args) {
        /*     1
              /    \
             2      3
            / \    / \
           4 5  6  7
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
```

```
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.println("Height of Tree: " + height(root));
    }
}
```

9. Invert Binary Tree

```
class Node{
    int data;
    Node left, right;
    public Node(int item){
        data = item;
        left = right = null;
    }
}
public class InvertBT {
    Node root;
    void mirror(){
        root = mirror(root);
    }
    Node mirror(Node node){
        if (node == null) {
            return node;
        }
        Node left = mirror(node.left);
        Node right = mirror(node.right);
        node.left = right;
        node.right = left;
        return node;
    }
    void inOrder(){
        inOrder(root);
    }
    void inOrder(Node node){
        if (node == null) {
            return;
        }
        inOrder(node.left);
        System.out.print(node.data + " ");
        inOrder(node.right);
    }
    public static void main(String[] args) {
        InvertBT tree = new InvertBT();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        System.out.println("Inorder Traversal: ");
        tree.inOrder();
        System.out.println();
        tree.mirror();
        System.out.println("Inverted Inorder Traversal: ");
```

```
        tree.inOrder();
    }
}
```

10. Kth Ancestor of Node

```
public class KthAncestorOfNode {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static int KAncestor(Node root, int k, int n){
        if (root == null) {
            return -1;
        }
        if (root.data == n) {
            return 0;
        }
        int leftDist = KAncestor(root.left, k, n);
        int rightDist = KAncestor(root.right, k, n);
        if (leftDist == -1 && rightDist == -1) {
            return -1;
        }
        int max = Math.max(leftDist, rightDist);
        if (max+1 == k) {
            System.out.println(root.data);
        }
        return max+1;
    }
    public static void main(String[] args) {
        /*
         *              1
         *            /   \
         *           2     3
         *          / \   / \
         *         4   5 6   7
         */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        int n=5;
        int k=2;
        KAncestor(root, k, n);
    }
}
```

11. Kth Level of Tree

```java
public class KthLevelOfTree {
    public static class Node{                //Time Complexity - O(n)
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static void KLevel(Node root, int level, int k){
        if (root == null) {
            return;
        }
        if (level == k) {
            System.out.print(root.data + " ");
            return;
        }
        KLevel(root.left, level+1, k);
        KLevel(root.right, level+1, k);
    }
    public static void main(String[] args) {
        /*
         *              1
         *            /   \
         *           2     3
         *          / \   / \
         *         4   5 6   7
         */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        int k = 3;
        System.out.print("Kth Level Elements of Tree are: ");
        KLevel(root, 1, k);
    }
}
```

12. Lowest Common Ancestor 1st Approach

```java
import java.util.*;
public class LowestCommonAncestorA1 {
    public static class Node{                    // Time Complexity - O(n)
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
```

```java
        }
    }
    public static boolean getPath(Node root, int n, ArrayList<Node>path){
        if (root == null) {
            return false;
        }
        path.add(root);
        if (root.data == n) {
            return true;
        }
        boolean foundLeft = getPath(root.left, n, path);
        boolean foundRight = getPath(root.right, n, path);
        if (foundLeft || foundRight) {
            return true;
        }
        path.remove(path.size()-1);
        return false;
    }
    public static Node lca(Node root, int n1, int n2){
        ArrayList <Node> path1 = new ArrayList<>();
        ArrayList <Node> path2 = new ArrayList<>();
        getPath(root, n1, path1);
        getPath(root, n2, path2);
        // Last Common Ancestor
        int i=0;
        for(; i<path1.size() && i<path2.size(); i++) {
            if (path1.get(i) != path2.get(i)) {
                break;
            }
        }
        // Last Equal Node -> i-1th
        Node lca = path1.get(i-1);
        return lca;
    }
    public static void main(String[] args) {
        /*
         *              1
         *            /   \
         *           2     3
         *          / \   / \
         *         4   5 6   7
         */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        int n1 = 6;
        int n2 = 5;
        System.out.println("Last Common Ancestor of Given Nodes is: " + lca(root, n1, n2).data);
    }
}
```

13. Lowest Common Ancestor 2nd Approach

```
public class LowestCommonAncestorA2 {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static Node lca(Node root, int n1, int n2){
        if (root == null || root.data == n1 || root.data == n2) {
            return root;
        }
        Node leftLca = lca(root.left, n1, n2);
        Node rigthLca = lca(root.right, n1, n2);
        if (rigthLca == null) {
            return leftLca;
        }
        if (leftLca == null) {
            return rigthLca;
        }
        return root;
    }
    public static void main(String[] args) {
        /*
         *              1
         *            /   \
         *           2     3
         *          / \   / \
         *         4   5 6   7
         */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        int n1 = 6;
        int n2 = 5;
        System.out.println("Last Common Ancestor of Given Nodes is: " + lca(root, n1, n2).data);
    }
}
```

14. Maximum Path Sum

```
class Node{
    int data;
    Node left, right;
    public Node(int item){
        data = item;
        left = right = null;
    }
```

```
}
class Res{
    public int val;
}
public class MaxPathSum {
    Node root;

    int findMaxUtil(Node node, Res res){
        if (node == null) {
            return 0;
        }
        int l = findMaxUtil(node.left, res);
        int r = findMaxUtil(node.right, res);
        int max_single = Math.max(Math.max(l, r) + node.data, node.data);
        int max_top = Math.max(max_single, l + r + node.data);
        res.val = Math.max(res.val, max_top);
        return max_single;
    }

    int findmaxSum(){
        Res res = new Res();
        res.val = Integer.MIN_VALUE;
        findMaxUtil(root, res);
        return res.val;
    }

    public static void main(String[] args) {
        MaxPathSum tree = new MaxPathSum();
        tree.root = new Node(10);
        tree.root.left = new Node(2);
        tree.root.right = new Node(10);
        tree.root.left.left = new Node(20);
        tree.root.left.right = new Node(1);
        tree.root.right.right = new Node(25);
        tree.root.right.right.left = new Node(3);
        tree.root.right.right.right = new Node(4);

        System.out.println("Maximum Path Sum: " + tree.findmaxSum());
    }
}
```

15. Minimum Distance Between N Nodes

```
public class MinDistanceBetweenNodes {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static Node lca(Node root, int n1, int n2){
```

```
        if (root == null || root.data == n1 || root.data == n2) {
            return root;
        }
        Node leftLca = lca(root.left, n1, n2);
        Node rigthLca = lca(root.right, n1, n2);
        if (rigthLca == null) {
            return leftLca;
        }
        if (leftLca == null) {
            return rigthLca;
        }
        return root;
    }
    public static int lcaDist(Node root, int n){
        if (root == null) {
            return -1;
        }
        if (root.data == n) {
            return 0;
        }
        int leftDistance = lcaDist(root.left, n);
        int rightDistance = lcaDist(root.right, n);
        if (leftDistance == -1 && rightDistance == -1) {
            return -1;
        } else if(leftDistance == -1){
            return rightDistance + 1;
        } else {
            return leftDistance + 1;
        }
    }
    public static int minDist(Node root, int n1, int n2){
        Node Lca = lca(root, n1, n2);
        int dist1 = lcaDist(Lca, n1);
        int dist2 = lcaDist(Lca, n2);
        return dist1 + dist2;
    }
    public static void main(String[] args) {
        /*
         *              1
         *            /   \
         *           2     3
         *         / \   / \
         *        4   5 6   7
         */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        int n1 = 4;
        int n2 = 6;
        System.out.println("Min Distance Between "+n1+" & "+n2+" is: " + minDist(root, n1, n2));
    }
}
```

16. Subtree of Another Tree

```java
public class SubtreeOfAnotherTree {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static boolean isIdentical(Node node, Node subRoot){
        if (node == null && subRoot == null) {
            return true;
        }
        else if(node == null || subRoot == null || node.data != subRoot.data){
            return false;
        }
        if (!isIdentical(node.left, subRoot.left)) {
            return false;
        }
        if (!isIdentical(node.right, subRoot.right)) {
            return false;
        }
        return true;
    }
    public static boolean isSubtree(Node root, Node subRoot){
        if (root == null) {
            return true;
        }
        if (root.data == subRoot.data) {
            if (isIdentical(root, subRoot)) {
                return true;
            }
        }
        return isSubtree(root.left, subRoot) || isSubtree(root.right, subRoot);
    }
    public static void main(String[] args) {
        /*
         *              1
         *            /   \
         *           2     3
         *          / \   / \
         *         4   5 6   7
         */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        /*
         *              2
         *            /   \
```

```
         *           4    5
         */
        Node subRoot = new Node(2);
        subRoot.left = new Node(4);
        subRoot.right = new Node(5);
        System.out.println(isSubtree(root, subRoot));
    }
}
```

17. Sum of Nodes

```
public class SumOfNodes {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static int sum(Node root){
        if (root == null) {
            return 0;
        }
        int leftSum = sum(root.left);
        int rightSum = sum(root.right);
        return leftSum + rightSum + root.data;
    }
    public static void main(String[] args) {
        /*     1
              /    \
             2     3
            / \   / \
           4 5  6  7
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.println("Sum of Nodes of a Binary Tree: " + sum(root));
    }
}
```

18. Top View of Tree

```
import java.util.*;
public class TopViewOfTree {
    public static class Node{
        int data;
        Node left;
        Node right;
```

```
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    static class Info{
        Node node;
        int hd;
        public Info(Node node, int hd){
            this.node = node;
            this.hd = hd;
        }
    }
    public static void topView(Node root){
        // Level order traversal
        Queue<Info> q = new LinkedList<>();
        HashMap<Integer, Node> map = new HashMap<>();
        int min=0, max=0;
        q.add(new Info(root, 0));
        q.add(null);
        while (!q.isEmpty()) {
            Info curr = q.remove();
            if (curr == null) {
                if (q.isEmpty()) {
                    break;
                } else {
                    q.add(null);
                }
            } else {
                if (!map.containsKey(curr.hd)) {
                    map.put(curr.hd, curr.node);
                }
                if (curr.node.left != null) {
                    q.add(new Info(curr.node.left, curr.hd-1));
                    min = Math.min(min, curr.hd-1);
                }
                if (curr.node.right != null) {
                    q.add(new Info(curr.node.right, curr.hd+1));
                    max = Math.max(max, curr.hd+1);
                }
            }
        }
        for(int i=min; i <= max; i++){
            System.out.print(map.get(i).data + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        /*
         *             1
         *           /   \
         *          2     3
         *         / \   / \
         *        4   5 6   7
         */
        Node root = new Node(1);
```

```
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.print("Top View of Given Tree is: ");
        topView(root);
    }
}
```

19. Transform Tree to Sum Tree

```
public class TransformToSumTree {
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    public static int transform(Node root){
        if (root == null) {
            return 0;
        }
        int leftChild = transform(root.left);
        int rightChild = transform(root.right);
        int data = root.data;
        int newLeft = root.left == null ? 0 : root.left.data;
        int newRight = root.right == null ? 0 : root.right.data;
        root.data = newLeft + leftChild + newRight + rightChild;
        return data;
    }
    public static void preorder(Node root){
        if (root == null) {
            return;
        }
        System.out.print(root.data + " ");
        preorder(root.left);
        preorder(root.right);
    }
    public static void main(String[] args) {
        /*
         *              1
         *            /   \
         *           2     3
         *          / \   / \
         *         4   5 6   7
         */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
```

```
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.print("Original Tree: ");
        preorder(root);
        transform(root);
        System.out.println();
        System.out.print("Transformation to Sum Tree: ");
        preorder(root);
    }
}
```

20. Binary Tree Traversals

```
public class TreeTraversals {
    // Nodes Declaration
    public static class Node{
        int data;
        Node left;
        Node right;
        Node(int data){
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    // PreOrder - O(n)
    public static void preorder(Node root){
        if (root == null) {
            return;
        }
        System.out.print(root.data + " ");
        preorder(root.left);
        preorder(root.right);
    }
    // InOrder - O(n)
    public static void inorder(Node root){
        if (root == null) {
            return;
        }
        inorder(root.left);
        System.out.print(root.data + " ");
        inorder(root.right);
    }
    // PostOrder - O(n)
    public static void postorder(Node root){
        if (root == null) {
            return;
        }
        postorder(root.left);
        postorder(root.right);
        System.out.print(root.data + " ");
    }
    public static void main(String[] args) {
        /*     1
            /    \
```

```
          2     3
         / \   / \
        4 5  6  7
        */
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.print("Pre-Order Traversal: ");
        preorder(root);
        System.out.print("InOrder Traversal: ");
        inorder(root);
        System.out.print("Post-Order Traversal: ");
        postorder(root);
    }
}
```