1:Maximum element

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

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

public class MaximumElementInBinaryTree {

    public static int findMax(TreeNode root) {
        if (root == null) {
            throw new IllegalArgumentException("The binary tree is empty.");
        }
        return findMaxUtil(root);
    }

    private static int findMaxUtil(TreeNode node) {
        if (node == null) {
            return Integer.MIN_VALUE;
        }

        int leftMax = findMaxUtil(node.left);
        int rightMax = findMaxUtil(node.right);

        return Math.max(node.val, Math.max(leftMax, rightMax));
```

```
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(5);
        root.right = new TreeNode(20);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(8);
        root.right.left = new TreeNode(15);
        root.right.right = new TreeNode(25);

        int maxElement = findMax(root);
        System.out.println("Maximum element in the binary tree: " + maxElement);
    }
}
```

2:Nodes at distance K

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

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

public class NodesAtDistanceKFromRoot {

    public static void printNodesAtDistanceK(TreeNode root, int k) {
```

```
    if (root == null || k < 0) {
        return;
    }

    printNodesAtDistanceKUtil(root, k, 0);
}

private static void printNodesAtDistanceKUtil(TreeNode node, int k, int distance) {
    if (node == null) {
        return;
    }

    if (distance == k) {
        System.out.print(node.val + " ");
        return;
    }

    printNodesAtDistanceKUtil(node.left, k, distance + 1);
    printNodesAtDistanceKUtil(node.right, k, distance + 1);
}

public static void main(String[] args) {
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);
    root.left.left = new TreeNode(4);
    root.left.right = new TreeNode(5);
    root.right.left = new TreeNode(6);
    root.right.right = new TreeNode(7);

    int k = 2;
```

```
        System.out.print("Nodes at distance " + k + " from the root: ");
        printNodesAtDistanceK(root, k);
    }
}
```

3: PostOrder

```
import java.util.Stack;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

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

public class PostOrderToTree {

    public static TreeNode buildTree(int[] postorder) {
        if (postorder == null || postorder.length == 0) {
            return null;
        }

        Stack<TreeNode> stack = new Stack<>();
        TreeNode root = new TreeNode(postorder[postorder.length - 1]);

        stack.push(root);
        for (int i = postorder.length - 2; i >= 0; i--) {
```

```
        TreeNode current = new TreeNode(postorder[i]);
        TreeNode lastPopped = null;

        while (!stack.isEmpty() && stack.peek().val < current.val) {
            lastPopped = stack.pop();
        }

        if (lastPopped != null) {
            lastPopped.right = current;
        } else {
            stack.peek().left = current;
        }

        stack.push(current);
    }

    return root;
}

public static void inOrderTraversal(TreeNode node) {
    if (node != null) {
        inOrderTraversal(node.left);
        System.out.print(node.val + " ");
        inOrderTraversal(node.right);
    }
}

public static void main(String[] args) {
    int[] postorder = {9, 15, 7, 20, 3};
    TreeNode root = buildTree(postorder);
```

```
        System.out.print("Inorder Traversal: ");
        inOrderTraversal(root);

    }
}
```