```java
// Q1. Binary Tree Traversals
import java.util.*;
class Q1Traversals {
    static class Node{int data;Node left,right;Node(int d){data=d;}}
    static Node insert(Node root,int x){
        if(root==null) return new Node(x);
        if(x<root.data) root.left=insert(root.left,x);
        else root.right=insert(root.right,x);
        return root;
    }
    static void preorder(Node root){
        if(root==null) return;
        System.out.print(root.data+" ");
        preorder(root.left);
        preorder(root.right);
    }
    static void inorder(Node root){
        if(root==null) return;
        inorder(root.left);
        System.out.print(root.data+" ");
        inorder(root.right);
    }
    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){
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        Node root=null;
        for(int i=0;i<n;i++){
            int x=sc.nextInt();
            root=insert(root,x);
        }
        preorder(root);System.out.println();
        inorder(root);System.out.println();
        postorder(root);
        sc.close();
    }
}
```

```java
// Q2. BST Functions
import java.util.*;
class Q2BSTFunctions {
    static class Node{int data;Node left,right;Node(int d){data=d;}}
    static Node insert(Node root,int x){
        if(root==null) return new Node(x);
        if(x<root.data) root.left=insert(root.left,x);
        else if(x>root.data) root.right=insert(root.right,x);
        return root;
    }
    static Node searchRec(Node root,int key){
        if(root==null || root.data==key) return root;
        if(key<root.data) return searchRec(root.left,key);
        else return searchRec(root.right,key);
    }
    static Node searchIter(Node root,int key){
        while(root!=null && root.data!=key){
            if(key<root.data) root=root.left;
            else root=root.right;
        }
        return root;
    }
    static Node minNode(Node root){
        if(root==null) return null;
        while(root.left!=null) root=root.left;
        return root;
    }
    static Node maxNode(Node root){
        if(root==null) return null;
        while(root.right!=null) root=root.right;
        return root;
    }
    static Node inorderSucc(Node root,int key){
        Node cur=root,succ=null;
        while(cur!=null){
            if(key<cur.data){succ=cur;cur=cur.left;}
            else if(key>cur.data) cur=cur.right;
            else{
                if(cur.right!=null){
                    Node t=cur.right;
                    while(t.left!=null)t=t.left;
                    succ=t;
                }
                break;
            }
        }
        return succ;
    }
    static Node inorderPred(Node root,int key){
        Node cur=root,pred=null;
        while(cur!=null){
            if(key>cur.data){pred=cur;cur=cur.right;}
            else if(key<cur.data) cur=cur.left;
            else{
```

```
                if(cur.left!=null){
                    Node t=cur.left;
                    while(t.right!=null)t=t.right;
                    pred=t;
                }
                break;
            }
        }
        return pred;
    }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        Node root=null;
        for(int i=0;i<n;i++){
            int x=sc.nextInt();
            root=insert(root,x);
        }
        int key=sc.nextInt();
        Node r1=searchRec(root,key);
        Node r2=searchIter(root,key);
        System.out.println(r1!=null?1:0);
        System.out.println(r2!=null?1:0);
        Node mn=minNode(root);
        Node mx=maxNode(root);
        System.out.println(mn!=null?mn.data:-1);
        System.out.println(mx!=null?mx.data:-1);
        Node s=inorderSucc(root,key);
        Node p=inorderPred(root,key);
        System.out.println(s!=null?s.data:-1);
        System.out.println(p!=null?p.data:-1);
        sc.close();
    }
}
```

```java
// Q3. BST: Insert, Delete, Max Depth, Min Depth
import java.util.*;
class Q3BSTOps {
    static class Node{int data;Node left,right;Node(int d){data=d;}}
    static Node insert(Node root,int x){
        if(root==null) return new Node(x);
        if(x<root.data) root.left=insert(root.left,x);
        else if(x>root.data) root.right=insert(root.right,x);
        return root;
    }
    static Node minNode(Node root){
        if(root==null) return null;
        while(root.left!=null) root=root.left;
        return root;
    }
    static Node deleteBST(Node root,int key){
        if(root==null) return null;
        if(key<root.data) root.left=deleteBST(root.left,key);
        else if(key>root.data) root.right=deleteBST(root.right,key);
        else{
            if(root.left==null && root.right==null) return null;
            else if(root.left==null) return root.right;
            else if(root.right==null) return root.left;
            else{
                Node t=minNode(root.right);
                root.data=t.data;
                root.right=deleteBST(root.right,t.data);
            }
        }
        return root;
    }
    static int maxDepth(Node root){
        if(root==null) return 0;
        return 1+Math.max(maxDepth(root.left),maxDepth(root.right));
    }
    static int minDepth(Node root){
        if(root==null) return 0;
        if(root.left==null && root.right==null) return 1;
        if(root.left==null) return 1+minDepth(root.right);
        if(root.right==null) return 1+minDepth(root.left);
        return 1+Math.min(minDepth(root.left),minDepth(root.right));
    }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        Node root=null;
        for(int i=0;i<n;i++){
            int x=sc.nextInt();
            root=insert(root,x);
        }
        int del=sc.nextInt();
        root=deleteBST(root,del);
        System.out.println(maxDepth(root));
```

```
        System.out.println(minDepth(root));
        sc.close();
    }
}



// Q4. Check if a Binary Tree is a BST or not
import java.util.*;
class Q4CheckBST {
    static class Node{int data;Node left,right;Node(int d){data=d;}}
    static Node insert(Node root,int x){
        if(root==null) return new Node(x);
        if(x<root.data) root.left=insert(root.left,x);
        else if(x>root.data) root.right=insert(root.right,x);
        return root;
    }
    static boolean isBST(Node root,long mn,long mx){
        if(root==null) return true;
        if(root.data<=mn || root.data>=mx) return false;
        return isBST(root.left,mn,root.data) &&
isBST(root.right,root.data,mx);
    }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        Node root=null;
        for(int i=0;i<n;i++){
            int x=sc.nextInt();
            root=insert(root,x);
        }

System.out.println(isBST(root,Long.MIN_VALUE,Long.MAX_VALUE)?"YES":"NO");
        sc.close();
    }
}
```