Code : 1  
package Topic\_17\_BinarySearchTree;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.Stack;

public class B\_AddNode {

public static class Node {

int data;

Node left;

Node right;

Node(int data, Node left, Node right) {

this.data = data;

this.left = left;

this.right = right;

}

}

public static class Pair {

Node node;

int state;

Pair(Node node, int state) {

this.node = node;

this.state = state;

}

}

public static Node construct(Integer[] arr) {

Node root = new Node(arr[0], null, null);

Pair rtp = new Pair(root, 1);

Stack<Pair> st = new Stack<>();

st.push(rtp);

int idx = 0;

while (st.size() > 0) {

Pair top = st.peek();

if (top.state == 1) {

idx++;

if (arr[idx] != null) {

top.node.left = new Node(arr[idx], null, null);

Pair lp = new Pair(top.node.left, 1);

st.push(lp);

} else {

top.node.left = null;

}

top.state++;

} else if (top.state == 2) {

idx++;

if (arr[idx] != null) {

top.node.right = new Node(arr[idx], null, null);

Pair rp = new Pair(top.node.right, 1);

st.push(rp);

} else {

top.node.right = null;

}

top.state++;

} else {

st.pop();

}

}

return root;

}

public static void display(Node node) {

if (node == null) {

return;

}

String str = "";

str += node.left == null ? "." : node.left.data + "";

str += " <- " + node.data + " -> ";

str += node.right == null ? "." : node.right.data + "";

System.out.println(str);

display(node.left);

display(node.right);

}

public static int size(Node node) {

if (node == null) {

return 0;

}

int ls = size(node.left);

int rs = size(node.right);

int ts = ls + rs + 1;

return ts;

}

public static int sum(Node node) {

if (node == null) {

return 0;

}

int ls = sum(node.left);

int rs = sum(node.right);

int ts = ls + rs + node.data;

return ts;

}

public static int max(Node node) {

if (node.right == null) {

return node.data;

} else {

return max(node.right);

}

}

public static int min(Node node) {

if (node.left == null) {

return node.data;

} else {

return min(node.left);

}

}

public static boolean find(Node node, int data) {

if (node == null) {

return false;

}

if (data > node.data) {

return find(node.right, data);

} else if (data < node.data) {

return find(node.left, data);

} else {

return true;

}

}

public static Node add(Node node, int data) {

if (node == null) {

return new Node(data, null, null);

}

if (data > node.data) {

node.right = add(node.right, data);

return node;

} else if (data < node.data) {

node.left = add(node.left, data);

return node;

} else {

return node;

}

}

public static Node remove(Node node, int data) {

if (node == null) {

return null;

}

if (data > node.data) {

node.right = remove(node.right, data);

return node;

} else if (data < node.data) {

node.left = remove(node.left, data);

return node;

} else {

if (node.left == null && node.right == null) {

return null;

} else if (node.left == null) {

return node.right;

} else if (node.right == null) {

return node.left;

} else {

int max = max(node.left);

node.data = max;

node.left = remove(node.left, max);

return node;

}

}

}

static int sum = 0;

public static void replaceWithSumOfLarger(Node node) {

if (node == null) {

return;

}

replaceWithSumOfLarger(node.right); // visit the right node

int od = node.data; // inorder process

node.data = sum;

sum += od;

replaceWithSumOfLarger(node.left); //visit left node

}

public static int lca(Node node, int d1, int d2) {

if (node == null) { //1

return 0;

}

if (d1 < node.data && d2 < node.data) { //2

return lca(node.left, d1, d2);

} else if (d1 > node.data && d2 > node.data) { //3

return lca(node.right, d1, d2);

} else { //4

return node.data;

}

}

public static void targetSumPair(Node root, Node node, int tar) {

if (node == null) {

return;

}

targetSumPair(root, node.left, tar);

int comp = tar - node.data;

if (comp > node.data) {

if (find(root, comp)) {

System.out.println(node.data + " " + comp);

}

}

targetSumPair(root, node.right, tar);

}

public static void printInRange(Node node, int d1, int d2) {

if (node == null) {

return;

}

if (node.data > d1 && node.data > d2) {

printInRange(node.left, d1, d2);

} else if (node.data < d1 && node.data < d2) {

printInRange(node.right, d1, d2);

} else {

printInRange(node.left, d1, d2);

System.out.println(node.data);

printInRange(node.right, d1, d2);

}

}

public static class ITPair {

Node node;

int state = 0;

ITPair() {

}

ITPair(Node node, int state) {

this.node = node;

this.state = state;

}

}

public static void bestApproach(Node node, int tar) {

Stack<ITPair> ls = new Stack<>();

Stack<ITPair> rs = new Stack<>();

ls.push(new ITPair(node, 0));

rs.push(new ITPair(node, 0));

Node left = getNextFromNormalInorder(ls);

Node right = getNextFromReverseInorder(rs);

while (left.data < right.data) {

if (left.data + right.data < tar) {

left = getNextFromNormalInorder(ls);

} else if (left.data + right.data > tar) {

right = getNextFromReverseInorder(rs);

} else {

System.out.println(left.data + " " + right.data);

left = getNextFromNormalInorder(ls);

right = getNextFromReverseInorder(rs);

}

}

}

public static Node getNextFromNormalInorder(Stack<ITPair> st) {

while (st.size() > 0) {

ITPair top = st.peek();

if (top.state == 0) {

if (top.node.left != null) {

st.push(new ITPair(top.node.left, 0));

}

top.state++;

} else if (top.state == 1) {

if (top.node.right != null) {

st.push(new ITPair(top.node.right, 0));

}

top.state++;

return top.node;

} else {

st.pop();

}

}

return null;

}

public static Node getNextFromReverseInorder(Stack<ITPair> st) {

while (st.size() > 0) {

ITPair top = st.peek();

if (top.state == 0) {

if (top.node.right != null) {

st.push(new ITPair(top.node.right, 0));

}

top.state++;

} else if (top.state == 1) {

if (top.node.left != null) {

st.push(new ITPair(top.node.left, 0));

}

top.state++;

return top.node;

} else {

st.pop();

}

}

return null;

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

Integer[] arr = new Integer[n];

String[] values = br.readLine().split(" ");

for (int i = 0; i < n; i++) {

if (values[i].equals("n") == false) {

arr[i] = Integer.parseInt(values[i]);

} else {

arr[i] = null;

}

}

int data = Integer.parseInt(br.readLine());

Node root = construct(arr);

//Size, Sum, Max, Min, Find In Bst

int size = size(root);

int sum = sum(root);

int max = max(root);

int min = min(root);

boolean found = find(root, data);

System.out.println(size);

System.out.println(sum);

System.out.println(max);

System.out.println(min);

System.out.println(found);

// Add node to BST

root = add(root, data);

display(root);

//Remove node from bst

root = remove(root, data);

display(root);

//Replace With Sum Of Larger

replaceWithSumOfLarger(root);

//Lca

int lca = lca(root, 12, 30);

System.out.println(lca);

//Print In Range

printInRange(root, 12, 65);

//target sum pair in bst

bestApproach(root, data);

}

}