BST Solutions

Solution 1:

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.*;

class Solution{

        static class Node{
        int val;
        Node left, right;
        };

        static Node newNode(int item){
        Node temp = new Node();
        temp.val = item;
        temp.left = temp.right = null;
        return temp;
        }


        static int sum = 0;

        static int rangeSumBST(Node root, int low,
                                                    int high){
                if (root == null)
                return 0;

                Queue<Node> q = new LinkedList<Node>();
                q.add(root);

                while (q.isEmpty() == false){

                        Node curr = q.peek();
                        q.remove();
                        if (curr.val >= low &&
                                curr.val <= high){
                                sum += curr.val;
                        }
                if (curr.left != null &&
                        curr.val > low) q.add(curr.left);
                if (curr.right != null &&
                        curr.val < high)
                        q.add(curr.right);
            }
            return sum;
    }

    static Node insert(Node node, int data){

            if (node == null)
                    return newNode(data);
            if (data <= node.val)
                    node.left = insert(node.left,
                                                        data);
            else
                    node.right = insert(node.right,
                                                data);
            return node;
    }

    public static void main(String[] args){

            Node root = null;
            root = insert(root, 10);
            insert(root, 5);
            insert(root, 15);
            insert(root, 3);
            insert(root, 7);
            insert(root, 18);

            int L = 7, R = 15;
            System.out.print(rangeSumBST(root, L, R));
    }
    }
```

Solution 2:

Time Complexity : o(h)
Space Complexity: o(1)

```
class Solution{
        static int min_diff, min_diff_key; static class Node{
        int key;
        Node left, right;
};

static Node newnode(int key){
        Node node = new Node();
        node.key = key;
        node.left = node.right = null;
        return (node);
}

         static void
 maxDiffUtil(Node ptr, int
     k){ if (ptr == null)
                return ;

        if (ptr.key == k){
                min_diff_key = k;
                return;
        }

        if (min_diff > Math.abs(ptr.key - k)){
                min_diff = Math.abs(ptr.key - k);
                min_diff_key = ptr.key;
        }

        if (k < ptr.key)
                maxDiffUtil(ptr.left, k);
        else
                maxDiffUtil(ptr.right, k);
}

static int maxDiff(Node root, int k){
        min_diff = 999999999;
        min_diff_key = -1;
        maxDiffUtil(root, k);
        return min_diff_key;
}
```

```
public static void main(String args[]){
        Node root = newnode(9); root.left = newnode(4);
        root.right = newnode(17);
        root.left.left = newnode(3);
        root.left.right = newnode(6);
        root.left.right.left = newnode(5);
        root.left.right.right = newnode(7);
        root.right.right = newnode(22);
        root.right.right.left = newnode(20);
        int k = 18;
        System.out.println( maxDiff(root, k));

}
}
```

Solution 3 :

Time Complexity : o(n)

Space Complexity: o(h)

```
import java.io.*;

class Node {
        int data;
        Node left, right;
        Node(int x)
        {
                data = x;
                left = right = null;
        }
}

class Solution {

        static int count = 0;
        public 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;
    }

    public static Node kthSmallest(Node root, int k){

        if (root == null)
            return null;

        Node left = kthSmallest(root.left, k);
        if (left != null)
            return left;
        count++;
        if (count == k)
            return root;
        return kthSmallest(root.right, k);
    }

    public static void printKthSmallest(Node root, int k){
        Node res = kthSmallest(root, k);
        if (res == null)
          System.out.println("There are less than k nodes in the BST");
        else
            System.out.println("K-th Smallest Element is " + res.data);
    }

    public static void main(String[] args){
        Node root = null;
        int keys[] = { 20, 8, 22, 4, 12, 10, 14 };
        for (int x : keys)
            root = insert(root, x);
        int k = 3;
        printKthSmallest(root, k);
    }
}
```

Solution 4 :

Time Complexity : o(n1+n2)
Space Complexity: o(h1+h2)

```
import java.util.Stack;
public class Solution {

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

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

    static Node root1;
    static Node root2;
    static int countPairs(Node root1, Node root2,
                                                int x)
    {
        if (root1 == null || root2 == null)
            return 0;
        Stack<Node> st1 = new Stack<>();
        Stack<Node> st2 = new Stack<>();
        Node top1, top2;

        int count = 0;
        while (true) {
            while (root1 != null) {
                st1.push(root1);
                root1 = root1.left;
            }
            while (root2 != null) {
                st2.push(root2);
                root2 = root2.right;
            }
            if (st1.empty() || st2.empty()) break;

            top1 = st1.peek();
            top2 = st2.peek();
            if ((top1.data + top2.data) == x) {
                count++;
                st1.pop();
                st2.pop();
                root1 = top1.right;
                root2 = top2.left;
            }

            else if ((top1.data + top2.data) < x) {
                st1.pop();
                root1 = top1.right;
            }

            else {
                st2.pop();
                root2 = top2.left;
            }
        }
        return count;
    }

    public static void main(String args[])
    {
        root1 = new Node(5);
        root1.left = new Node(3);
        root1.right = new Node(7);
        root1.left.left = new Node(2);
        root1.left.right = new Node(4);
        root1.right.left = new Node(6);
        root1.right.right = new Node(8);

        root2 = new Node(10);
        root2.left = new Node(6);
        root2.right = new Node(15);
        root2.left.left = new Node(3);
        root2.left.right = new Node(8); root2.right.left = new Node(11);
        root2.right.right = new Node(18);

        int x = 16;
        System.out.println("Pairs = "
            + countPairs(root1, root2, x));
    }
}
```

Solution 5 :

Time Complexity : o(n)
Space Complexity: o(n)

```
class Solution {

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

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

static class Info{
    int max;
    int min;
    boolean isBST;
    int sum;
    int currmax;

    Info(int m,int mi, boolean is,
        int su, int cur) {
```

```
        max = m;
                min = mi;
                isBST = is;
                sum = su;
                currmax = cur;
        }
        Info(){}
};

static class INT{
        int a;
}

static Info MaxSumBSTUtil( Node root, INT maxsum){
        if (root == null)
                return new Info( Integer.MIN_VALUE,
                                        Integer.MAX_VALUE, true, 0, 0 );
        if (root.left == null && root.right == null){
                maxsum.a = Math.max(maxsum.a, root.data);
                return new Info( root.data, root.data,
                                        true, root.data, maxsum.a );
        }

        Info L = MaxSumBSTUtil(root.left, maxsum);
        Info R = MaxSumBSTUtil(root.right, maxsum);

        Info BST=new Info();
        if (L.isBST && R.isBST && L.max < root.data &&
                                                R.min > root.data) {

                BST.max = Math.max(root.data, Math.max(L.max, R.max));
                BST.min = Math.min(root.data, Math.min(L.min, R.min));

                maxsum.a = Math.max(maxsum.a, R.sum +
                root.data + L.sum); BST.sum = R.sum + root.data +
                L.sum;
                BST.currmax = maxsum.a;

                BST.isBST = true;
                return BST;
        }
        BST.isBST = false;
        BST.currmax = maxsum.a;
        BST.sum = R.sum + root.data + L.sum;
                                                    return BST;

}


static int MaxSumBST( Node root){
        INT maxsum = new INT();
        maxsum.a = Integer.MIN_VALUE;
        return MaxSumBSTUtil(root,
maxsum).currmax; }

public static void main(String args[]){
        Node root = new Node(5);
        root.left = new Node(14);
        root.right = new Node(3);
        root.left.left = new Node(6);
        root.right.right = new Node(7);
        root.left.left.left = new Node(9);
        root.left.left.right = new Node(1);

        System.out.println( MaxSumBST(root));
}
}
```