1.https://leetcode.com/problems/recover-binary-search-tree/

```
public class Solution {
    public void recoverTree(TreeNode root) {
        //morris inorder traversal
        if(root == null) return;
        TreeNode prev = null;
        TreeNode first = null;
        TreeNode second = null;
        while(root != null){

            if(root.left == null){
                //print root
                if(prev != null && prev.val > root.val){
                    if(first == null)  first =  prev;
                    second = root;
                }
                prev = root;
                root = root.right;
            }else{
                TreeNode cur = root.left;
```

```
            while(cur.right != null && cur.right != root)
cur = cur.right;

            if(cur.right == null){
                cur.right = root;
                root = root.left;
            }else{
                cur.right = null;
                //print root
                if(prev != null && prev.val > root.val){
                    if(first == null)  first =  prev;
                        second = root;
                    }
                    prev = root;

                root = root.right;
            }
        }
    }
    int temp = first.val;
    first.val = second.val;
    second.val = temp;

  }
```

}

2.https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

```
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        return lca(root,p.val,q.val);
    }
    public static TreeNode lca(TreeNode root,int n1,int n2){//O(n)
        ArrayList<TreeNode> path1=new ArrayList<>();
        ArrayList<TreeNode> path2=new ArrayList<>();
```

```
        getPath(root,n1,path1);
        getPath(root,n2,path2);

        int i=0;

        for(i=0;i<path1.size() && i<path2.size();i++){
            if(path1.get(i)!=path2.get(i)){
                break;
            }
        }
        TreeNode lastComman=path1.get(i-1);
        return lastComman;
    }
    public static boolean getPath(TreeNode root,int
n,ArrayList<TreeNode> path){
        if(root==null){
            return false;
        }
        path.add(root);
        if(root.val==n){
            return true;
        }
        boolean left=getPath(root.left, n, path);
        boolean right=getPath(root.right, n, path);
        if(left||right){
```

```
            return true;
        }
        path.remove(path.size()-1);
        return false;
    }
}
```

3.https://leetcode.com/problems/diameter-of-binary-tree/

```
public class Solution
 {
static class Info
{
int diameter; int height;
 public Info(int diameter,int height)
{
this.diameter=diameter; this.height=height;
 }
```

```
}

public int diameterOfBinaryTree(TreeNode root)
{ return --optimizeDiameter(root).diameter;
 }

public static Info optimizeDiameter(TreeNode root)

{
if(root==null)
{
return new Info(0, 0);
}

Info leftInfo=optimizeDiameter(root.left);
Info rightInfo=optimizeDiameter(root.right);

int
currDiameter=Math.max(leftInfo.height+rightInfo.hei
ght+1,Math.max(leftInfo.diameter,
rightInfo.diameter) );

int currHeight=Math.max(leftInfo.height,
rightInfo.height)+1;
```

```
return new Info(currDiameter, currHeight);
 }
}
```

4.https://practice.geeksforgeeks.org/problems/print-common-nodes-in-bst/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

```
//User function Template for Java
class Solution
{
    //Function to find the nodes that are common in both BST.
```

```
    public static ArrayList<Integer>
findCommon(Node root1,Node root2)
    {
        //code here
        ArrayList<Integer> list = new ArrayList<>() ;
        Set<Integer> set = new HashSet<>() ;
        Stack<Node> stack = new Stack<>() ;
        Node cur = root1 ;
        while(!stack.isEmpty() || cur!=null){
            while(cur!=null){
                stack.push(cur) ;
                cur = cur.left ;
            }
            Node n = stack.pop() ;
            set.add(n.data);
            cur = n.right ;
        }
        stack = new Stack<>() ;
        cur = root2 ;
        while(!stack.isEmpty() || cur!=null){
            while(cur!=null){
                stack.push(cur) ;
                cur = cur.left ;
            }
            Node n = stack.pop() ;
```

```
            if(set.contains(n.data)){
                list.add(n.data);
            }
            cur = n.right ;
        }
        return list ;
    }
    // public static void func(Node node ,
ArrayList<Integer>){

    // }
}
```

5.https://leetcode.com/problems/same-tree/

/** * Definition for a binary tree node. * public class TreeNode { * int val; * TreeNode left; * TreeNode

```
right; * TreeNode() {} * TreeNode(int val) { this.val =
val; } * TreeNode(int val, TreeNode left, TreeNode
right) { * this.val = val; * this.left = left; * this.right =
right; * } * } */
***/


public class Solution
{
public boolean isSameTree(TreeNode p, TreeNode q)
 {
if(p==null && q==null)
{
return true;
}
if( p==null || q==null || (p.val!=q.val))
 return false;

boolean left=isSameTree(p.left,q.left);
boolean right= isSameTree(p.right,q.right);
 return left && right;
 }
}
```

6.https://leetcode.com/problems/kth-smallest-element-in-a-bst/

```
public class Solution {
    int count =0;
    int result =Integer.MIN_VALUE;
    public int kthSmallest(TreeNode root, int k)
    {
    helper(root,k);
    return result;
    }
    public void helper( TreeNode root ,int k )
    {
          if(root==null)
            return ;

        kthSmallest(root.left,k);
         if(++count ==k)
          result = root.val;
         kthSmallest(root.right,k);
    }
}
```

7.
https://www.interviewbit.com/problems/path-to-given-node/

```
import java.util.ArrayList;

public class PathToGivenNode {

    public static ArrayList<Integer> findPath(Node root, int data) {
        if (root == null) {
            return null;
        }

        if (root.data == data) {
            ArrayList<Integer> path = new ArrayList<>();
            path.add(root.data);
            return path;
        }

        ArrayList<Integer> leftPath = findPath(root.left,
```

```
data);
        if (leftPath != null) {
            leftPath.add(root.data);
            return leftPath;
        }

        ArrayList<Integer> rightPath =
findPath(root.right, data);
        if (rightPath != null) {
            rightPath.add(root.data);
            return rightPath;
        }

        return null;
    }

    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.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
```

```
        ArrayList<Integer> path = findPath(root, 5);
        System.out.println(path);
    }
}

class Node {

    int data;
    Node left;
    Node right;

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

8.https://leetcode.com/problems/validate-binary-search-tree/

```
/** * Definition for a binary tree node. * public class
```

```
TreeNode { * int val; * TreeNode left; * TreeNode
right; * TreeNode() {} * TreeNode(int val) { this.val =
val; } * TreeNode(int val, TreeNode left, TreeNode
right) { * this.val = val; * this.left = left; * this.right =
right; * } * } */
**/

public class Solution
{
public boolean isValidBST(TreeNode root)
 {
boolean
result=bstHelper(root,Long.MIN_VALUE,Long.MAX_V
ALUE);
return result;
}

public boolean bstHelper(TreeNode root,long
min,long max)
{
if(root==null)
 {
return true;
}
boolean leftRes=bstHelper(root.left,min,root.val);
```

```
boolean rightRes=bstHelper(root.right,root.val,max);
return (root.val>min && root.val<max)? leftRes &&
rightRes:false;

}
}
```