Binary Search Tree

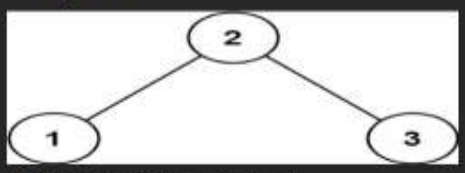
1.Validate Binary Search Tree

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*

A **valid BST** is defined as follows:

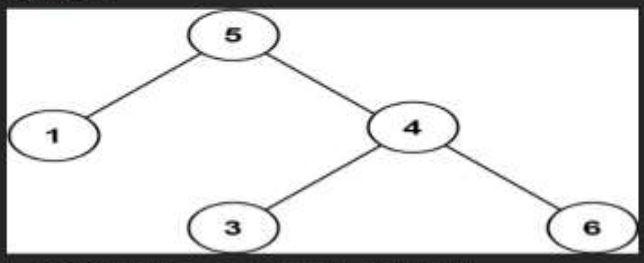
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:




```
Input: root = [2,1,3]
Output: true
```

Example 2:



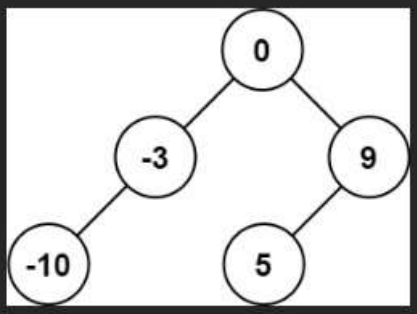

```
Input: root = [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's value is 4.
```

```java
*/
class Solution {
    public boolean isValidBST(TreeNode root) {

        return valid(root,null,null);
    }
    public boolean valid(TreeNode root,TreeNode min,TreeNode max){
        if(root==null){
            return true;
        }
        if(min !=null && root.val<=min.val){
            return false;
        }
        if(max !=null && root.val>=max.val){
            return false;
        }
        return valid(root.left,min,root) && valid(root.right,root,max);
    }
}
```

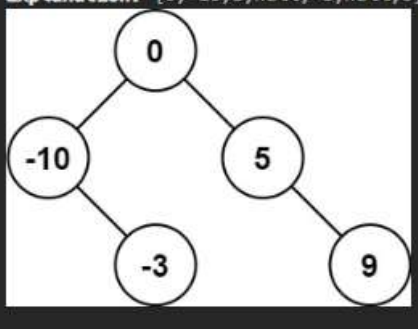
2.Converted sorted array to BST

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a height-balanced binary search tree.*

Example 1:



Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:




```java
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        if(nums.length==0){
            return null;
        }

        return helper(nums,0,nums.length-1);
    }
    public TreeNode helper(int nums[],int low,int high){
        if(low>high){
            return null;
        }
        int mid=(low+high)/2;
        TreeNode node=new TreeNode(nums[mid]);
        node.left=helper(nums,low,mid-1);
        node.right=helper(nums,mid+1,high);

        return node;
    }

}
```

3.Flatten BST to sorted List

You are given a **Binary Search Tree (BST)** with **n** nodes, each node has a distinct value assigned to it. The goal is to flatten the tree such that, the **left child** of each element points to nothing **(NULL),** and the **right child** points to the next element in the sorted list of elements of the **BST** (look at the examples for clarity). You must accomplish this **without using any extra storage**, except for recursive calls, which are allowed.

Note: If your **BST** does have a **left child**, then the system will print a **-1** and will skip it, resulting in an **incorrect solution**.

Example 1:

Input:

Output: 2 3 4 5 6 7 8

Explanation:
After flattening, the tree looks like this

Here, left of each node points to NULL and right contains the next node.

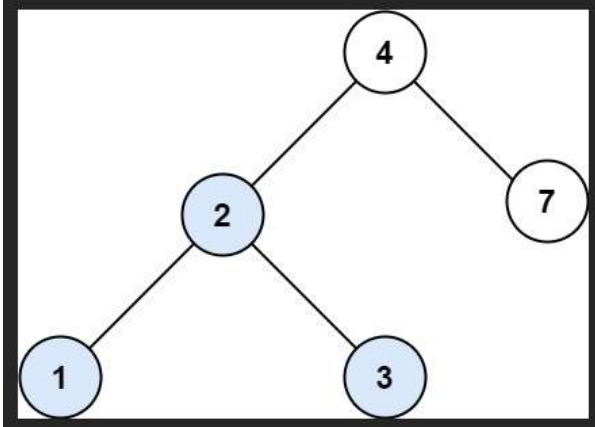
```java
class Solution {
    public Node flattenBST(Node root) {
        // Code here
        if(root == null){
            return null;
        }

        Node head = flattenBST(root.left);
        root.left = null;

        Node temp = head;
        if(temp == null){
            head = root;
        } else {
            while(temp != null && temp.right != null){
                temp = temp.right;
            }
            temp.right = root;
        }

        root.right = flattenBST(root.right);

        return head;
    }
}
```

4.Search in Binary Search Tree

You are given the `root` of a binary search tree (BST) and an integer `val`.

Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node. If such a node does not exist, return `null`.

Example 1:



Input: root = [4,2,7,1,3], val = 2
Output: [2,1,3]

```java
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {

        if(root==null || root.val==val){
            return root;
        }
        if(root.val>val){
            return searchBST(root.left,val);
        }else{
            return searchBST(root.right,val);
        }
    }
}
```

5.Preorder Traversal and Bst

Given an array **arr[ ]** of size **N** consisting of **distinct** integers, write a program that returns **1** if given array can represent preorder traversal of a possible BST, else returns **0**.

Example 1:

Input:
N = 3
arr = {2, 4, 3}
Output: 1
Explanation: Given arr[] can represent preorder traversal of following BST:

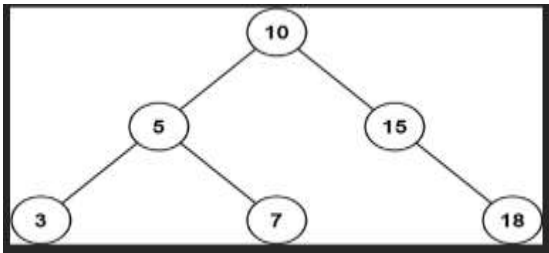
Example 2:

Input:
N = 3
Arr = {2, 4, 1}
Output: 0
Explanation: Given arr[] cannot represent preorder traversal of a BST.

```java
// User function Template for Java

class Solution {
    static int canRepresentBST(int arr[], int n) {
        // code here
       Stack<Integer> st = new Stack<>();
       int root = -1;
       for(int i=0;i<n;i++) {
          if(arr[i]<root) return 0;
           while(!st.empty() && arr[i]>st.peek()) {
               root = st.peek();
               st.pop();
           }
           st.push(arr[i]);
       }
       return 1;
    }
}
```

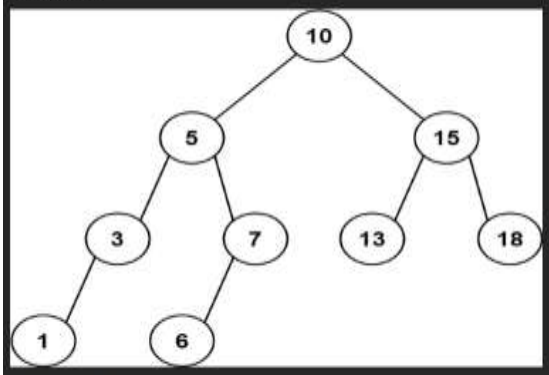
6.Range Sum of BST

Given the root node of a binary search tree and two integers low and high, return the sum of values of all nodes with a value in the inclusive range [low, high].



Input: root = [10,5,15,3,7,null,18], low = 7, high = 15
Output: 32
Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32.

Example 2:



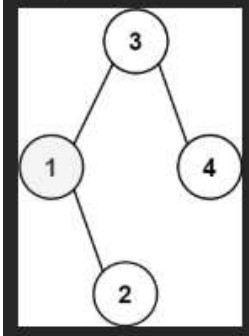
Input: root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10
Output: 23
Explanation: Nodes 6, 7, and 10 are in the range [6, 10]. 6 + 7 + 10 = 23.

```java
*/
class Solution {
    public int rangeSumBST(TreeNode root, int low, int high) {

        if(root==null){
            return 0;
        }
        int sum=root.val>=low && root.val<=high ? root.val:0;

        if(root.left !=null){
            sum +=rangeSumBST(root.left,low,high);
        }
        if(root.right !=null){
            sum +=rangeSumBST(root.right,low,high);
        }
        return sum;

    }
}
```

7.Kth Smallest Element in BST

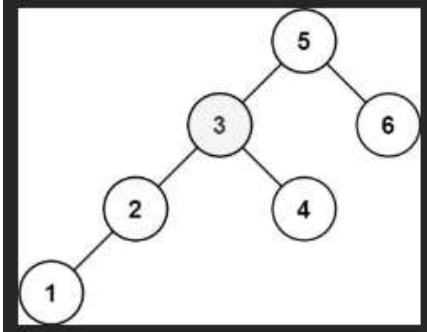
Given the `root` of a binary search tree, and an integer `k`, return the k^{th} *smallest value (**1-indexed**) of all the values of the nodes in the tree.*

Example 1:



Input: root = [3,1,4,null,2], k = 1
Output: 1

Example 2:



Input: root = [5,3,6,2,4,null,null,1], k = 3
Output: 3

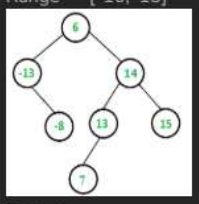
```
15  */
16  class Solution {
17      int ans=Integer.MAX_VALUE;
18      int count=1;
19      public int kthSmallest(TreeNode root, int k) {
20          if(root==null){
21              return 0;
22          }
23          kthSmallest(root.left,k);
24          if(count==k){
25              ans=root.val;
26          }
27          count++;
28          kthSmallest(root.right,k);
29          return ans;
30      }
31  }
```

8.Remove BST key outside given range

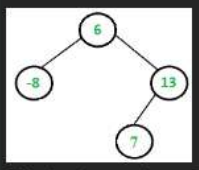
Given a Binary Search Tree (BST) and a range [min, max], remove all keys which are outside the given range. The modified tree should also be BST.

Example 1:

Input:
Range = [-10, 13]



Output:
-8 6 7 13
Explanation:
Nodes with values -13, 14 and 15 are outside the given range and hence are removed from the BST.



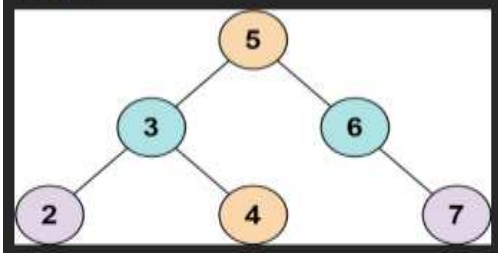
This is the resultant BST and it's inorder traversal is -8 6 7 13.

```
class Solution {
    Node removekeys(Node root, int l, int r) {
        // code here
        if(root==null){

            return null;

        }

        if(root.data>r){

            return removekeys(root.left,l,r);

        }

        if(root.data<l){

            return removekeys(root.right,l,r);

        }

        root.left=removekeys(root.left,l,r);

        root.right=removekeys(root.right,l,r);

        return root;

    }

}
```

9.Two Sum IV – Input is a BST

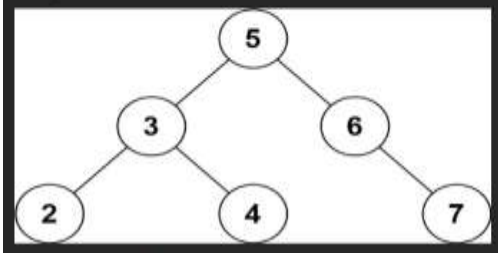
Given the `root` of a binary search tree and an integer `k`, return `true` *if there exist two elements in the BST such that their sum is equal to* `k`, *or* `false` *otherwise.*

Example 1:




```
Input: root = [5,3,6,2,4,null,7], k = 9
Output: true
```

Example 2:




```
Input: root = [5,3,6,2,4,null,7], k = 28
Output: false
```

```java
 */
class Solution {
    public void inOrder(TreeNode node,ArrayList<Integer> list){
        if(node==null){
            return;
        }
        inOrder(node.left,list);
        list.add(node.val);
        inOrder(node.right,list);
    }
    public boolean findTarget(TreeNode root, int k) {
       ArrayList<Integer> list=new ArrayList<>();
       inOrder(root,list);
       int left=0;
       int right=list.size()-1;

       while(left<right){
            int sum=list.get(left)+list.get(right);
            if(sum==k){
                return true;
            }else if(sum<k){
                left++;
            }else{
                right--;
            }
       }
       return false;

    }
    /*
    HashSet<Integer> set=new HashSet<>();
```

10.Delete node in a BST

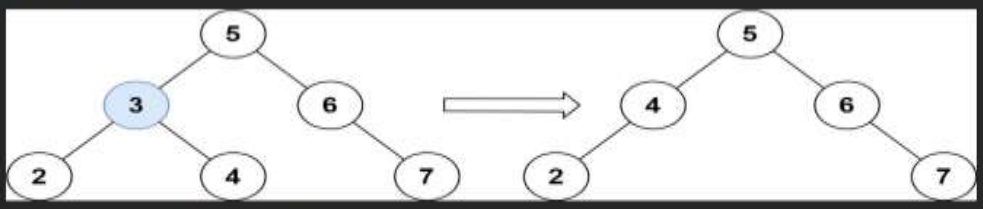
Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the **root node reference** (possibly updated) of the BST.*

Basically, the deletion can be divided into two stages:

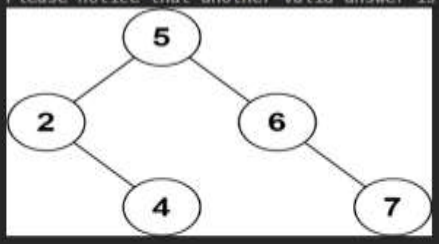
1. Search for a node to remove.

2. If the node is found, delete the node.

Example 1:




```
Input: root = [5,3,6,2,4,null,7], key = 3
Output: [5,4,6,2,null,null,7]
Explanation: Given key to delete is 3. So we find the node with value 3 and delete it.
One valid answer is [5,4,6,2,null,null,7], shown in the above BST.
Please notice that another valid answer is [5,2,6,null,4,null,7] and it's also accepted.
```



Example 2:

```
Input: root = [5,3,6,2,4,null,7], key = 0
Output: [5,3,6,2,4,null,7]
Explanation: The tree does not contain a node with value = 0.
```

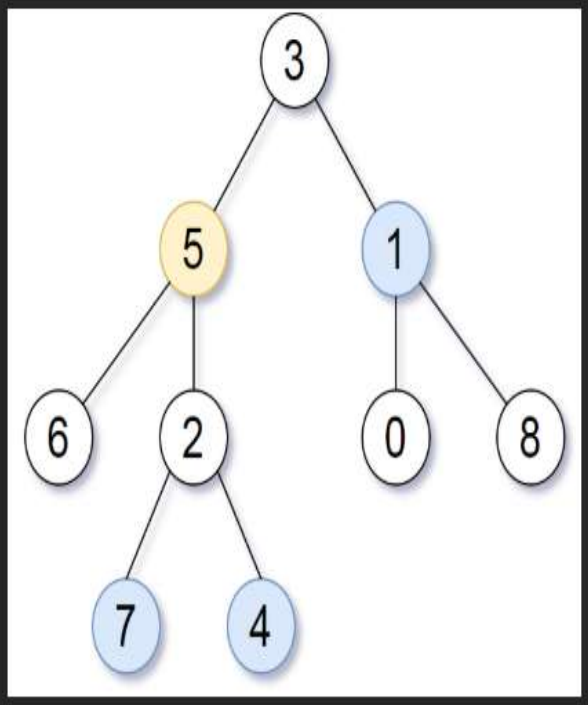
```java
*/
class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if(root==null){
            return null;
        }
        if(key<root.val){
            root.left=deleteNode(root.left,key);
        }else if(key>root.val){
            root.right=deleteNode(root.right,key);
        }else{
            if(root.left==null){
                return root.right;
            }else if(root.right==null){
                return root.left;
            }
            TreeNode min=findMin(root.right);
            root.val=min.val;
            root.right=deleteNode(root.right,root.val);

        }
        return root;
    }
    public TreeNode findMin(TreeNode root){
        while(root.left!=null){
            root=root.left;

        }
        return root;
    }
}
```

11.All Node Distance K in Binary Tree

Given the `root` of a binary tree, the value of a target node `target`, and an integer `k`, return *an array of the values of all nodes that have a distance* `k` *from the target node.*

You can return the answer in **any order.**

Example 1:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], target = 5, k = 2
Output: [7,4,1]
Explanation: The nodes that are a distance 2 from the target node (with value 5) have values 7, 4, and 1.
```

Example 2:

```
Input: root = [1], target = 1, k = 3
Output: []
```

```java
class Solution {
    public List<Integer> distanceK(TreeNode root, TreeNode target, int K) {
        // Step 1: Build adjacency list graph
        Map<TreeNode, List<TreeNode>> graph = new HashMap<>();
        buildGraph(root, null, graph);

        // Step 2: Perform BFS from the target node
        Queue<Pair<TreeNode, Integer>> queue = new LinkedList<>();
        Set<TreeNode> visited = new HashSet<>();
        List<Integer> result = new ArrayList<>();

        queue.add(new Pair<>(target, 0));
        visited.add(target);

        while (!queue.isEmpty()) {
            Pair<TreeNode, Integer> pair = queue.poll();
            TreeNode node = pair.getKey();
            int distance = pair.getValue();

            if (distance == K) {
                result.add(node.val);
            }

            if (distance > K) {
                break;
            }

            for (TreeNode neighbor : graph.get(node)) {
                if (!visited.contains(neighbor)) {
                    visited.add(neighbor);
                    queue.add(new Pair<>(neighbor, distance + 1));
                }
            }
        }

        return result;
    }
```

ved

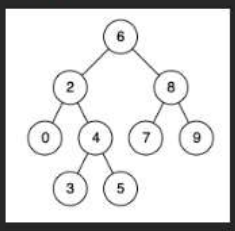
```java
        return result;
    }

    private void buildGraph(TreeNode node, TreeNode parent, Map<TreeNode, List<TreeNode>> graph) {
        if (node == null) {
            return;
        }

        if (!graph.containsKey(node)) {
            graph.put(node, new ArrayList<>());
        }

        if (parent != null) {
            graph.get(node).add(parent);
            graph.get(parent).add(node);
        }

        buildGraph(node.left, node, graph);
        buildGraph(node.right, node, graph);
    }
}
```

12.Lowest Common Ansector of a Binary Search Tree

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

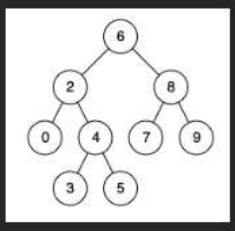
According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
```

Example 2:




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.
```

```
9  */
10
11 public class Solution {
12     public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
13         if(root.val > p.val && root.val > q.val){
14             return lowestCommonAncestor(root.left, p, q);
15         }else if(root.val < p.val && root.val < q.val){
16             return lowestCommonAncestor(root.right, p, q);
17         }else{
18             return root;
19         }
20     }
21 }
```

13.Inorder Succesor in BST

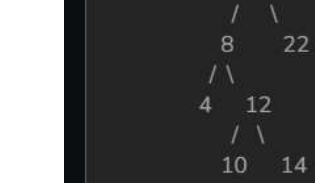
Given a BST, and a reference to a Node **k** in the BST. Find the Inorder Successor of the given node in the BST. If there is no successor, return -1.

Examples :

Input: root = [2, 1, 3], k = 2

Output: 3
Explanation: Inorder traversal : 1 2 3 Hence, inorder successor of 2 is 3.

Input: root = [20, 8, 22, 4, 12, N, N, N, N, 10, 14], k = 8



Output: 10
Explanation: Inorder traversal: 4 8 10 12 14 20 22. Hence, successor of 8 is 10.

Input: root = [2, 1, 3], k = 3

Output: -1
Explanation: Inorder traversal : 1 2 3 Hence, inorder successor of 3 is null.

```
*/
class Solution {
    // returns the inorder successor of the Node x in BST (rooted at 'root')
    public int inorderSuccessor(Node root, Node x) {
        // add code here.
        int successor=-1;
        while(root!=null){
            if(x.data<root.data){
                successor=root.data;
                root=root.left;
            }
            else root=root.right;
        }
        return successor;
    }
}
```

14.Ceil in BST

Given a BST and a number **X**, find **Ceil of X**.
Note: Ceil(X) is a number that is either equal to X or is immediately greater than X.

If Ceil could not be found, return -1.

Example 1:

Input: root = [5, 1, 7, N, 2, N, N, N, 3], X = 3

Output: 3
Explanation: We find 3 in BST, so ceil of 3 is 3.

Example 2:

Input: root = [10, 5, 11, 4, 7, N, N, N, N, N, 8], X = 6

Output: 7
Explanation: We find 7 in BST, so ceil of 6 is 7.

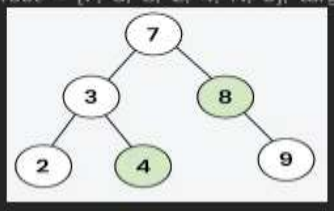
```
09
10 class Tree {
11     // Function to return the ceil of given number in BST.
12     int findCeil(Node root, int key) {
13         if (root == null) return -1;
14         // Code here
15         if (root == null) return -1;
16         int ceil=-1;
17         while(root!=null){
18             if(key==root.data)return root.data;
19             else if(key<root.data){
20                 ceil=root.data;
21                 root=root.left;
22             }
23             else root=root.right;
24         }
25         return ceil;
26     }
27 }
28
```

15.Pair Sum in BST

Given a Binary Search Tree(BST) and a **target**. Check whether there's a pair of Nodes in the BST with value summing up to the target.

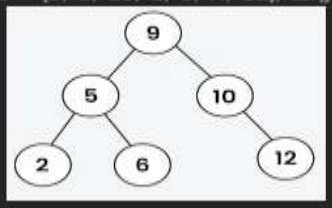
Examples:

Input: root = [7, 3, 8, 2, 4, N, 9], target = 12



Output: True
Explanation: In the binary tree above, there are two nodes (8 and 4) that add up to 12.

Input: root = [9, 5, 10, 2, 6, N, 12], target = 23



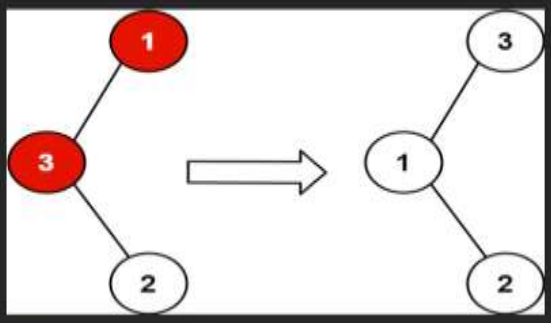
Output: False
Explanation: In the binary tree above, there are no such two nodes exists that add up to 23.

```java
class Solution {
    public void inOrder(Node node,ArrayList<Integer> list){
        if(node==null){
            return;
        }
        inOrder(node.left,list);
        list.add(node.data);
        inOrder(node.right,list);
    }
    boolean findTarget(Node root, int target) {
        // Write your code here
        ArrayList<Integer> list=new ArrayList<>();
        inOrder(root,list);

        int left=0;
        int right=list.size()-1;

        while(left<right){
            int sum=list.get(left)+list.get(right);
            if(sum==target){
                return true;
            }else if(sum<target){
                left++;
            }else{
                right--;
            }
        }
        return false;
    }
}
```

16.Recover Binary Search Tree

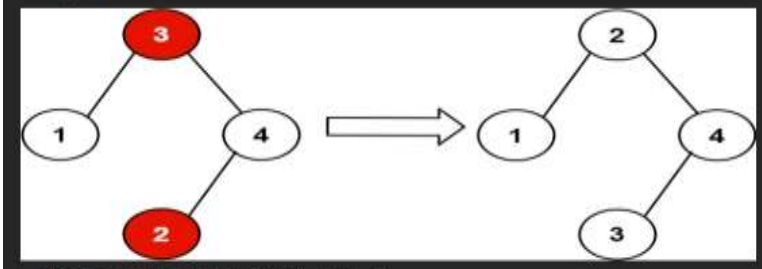
You are given the `root` of a binary search tree (BST), where the values of **exactly** two nodes of the tree were swapped by mistake. *Recover the tree without changing its structure.*

Example 1:




```
Input: root = [1,3,null,null,2]
Output: [3,1,null,null,2]
Explanation: 3 cannot be a left child of 1 because 3 > 1. Swapping 1 and 3 makes the BST valid.
```

Example 2:




```
Input: root = [3,1,4,null,null,2]
Output: [2,1,4,null,null,3]
Explanation: 2 cannot be in the right subtree of 3 because 2 < 3. Swapping 2 and 3 makes the BST valid.
```

```java
*/
class Solution {
    TreeNode prev=null,first=null,second=null;
    void inorder(TreeNode root){
        if(root==null)
            return ;
        inorder(root.left);
        if(prev!=null&&root.val<prev.val){
            if(first==null)
                first=prev;
            second=root;
        }
        prev=root;
        inorder(root.right);
    }
    public void recoverTree(TreeNode root) {
        if(root==null)
            return ;
        inorder(root);
        int temp=first.val;
        first.val=second.val;
        second.val=temp;
    }
}
```

https://leetcode.com/problems/serialize-and-deserialize-bst/https://leetcode.com/problems/serialize-and-deserialize-bst/