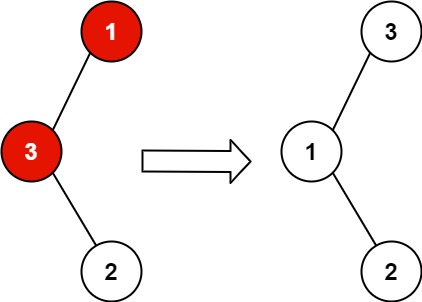
You are given the root of a binary search tree (BST), where the values of **exactly** two nodes of the tree were swapped by mistake. *Recover the tree without changing its structure*.

**Example 1:**

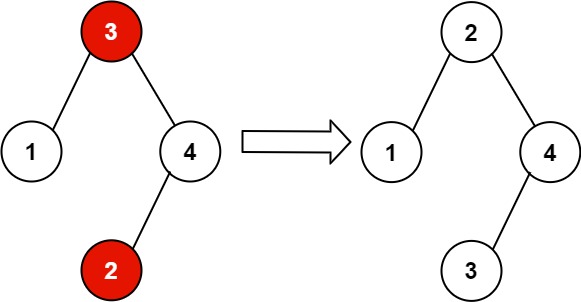


**Input:** root = [1,3,null,null,2]

**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.

**Solution:**

/\*\*

\* 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;

\* }

\* }

\*/

class Solution {

private TreeNode first;

private TreeNode middle;

private TreeNode prev;

private TreeNode last;

public void recoverTree(TreeNode root) {

first = middle = last = null;

prev = new TreeNode(Integer.MIN\_VALUE);

inorder(root);

if(first != null && last!= null){

int t = first.val;

first.val = last.val;

last.val = t;

}

else if(first != null && middle!= null){

int t = first.val;

first.val = middle.val;

middle.val = t;

}

}

private void inorder(TreeNode root){

if(root == null)

return;

inorder(root.left);

if(prev!= null && (prev.val>root.val)){

if(first == null){

first = prev;

middle = root;

}

else

last = root;

}

prev = root;

inorder(root.right);

}

}

T.C: = O(N), S.C:= O(1)