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

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

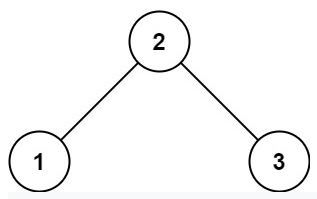
A **valid BST** is defined as follows:

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.

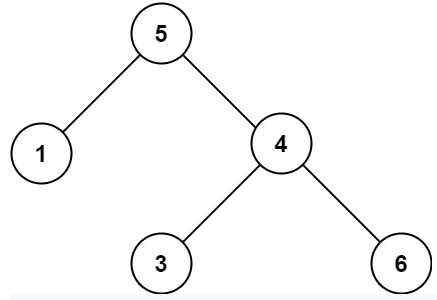
**Example 1:**



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.

**Constraints:**

The number of nodes in the tree is in the range [1, 10^4].

-2^31 <= Node.val <= 2^31 - 1

**Attempt 1: 2022-11-17**

**Solution 1:  Divide and Conquer (10min)**

/\*\*

\* 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 {

    public boolean isValidBST(TreeNode root) {

        // Refer to

        // https://leetcode.com/problems/validate-binary-search-tree/discuss/1389569/Input-2147483647-ouput%3A-true-why

        // Must use Long to process, as using Integer.MIN\_VALUE or

    // Integer.MAX\_VALUE will not able to handle corner case

    // as root.val = 2147483647 (Its a single node with val = 2147483647

        // ,so a bst also), which expected return true, but wrongly return false

        return helper(root, Long.MIN\_VALUE, Long.MAX\_VALUE);

    }

    private boolean helper(TreeNode root, long min, long max) {

        // Base case

        if(root == null) {

            return true;

        }

        if(root.val <= min || root.val >= max) {

            return false;

        }

        // Divide

        boolean left = helper(root.left, min, root.val);

        boolean right = helper(root.right, root.val, max);

        // Conquer

        return left && right;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Did everyone who use a long tried int before and failed by the test case 2147483647？**

**Refer to**

<https://leetcode.com/problems/validate-binary-search-tree/discuss/32109/My-simple-Java-solution-in-3-lines/656786>

2147483647 is the maximum integer value. The code is written to say the node must have a value ***less than*** whatever is passed as the max of the range, which is good because we want to be sure the same value isn't in the tree twice. **However, any possible int value should be allowed, so setting the max to 2147483647 (Integer.MAX\_VALUE) at the start doesn't work because then we get a failure for a tree that has the value 2147483647.**

If you use a long for the max and min that are passed around, you can use Long.MAX\_VALUE instead (which will be larger) as well as Long.MIN\_VALUE. Alternatively, you can use an Integer instead of int and set the initial value to null, then write some conditional logic that if max is set to null you don't need to do the comparison.

**Solution 2:  Inorder Iterative traversal (10min)**

/\*\*

\* 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 {

    public boolean isValidBST(TreeNode root) {

        if(root == null) {

            return true;

        }

        TreeNode prev = null;

        Stack<TreeNode> stack = new Stack<TreeNode>();

        while(root != null || !stack.isEmpty()) {

            while(root != null) {

                stack.push(root);

                root = root.left;

            }

            root = stack.pop();

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

                return false;

            }

            prev = root;

            root = root.right;

        }

        return true;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/validate-binary-search-tree/discuss/32112/Learn-one-iterative-inorder-traversal-apply-it-to-multiple-tree-questions-(Java-Solution)>

**Refer to**

[L94.Binary Tree Inorder Traversal (Ref.L98,L230,L144,L145)](note://5CCEDD558CF9429FA837D734A5B29789)

[L333.Largest BST Subtree (Ref.L98)](note://27750017E06842BB91F3070EF9BC6DFB)

[L230.Kth Smallest Element in a BST (Ref.L98)](note://28B70EB5DF2F48F6803E1D69A522EA24)