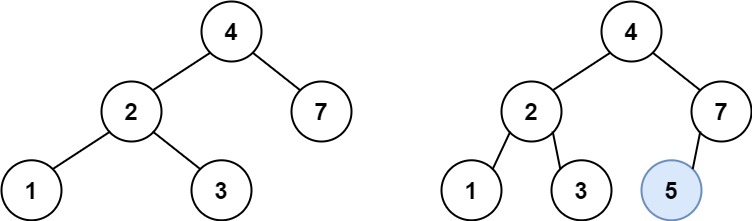
# Question

You are given the root node of a binary search tree (BST) and a value to insert into the tree. Return *the root node of the BST after the insertion*. It is **guaranteed** that the new value does not exist in the original BST.

**Notice** that there may exist multiple valid ways for the insertion, as long as the tree remains a BST after insertion. You can return **any of them**.

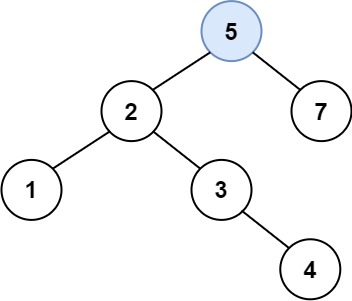
**Example 1:**



**Input:** root = [4,2,7,1,3], val = 5

**Output:** [4,2,7,1,3,5]

**Explanation:** Another accepted tree is:



**Example 2:**

**Input:** root = [40,20,60,10,30,50,70], val = 25

**Output:** [40,20,60,10,30,50,70,null,null,25]

**Example 3:**

**Input:** root = [4,2,7,1,3,null,null,null,null,null,null], val = 5

**Output:** [4,2,7,1,3,5]

**Constraints:**

* The number of nodes in the tree will be in the range [0, 104].
* -108 <= Node.val <= 108
* All the values Node.val are **unique**.
* -108 <= val <= 108
* It's **guaranteed** that val does not exist in the original BST.

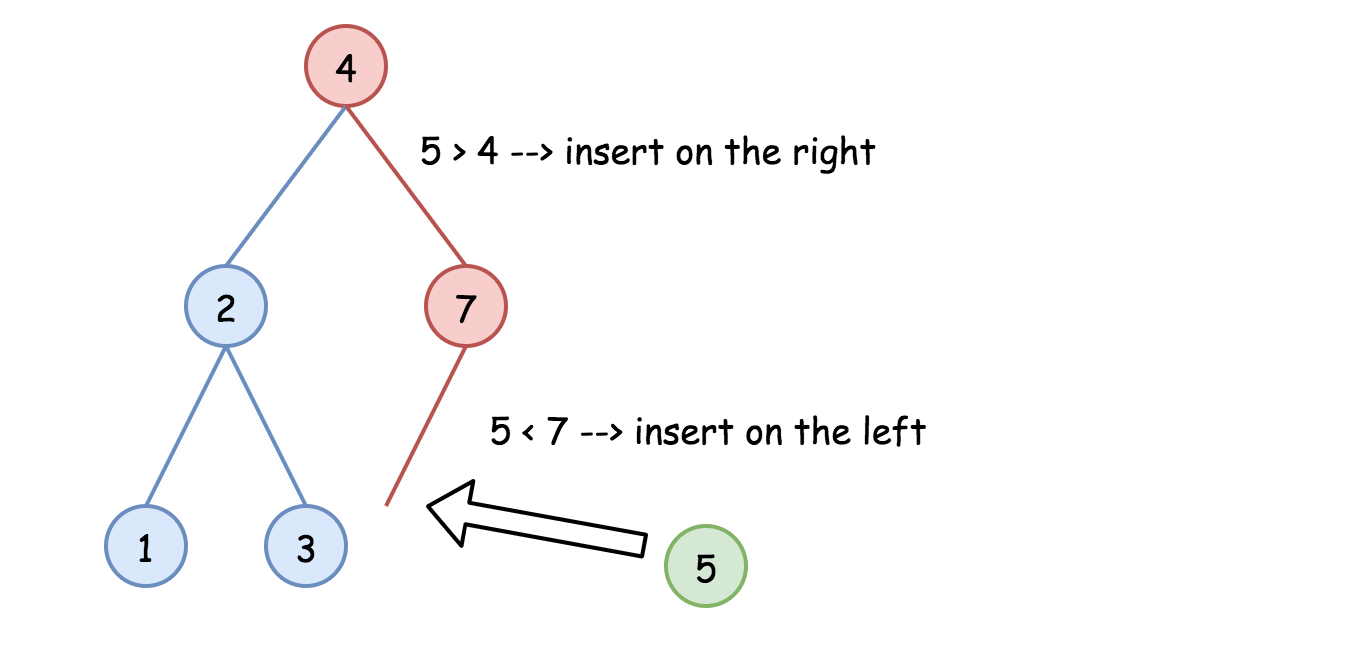
# Solution

#### **Intuition**

One of the huge BST advantages is a [search](https://leetcode.com/problems/search-in-a-binary-search-tree/) for arbitrary element in \mathcal{O}(\log N)O(log*N*) time. Here we'll see that the insert time is \mathcal{O}(\log N)O(log*N*), too, in the average case.

The problem solution is very simple - one could always insert new node as a child of the leaf. To define which leaf to use, one could follow the standard BST logic :

* If val > node.val - go to insert into the right subtree.
* If val < node.val - go to insert into the left subtree.



#### **Approach 1: Recursion**

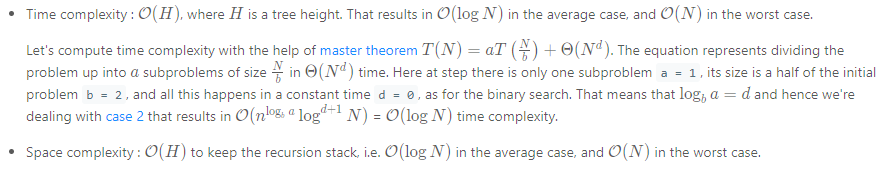
The recursion implementation is very straightforward :

* If root is null - return TreeNode(val).
* If val > root.val - go to insert into the right subtree.
* If val < root.val - go to insert into the left subtree.
* Return root.

1 / 7

|  |
| --- |
| **class Solution {**  **public TreeNode insertIntoBST(TreeNode root, int val) {**  **if (root == null) return new TreeNode(val);**  **// insert into the right subtree**  **if (val > root.val) root.right = insertIntoBST(root.right, val);**  **// insert into the left subtree**  **else root.left = insertIntoBST(root.left, val);**  **return root;**  **}**  **}** |

**Complexity Analysis**



#### **Approach 2: Iteration**

The recursion above could be converted into the iteration

|  |
| --- |
| class Solution {  public TreeNode insertIntoBST(TreeNode root, int val) {  TreeNode node = root;  while (node != null) {  // insert into the right subtree  if (val > node.val) {  // insert right now  if (node.right == null) {  node.right = new TreeNode(val);  return root;  }  else node = node.right;  }  // insert into the left subtree  else {  // insert right now  if (node.left == null) {  node.left = new TreeNode(val);  return root;  }  else node = node.left;  }  }  return new TreeNode(val);  }  } |

**Complexity Analysis**

