

Insert into BST

● MEDIUM

■ Introduction

Think of inserting a new employee into a company org-chart that is always sorted by employee ID. You walk down the hierarchy, at each manager asking "is your ID smaller or larger?" until you find an empty desk — that's where the new person sits. BST Insert works exactly the same way.

■ Core Intuition

Key insight: **A new node ALWAYS becomes a LEAF.** Why? Because we navigate using the BST property until we fall off the tree — and that null spot is where we place the new node. We never insert in the middle.

The Golden Rule

- Navigate exactly like Search.
- When you would return null (not found), instead CREATE and return a new node.
- The BST property is automatically maintained because you followed it to get there.

■■ How It Works Internally

Step 1: If root is null → create new node here (this IS the insertion point).

Step 2: If $val < root.val$ → recursively insert into LEFT subtree. Attach result back to root.left.

Step 3: If $val > root.val$ → recursively insert into RIGHT subtree. Attach result back to root.right.

Step 4: Return root at every level — this rebuilds the tree structure on the way back up.

■ Visual — Insert 6 into BST

Insert Visualization

Before Insert(6): After Insert(6):

10 10

/ \ / \

5 15 5 15

\ \

7 7

/

6 ← NEW LEAF

Path taken: 10→left→5→right→7→left→null → INSERT HERE


```

while (true) {
    if (val < curr.val) {
        if (curr.left == null) { curr.left = newNode; break; }
        curr = curr.left;
    } else {
        if (curr.right == null) { curr.right = newNode; break; }
        curr = curr.right;
    }
}
return root;
}
}

```

■ Time & Space Complexity

Case	Time	Space	Why
Best	$O(\log n)$	$O(\log n)$	Balanced BST, insert near top
Average	$O(\log n)$	$O(\log n)$	Random insertions, tree stays balanced
Worst	$O(n)$	$O(n)$	Sorted input → skewed tree (like linked list)

■ Common Mistakes

Common Errors

- NOT returning root at each level — without "return root", you lose the tree structure.
- Not re-assigning root.left/right — the line `root.left = insertIntoBST(...)` is crucial.
- Inserting duplicates — standard BST does not allow them; clarify with interviewer.
- Forgetting iterative needs explicit break — infinite loop if you don't break on insertion.

■ Patterns & Connections

This "navigate and place" pattern appears in Tries (insert character by character), Segment Trees (update a position), and AVL/Red-Black trees (insert + rebalance). The recursive return-and-reattach idiom is a fundamental tree pattern.

■ Practice Problems

Case	Time	Space	Why
------	------	-------	-----

Medium	LC 701	Insert into a Binary Search Tree	Direct application
Medium	LC 1305	All Elements in Two Binary Search Trees	Insert + merge
Hard	LC 449	Serialize and Deserialize BST	Insert to reconstruct

■ Mental Model

Remember This Forever

- "Follow the search path. Where you would return NOT FOUND — plant the new node."
- New nodes are ALWAYS leaves. The tree's past structure is never disturbed.