



## LeetCode 98 — Validate Binary Search Tree

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

### 1. Problem Title & Link

**Title:** LeetCode 98: Validate Binary Search Tree

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

### 2. Problem Statement (Short Summary)

Given the root of a binary tree, determine whether it is a **Valid Binary Search Tree (BST)**.

A tree is a valid BST if:

1. **Left subtree** values < node value
2. **Right subtree** values > node value
3. **All subtrees** must satisfy the same condition

Not just *immediate children* — **entire subtrees**.

### 3. Examples (Input → Output)

#### Example 1

Tree:

```
2
/\ 
1 3
```

Output: true

#### Example 2

Tree:

```
5
/\ 
1 4
 /\ 
3 6
```

Output: false

Reason:

Node 3 is in the right subtree of 5, but  $3 < 5 \rightarrow$  violates BST rule.

#### Example 3

Tree:

```
2
```



/\

2 2

Output: false

BST rule → duplicates not allowed.

#### 4. Constraints

- Node count  $\leq 10^4$
- Node values may be negative
- **BST does NOT allow duplicates**
- Must validate global property, not local

#### 5. Core Concept (Pattern / Topic)

##### ⭐ DFS with Range Validation

Also related: Inorder Traversal (BST gives sorted array)

#### 6. Thought Process (Step-by-Step Explanation)

##### ✗ Wrong Intuition (Most Students Mistake)

Checking only:

left &lt; node &lt; right

IS NOT ENOUGH.

You must ensure:

- Entire left subtree  $< \text{node}$
- Entire right subtree  $> \text{node}$

##### ✓ Correct Thinking — Maintain Valid Value Range

Every node must lie within a valid range:

( $-\infty, +\infty$ )

For root.

For each left child:

max allowed = parent.val

For each right child:

min allowed = parent.val

#### Approach — Recursive DFS

check(node, minVal, maxVal)

At every node:



- Ensure  $\text{minVal} < \text{node.val} < \text{maxVal}$
- Recurse left with  $(\text{minVal}, \text{node.val})$
- Recurse right with  $(\text{node.val}, \text{maxVal})$

### Alternate Approach — Inorder Traversal

Inorder of BST is **strictly increasing**.

Store inorder → check if sorted without duplicates.

### 7. Visual / Intuition Diagram

Tree:

```

 5
 / \
1  7
 / \
6  8

```

Ranges:

5:  $(-\infty, +\infty)$

1:  $(-\infty, 5)$

7:  $(5, +\infty)$

6:  $(5, 7)$

8:  $(7, +\infty)$

All satisfy → valid.

### 8. Pseudocode

```

function isValid(node, min, max):
    if node is null:
        return true

    if node.val <= min or node.val >= max:
        return false

    return isValid(node.left, min, node.val) AND
          isValid(node.right, node.val, max)

```

### 9. Code Implementation

#### Python

```

class Solution:
    def isValidBST(self, root):

```



```

def dfs(node, minVal, maxVal):
    if not node:
        return True
    if not (minVal < node.val < maxVal):
        return False
    return dfs(node.left, minVal, node.val) and \
           dfs(node.right, node.val, maxVal)

return dfs(root, float('-inf'), float('inf'))

```

## ✓ Java

```

class Solution {
    public boolean isValidBST(TreeNode root) {
        return validate(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean validate(TreeNode node, long min, long max) {
        if (node == null) return true;

        if (node.val <= min || node.val >= max)
            return false;

        return validate(node.left, min, node.val) &&
               validate(node.right, node.val, max);
    }
}

```

## 10. Time & Space Complexity

### DFS Range Method

- **Time:**  $O(n)$
- **Space:**  $O(h)$  recursion stack
  - $h = \text{tree height}$

### Inorder Method

- **Time:**  $O(n)$
- **Space:**  $O(n)$  for inorder list

## 11. Common Mistakes / Edge Cases

✗ Only comparing parent and child

✗ Allowing duplicates (not allowed in BST)



✗ Using MIN\_VALUE / MAX\_VALUE instead of long range (Java overflow)

✗ Incorrect comparison: use strict < not <=

Edge cases:

- ✓ Empty tree → valid
- ✓ Single node → valid
- ✓ Entire left chain / right chain
- ✓ Very large/small values

## 12. Detailed Dry Run (Step-by-Step Table)

Tree:

```

 5
 / \
1  4
 / \
3  6
  
```

Node	Range	Valid?	Next
5	( $-\infty, \infty$ )	✓	left: ( $-\infty, 5$ ), right: ( $5, \infty$ )
1	( $-\infty, 5$ )	✓	leaf
4	( $5, \infty$ )	✗	because $4 < 5$

Return: **false**

## 13. Common Use Cases

- Validating BST before operations
- Checking if tree can be used for binary search
- Database indexing trees
- Memory-optimized search structures

## 14. Common Traps

- ⚠ Node in right subtree smaller than root (classic mistake)
- ⚠ Forgetting to update max/min range properly
- ⚠ Using inorder but allowing equal values
- ⚠ Only checking immediate children

## 15. Builds To (Related Problems)



- LC 700 — Search in BST
- LC 701 — Insert in BST
- LC 450 — Delete in BST
- LC 1008 — BST from preorder
- LC 230 — Kth Smallest in BST
- LC 530 — Minimum Absolute Difference in BST

## 16. Alternate Approaches + Comparison

Approach	Time	Space	Strength
DFS with range	$O(n)$	$O(h)$	Most reliable
Inorder traversal	$O(n)$	$O(n)$	Easy but uses extra array
Iterative inorder	$O(n)$	$O(h)$	No array; maintain prev

## 17. Why This Solution Works (Short Intuition)

A BST is valid only if **every node** lies in a valid range determined by all its ancestors.

The DFS approach enforces these ranges correctly, not just parent-child.

## 18. Variations / Follow-Up Questions

- Allow duplicates but only on one side?
- Return the exact node where violation occurs
- Convert tree to valid BST
- Check if array is valid preorder of BST