

13
.6

Balanced Binary Tree

EASY

■ What You Will Learn

- ✓ Definition of a height-balanced binary tree
- ✓ Naive $O(n^2)$ check vs optimized $O(n)$ check
- ✓ Bottom-up approach: return height or -1 (unbalanced signal)
- ✓ Why early termination makes the optimized solution elegant

■ Intuition First

A balanced binary tree is one where, for every node, the height of the left subtree and the height of the right subtree differ by at most 1. It's not just about the root — every single node must satisfy this.

Naive idea: for each node, compute heights of both subtrees and check the difference. Problem: you recompute heights redundantly — $O(n^2)$. Smart idea: compute height bottom-up AND check balance simultaneously. Return -1 as a special 'unbalanced' signal that propagates up instantly.

■ The $O(n)$ Trick — Return Height OR -1

- Helper function returns: height if subtree is balanced, -1 if unbalanced
- At each node: get $\text{leftH} = \text{helper}(\text{left})$, $\text{rightH} = \text{helper}(\text{right})$
- If $\text{leftH} == -1$ OR $\text{rightH} == -1$: already unbalanced below $\rightarrow$ return -1
- If $|\text{leftH} - \text{rightH}| > 1$: this node unbalanced $\rightarrow$ return -1
- Otherwise: return $1 + \max(\text{leftH}, \text{rightH})$ (the height)

■ Visual

BALANCED: NOT BALANCED:

```

1 1
/ \ /
2 3 2
/ /
4 3
/
4

```

$|h(\text{left}) - h(\text{right})| \leq 1$ $|h(\text{left}) - h(\text{right})| = 3 > 1$ $\times$

■ Dry Run — Optimized

This tree is NOT balanced — left side too deep

Tree: 1
 / \
2 2
 /
3
 /
4

Step 1	helper(4) → null children → return 1
Step 2	helper(3) → leftH=1, rightH=0. $ 1-0 =1 \leq 1$. return 2
Step 3	helper(2-left) → leftH=2, rightH=0. $ 2-0 =2 > 1$ → return -1!
Step 4	helper(1) → leftH=-1 → immediately return -1 (no need to check right!)
Step 5	isBalanced = (helper(root) != -1) = (-1 != -1) = FALSE ✓

■ Java Code — O(n) Optimized

```
public static boolean isBalanced(TreeNode root) {
    return checkHeight(root) != -1;
}

// Returns height if balanced, -1 if unbalanced
private static int checkHeight(TreeNode node) {
    if (node == null) return 0;
    int leftH = checkHeight(node.left);
    if (leftH == -1) return -1; // Early exit: left already unbalanced
    int rightH = checkHeight(node.right);
    if (rightH == -1) return -1; // Early exit: right already unbalanced
    if (Math.abs(leftH - rightH) > 1) return -1; // This node unbalanced
    return 1 + Math.max(leftH, rightH); // Return height
}
```

■■ Complexity

Case	Time	Space	What it means
Naive O(n²)	O(n²)	O(h)	Recomputes height for every node at every level
Optimized	O(n)	O(h)	Each node visited once — height computed alongside balance check

■ Common Mistakes

- Checking only the root's subtrees — EVERY node must be balanced, not just root
- Using naive height computation inside isBalanced → $O(n^2)$ — always use the optimized approach
- Using -1 as both invalid signal AND valid height for empty subtrees — always return 0 for null

■ Why Balanced Trees Matter

- BST operations (search, insert, delete) are $O(\log n)$ ONLY if tree is balanced
- Worst case unbalanced BST degenerates to a linked list → $O(n)$ operations
- AVL trees and Red-Black trees maintain balance automatically
- LeetCode #110 — this exact problem

■ Visual Thinking

Imagine a scale at every node. Left pan holds left subtree height, right pan holds right subtree.

If any scale tips more than 1 unit → the whole tree is unbalanced.

The -1 signal is like a red alert that propagates up instantly — short-circuits the whole check.

Animation: heights bubble up from leaves. Nodes turn red when their scale tips too far.

■ Practice Problems

[Easy] Check if [1,2,3,4,5,6,7] is balanced

[Med] Convert sorted array to balanced BST

[Med] Balance a BST that has become skewed

[Hard] AVL Tree insertion with rotations

■ Quick Revision Box

- Balanced = every node has $|\text{leftH} - \text{rightH}| \leq 1$
- Optimized: return height or -1 (unbalanced signal)

- Early exit when -1 received from child
- $O(n)$ time — each node visited once
- Naive $O(n^2)$ recalculates height redundantly — avoid it