Documentation For Binary Tree Tilt

1. Problem Statement

Given the root of a binary tree, return the sum of every tree node's tilt.

Definition:

- The tilt of a node is the absolute difference between the sum of values in its left subtree and the sum of values in its right subtree.
- If a node is null or has no children, its tilt is considered 0.

2. Intuition

We need to compute the subtree sums for every node to calculate their tilts. A post-order traversal (left → right → root) naturally allows us to:

- Traverse to the leaf nodes first,
- Calculate their sums bottom-up,
- And compute tilt on the way back up.

This recursive traversal ensures no repeated subtree summation, making it efficient.

3. Key Observations

- Leaf nodes always have a tilt of 0.
- We must sum values recursively in each subtree.
- The total tilt is the sum of all nodes' tilts.
- A global accumulator can be used to keep track of the total tilt.

4. Approach

- Use a helper recursive function (dfs) that:
    - Computes the sum of the left subtree.
    - Computes the sum of the right subtree.
    - Calculates the tilt of the current node.
    - Accumulates the tilt into a global variable.
    - Returns the total sum of the subtree rooted at this node.

5. Edge Cases

- Empty tree (root = None) → Return 0.
- Single-node tree → Tilt is 0.
- Unbalanced tree → Still works since tilt is local per node.

6. Complexity Analysis

Time Complexity

- O(n), where n is the number of nodes in the tree.
- Each node is visited once, and computation per node is O(1).

Space Complexity

- O(h), where h is the height of the tree (due to recursion stack).
- Worst case: skewed tree → O(n).
- Best case: balanced tree → O(log n).

7. Alternative Approaches

Approach	Time	Space	Notes
Recursive (post-order)	O(n)	O(h)	Efficient and simple
Iterative with stack	O(n)	O(n)	More complex to track subtree sums
Level-order BFS + Map	O(n)	O(n)	Inefficient as it requires extra bookkeeping

8. Test Cases

✅ Example 1

Input: root = [1,2,3]

Output: 1

Explanation:

Tilt(2)=0, Tilt(3)=0, Tilt(1)=|2-3|=1 → Total Tilt = 1

✅ Example 2

Input: root = [4,2,9,3,5,null,7]

Output: 15

Explanation: Total Tilt = 0+0+0+2+7+6 = 15

✅ Example 3

Input: root = [21,7,14,1,1,2,2,3,3]

Output: 9

✅ Edge Case

Input: root = None

Output: 0

9. Final Thoughts

- This is a great example of divide-and-conquer using recursion.
- The problem teaches how to use global state in recursion, and why post-order is preferred when processing bottom-up.
- Easily extendable to problems like:
  - Calculating subtree sums
  - Checking balance
  - Counting nodes or heights