Count Complete Tree Nodes

This problem asks for counting nodes in a complete binary tree in less than $O(n)$ time. The trick is to leverage the structure of a complete binary tree:

- All levels are completely filled except possibly the last level.
- If the leftmost height equals the rightmost height, the tree is perfect and we can compute nodes directly with $2^h - 1$.
- Otherwise, we recursively count nodes in left and right subtrees.

Approach:

1. Compute left height: Keep going left until null.
2. Compute right height: Keep going right until null.
3. If left height == right height:
   - The tree is perfect (completely filled).
   - Return $2^h - 1$ (i.e. $(1 << h) - 1$).
4. Else:
   - Recursively compute countNode(left) + countNodes(right) + 1.

Complexity:

- Each recursion reduces the tree height by 1.
- Height computation = $O(\log n)$.
- Recursion depth = $O(\log n)$.
- Total: $O(\log^2 n)$ (much better than $O(n)$).