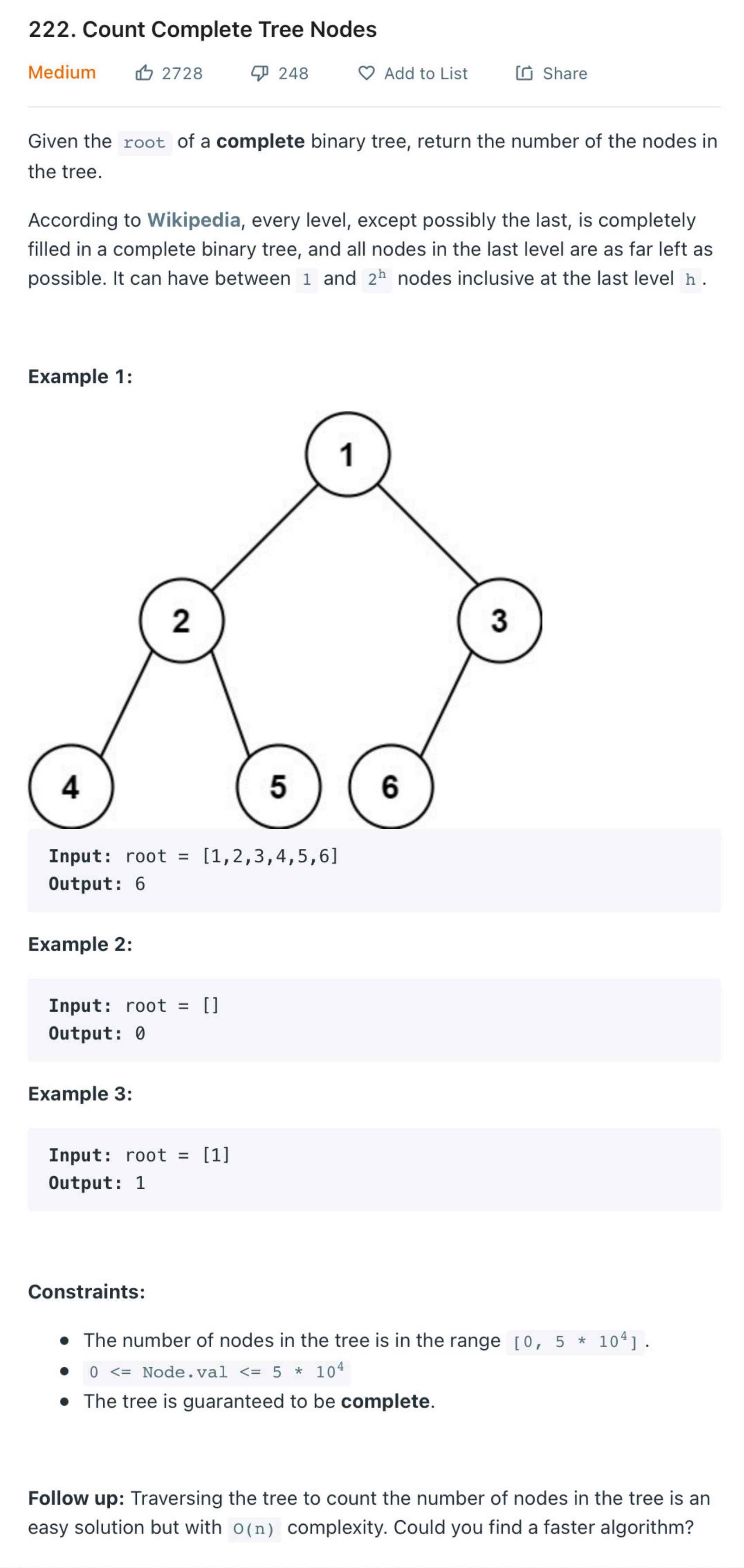


222. Count Complete Tree Nodes

Given the `root` of a **complete** binary tree, return the number of the nodes in the tree.

According to **Wikipedia**, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between `1` and 2^h nodes inclusive at the last level `h`.

Example 1:

```
Input: root = [1,2,3,4,5,6]
Output: 6
```

Example 2:

```
Input: root = []
Output: 0
```

Example 3:

```
Input: root = [1]
Output: 1
```

Constraints:

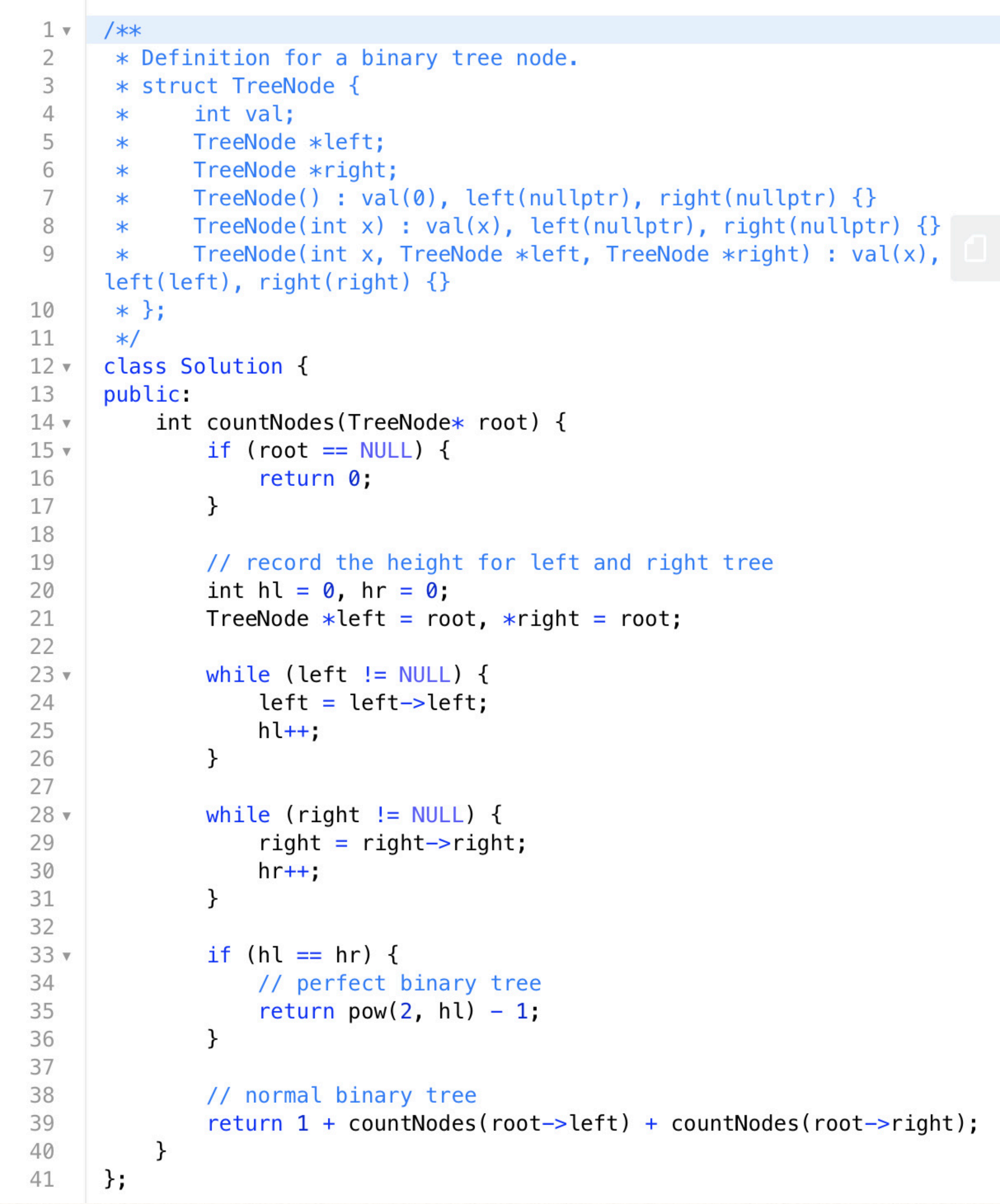
- The number of nodes in the tree is in the range `[0, 5 * `10^4`]`.
- `0 <= Node.val <= 5 * `10^4
- The tree is guaranteed to be **complete**.

Follow up: Traversing the tree to count the number of nodes in the tree is an easy solution but with `O(n)` complexity. Could you find a faster algorithm?

Normal binary tree: iteration
Time complexity: O(N)

Perfect binary tree: $2^h - 1$
Time complexity: O(1)

Overall time complexity: O(log N * log N)



```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int countNodes(TreeNode* root) {
        if (root == NULL) {
            return 0;
        }

        // record the height for left and right tree
        int hl = 0, hr = 0;
        TreeNode *left = root, *right = root;

        while (left != NULL) {
            left = left->left;
            hl++;
        }

        while (right != NULL) {
            right = right->right;
            hr++;
        }

        if (hl == hr) {
            // perfect binary tree
            return pow(2, hl) - 1;
        }

        // normal binary tree
        return 1 + countNodes(root->left) + countNodes(root->right);
    }
};
```