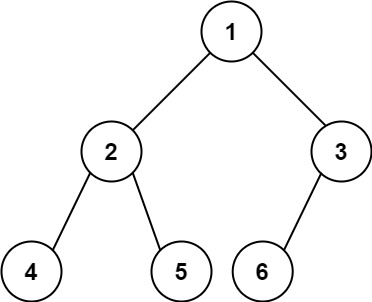
Given the root of a **complete** binary tree, return the number of the nodes in the tree.

According to [**Wikipedia**](http://en.wikipedia.org/wiki/Binary_tree#Types_of_binary_trees), 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 2h nodes inclusive at the last level h.

Design an algorithm that runs in less than O(n) time complexity.

**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

Solution:

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public int countNodes(TreeNode root) {

if(root == null)

return 0 ;

int lHeight = countLeftheight(root);

int rHeight = countRightHeight(root);

if(lHeight == rHeight)

return ((2<<lHeight)-1);

else

return countNodes(root.left)+countNodes(root.right)+1;

}

public int countLeftheight(TreeNode root){

int count =0;

while(root.left != null){

count++;

root = root.left;

}

return count;

}

public int countRightHeight(TreeNode root){

int count =0;

while(root.right != null){

count++;

root = root.right;

}

return count;

}

}

T.C = (logN)2

S.C = logN