222. Count Complete Tree Nodes

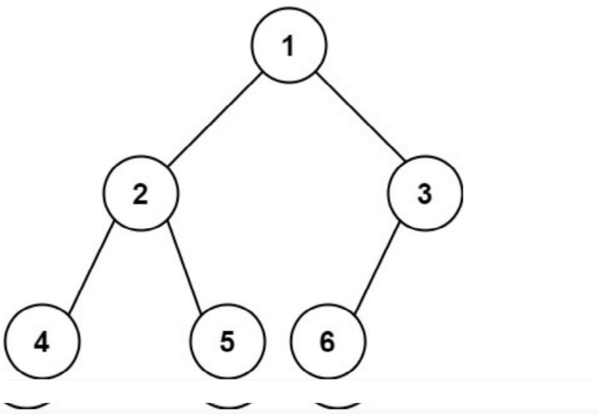
30 March 2022 05:41 PM

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`.

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

Example 1:



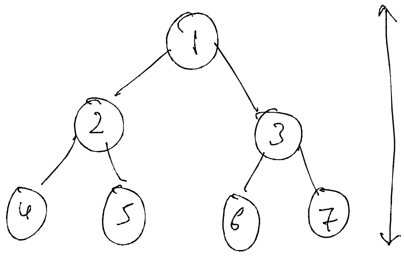
```
Input: root = [1,2,3,4,5,6]
Output: 6
```

Example 2:

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

Example 3:

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



S.C = worst case O(log n)

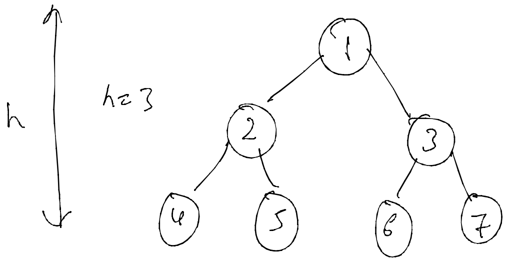
Brute force

Whenever you see count node you can do in-order, pre-order, post-order and level order.

T.C ⇒ O(n) S.C ⇒ O(h)

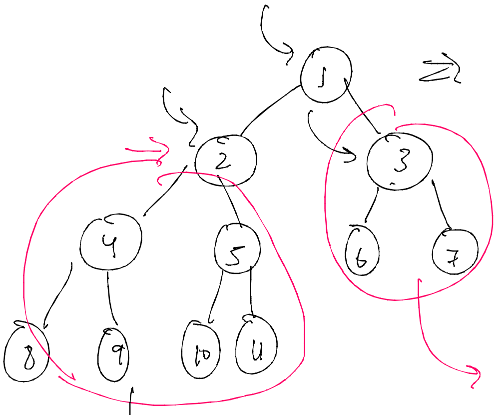
```
inorder (node, & cnt) {

    if (root == null)
        return

    cnt ++

    inorder (node -> left)
    inorder (node -> right)
}
```

}



no of nodes $= 2^3 - 1 = 2^h - 1$

$= 7$



$\Rightarrow 1 + 7 + 3 = 11$ nodes

$2^3 - 1$

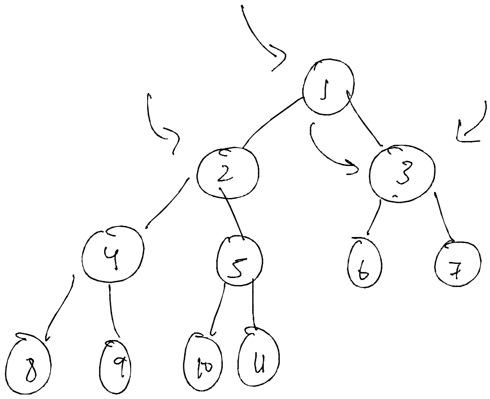
$= 7$

$2^2 - 1$

$= 3$

* you can check for every subtree
* you directly apply the formula

DRY RUN



lh = 4 rh = 3

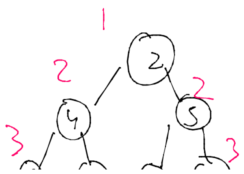
if you stand at 1 node the lh ≠ rh

So you cannot apply the formula

so the ans ll be 1.

$\Rightarrow 1 + \underbrace{(7)}_{lh} + \underbrace{(3)}_{rh} = 11$

now left sub tree 2

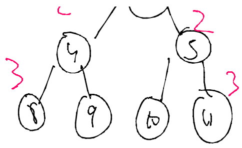


lh=3 rh=3

lh == rh

So the formula is

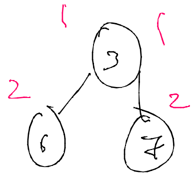
now right subtree 3



So the formula is

$2^3 - 1 = 7$

$lh = 7$



$lh == rh$ | $2^2 - 1 = 3$

$1 + 7 + 3 = 11$

Self Notes:
1. Formula is (2^TreeLevel - 1). Only works for perfect tree.
2. To determine if its a perfect tree, go all the way down and count the nodes on left and right side, If they match, its a perfect tree and our formula applies.
3. If its not a perfect tree, we go on left and right subtree and again determine if they are perfect tree.
4. When we have our left and right heights, we do (1 + left + right)
5. If we reach null, return 0
6. C++ note: 1 << n is the same as raising 2 to the power n, or 2^n

```java
class Solution {
    public int countNodes(TreeNode root) {
        if(root == null)
            return 0;

        int left = findHeightLeft(root);
        int right = findHeightRight(root);

        // if left and right are equal it means the tree is complete and hence use the formula 2^h - 1
        if(left == right)
            return ((2<<(left)) - 1);
         //else recursively calculate the number of nodes in left and right and add 1 for root.
        else return countNodes(root.left) + countNodes(root.right) + 1;
    }

    public int findHeightLeft(TreeNode root){
        int count = 0;
        while(root.left != null){
            count++;
            root = root.left;
        }
        return count;
    }

     public int findHeightRight(TreeNode root){
         int count = 0;
         while(root.right != null){
             count++;
             root = root.right;
         }
         return count;
     }
}
```

T.C => $O((\log n)^2)$

S.C => $O(\log h)$