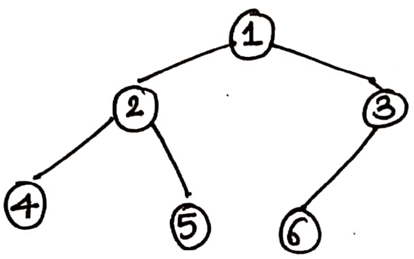
#

Count Total Nodes In A Complete Binary Tree

$O(\log_2 N)$?

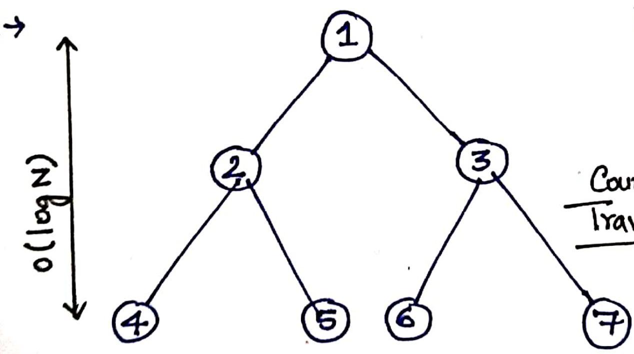
- Given the root of a complete Binary Tree, return the number of the nodes in the tree. Every level, except possibly the last level, ~~The les~~ completly filled, in a complete Binary Tree, and all nodes in the last level are as far ~~as~~ left as possible.
  It can have between 1 and 2^h nodes inclusive at the last level h.

Design an algorithm that runs in less than $O(n)$ Time Complexity.



For counting nodes we can do either INORDER, PREORDER, POSTORDER.

For Example :→




```
inorder (node, &cnt)
{
  if (root == null)
      return;
  cnt++;
  inorder(node->left);
  inorder(node->right);
}
```

Count Traversal

$\boxed{\begin{aligned} T.C &= O(N) \\ A.S.C &= O(h) \end{aligned}}$

No. of nodes $= 2^3 - 1 = 7$

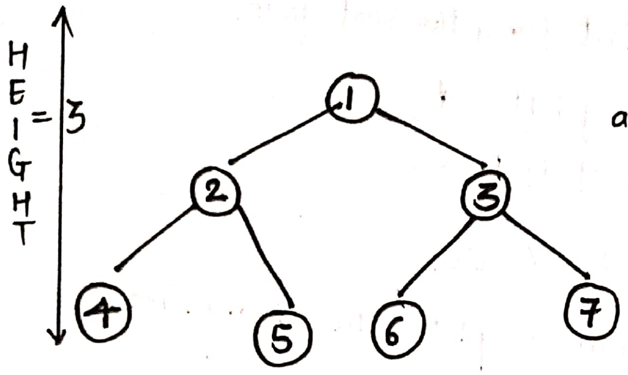
It is the complexity of Brute Force.

TIME COMPLEXITY
↳ $O(N)$
∴ Because Traversing for every node.

SPACE COMPLEXITY
↳ $O(H)$

where H is the height of the Binary Tree. → (Since it is a complete Tree, the Height of the Binary Tree - $O(\log N)$)

N = No. of Nodes in a Tree.



* It is completly fill, as none of the level, has any Node, shortage. →

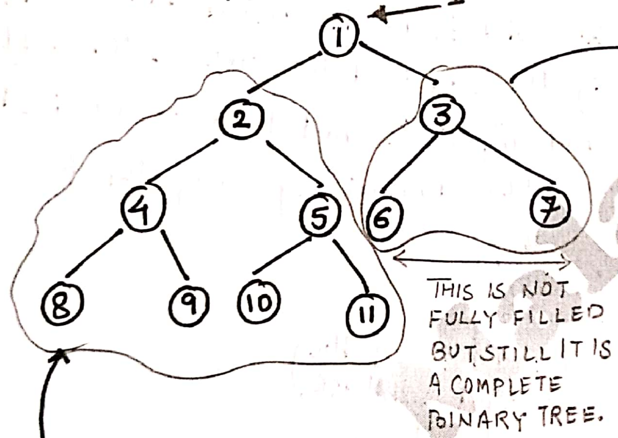
- No. of Nodes $= 2^3 - 1 = 7$.

  So, if we some how compute the height of the Tree.

  But there may be case where the Binary Tree may not be complete.

For exomple :→

ROOT - height is 1 +



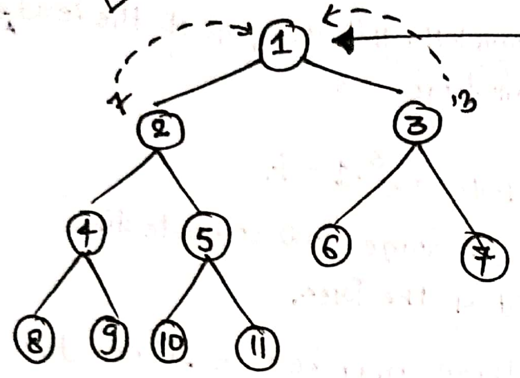
Height of the SubTree is 2
No. of Nodes $= 2^2 - 1 = 4 - 1 = 3$.

Now let from the Node 2 → height of the SubTree is 3
So no. of node in the Subtree is $2^3 - 1 = 7$.

* $\boxed{\text{So, Total NUMBER OF NODES} = 1\text{(Root Node)} + 7 \text{ Nodes in Left SubTree} + 3 \text{ Nodes in Right SubTree.} \quad 11}$

So CHECK FOR EVERY SUB-TREE,

$\boxed{7+3+1=11}$



if we start from the root node.
height of left = 4.
height of right = 3.
So, lh ≠ rh.

∴ we can complete the Binary Tree, traversal.

1 + (Traverse on the left) + (Traverse on the right)

We will do the Recursion and check, if the height of the tree on the left and right is equal than we can directly do the analysis by using the formula 2^n-1.

1 + (Traverse on the left Sub Tree) + (Traverse on the right Sub Tree)

↓ lh=3, rh=3
↓ This Sub tree is indeed the complete tree.
return $(2^3-1) = 7$
↑ no. of nodes.
Thus, no need of Traverse anymore

↓ lh=2, rh=2
↓ This Sub tree is indeed the complete tree.
return $(2^2-1) = 3$.
↑ no. of nodes.
Thus, no need of Traverse any more.

Data Structure and Algorithm
By
Malay Tripathi

TREES.

C++ code.

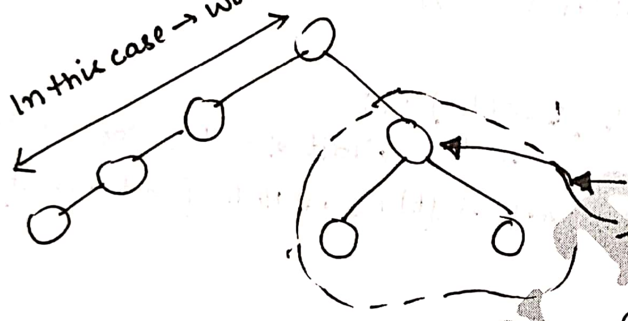
```
class Solution {
public:
   int countNodes(TreeNode* root) {
    if(root == NULL) return 0;

    int lh = findHeightLeft(root);
    int rh = findHeightRight(root);

    if (lh==rh) return (1<<lh) -1;

    return 1 + countNodes(root->left) + countNodes(root->right);

   }

  int findHeightLeft(TreeNode* node) {
     int hght=0;
     while (node) {
        hght++;
         node = node->left;
     }
     return hight;
  }
  int findHeightRight(TreeNode* node) {
       int hght=0;
       while(node){
         hght++;
          node = node->right;
       }
       return hght;
     }
   };
```

$\underbrace{1<<lh}$ → Bit wise operator → return 2^{lh}.

Data Structure and Algorithm
By
Malay Tripathi

$\boxed{T.C = (\log n)^2}$

In the Worst case → you end up using O(logN) is complete SubTree.
height of tree even in worst case O(log N), at max you traverse for logN nodes,



$$T.C = \underbrace{\log N}_{\text{for Traversing}} \times \underbrace{\log N}_{\text{for finding Height.}}$$

S.C = No external Space used,
except the Recursive Call of the Auxillary Space.

O(log N) → because it is the height of the Tree.