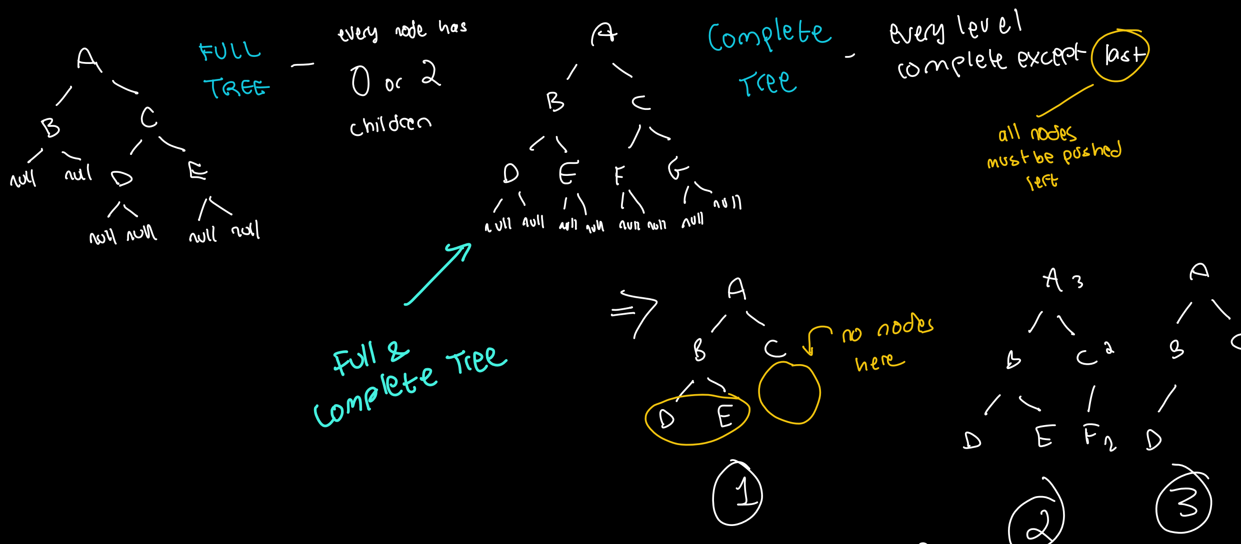
Problem: Given complete binary tree, count # of nodes



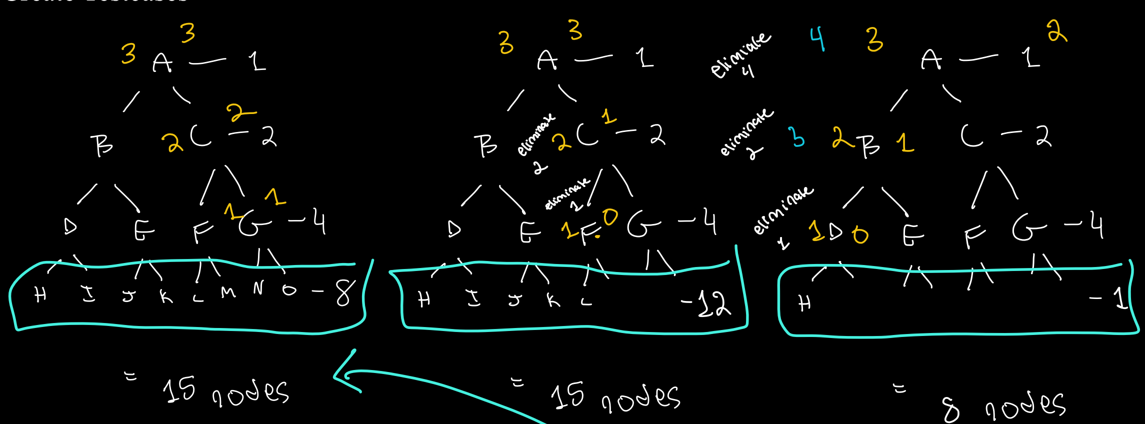
main function
2nd function
max # of nodes = $2^h - 1$

1 function
keeps traversing left to return height of tree
height - 1

3rd function
if no right node
max # of nodes -2

Intro:

- Verify Constraints
- Create Testcases



BFS or DFS T: O(n) S: O(n) ← Brute Force

Hint: leverage tree is complete for more optimal sol'ns

height = 4

# of nodes: $2^4 - 1 = 15$

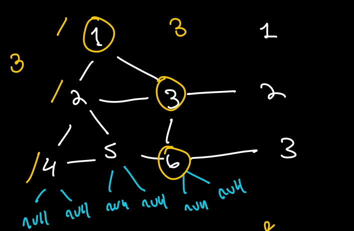
at least: $2^3 - 1 = 7$

$2^h - 1 = n$

⇒ If we know the HEIGHT

we know there are @ $\boxed{\text{least } 2^{h-1}}$ nodes

⇒ How do we know the # of nodes in the last level?



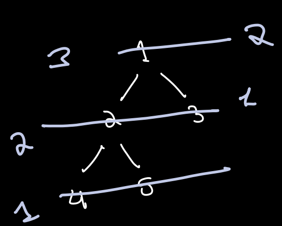
keep track of where it's going

taking the height of the tree

[1,3,6]

Post-Order Traversal

root
right
left



3 2

return root.left

# of nodes in full + complete @ last level

$= n/2$

$15/2 = 8$

7.5

$2^h = n+1$

$h = \log_2(n+1)$

from method

$\text{round}\left(\frac{2^{\text{height}+1} - 1}{2}\right)$

# of nodes in bottom level

# of nodes = 8

anytime go left

⇒ # of nodes/2

sum of eliminated nodes

if go down left node

sum += # of nodes − $\frac{\text{\# of nodes}}{2}$

every level

$\boxed{\text{\# of nodes} = \text{\# of nodes}/2}$

return sum

<u>Brute Force:</u>

- Brainstorming & Pattern Observations
- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity
  - Time: O(n)
  - Space: O(n)

```
def findMaxHeightRoot(self, root, list)

    if root.right == None and root.left == None

        return 1
    leftEmpty = [], rightEmpty = []
    left = 1+ self.findMaxHeight(root.left, leftEmpty)
    right = 1 + self.findMaxHeight(root.right, rightEmpty)

    if left > right:
        list.append(leftEmpty)

    else:
        list.append(rightEmpty)
```

<u>Optimal:</u>

- Brainstorming & Pattern Observations
- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity
  - Time: O(log2n) (o of log squared n)
  - Space: O(1)

Pseudocode:

base

\# of nodes : $2^{height-1} - 1$

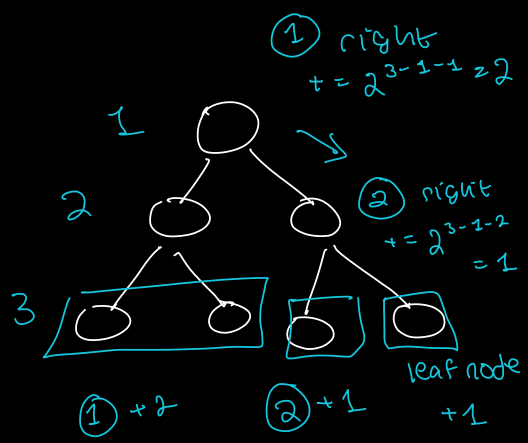
add to base

\# of nodes

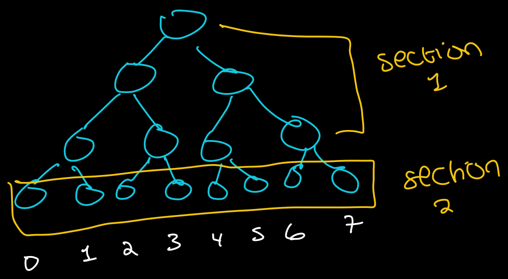
left: don't add

right: add $2^{height - 1 - currentHeight}$

leaf node : +1



$\boxed{\text{Udemy Notes}}$ – not traversal, related to property $\boxed{\text{COMPLETE}}$



→ reach optimal solution in $O(\log n)$ or $O(1)$ b/c

BFS
DFS : $O(n)$

$2^0 + 2^1 + 2^2 + 2^3 = 15$ nodes

$1 + 2 + 4 + 8$

$2^{h-1} - 1 : O(1)$

Time to get height of tree $+$: $O(h) = O(\log n)$

$= O(\log n)$

section 1

min: 1

max: 2^{h-1}

↗ nodes in last level

① determine rightmost node

② determine steps to get to rightmost node

section 2

last level size $= n/2$

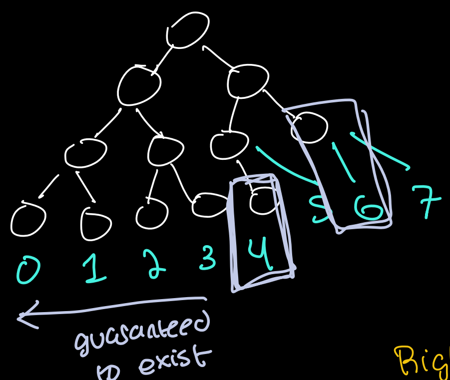
$\Rightarrow O(n) \neq O(\log n)$

X

Binary Search on sorted array

$\Rightarrow$ have at last level

$[0, \ldots, 2^{h-1} - 1]$



Left	Right
0	7

if value exists : left = mid

else : right = mid-1

mid : $(7+0)/2 = 3.5 \Rightarrow$ round up $= 4$

why? inclusive

Left	Right
4	7

mid: $(4+7)/2 = 6$ — node does not exist

right : 6-1 = 5

Left	Right
4	5

mid: $(4+5) = 5$ — node dne

right = 4

$\Rightarrow$ left == right

$\Rightarrow$ 4 rightmost value

How to get to index 4?

mid

4 < 4 false

L: 0
R: 7
$\boxed{mid = 4}$

on right side

traverse right

L: 4
R: 7
$\boxed{mid = 6}$

4 < 6 true

go left

4 < 5 true

L: 4
R: 5
$\boxed{mid = 5}$

go left

$\Rightarrow$ run $O(n) = O(\log n)$ runs 4 times

time