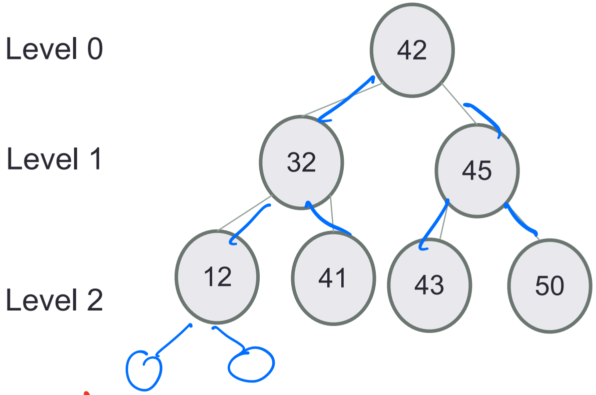
BST RUNNING TIME ANALYSIS

Problem Solving with Computers-II

Types of BSTs

n: is the number of nodes in a BST
H : is the height of the BST



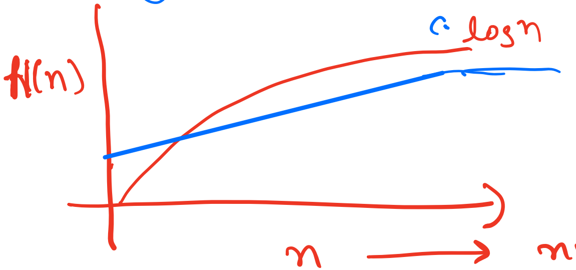
$H(n) = O(\log n)$

Balanced BST:

AVL, Red-Black Trees, Full BST

Full Binary Tree: Every node other than the leaves has two children.

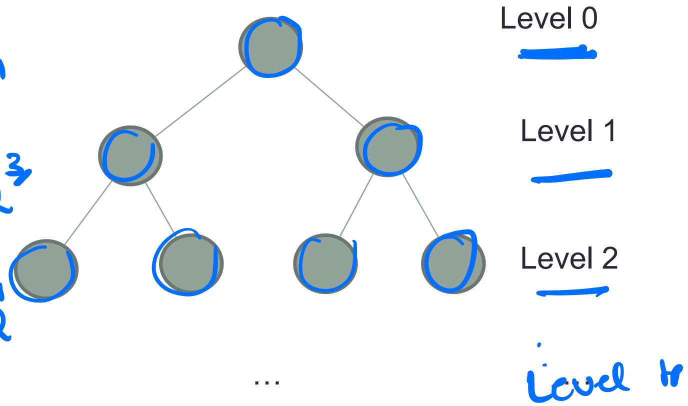
Complete Binary Tree: Every level, except possibly the last, is completely filled, and all nodes are as far left as possible



→ number of nodes

Relating H (height) and n (#nodes) for a full binary tree

Height	n
0	$1 = 2^{0+1} - 1$
1	$1+2 = 2^{1+1} - 1$
2	$1+2+2^2$
3	$1+2+2^2+2^3$
⋮	
H	$1+2+2^2+2^3 \dots 2^H$



Claim: $1+2+2^2+\dots 2^H = 2^{H+1} - 1$

$$n = 2^{H(n)+1} - 1$$

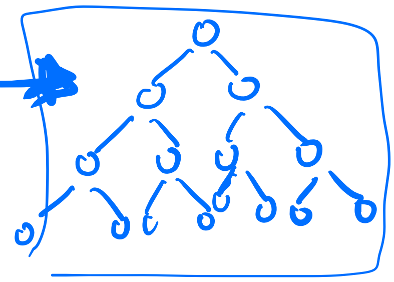
$$n+1 = 2^{H(n)+1}$$

$$\log_2(n+1) = H(n) + 1$$

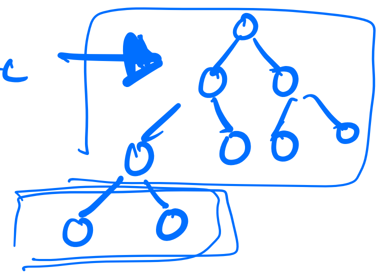
Therefore $H(n) = \log_2(n+1) - 1$, $H(n) = O(\log_2 n)$

Height a tree with n nodes

$$\boxed{H(n) = O(\log n)}$$

Full 

$$H(n) = O(\quad)$$

Complete 

$$H_c(n) = \log_2(n+1) - 1 \pm 1$$

$$= O(\log_2(n))$$

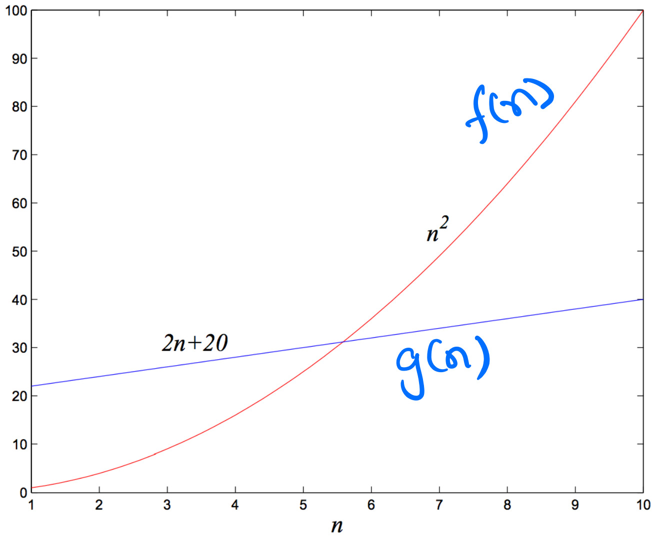
Big-Omega

- f(n) and g(n) map positive integer inputs to positive reals.

We say $f = \Omega(g)$ if there are constants $c > 0$, $k > 0$ such that
$c \cdot g(n) \leq f(n)$ for $n >= k$

$f = \Omega(g)$
means that "f grows at least as fast as g"

$$f(n) = \Omega(g(n))$$



Big-Theta

Θ ← In Practice Big-Oh is really asking Big Theta

- f(n) and g(n) map positive integer inputs to positive reals.

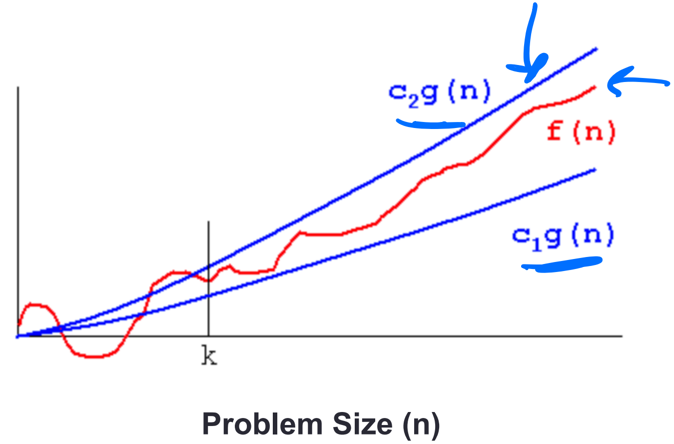
We say $f = \Theta(g)$ if there are constants c_1, c_2, k such that

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for $n >= k$

$f(n) = \Theta(g(n))$

$f(n) = O(g(n))$

$f(n) = \Omega(g(n))$



Big-O analysis of iterative fibonacci

```
function F(n){
 Create an array fib[1..n]
 fib[1] = 1
 fib[2] = 1
 for i = 3 to n:
     fib[i] = fib[i-1] + fib[i-2]
 return fib[n]
}
```

$O(1)$

$n-3$

$\rightarrow O(1)$

$T(n)$ is the running time of calculating $F(n)$

$T(n) = O(1) + (n-3) \cdot O(1)$

$= O(n)$

Big-O analysis of recursive fibonacci

What takes so long? Let's unravel the recursion…

```
function F(n){
    if(n == 1) return 1
    if(n == 2) return 1
return F(n-1) + F(n-2)
}
```

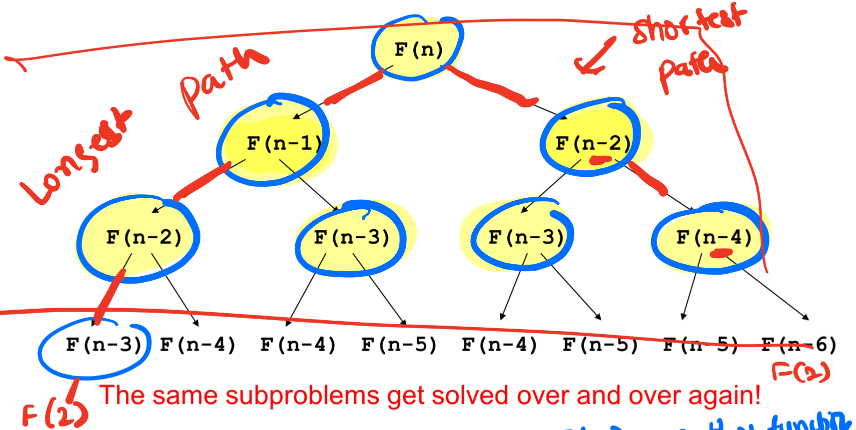
$T(1) = 2$

$T(2) = 3$

$n > 2$

$T(n) = 1 + 1 + 2 + 1 + T(n-1) + T(n-2)$

$= T(n-1) + T(n-2) + 5$



The same subproblems get solved over and over again!

$T(n) = c$ # of function calls

Length of the shortest path $\simeq n/2$

Length of the longest path $= n-2$

min number of function calls

$= 2^{n/2+1} - 1$

max number of function calls

$= 2^{n+1} - 1$

$T(n) = \Omega(2^{n/2})$ ← exp.

$T(n) = O(2^n)$ ← exp.

$T(n) = \Theta(2^{0.69n})$

Balanced trees

- Balanced trees by definition have a height of O(log N)
- A completely filled tree is one example of a balanced tree
- Other Balanced BSTs include AVL trees, red black trees and so on
- Visualize operations on an AVL tree: https://visualgo.net/bn/bst

Summary of operations

Operation	Sorted Array	Binary Search Tree	Linked List
Min			
Max			
Median			
Successor			
Predecessor			
Search			
Insert			
Delete			