CS1800

Day 21

Admin:

- exam2, hw6 & hw7:results week we get back

Content:

- function growth
- big-o, big-theta, big-omega notation

In Class Activity

Which gift will produce more value in one's lifetime?

- a magic penny which doubles it value every 3 years
- $10 a day

1. write first impressions (before computing) what do you think?
2. explicitly label your assumptions
3. compute & explain

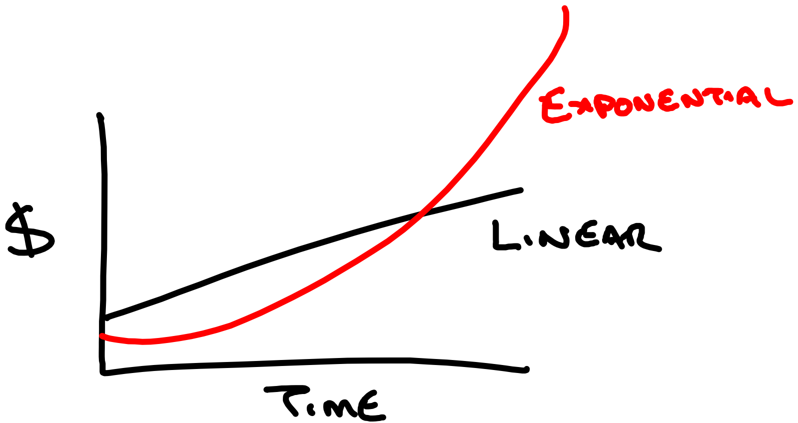
In Class Activity:

Which gift will produce more value over an infinite amount of time?
- a magic penny which doubles its value every 100000 years
- $1000000000000000 a second

1. write first impressions (before computing) what do you think?
2. explicitly label your assumptions
3. explain (maybe don't compute ...)

Punchline: some functions grow faster than others

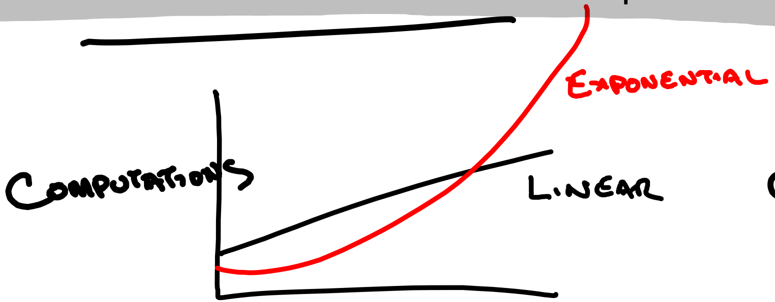
"doubling" (exponential) is eventually larger than "constant" (linear) growth
- no matter how small initial value of doubling is
- no matter how large initial value of linear growth is
- no matter how often the doubling occurs
- no matter how steep the linear growth occurs



Why do we care that some functions grow faster than others?

Suppose we have two algorithms (i.e. computer programs) which accomplish the same task on an input of size n.

Algorithm 1 takes 2^n computations (exponential)
Algorithm 2 takes 99999999 + 99999999n computations (linear)



$$9+9n = O(2^n)$$

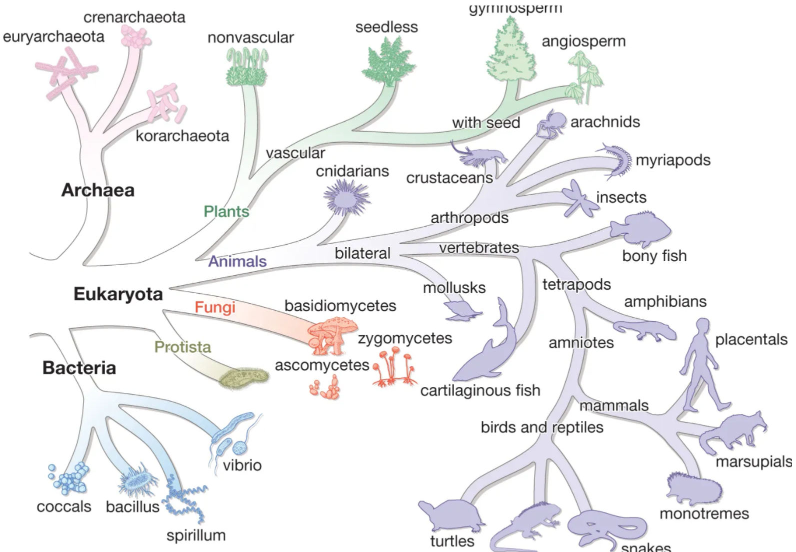
Our previous example shows that for sufficiently large input size (n), algorithm 2 will take fewer computations

(intuition from previous example: exponential functions grows faster than linear)

Objective:

Create a taxonomy of functions which allows us to organize them based on how quickly they grow.

Taxonomy (organization) of life:



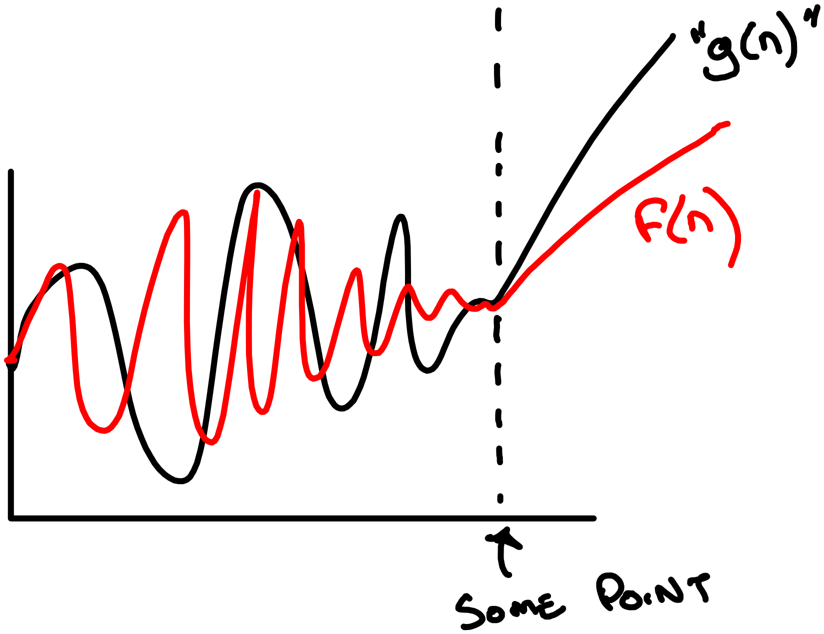
Big-O Notation (First Intuition): Big-O notation is kind of like "less than"

$f(n) = O(g(n))$ IS KIND OF LIKE "$f(n) < g(n)$"

PRONOUNCED "BIG OH OF G OF N"

g GROWS FASTER THAN f

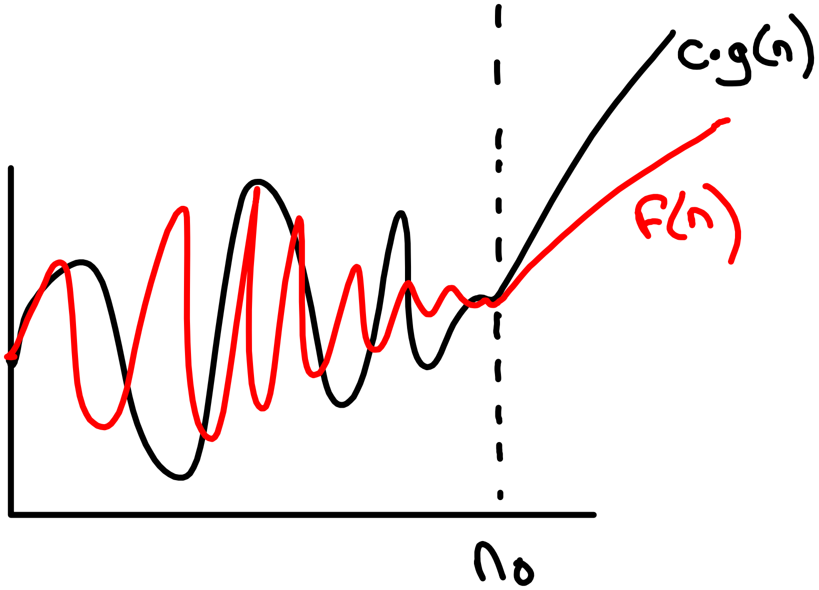
Big-O Notation (Intution): f(n) = O(g(n)) means g(n) grows faster than f(n)



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

Means "g(n)" is always larger than f(n) beyond some point

Big-O Notation (Intution): f(n) = O(g(n)) means g(n) grows faster than f(n)



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

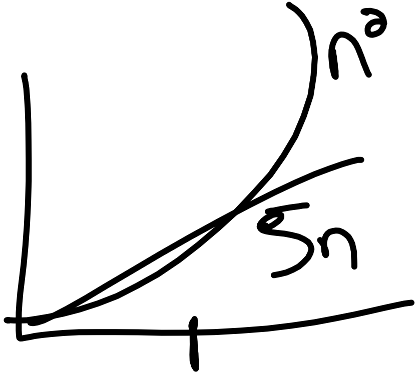
MEANS THERE EXISTS VALUES n_0 AND c WITH

$$n_0 \le n \rightarrow 0 \le f(n) \le c \cdot g(n)$$

Big-O Notation: Showing that one function is big-O (bounded above) by another

How do we show f(n) = O(g(n))? Choose n_0 and c to satisfy the definition

Example: Show that 5n = O(n^2)



$n_0 = 2$
$c = 5$

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

MEANS

THERE EXISTS VALUES

n_0 AND c WITH

$2 \leq n \rightarrow 0 \leq 5n \leq 5n^2$

Proving Big-O notation: FAQ

Aren't there many choices for n_0 and c?

There are!

So why do you choose these particular ones?

Remember, our purpose in writing a proof is to be compelling.
For this reason, choose the n_0 and c which are as simple as possible.

How will I know if my values are the simplest? Will credit be taken if I don't get the absolute simplest values?

There are many n_0, c pairs which are equally compelling. Avoid blindly choosing really large values (even if they "work" they're hard to understand)

In Class Activity: Proving Big-O relations

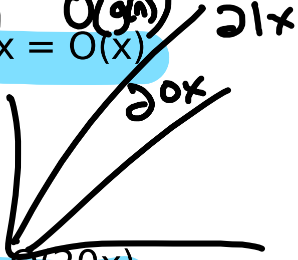
Prove each true statement below. If a statement is false, give a justification of why it is false (sketching a graph is often a good idea here).

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

20 x = O(x)

$c=21$

$n_0=0$



choose n_0 c

$$0 \le n \rightarrow 0 \le 20x \le 21 \cdot x$$

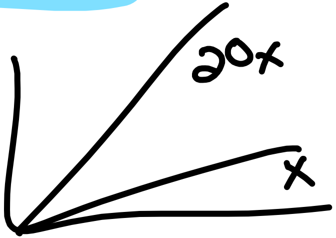
x^3 = O(x^2)

x = O(20x)

x^2 = O(x^3)

$c=1$

$n_0=0$

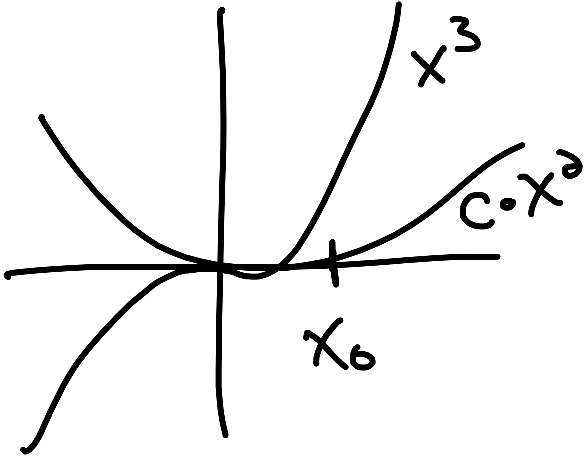


$$0 \le n \rightarrow 0 \le x \le 1 \cdot 20x$$

$x^3 = O(x^2)$

FALSE

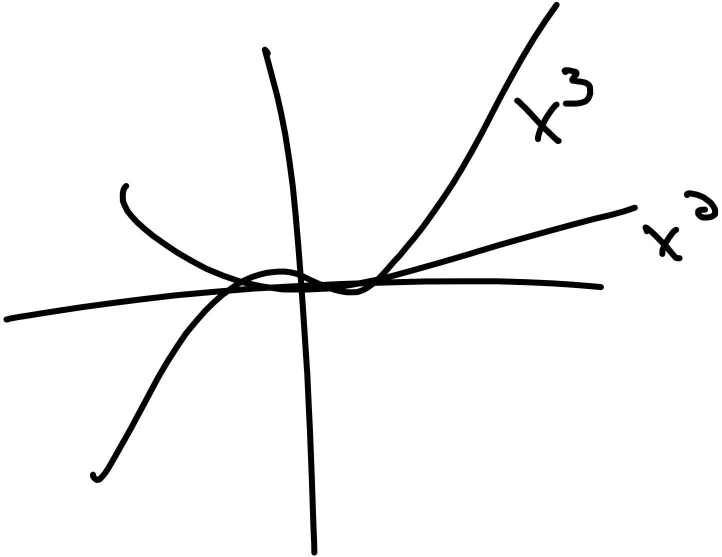
x^3 GROWS FASTER THAN x^2



$x^3 \leq c \cdot x^2$

$x \leq c$

$x^2 = O(x^3)$



$n_0 = 1$

$c = 1$

$1 \leq x \qquad 0 \leq x^2 \leq 1 \cdot x^3$

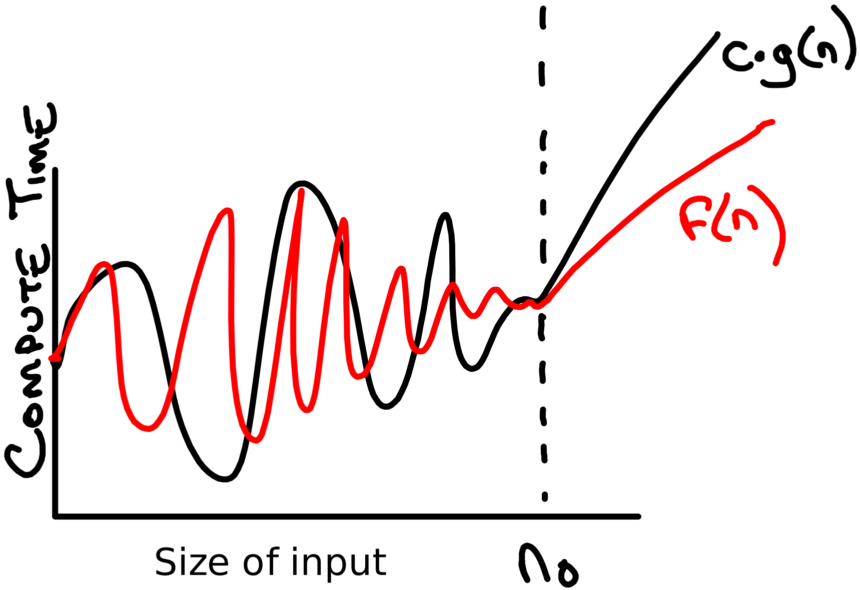
$$mx+b = O(x)$$

A FINAL INTUITION OF BIG-O

$f(n) = O(g(n))$ MEANS $g(n)$ GROWS AT LEAST AS QUICKLY AS $f(n)$

"$f(n) \leq g(n)$"

Critiquing the Big-O definition: Why do we only care about large n?



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

MEANS

THERE EXISTS VALUES

n_0 AND c WITH

$$n_0 \le n \rightarrow 0 \le f(n) \le c \cdot g(n)$$

In our context (n=input size, f(n) = compute time) we don't care about small n, they're easily computed anyways!

Critiquing the Big-O definition: why allow a multiplicative constant c?

FROM IN CLASS ACTIVITY

$20x = O(x)$ AND $x = O(20x)$

X AT LEAST AS FAST AS 20X

20X AT LEAST AS FAST AS X

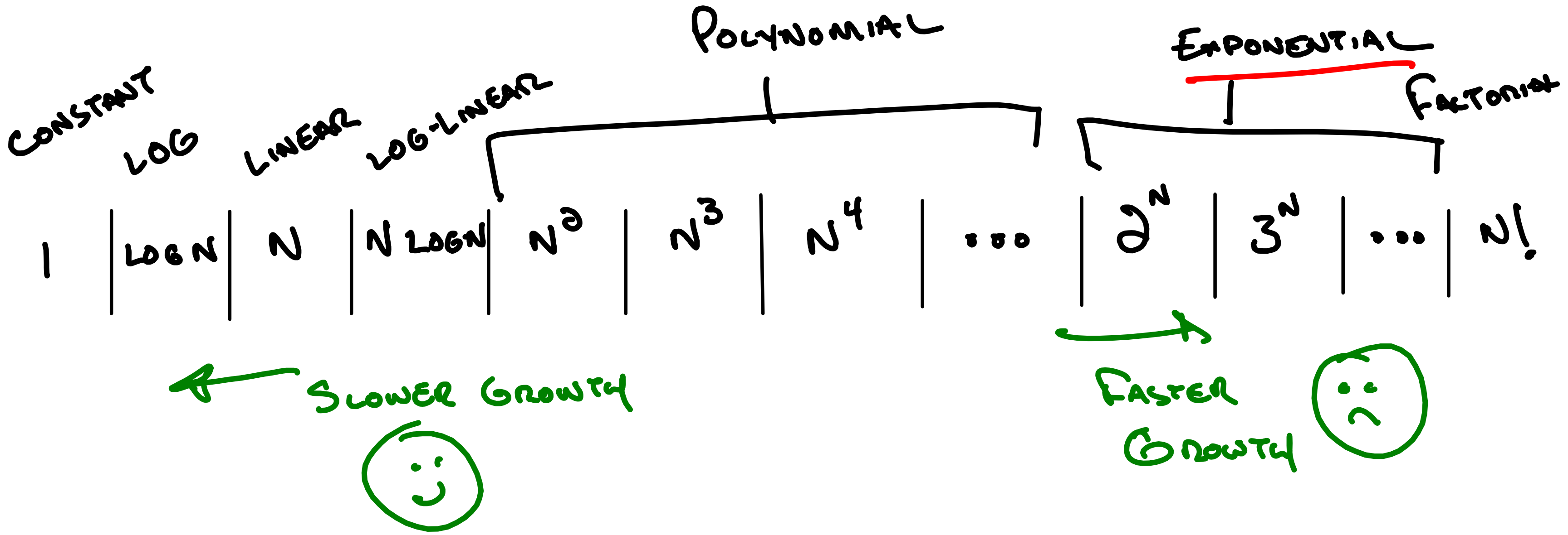
SO X AND 20X GROW EQUALLY QUICKLY

Inclusion of c allows a notion of functions which grow equally quickly.

Useful insight 1:
Ignore constant multipliers in a function when considering Big-O

Motivation:
Simplifies how we define function growth (there are many functions in the same "growth bucket", all grow equally quickly)

Function Growth Buckets:

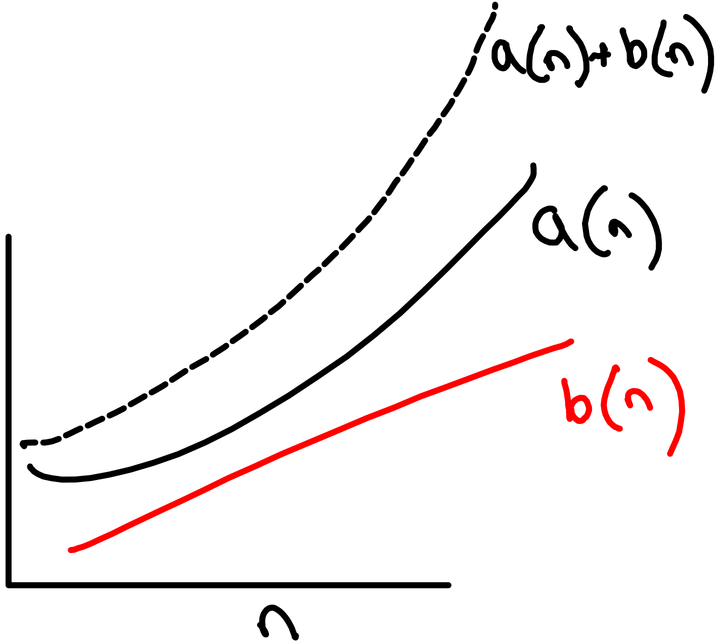


Function Growth: Why do we care again? (taken from Fell / Aslam's "Discrete Structures")

	n			
	10	50	100	1,000
$\lg n$	0.0003 sec	0.0006 sec	0.0007 sec	0.0010 sec
$n^{1/2}$	0.0003 sec	0.0007 sec	0.0010 sec	0.0032 sec
n	0.0010 sec	0.0050 sec	0.0100 sec	0.1000 sec
$n \lg n$	0.0033 sec	0.0282 sec	0.0664 sec	0.9966 sec
n^2	0.0100 sec	0.2500 sec	1.0000 sec	100.00 sec
n^3	0.1000 sec	12.500 sec	100.00 sec	1.1574 day
n^4	1.0000 sec	10.427 min	2.7778 hrs	3.1710 yrs
n^6	1.6667 min	18.102 day	3.1710 yrs	3171.0 cen
2^n	0.1024 sec	35.702 cen	4×10^{16} cen	1×10^{166} cen
$n!$	362.88 sec	1×10^{51} cen	3×10^{144} cen	1×10^{2554} cen

Table 14.1: Time required to process n items at a speed of 10,000 operations/sec using ten different algorithms. *Note:* The units above are seconds (sec), minutes (min), hours (hrs), days (day), years (yrs), and centuries (cen)!

Useful insight 2: When assessing functions growth, slower growing terms don't impact Big-O



IF a GROWS FASTER THAN b THEN a+b GROWS AS QUICKLY AS a

=

IF $b(n) = O(a(n))$

THEN $a(n) + b(n) = O(a(n))$

Quickly Assessing (but not proving) Function Growth:

1	LOG N	N	N LOG N	N^2	N^3	N^4	...	2^N	3^N	...	N!

Insight1: ignore constant multipliers

Insight2: discard slower growing terms

$$\cancel{1} + \cancel{\log_{10} N} + \cancel{14N} + \cancel{\pi N^3} + \cancel{.0001} \cdot 2^N = O(2^N)$$

$$\Omega(2^N) = \cancel{100} + \cancel{10^{10}} \cdot 2^N = O(2^N) = \Theta(2^N)$$

Quick In Class Activity:

1	$\log N$	N	$N \log N$	N^2	N^3	N^4	$\dots$	1.01^N	2^N	3^N	$\dots$	$N!$

Give the simplest, slowest growing function g(n) such that each f(n) = O(g(n))
(see previous slide)

$$f_1(n) = \cancel{2n} + \cancel{3}n^2 = O(n^2)$$

$$f_2(n) = \cancel{1234} + \cancel{N \log N} + \cancel{7} + \cancel{4} + \cancel{3} + \cancel{n^{1000}} + 1.01^n$$

$$= O(1.01^N)$$

Big-Omega is the opposite of Big-O

BIG O

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

THERE EXISTS VALUES n_0 AND c WITH

$$n_0 \leq n \rightarrow 0 \leq f(n) \leq c \cdot g(n)$$

$g(n)$ IS UPPER BOUND ON $f(n)$

BIG OMEGA

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

THERE EXISTS VALUES n_0 AND c WITH

$$n_0 \leq n \rightarrow 0 \leq c \cdot g(n) \leq f(n)$$

$g(n)$ IS LOWER BOUND ON $f(n)$

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

$\leq$ $\geq$

Big Theta: when two functions grow equally quickly

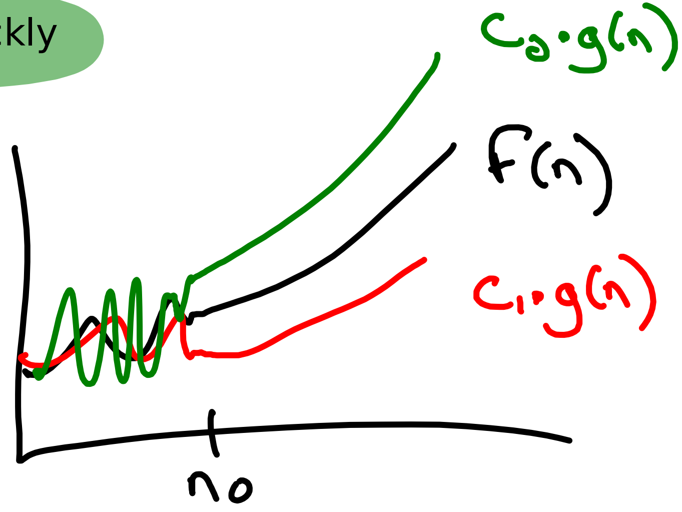
BIG THETA

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

THERE EXISTS c_1 c_2 n_0

WITH

$$n_0 \le n \rightarrow 0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$



BIG-O AND BIG OMEGA = BIG THETA

$$f(n) = O(\underline{g(n)})$$

AND

$$f(n) = \Omega(\underline{g(n)})$$

$$\longleftrightarrow$$

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

$$2^n + 3^n = O(3^n)$$

$$20x = \Theta(x)$$

$$x = O(20x)$$

$$\Omega(20x) = x$$