Professor Hamlin
Day 19

1) Admin
2) Review
3) Function growth
4) Big-O, Big-Θ, Big-Ω

Review

Summary of Arithmetic, Geometric & Quadratic
($k=0$)

	Arithmetic	Geometric	Quadratic
How to identify	2 4 6 8 10 … (+2 +2 +2 +2) Difference constant	1 2 4 8 16 … (×2 ×2 ×2 ×2) Constant ratio	1 3 7 13 21 (2 4 6 8) (+2 +2 +2) constant second difference
Expression of single term	$a_0 + dk$	$a_0 \cdot r^k$	$ak^2 + bk + c$
Computing Partial Sum	$\sum_{k=0}^{N} a_0 + dk = (a_0 + a_N)\left(\frac{N+1}{2}\right)$	$\sum_{k=0}^{N} a_0 r^k = \frac{a_0(1-r^{N+1})}{1-r}$	Calculus fun (not CS1800)

Exercise: 1) $\sum_{k=0}^{4} 3 + dk = 35$ - What is d?

$k=0$ to $4 \rightarrow 5$ terms

$3 + 0d + 3 + 1d + 3 + 2d + 3 + 3d + 3 + 4d = 35$

$15 + 10d = 35$

$d = 2$

Activity 1

Which gift would be worth more over 75 years?

1) a magic penny whose value doubles every 3 years
2) $10 a day

a) Write 1st impression before math
b) Compute and compare.

a) 10 $ a day?

b) $10 day

```
10*(75*365)
↳ = 273750
```

exponential penny

```
(2^(75/3))/100.
↳ = 335544.32
```

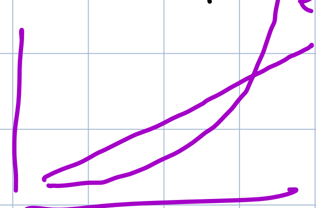
Activity 2

Which would produce more value over the life of the universe?

1) A magic penny whose value doubles every 100,000 yrs
2) $1,000,000,000,000 a second

a) what are your 1st impression pre-math
b) explain (maybe don't compute)

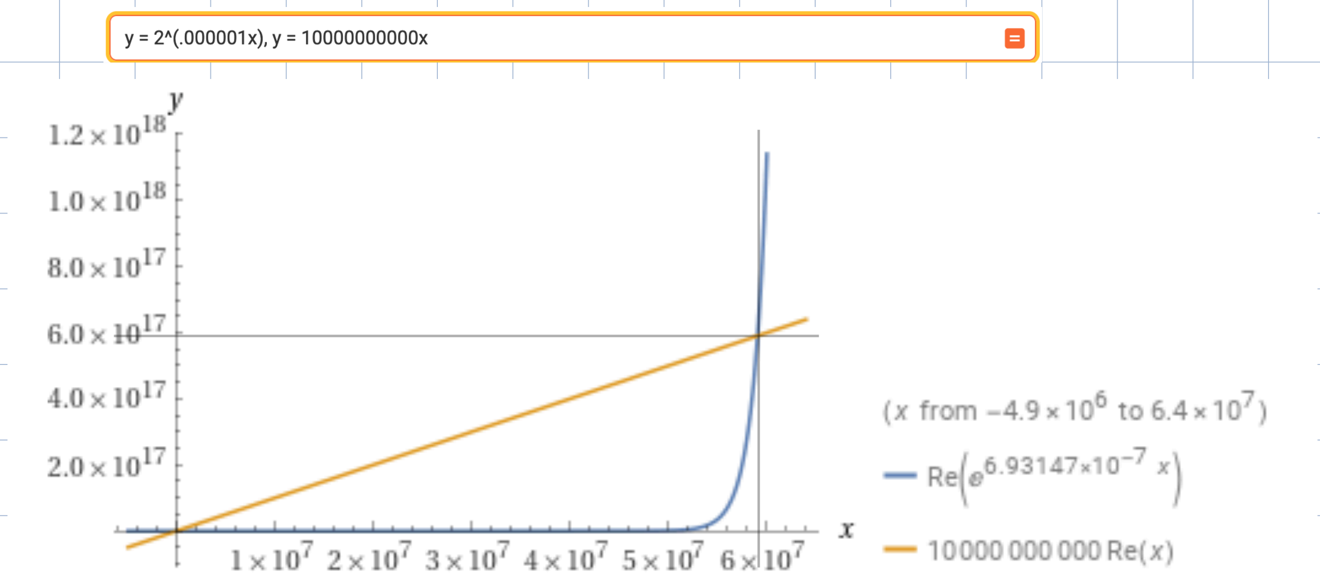
exponential ¢



Takeaway: Some functions grow faster than others

An exponential function will become larger than linear growth no matter:

- how small the initial value to double is
- how large the initial value for linear growth
- how often the doubling occurs
- how steep the linear growth is



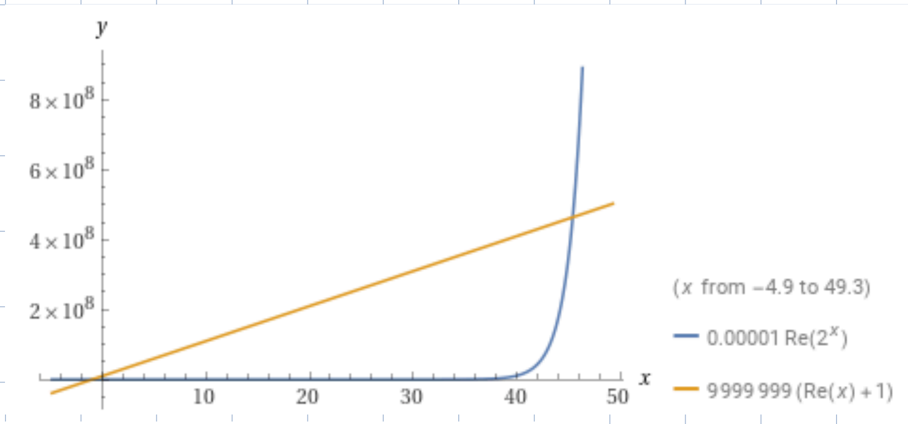
Why do we care about how fast a function grows?

Consider two different computer programs that accomplish the same thing. However, on input size n

Algorithm 1: takes $.00001 \cdot 2^n$ steps

Algorithm 2: takes $9999999 + 9999999n$

We know Algorithm 2 will at some point take fewer steps.



$x \approx 45.3992, \quad y \approx 4.63992 \times 10^8$

In fact at $n=46$, Algorithm 2 is slower....

Comparing how fast algorithms run

.... super slow?

on input of length n; how would we classify these algorithms?

n	$n + 100000000000$	n^n
n^2	$n^2 + \log n$	3^n
2^n	$n^{10000000}$	$n \log n$

We need a better way of putting our run-times in "buckets" so we can compare them

aka. a way to say a function $(f(n))$ is...

...."bigger than"...

...."smaller than"...

...."the same as"....

... another function $g(n)$

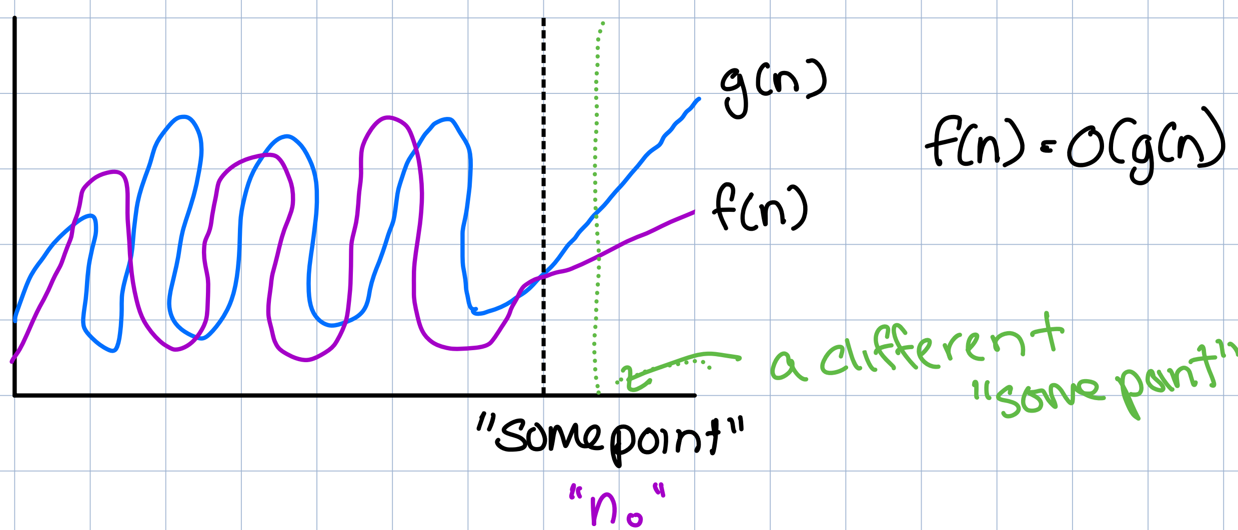
Big-O Notation

$f(n) = O(g(n))$ is similar to "$f(n) < g(n)$"

"Big-O of g of n"

g grows faster than f

It means that at a certain point $g(n)$ will be larger than $f(n)$ past some point



Formally, $\exists\ n_0, c \in \mathbb{N}$ s.t. $\forall n \geq n_0$;

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

"There exists two natural numbers n_0 & c s.t. for any value of n greater than n_0, if you evaluate $f(n)$ its result will be smaller than $c \cdot g(n)$"

Cool so we have our definition how do we use it?

e.g. $\quad 5n = O(n^2)$

we need to find values of c & n_0 s.t.

$\forall n,\ n_0 \leq n \qquad 0 \leq 5n \leq c \cdot n^2$

Let's start $n_0 = 1$, what does c need to be?

$n=1$	$f(n)$ $5n$	$g(n)=n^2$
	5	1

c needs to be 5

Does this work for $n \geq 1$? $n=3$

$5 \cdot 3 \leq 5 \cdot 3^2$ ✓

$c=5,\ n_0=1$

weak induction

$$\boxed{5n \leq 5n^2 \quad \forall n \geq 1}$$

Big-O FAQs

1) Aren't there infinitely many choices for c, n_0?

Yep!

could have $n_0 = 1$, $c = 5$

$n_0 = 10{,}000$ $c = 1$

$n_0 = 10{,}000$ $c = 10{,}000$

2) So why choose the ones we did?

We want our proof to convince other people than us. So $c = 100000000$ and $n_0 = 100000$ is hard to think about

3) How do I know what value to use? Will I lose points if I don't choose correctly?

There are many values of c/n_0 that work → try to choose values for c/n_0 that are < 20.

<u>Exercise</u> For true statements state c & n_0

For false statements, give justification (possibly a graph)

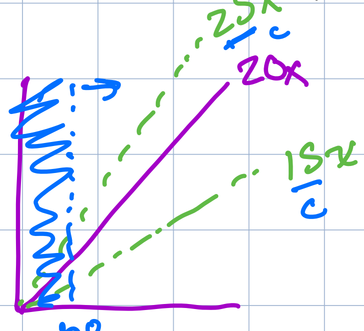
c allows us to control the "slope" of g(n)

1) $20x = O(x)$

$n_0 = 2$ $c = 30$

$\forall\ x \geq 2$

$0 \leq 20x \leq \underline{30}x$



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

$0 \leq x^3 \leq c\,x^2$

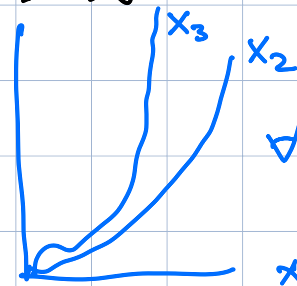
False

no value of c works

3) $x = O(20x)$

$n_0 = 1 \quad c = 1$

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



$n_0 = 2 \quad c = 1$

$\forall x > 1$

$0 \leq x^2 \leq x^3$

$x = 2 \quad 4 \leq 8$

$\boxed{f(n) = O(g(n)) : g(n) \text{ grows atleast as quickly as } f(n)}$

Why do we only care about really big values of n?

n = input size, f(n) is compute time.
This type of abstraction isn't really needed for small values of n, which can easily be computed.

What is up w/ this whole "c" thing?

Recall:

$20x = O(x)$ x is atleast as fast as 20x	$x = O(20x)$ 20x is atleast as fast as x

→ it seems x & 20x belong in the same "bucket" of linear growth

→ c captures idea of functions being mutually Big-O to each other

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

$f_1(n) = 1000n$ $\quad O(n)$ $\qquad f_2(n) = 9999999n$ $\quad O(n)$

Common "Buckets"

log $\quad$ log-linear $\quad$ exponential

$1 \mid \log N \mid N \mid N\log N \mid N^2 \mid N^3 \mid \dots \mid 2^N \mid 3^N \mid \dots \mid N! \mid N^N$

constant $\quad$ linear $\quad$ polynomial $\quad$ factorial

:) slower growth $\qquad$:(faster growth

A concrete comparison:

input size

run times

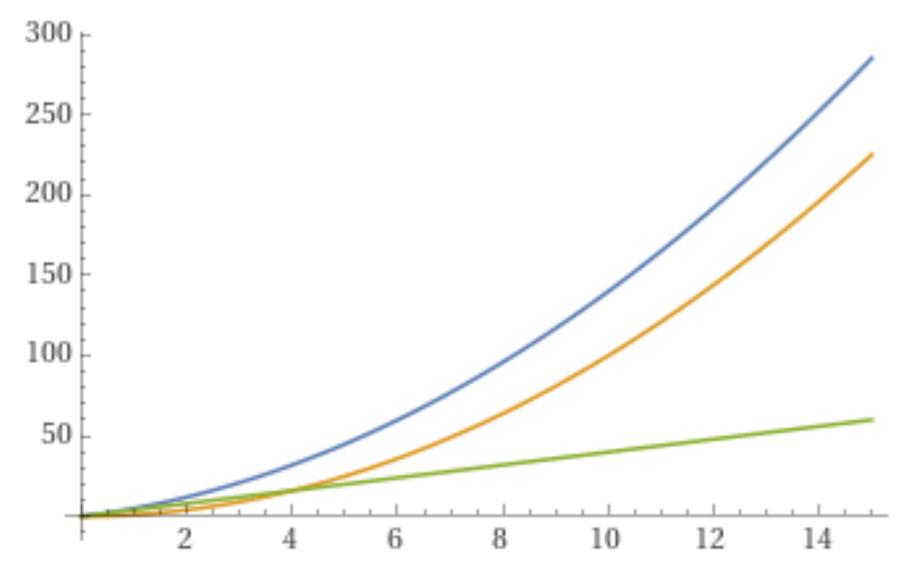
	n			
	10	50	100	1,000
$\lg n$	0.0003 sec	0.0006 sec	0.0007 sec	0.0010 sec
$n^{1/2}$	0.0003 sec	0.0007 sec	0.0010 sec	0.0032 sec
n	0.0010 sec	0.0050 sec	0.0100 sec	0.1000 sec
$n \lg n$	0.0033 sec	0.0282 sec	0.0664 sec	0.9966 sec
n^2	0.0100 sec	0.2500 sec	1.0000 sec	100.00 sec
n^3	0.1000 sec	12.500 sec	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

$O(n)$ vs $O(2^n)$

$100n$

Only the biggest function of n matters

$a(n) + b(n)$



$c(n) = n^2 + 4n$

$a(n) = n^2$

$b(n) = 4n$

<u>informal</u>: if $a(n)$ grows faster than $b(n)$ then $c(n) = a(n) + b(n)$ grows as quickly as $a(n)$

"largest power of n determines bucket"

<u>Formally</u>: if $b(n) = O(a(n))$ then $c(n) = O(a(n))$

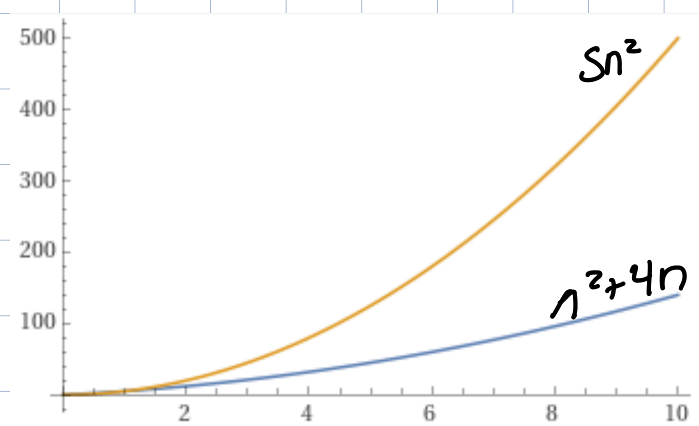
Formally showing it w/ n_0 & c: $c(n) = O(n^2)$

$c(n) = n^2 + 4n$ $\quad n_0 = 1$ $\quad c = 5$

$n = 1 \qquad 1^2 + 4 \cdot 1 \leq 5 \cdot 1^2$ ✓

$n = 2 \qquad 2^2 + 4 \cdot 2 \leq 5 \cdot 4$ ✓

$n = 3 \qquad 3^2 + 4 \cdot 3 < 5 \cdot 9$ ✓



Quickly assessing function growth

1	log N	N	N log N	N^2	N^3	....	2^N	3^N	...	$N!$	N^N

log

log-linear

exponential

constant

linear

polynomial

factorial

Insight 1: ignore constants
Insight 2: discard slower growing terms

$$f(n)\ 1 + \log_{10} n + 14n + \pi n^2 + .00001 \cdot 2^N$$

$$\boxed{O(2^N)}$$

Exercise Find simplest Big-O

1	log N	N	N log N	N^2	N^3	...	2^N	3^N	...	$N!$	N^N

log

log-linear

exponential

constant

linear

polynomial

factorial

1) $f_1(n) = 2n + 3n^2$

$O(n^2)$

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

$O(1.01^n)$

So far we wanted...

aka. a way to say a function ($f(n)$) is...
...."bigger than"....
...."smaller than"... Big-O
...."the same as"....
... another function $g(n)$

Now how we capture $f(n)$ is "bigger than" $g(n)$

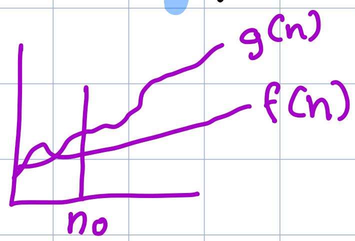
$\boxed{\text{Big-Omega}}$

Big-O

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

$\exists\ n_0, c \in \mathbb{N}$ s.t.
$\forall n \geq n_0 \rightarrow 0 \leq f(n) \leq c \cdot g(n)$

g(n) is an "upper bound" for f(n)



Big-Omega

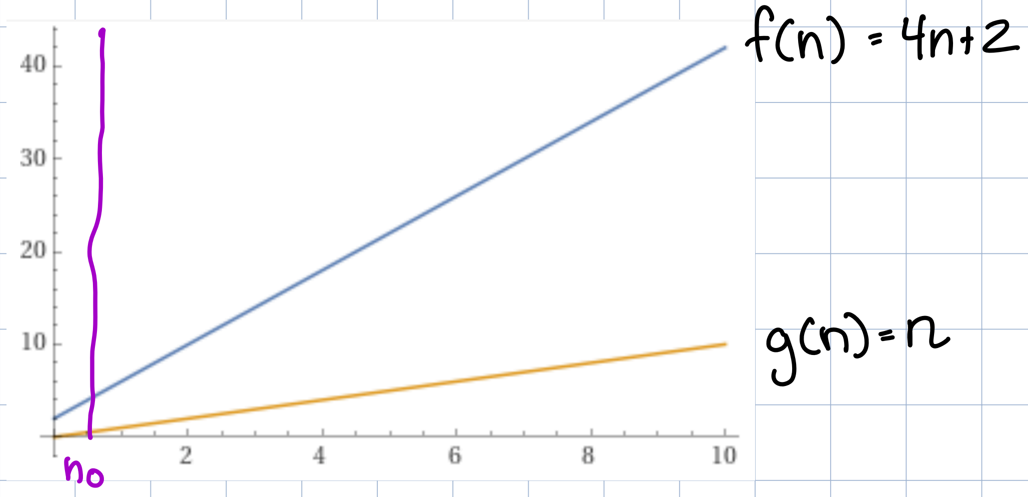
$f(n) = \Omega(g(n))$

$\exists\ n_0, c \in \mathbb{N}$ s.t.
$\forall n \geq n_0 \rightarrow 0 \leq c \cdot g(n) \leq f(n)$

g(n) is a "lower bound" for f(n)

Example! $f(n) = 4n + 2$ $\quad f(n) = \Omega(n)$

$n_0 = 1 \quad c = 1$



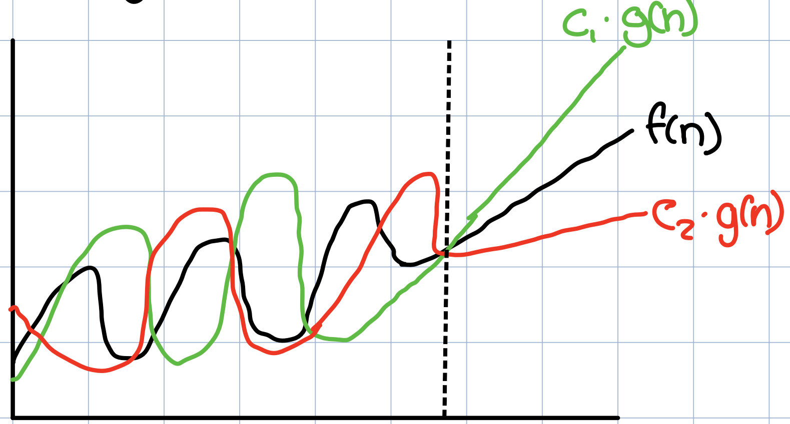
So far we wanted...

aka. a way to say a function $(f(n))$ is...
...."bigger than"... Big-Omega
...."smaller than"... Big-O
...."the same as".... Big-Theta
... another function $g(n)$

Big-Theta: two functions grow equally quickly

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

$\exists\ c_1, c_2, n_0 \in \mathbb{N}$ s.t
$\forall n > n_0$:
$0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$



Big-O and Big-Omega = Big-Theta

$$f(n) = O(g(n)) \text{ AND } f(n) = \Omega(g(n))$$

$$\Updownarrow$$

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

$x \le 1$ &
$x \ge 1 \Rightarrow$
$x = 1$

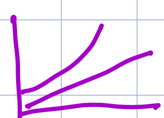
Example $f(n) = 5n^2 + 3n + 5$

$f(n) = O(n^2)$ $\qquad$ $f(n) = \Omega(n^2)$

then

$$f(n) = \Theta(n^2)$$

Example Are the following statements true or false

1) $n = O(n^3)$  T

2) $n^2 = \Omega(n)$ $\quad$ n "≤" n^2 $\quad$ T

3) $\boxed{10n} + \underbrace{\log n}_{O(n)} = \underbrace{O(.1n)}_{O(n)}$ $\quad$ T

4) $\underbrace{14 + n\log_2 n}_{n\log_2 n} = \underbrace{\Theta(15n\log_2 n)}_{n\log_2 n}$ $\quad$ T

5) $\log_2 n = \Theta(\log_{10} n)$ $\quad$ T

$\downarrow$

$(\log_2 n)$

hint: $\log_b x = \dfrac{\log_a x}{\log_a b}$