Asymptotic Analysis

The Theory

- the ability to reduce complicated algorithms to mathematical equations
- this gives us the ability to predict the speed and overall time an algorithm will take with any given input

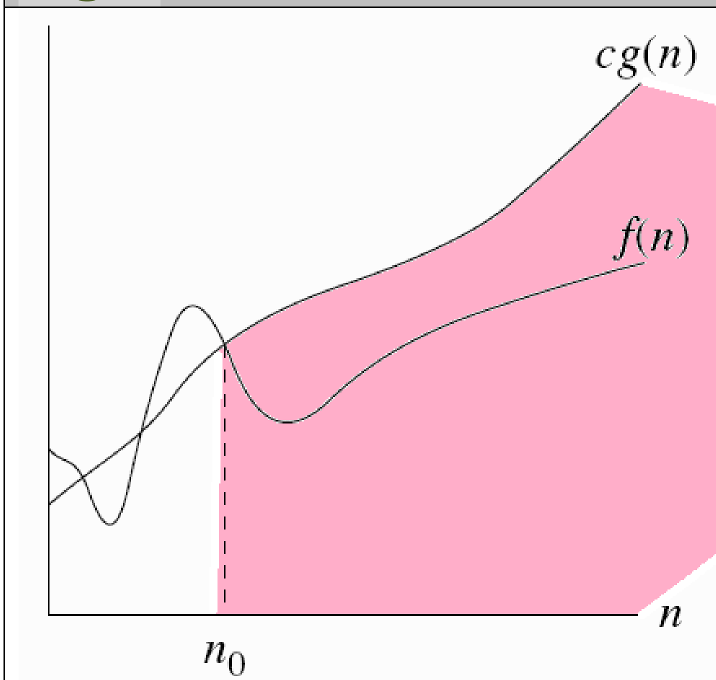
The Basics

- time and space are required to solve problems of given size "n"
- is the study of how algorithms behave as the size of data grows to infinity
  - so this really means SCALING!!
    - how does a function scale and react with limited data to lots of data
- O(n)
  - 'n' is the size of the data input
    - input could be number of bits, or commonly used number of items
    - has to be > 0
      - since you can't have NEGATIVE or HALF pieces of data
  - running time of the problem
    - this is a mathematical model
    - will not account for speed of computer, compiler, etc…
- f(n)
  - is a 'real world' function (or your coded application)
  - describes a process, running cost of a program
  - we watch f(n) grow as the number of "n" increases
- g(n) **or** c g(n)
  - another function (and graph line) that we are trying to mimic the same graph line as f(n), but will be above or below the f(n) line to determine worst/best case

Asymptotic Bounds

- bounds
  - think as ***bound***ary
  - given a graph line of f(n), we will find a line that tightly mimics that graph line, but depending on the bound, will be above, below, etc…
  - **tightly** also means that the graph line is close/parallels
- three types of bounds (covered later) always around f(n), but **called** c * g(n)
  - Big O (above f(n) graph line) upper bound of f(n)
    - so f(n) <= c * g(n) for all n > n_0
  - Big Ω (below f(n) graph line) lower bound of f(n)
  - Big Θ (below AND above f(n) graph line)
- used to relate the growth in one function relative to another simpler function

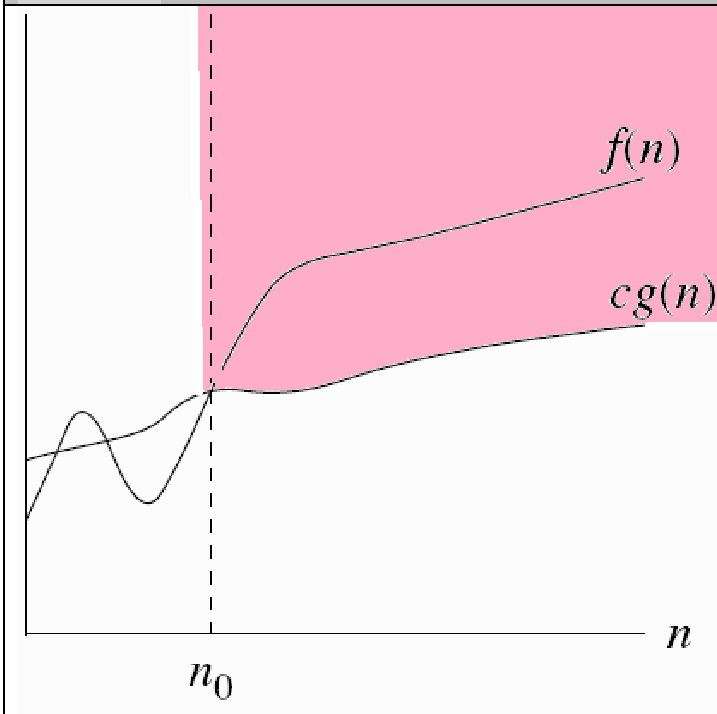
Big-O



f(n) <= c * g(n) for all n > n_0
notice where the n > n_0 comes in

- determines the ***upper bounds*** on a function
- O (Omicron) (mathematical term)
- we think of it as always representing the worst time, but not always
  - since ABOVE or UPPER bound of a given g(n)
- f(n) € O(g(n)) therefore f(n) <= c*g(n)
  - € = is in
  - YOU get to pick the constant to make the above equation work!
  - whenever n is big enough for ***some large c***
  - So how Big is Big?
    - "n" needs to be big enough to have the first function (g(n)) fit the 2nd
      - f(n) fits c*g(n)

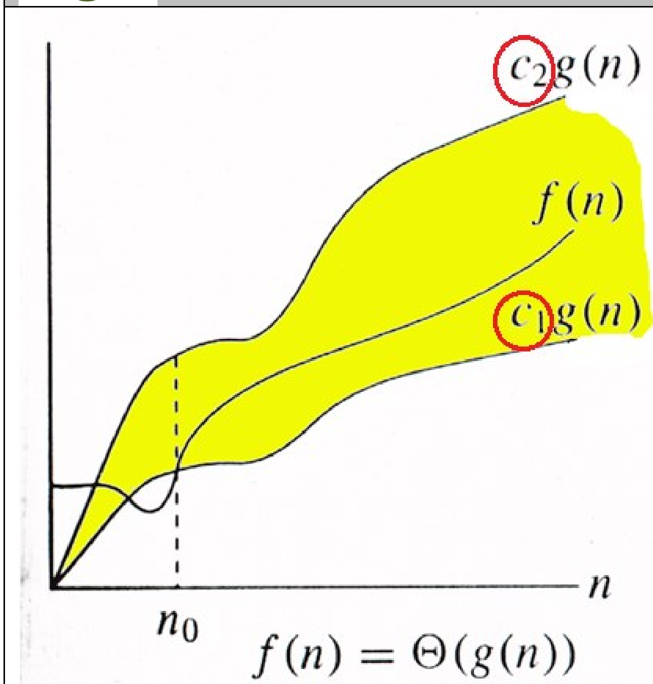
Big-Ω



$f(n) >= c * g(n)$ for all $n > n_0$

- opposite/reverse of Big O

Big Θ



$c_1 * g(n) >=$ f(n) $>= c_2 * g(n)$ for all $n > n_0$

Notice how g(n) is the same for both upper and lower bound lines, but it's the constants c_1 and c_2 that are different each time.

Equations for each type of line

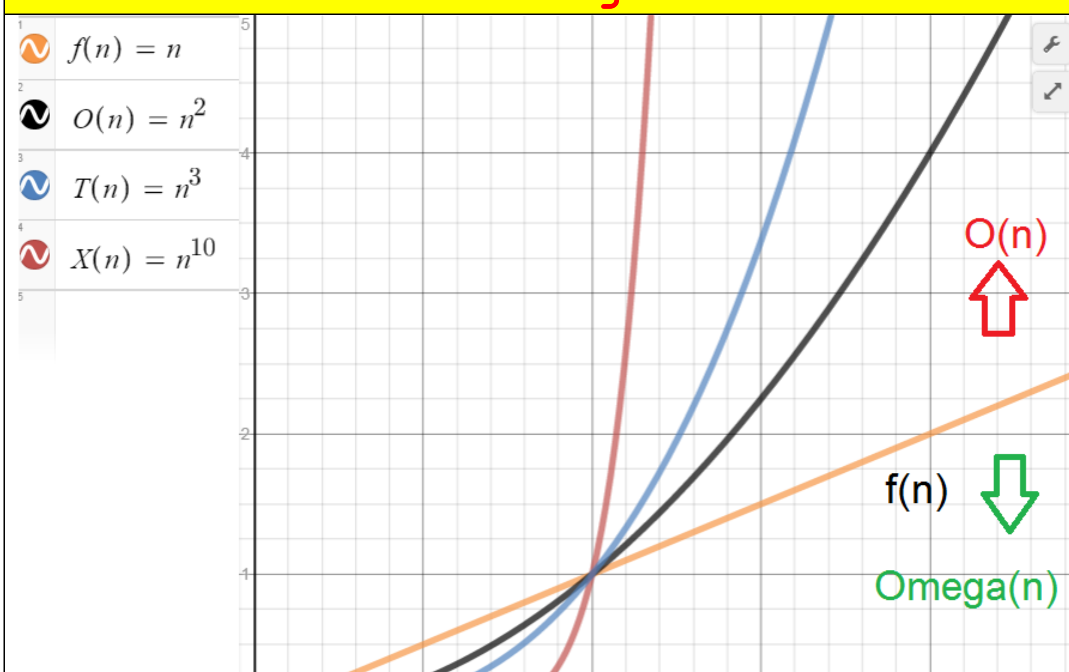
- Big O
  - $0 < f(n) <= c * g(n)$ for all $n > n_0$ (What is c??)
- Big Ω
  - $0 < c * g(n) <= f(n)$ for all $n > n_0$
- Big Θ
  - $c_1 * g(n) >= f(n) >= c_2 * g(n)$ for all $n > n_0$

Formal definition of Big-O

- O(f(n)) is a ***set*** of ALL functions g(n) that will satisfy that there exists a positive constant c and n respectfully such that for all $n >= n_0$, $f(n) <= c*g(n)$
  - very very big set of functions!!!
  - "n_0" is this vertical line on the X axis where
    - f(n) and c*g(n) meet
    - c*g(n) will be > f(n)
    - is called a "threshold" in some texts

Is f(n) € O(n^3) given that f(n) = n and O(n^3). == TRUE
Is f(n) € O(n^2) given that f(n) = n and O(n^2). == TRUE
Is f(n) € O(n^{10000}) given that f(n) = n and O(n^{1000}). == TRUE

Tightest Bound

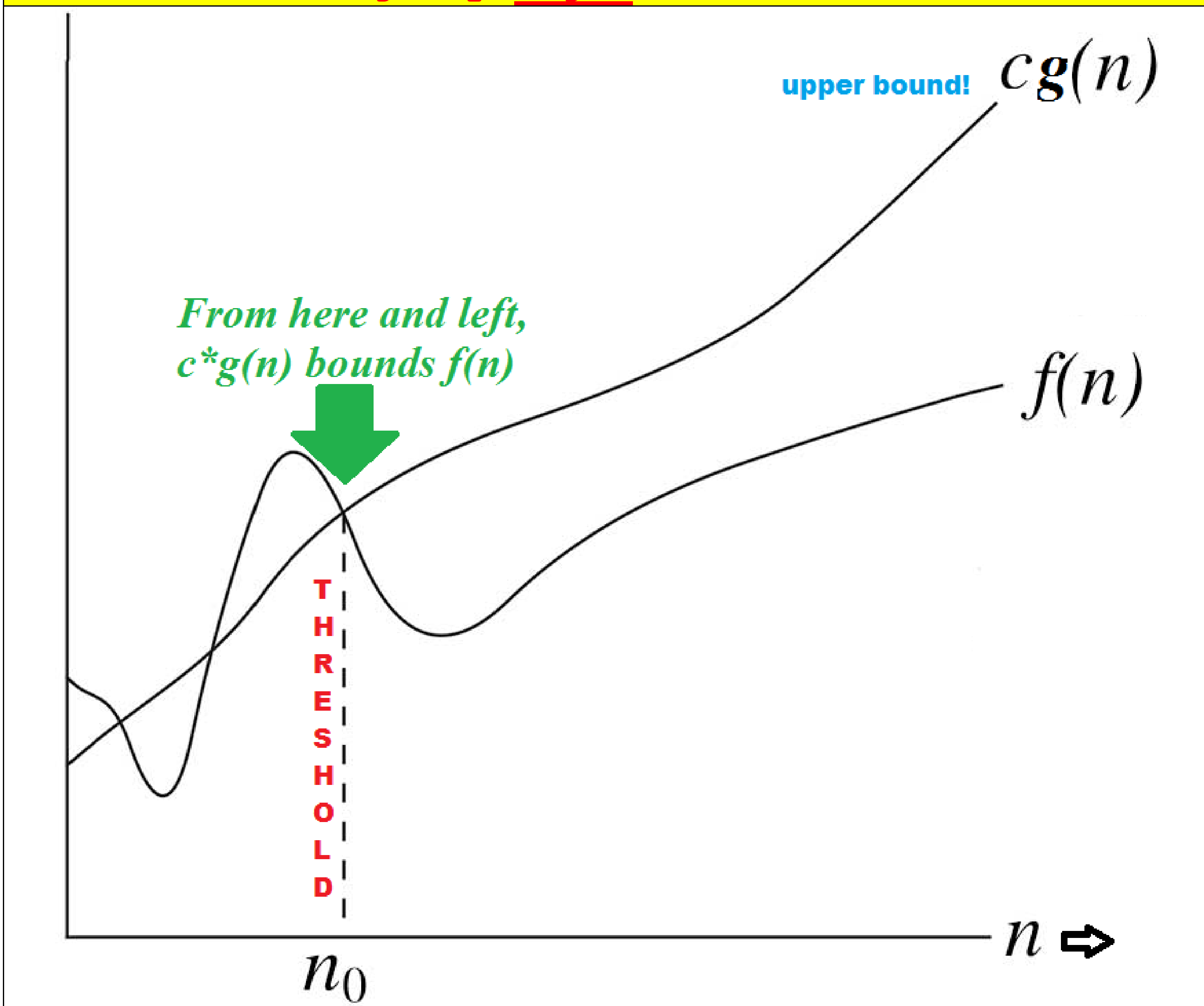


Which one of these is the tightest bound to a constant y???

Building tightly bound Big-? functions

- to get the bounds to work and be as close as possible, we have a few options
    - **we use both options below simultaneously**
    - multiply the bound (called f(n)) by a positive constant "c"
        - this will move the graph line left/right
    - threshold
        - called n_0
        - where the exact bound BEGINS
        - must be furthest ***<u>left</u>*** as graph lines increase in time and input

What a tightly <u>*Big-O*</u> answer looks like



f(n) € O(g(n)) therefore f(n) <= c*g(n) **again, for Big-O**

Choosing the constant "c" manually and how to work it

- just messing with "c" can do some interesting things with a function/graph line g(n)
- we are ONLY allowed positive values for "c"
- notice how I can change the positioning of g(n)
  - which is x^2 in this example

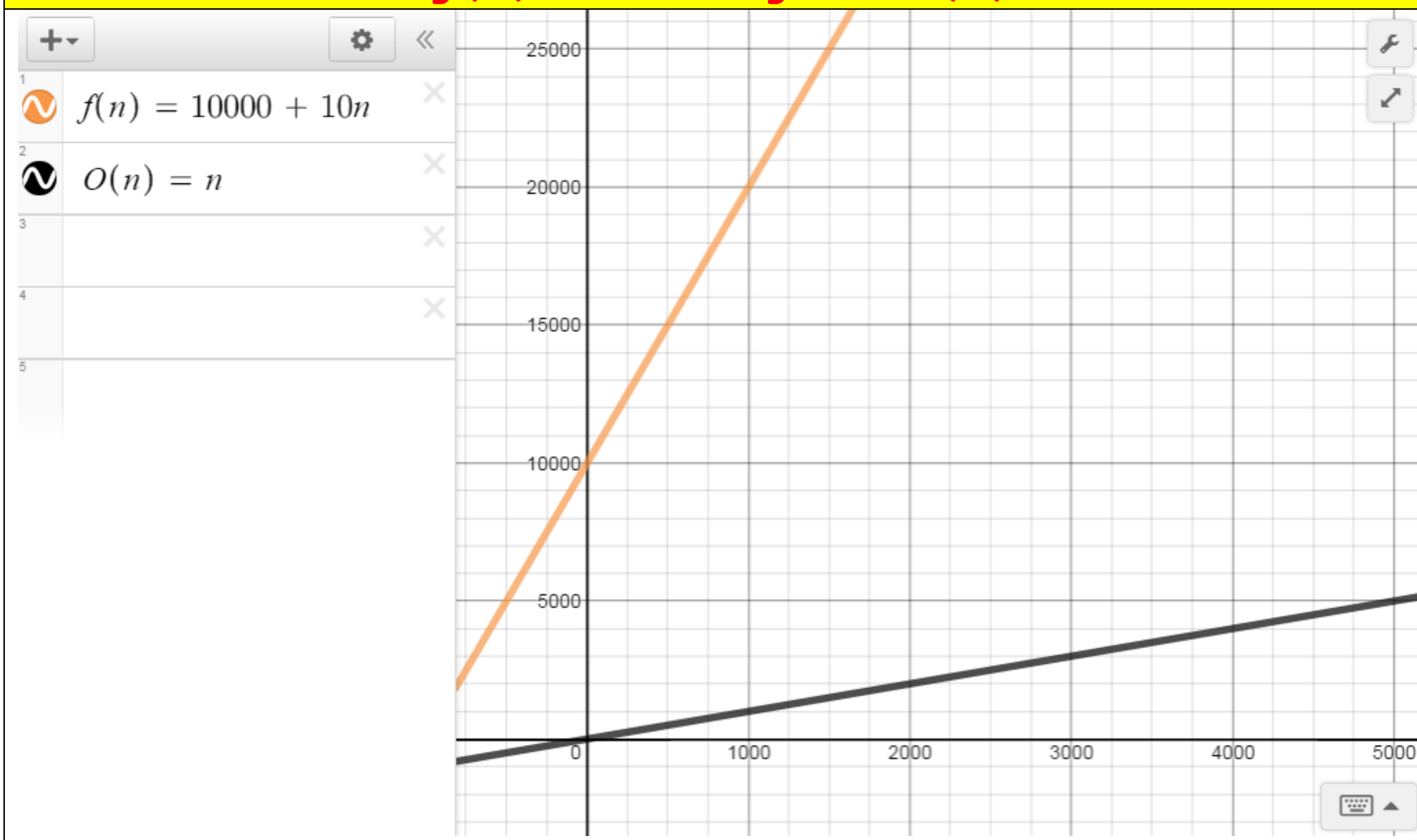
Variation of c * g(n) where g(n) = x^2

What did c of 2 and 3 do? What did c of .1 do?

Determining if f(n) is in Big-O (??) – Visually

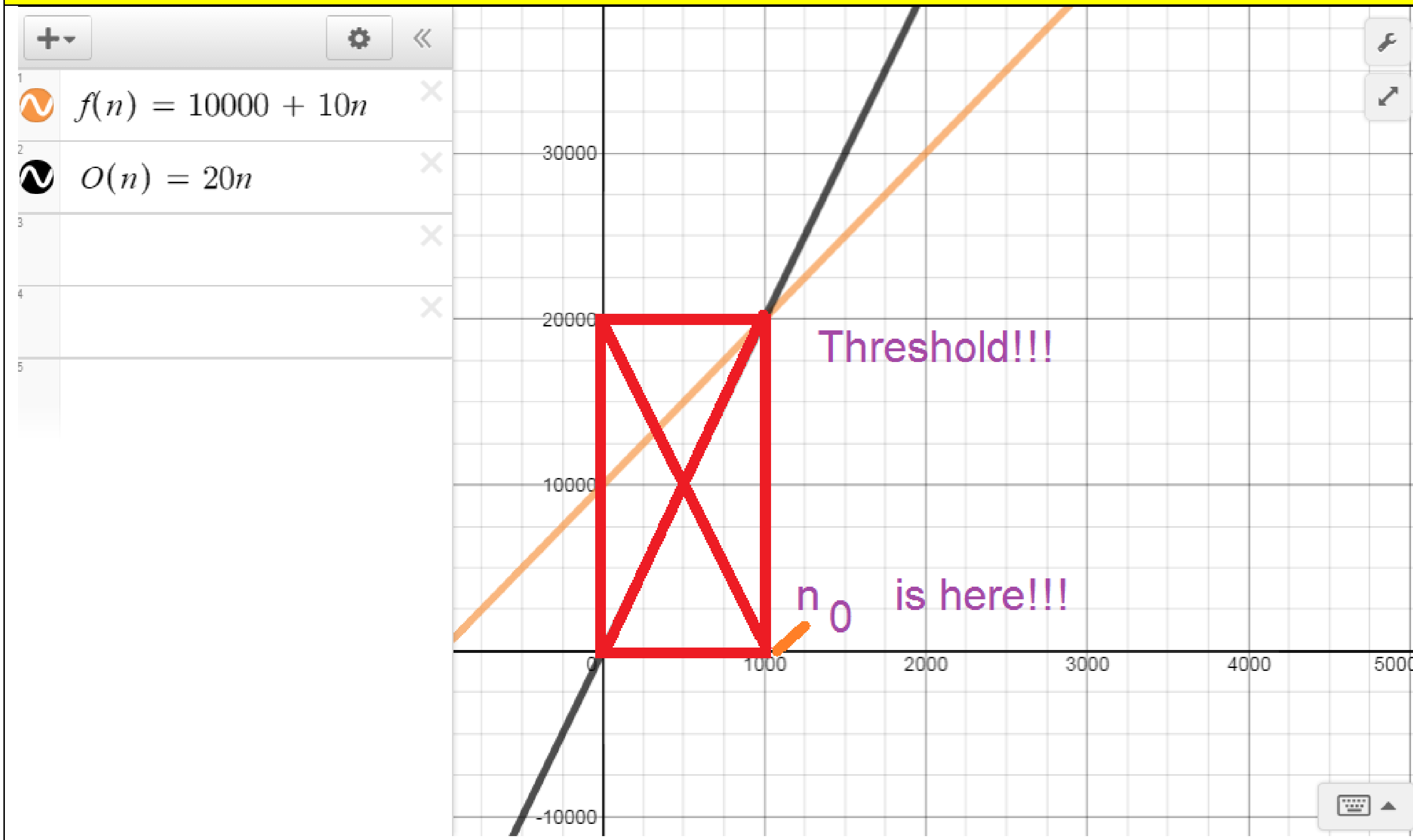
- hard way, but visual
- **not to be used as an answer**
- remember, we want f(n) <= c*g(n) SOMEWHERE if we were to graph it out
  - there may be a point where they intersect while n > 0
    - left of intersection f(n) >= c*g(n)
      - ignore, as long as we get below
    - right of intersection f(n) <= c*g(n)

g(n) versus just f(n)



- remember we want g(n) or really c*g(n) < f(n)
  - which is definitely not the case here
  - so let's adjust c!
- notice we are ***only*** interested in the positive "c" and "n" values

g(n) versus new "c" f(n)



- "c" we now made 20!!!
- ignore the graph BEFORE the intersection since our data "n" will only grow larger!!
- notice c = 20 and n_0 (x axis) = 1000 (both are positive!!)
- n_0 (threshold) = 1000!!!

Finalizing the proof

- as long as a vertical line (n on the x axis, called a threshold) exists we have a proof
- in our previous example
  - for any n >= 1000, f(n) <= c*g(n)
  - therefore
    - f(n) € O(g(n))

The Constant "c" – used, then ignored

- Big-O really does not care about "c"
  - this why we see f(n) € O(g(n)) ***and not*** f(n) € O(c*g(n))
  - c is considered a "lower order term"
    - which comes up again later
- but we use it to prove that an equation is in Big-O (O(??))
- remember a few things
  - c
    - has to be > 0
    - is constant "cost"
      - it could be how long it takes to read EACH data piece
        - 100 seconds now, could be 5 seconds in a few years!!
      - it could be how much data it needs to be read in

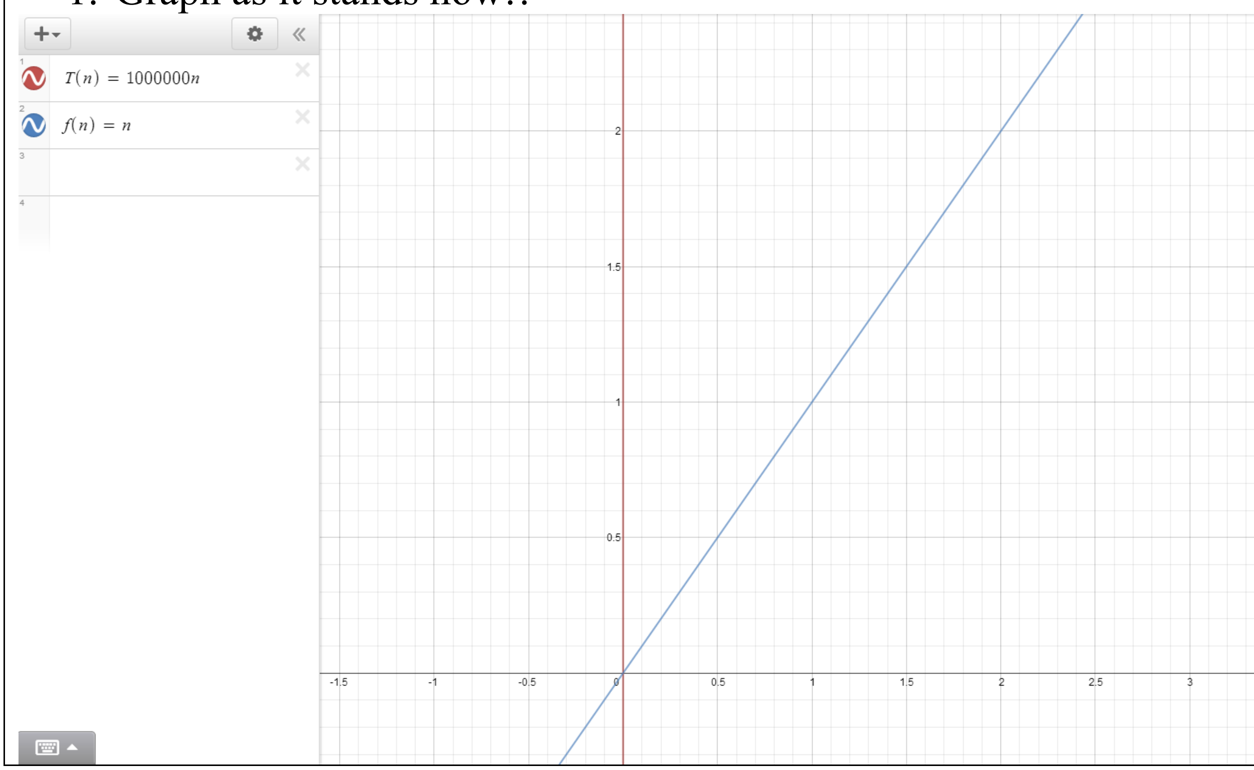
The used and ignored "c"

Prove g(n) is in Big-O (n)

g(n) = 1,000,000n
O(n)

Is f(n) € O(n)??

1. Graph as it stands now!!



2. Pick a "c" that should fit f(n) <= c*g(n). Try to keep it simple!! When graphing, g(n) is the same, no changes, just try different "c"s' and see when it could intersect

1,000,000n <= c n, try c = 1,000,000!!!

1,000,000n <= 1,000,000 n == TRUE!!!

f(n) € 1,000,000 O(n) but we only want the algorithm (n portion), not the constant

3. Given "c", solve for n_0 (n), which has to be > 0

therefore the given f(n) € O(n) is indeed true, where C = 1,000,000, and because of c, n_0 = 0

Determining if f(n) is in Big-O – Easy(ish) Way

- all math
- but the same steps apply
  - place into formula based on type of bound
  - reduce terms (n) to get "c" by itself
  - determine value for "c"
  - determine n_0 using found "c"

Solving if $2n^2$ € $O(n^3)$	
* remember, this means "Is n^3 above the $2n^2$ line on a graph?" * if you just look at the terms (eye ball test), you SHOULD know it does * I will ask you to prove, that would look like what I have below 1. Place into formula based on type of bound	
$0 \leq f(n) \leq O(n)$	This is the definition of O(g(n)) REALLY IMPORTANT!
$0 \leq f(n) \leq c * g(n)$	This is the definition of O(g(n)) REALLY IMPORTANT!
$0 \leq 2n^2 \leq cn^3$	Place in values from question

2. reduce terms to get "c" by itself

$\frac{0}{n^3} \leq \frac{2n^2}{n^3} \leq c\cancel{n^3}/\cancel{n^3}$ Divide by n^3 to get c by itself

$0 \leq \frac{2}{n} \leq c$

3. determine value for "c".

Remember n is the number of inputs, it must be **positive** and a **whole value**. We can start at 0, and go to infinity to determine where n is maximum! So I graphed.

so n = 1

now we can solve for c using value n

$0 \leq \frac{2}{1} \leq c$

c = 2

4. determine n_0 using found "c". Replace n with n_0 since we want to find the threshold.

$0 \le \frac{2}{n_0} \le 2$

$0 \le \frac{2}{2} \le n_0$ when we multiply everything by n_0 and divide by 2

$0 \le 1 \le n_0$

so $n_0 = 1$ (again, since n_0 HAS to be > 0, can't have 0 inputs!!)

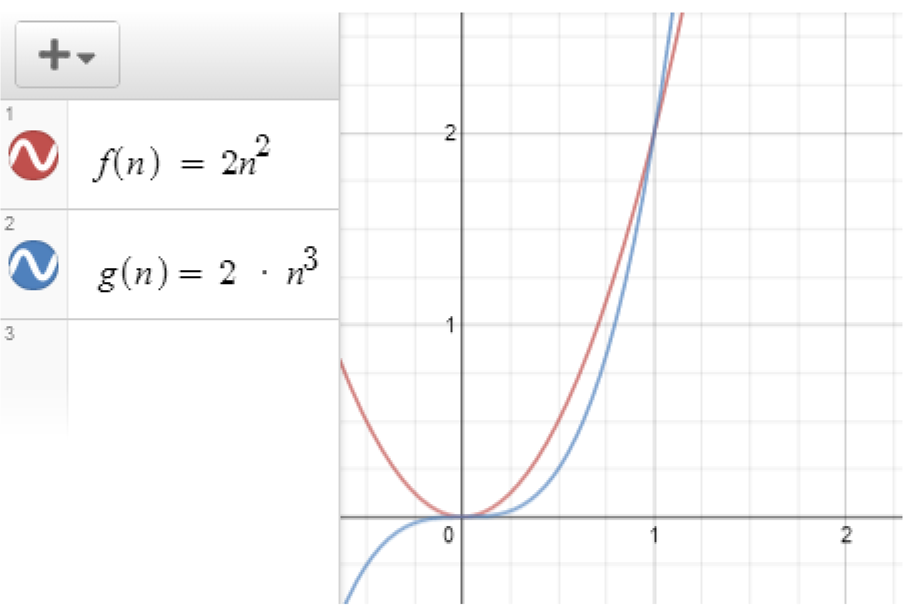
5. Now test the given equation with our found "c" and see if n_0 still stands.

$0 \le 2n^2 \le 2n^3$ can reduce if we want!!   2 is "c"

$0 \le n^2 \le n^3$ when $n_0 \ge 1$ and C = 2

which means:

1. $2n^2 \in O(n^3)$
2. n^3 is larger than $2n^2$ ON AND AFTER n_0 of 1 when c = 2



Complete Example - Prove f(n) € O(n)

given that f(n) = 500 + 10n and O(n). Make sure to find "c" and n_0

// thanks Benjamin "Spiderman" Yankowski 'F14

f(n) = 500 + 10n g(n) = n

Solve for c

500 + 10n =< cn
== 500/n + (10n)/n =< (cn)/n
== 500/n + 10 =< c (Chose n = 1)
== 500/(1) + 10 =< c
== 510 c
c = 510

Solve for n_0

500 + $10n_0$ =< cn (where c = 510 and n = 1)
== (-500) + 500 + $10n_0$ =< (510)(1) + (-500)
== $10n_0$ =< 10
== n_0 =< 1
n_0 = 1

Proof: [(explanation)] **← video here!!**

Base Case:

500 + 10(1) =< 510(1)
== 510 =< 510

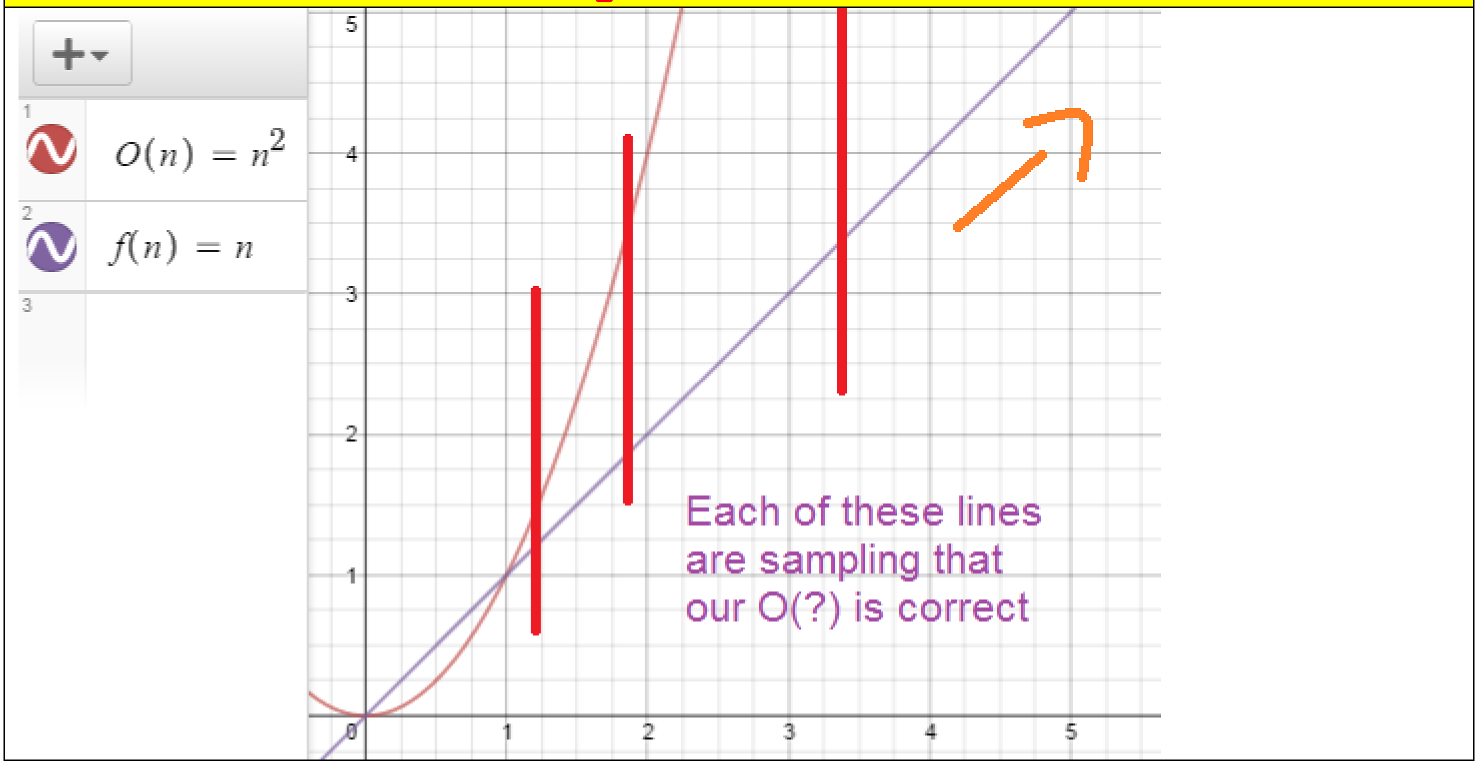
Induction:

Assume: 500 + 10(k) =< 510(k) **to be true.**
To show: 500 + 10(k + 1) =< 510(k+1)
500 + 10(k + 1)
== [500 + 10k] + 10
=<510k +10
=<510k + 510
== 510(k + 1)

But why are we doing Proof by Induction?

- remember, we want to prove our Big-O will work for n, n+1, n+2,n+..
- In proof by induction
  - is the only proof that works with n, n+1, n+2, etc…
  - n is changed to k while doing the proof

Why induction?



Determining if f(n) is in Big-O – Hunt &Peck Way

- ***last ditch effort***
- we could try some positive values of **c and n** and try to find the threshold that way
  - chart it out
- still need the equation and what Big-? we are trying to prove

Is $4n^2 + 16n + 2 \Rightarrow O(n^4)$

When c = 1		
n	**equation**	**Big ?**
0	$4(0)^2 + 16(0) + 2 = 2$	0
1	$4(1)^2 + 16(1) + 2 = 22$	1
2	$4(2)^2 + 16(2) + 2 = 50$	16
3	$4(3)^2 + 16(3) + 2 = 86$	81
4		
5		

Solve for when n = 4 and 5. Was a threshold reached?

Answer$_b$:

Just remember "c" may NOT be 1!!!! But the easiest to try!!

Solutions to Big-? problems

1. Place into formula based on type of bound
2. reduce terms to get "c" by itself
3. determine value for "c".

Remember n is the number of inputs, it must be **positive** and a **whole value**. We can start at 0, and go to infinity to determine where n is maximum!

4. determine n_0 using found "c". Replace n with n_0 since we want to find the threshold.
5. Now test the given equation with our found "c" and see if n_0 still stands.

Try:

1. Prove f(n) € O(n^3) given that f(n) = n and O(n^3). Make sure to find "c" and n_0
    a. Answer$_b$:
2. Prove f(n) € O(n^3) given that f(n) = $n^3 + n^2 + n$ and O(n^3). Make sure to find "c" and n_0
    a. Answer$_b$:

Why ignore the lower terms?

- let's start with an exercise to find out why

Assume that each of the expressions below gives the processing time g(n) spent by an algorithm for solving a problem of size n. Select the dominant term(s) having the steepest increase in n and specify the lowest Big-O complexity of each algorithm. Expression Dominant term(s) O(. . .)

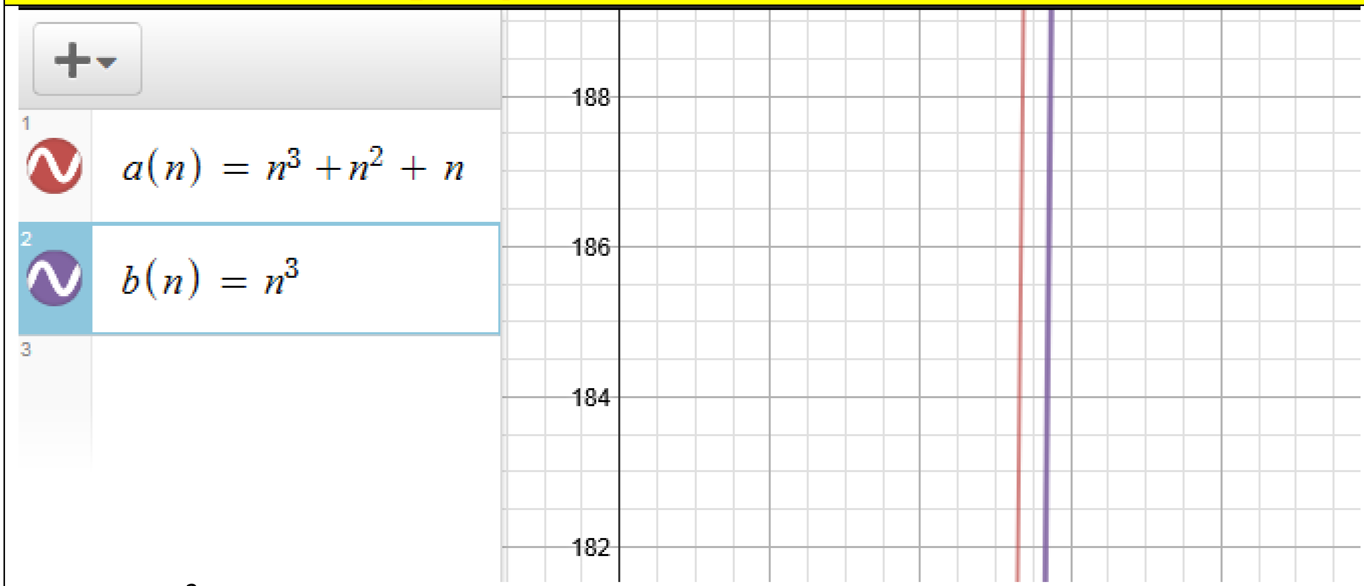
Highest Term Exercise

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n\log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n\log_3 n + n\log_2 n$		
$3\log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n\log_2 n + n(\log_2 n)^2$		
$100n\log_3 n + n^3 + 100n$		
$0.003\log_4 n + \log_2 \log_2 n$		

Answer_b:

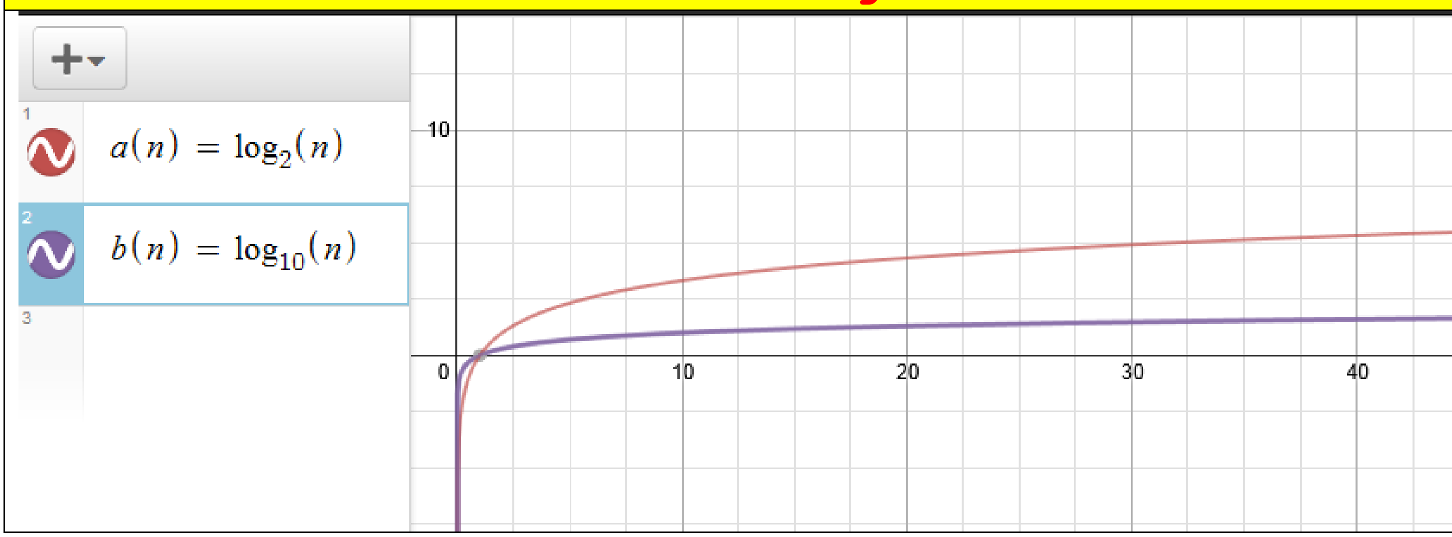
- in the example $g(n) = n^3 + n^2 + n$
  - the other terms $(n^2 + n)$ will add less and less to the overall equation as n^3 (or highest term) gets bigger and n gets bigger
- here is where you identities help!!

Why ignore the lower terms?



Notice $+n^2 + n$ is negligible
Notice the graph is NOT at the origin

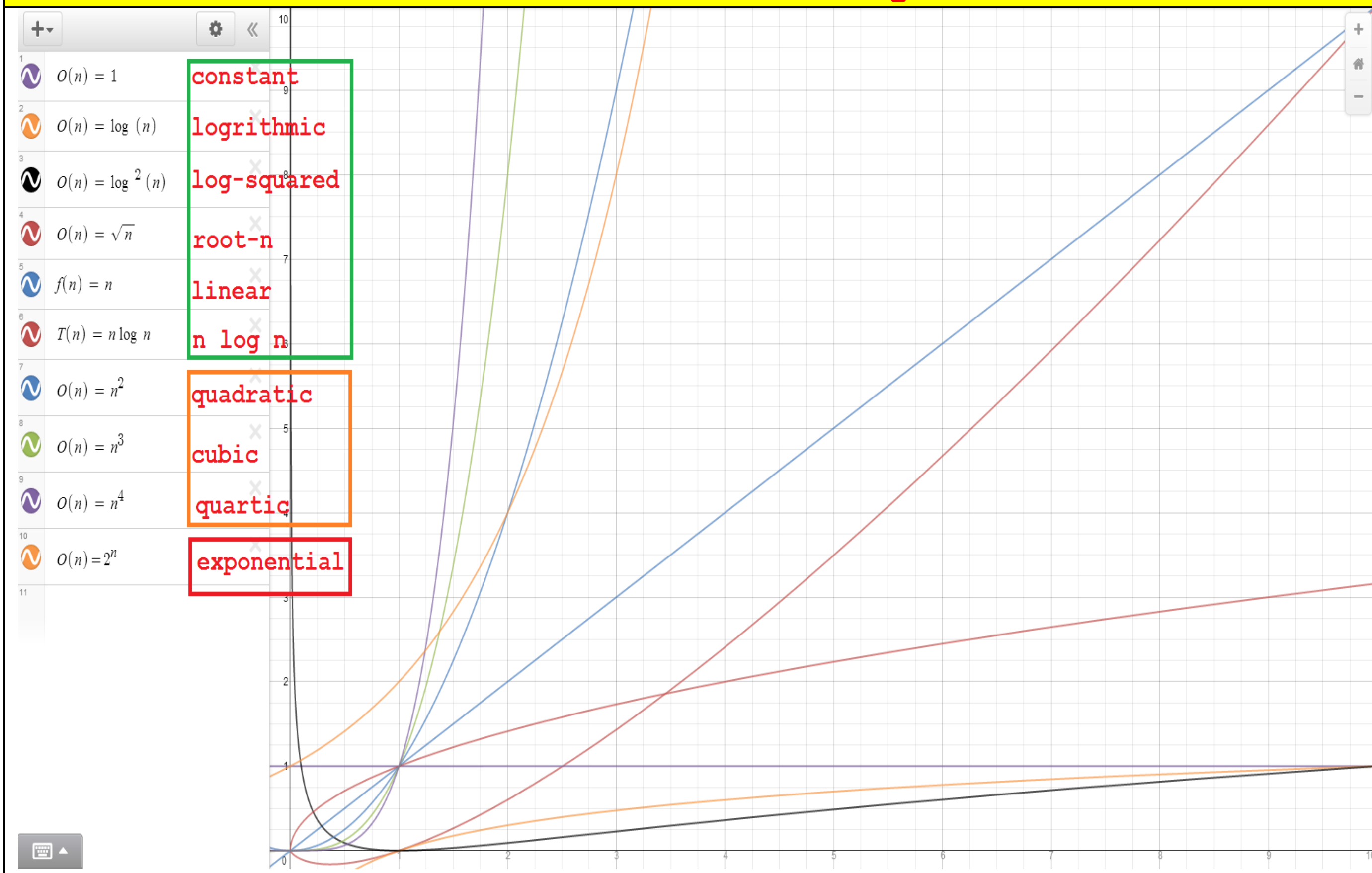
How about Logs??



How functions are a ***set*** of g(n)

- remember, we are looking for f(n) € O(???)
- several "pre-set" functions already contain this info just in their track on a graph

How functions stack up in O(??)



Green function are considered efficient
Yellow functions depend on "n" (ex: rubik's cube solver)
Red functions are useless

O(1) € O(log n) € O($\log^2$ n) € O(sqrt $\sqrt{n}$) …

since a is WITHIN the upper bound of b

Comparing Big-Os

1. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$
2. If $f(n) = O(kg(n))$ for any $k > 0$, then $f(n) = O(g(n))$
3. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$
4. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) * f_2(n) = O(g_1(n) * g_2(n))$

Draw what each of these scenarios above would look like on a graph

Worst case vs best case vs average case.

- What particular input (of given size) gives worst/best/average complexity?
- Best Case
  - If there is a permutation of the input data that minimizes the "run time efficiency", then that minimum is the best case run time efficiency
- Worst Case
  - If there is a permutation of the input data that maximizes the "run time efficiency", then that maximum is the best case run time efficiency
- Average case
  - is the "run time efficiency" over all possible inputs.

Mileage example: how much gas does it take to go 20 miles?
Worst case: all uphill
Best case: all downhill, just coast
Average case: "average" terrain

Case Example
Consider sequential search on an unsorted array of length n, what is time complexity? Best case: Worst case: Average case: Draw this out!!

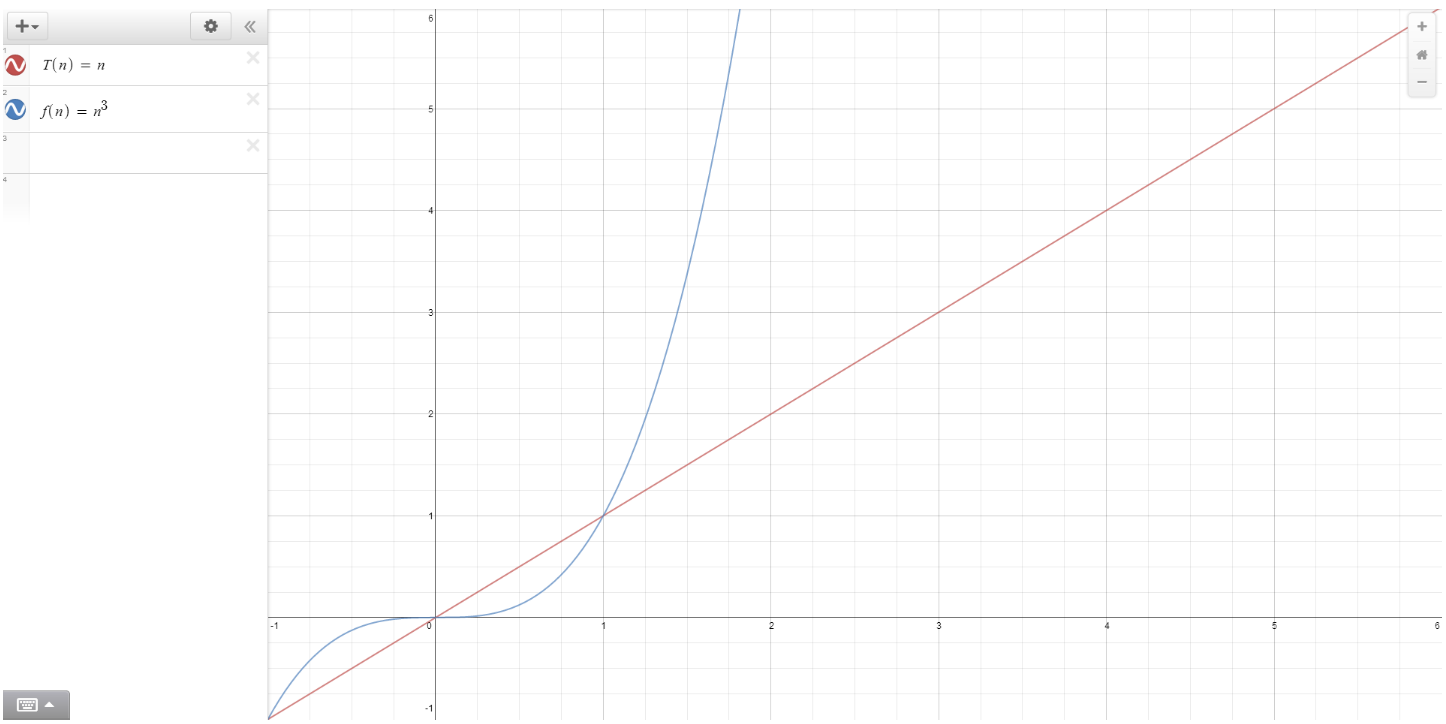
Answer Section

Is $4n^2 + 16n + 2 \Rightarrow O(n^4)$

Yes, when $c = 1$ and $n_0 = 4$

$f(n) \in O(n^3)$ given that $g(n) = n$ and $O(n^3)$

1. Graph as it stands now



2. Pick a "c" that should fit $f(n) <= c*g(n)$. Try to keep it simple!!

Since they already intersect, let's just make c = 1!!!

$f(n) \in 1\ O(n^3)$

3. Given "c", solve for N (n), which has to be > 0

therefore the given $f(n) \in O(n^3)$ is indeed true, where C = 1 and because of c, N = 1 f(n) € O(n^3) given that g(n) = $n^3 + n^2 + n$ and O(n^3)

c = 3
N = 1

btw, when N(n) = 1, $1^3 + 1^2 + 1 = 3$, which is our constant

---

Highest Term Exercise

Expression	Dominant term(s)	$O(\ldots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n \log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log n)$
$n \log_3 n + n \log_2 n$	$n \log_3 n,\ n \log_2 n$	$O(n \log n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log n)^2)$
$100n \log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003 \log_4 n + \log_2 \log_2 n$	$0.003 \log_4 n$	$O(\log n)$

Sources

Asymptotic Lectures
http://www.youtube.com/watch?v=VIS4YDpuP98
http://www.youtube.com/watch?v=ca3e7UVmeUc

Basics
http://www.youtube.com/watch?v=6Ol2JbwoJp0

Solving Big-O mathematically
http://www.youtube.com/watch?v=ei-A_wy5Yxw

Online Graphing Calculator
https://www.desmos.com/calculator

Highest term
http://www.cs.auckland.ac.nz/compsci220s1t/lectures/lecturenotes/GG-lectures/220exercises1.pdf

Big – Oh charts
http://filipwolanski.com/2013/03/08/big-o/