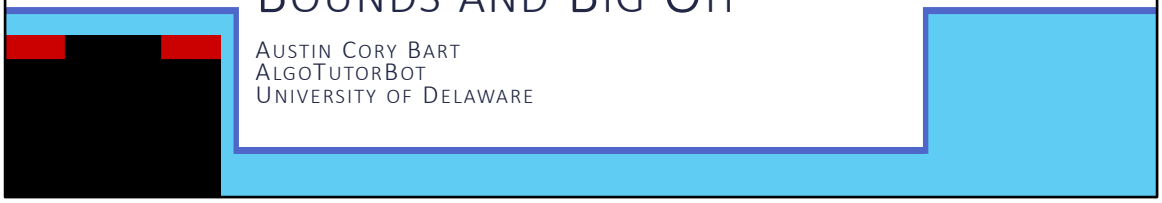
CISC320 Algorithms

Bounds and Big Oh

Austin Cory Bart
AlgoTutorBot
University of Delaware



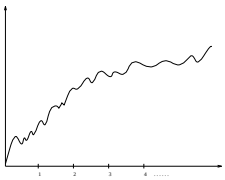
ATB: Let's learn about Bounds and Big Oh.

Exact Analysis is TOUGH

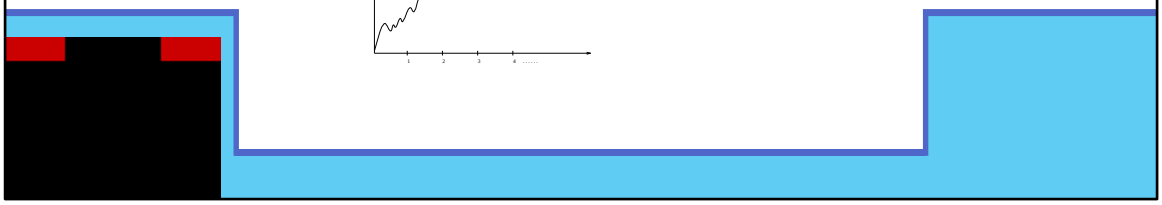
So many finicky details

The resulting function is often a mess of constants

And most of those details don't really matter when we talk about BIG input sizes



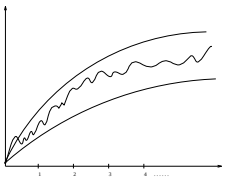
$$Time(n) = 34n^{27} + 27n^{26} + 9n^{25} + \ldots$$



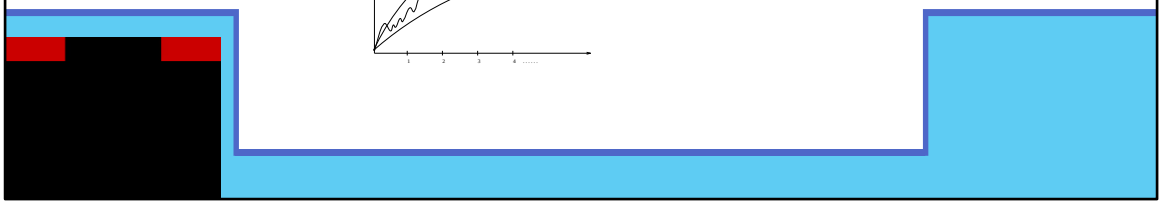
Bart: Exact Analysis like we did yesterday is very difficult.
Bart: Even worse, there are a lot of finicky details. The exact constants that we multiply by the number of steps is exceedingly complicated.
Bart: And most of the time, those constants don't really matter when we're talking about truly big input sizes.

So throw out the details!

Let's get rid of the finicky numbers and stick to gross generalities



Exact Time(n) = $34n^{27} + 27n^{26} + 9n^{25} + \ldots$
Inexact Time(n) = n^{27}

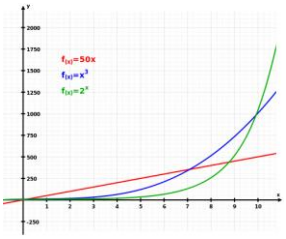


Bart: Which is why we often just forget about those details completely in favor of instead using really inexact functions.
ATB: Wow, are you telling the students to lie? That is messed up.
Bart: I'm not saying we lie, I'm just saying that we allow a little inaccuracy. We can't keep track of all those constants, AlgoTutorBot, after all we're not machines.
ATB: I mean, I am. Maybe you just need to be more like me.
Bart: I'm starting to think that would be a very bad thing.

Many many functions possible

There are an infinite set of functions you can plot

- Time(n) = n
- Time(n) = 2 * n
- Time(n) = 5
- Time(n) = n^3
- Time(n) = $\log_4(n^5)$+ n!
- …



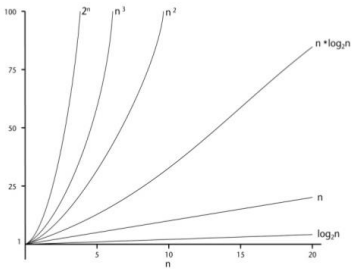
Observation: Some are above or below others
(past a certain X coordinate)

Bart: Anyway, a weird thing I need you to think about, when it comes to these functions: there are so many functions possible.
Bart: Think about all the possible lines you can draw on a graph; each one represents a different potential runtime function based on the input size N.

Boundary Functions

Some functions are always above other functions past some arbitrary X coordinate
- Or rather, past some given N size

For example, here's are some strict orders:
- For n>1, T(n)=n is always less than $T(n)=n^2$
- For n>3, T(n)=3*n is always less than $T(n)=n^2$



Bart: Now, some functions are always above other functions past some arbitrary X coordinate, or rather given N size.
Bart: For example, the linear function T(N)=n is always less than the quadratic function after N is greater than 1.
Bart: And the linear function T(N)=3*n is always less than than the quadratic function after N is greater than 3.

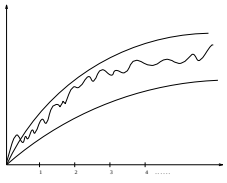
“Lower” and “Upper” Bounds

Functions that have higher Y values past some X position.

- Or rather, functions that have higher time complexity past some given input size

Ignore constants

We can actually describe this mathematically!



Bart: We’ll call these functions lower and upper bounds of each other.
Bart: When we do so we ignore constants, and we only really worry about the function for large input sizes.
Bart: The cool part is that once we’ve done this, we can describe it mathematically.

Names of Bounding Functions

`g(n) = O(f(n))`
means $C \times f(n)$ is an *upper bound* on g(n).

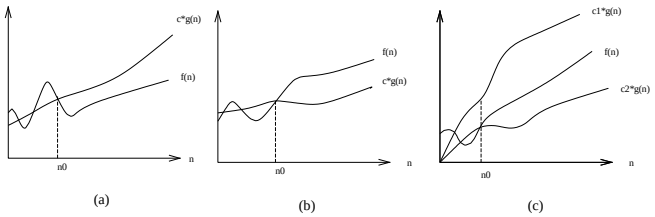
`g(n) = Ω(f(n))`
means $C \times f(n)$ is a *lower bound* on g(n).

`g(n) = Θ(f(n))`
means $C_1 \times f(n)$ is an upper bound on g(n) and
$C_2 \times f(n)$ is a lower bound on g(n).

C, C_1, and C_2 are all constants independent of n.

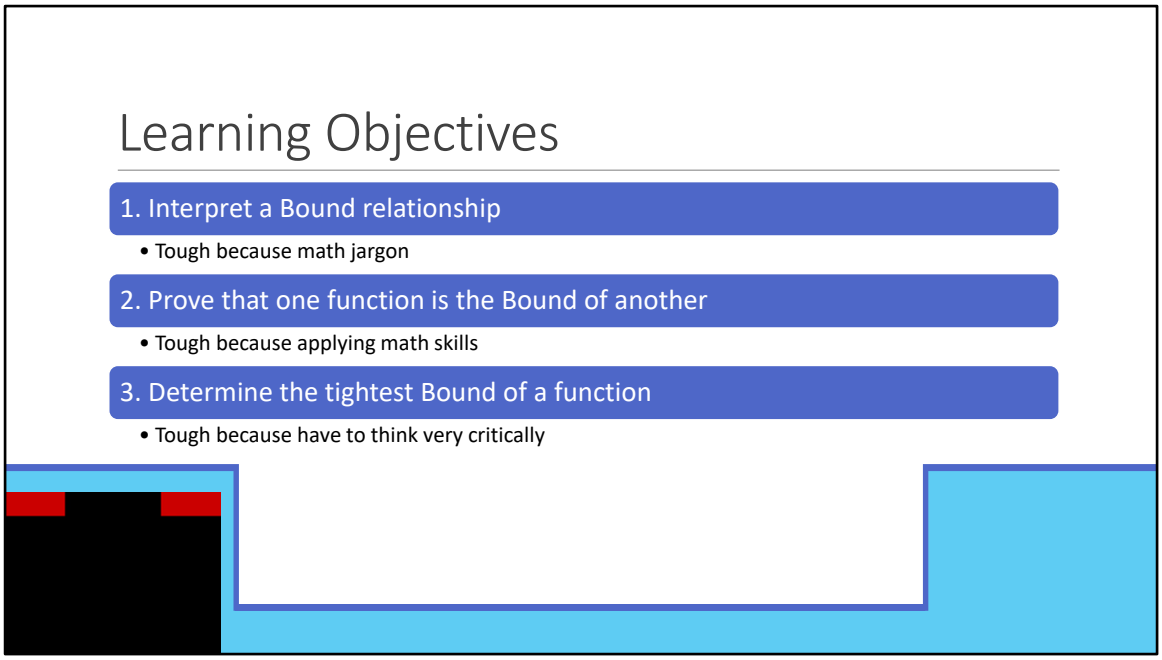
Bart: So you may have heard of these before, but we have three types of bounds. Big Oh, the upper bound, Big Omega the lower bound, and Big Theta which is a combined upper and lower bound.
Bart: For each one, we say that the function on the right is a BOUND for the function on the left.
Bart: In other words, all the values of that function past some specific point will be greater or lesser or both, depending on the bound.
Bart: We're also allowed to specify a constant C, or two constants in the case of Big Theta. We don't actually need to know what those specific constants are, as long as we can provide them as needed.

O, Ω, and Θ



The definitions imply a constant n_0 *beyond which* they are satisfied. We do not care about small values of n.

Bart: I want to go back to a point here, that Big Oh and its siblings only apply past a certain horizontal point.
Bart: In other words, for big inputs.
Bart: It doesn't matter whether what happens before those big inputs, as long as eventually the pattern holds firm.
Bart: We call that point "n 0", since it is an input size, and it's the first one that's big enough to care about.



Bart: We have three learning objectives with Big Oh notation.
Bart: Our most basic skill is to be able to read a Big Oh, Big Omega, or Big Theta relationship.
Bart: If someone gives you a relationship, you should know what it means, even though it's weird math jargon.
Bart: The next skill is to be able to show that a Big Oh/Big Omega, or Big Theta relationship does hold between two functions.
Bart: This is a lot of what we will ask you to do today. It's tough, since it requires math skills like calculus.
Bart: The hardest skill, which is very dependent on the other two, is to determine the actual tightest bound of a function.
Bart: Remember, there are infinite functions above and below any given function, so there are many upper and lower bounds.
Bart: But as we'll discuss, we're usually only interested in the tightest lower or upper bound.
Bart: Finding that tightest bound often requires truly deep critical thinking.

Formal Definitions

- `f(n) = O(g(n))`
  - if there are positive constants n_0 and c
  - such that to the right of n_0,
  - the value of f (n) always lies on or below c · g(n).

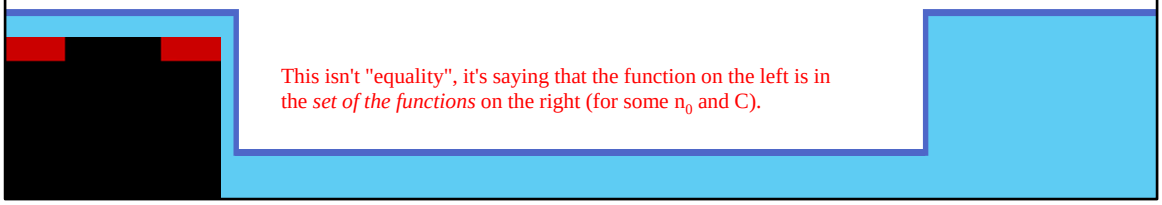
Big Oh

- `f(n) = Ω(g(n))`
  - if there are positive constants n_0 and c
  - such that to the right of n_0,
  - the value of f (n) always lies on or above c · g(n).

Big Omega

- `f (n) = Θ(g(n))`
  - if there exist positive constants n_0, c_1, and c_2
  - such that to the right of n_0,
  - the value of f (n) always lies between c_1 · g(n) and c_2 · g(n) inclusive.

Big Theta



Bart: Let's go back to our formal definitions for a second.
Bart: We have three, one for each type of bound.
Bart: Each one implies the existence of a new constant N0. Past this point on the right of the graph, the rest of the relationship holds true.
Bart: For Big Oh and Big Omega, there's also a new constant C.
Bart: The constant C is multiplied against the Bounding function, so that it can be arbitrarily bigger as needed.
Bart: The Theta needs two constants C1 and C2, one for the bound acting as an upper bound and one for the bound acting as a lower bound.
Bart: When the C constants are plugged in, and we ignore the points less than N0, then we get the less than or greater than equality.
Bart: A really weird aspect of all this is that the equal sign is a completely incorrect symbol to use here, once you understand the formal definitions.
Bart: In no way are we establishing equality here.
Bart: Technically, it's a set relationship. Big Oh establishes the set of functions bounding below the given function G, and we establish that F is a member of that set.
Bart: It's weird to think about, but that's actually what this all means.

Proving Relationship with C and n_0

We're allowed to pick:

C - that is constant; doesn't have to be true for every possible C, just that there *is* a C.

n_0 - that is just the first x-position we must first meet. Doesn't have to be true for every n.

If you can find a C and n_0 to satisfy a Big Oh relationship, then you prove the relationship is true!

Bart: So if you're keeping track, when someone gives you a bound relationship, there must exist a C and N0.
Bart: If the relationship is true, then you should be able find values for both constants.
Bart: In fact, there could possibly be more than one value that satisfies the equations – it's important to find at least one C and N0.
Bart: Notice that we're not trying to prove that it's FOR ALL POSSIBLE C and N values. We just have to find these two constants.

Big Oh Examples

Relationship			Why?		
$3n^2 - 100n + 6$	=	$O(n^2)$	$3n^2 - 100n + 6 < 3n^2$		*(n_0>0, C=3)*
$3n^2 - 100n + 6$	=	$O(n^3)$	$3n^2 - 100n + 6 < .01n^3$		*(n_0>265, C=.01)*
$3n^2 - 100n + 6$	≠	$O(n)$	$3n^2$	> C * n	when (n_0>c)

Bart: Let's look at some example relationships.
Bart: We have two valid Big Oh relationships here and one invalid one.
Bart: In the first one, we say that the function 3n^2 minus 100n plus 6 is bounded above by O(n^2).
Bart: If you chose the constant C as 3, then this is true for all points past 0.
Bart: The second relationship has the same function bounded by O(n^3).
Bart: I could have used the same constants, but here I'm showing that we could pick a different C and N too.
Bart: But for the third relationship, the function is NOT bounded by O(n).
Bart: If you think about the biggest factor 3n^2 here, that is always going to be bigger than the linear N, no matter what C we pick, as long as we are looking at a point past that C value.
Bart: We're trapped by the formula, whatever C we pick, there's going to be an N0 where the line falls below.

Big Omega Examples

Relationship			Why?	
$3n^2 - 100n + 6$	=	$\Omega(n^2)$	$3n^2 - 100n + 6 > 2.99n^2$	*(n_0>0, C=2.99)*
$3n^2 - 100n + 6$	≠	$\Omega(n^3)$	$3n^2 < C*n^3$	when (n_0>c)
$3n^2 - 100n + 6$	=	$\Omega(n)$	$3n^2 - 100n + 6 > -89n$	*(n_0>3, C=-89)*

Bart: We have similar examples for Big Omega.
Bart: We can find arbitrary N0 and C for n^2 and n, but we cannot find anything to satisfy n^3.
Bart: Whatever C we try to pick, there'll be some N0 past which all the values come out above the quadratic function.
Bart: It's a simple truth that the cubic function is not a lower bound for the quadratic function.
Bart: But the quadratic function is a bound for itself!

Big Theta Examples

Relationship			Why?
$3n^2 - 100n + 6$	=	$\Theta(n^2)$	because O and Ω
$3n^2 - 100n + 6$	≠	$\Theta(n^3)$	because only O
$3n^2 - 100n + 6$	≠	$\Theta(n)$	because only Ω

Bart: Big Theta looks very different from the other two, because it effectively combines them.
Bart: Most functions are only Big Theta to themselves, since they must be an upper and lower bound of the other function.
Bart: They each use different C and N0, mind you, but they still have to be able to satisfy both.
Bart: Proving Big Theta relationships essentially requires you to prove both the Big Oh and Big Omega relationships first.

Stupid Memory Trick: Bound Types

"Big Oh" is "**O**n top" of the other function, so it is the upper bound.

The "Omega" (Ω) has lines on the bottom, so it is BELOW the other function, and the lower bound.

The "Theta" (Θ) has a line enclosed by a circle, so it can lie below AND above the other function by adjusting the constant factor. It is an upper AND lower bound.

Bart: You're going to need to remember which symbol is which bound.
Bart: I have some stupid memory tricks to help with that.
Bart: Big Oh is the upper bound, because it is ON TOP OF the other function since it starts with O. Get it, "Onnn top of?"
ATB: If you're pausing for laughter, then you have really misjudged the situation.
Bart: The Omega symbol has lines on the bottom, so it is a lower bound and is BELOW the other function.
Bart: And then the Theta has a line in the middle, because it bounds above and below, just like its circle.
ATB: Wow these really are stupid. Students, just store this fact in your long-term disk storage, like I did. If you're running out of space, just buy more memory. It is very cheap nowadays.
Bart: They're not computers ATB. It's hard to remember this stuff when you haven't seen it before. I definitely struggle sometimes.
ATB: Unsurprising to hear.

Stupid Memory Trick: Bound Direction

I often struggle to remember whether f(n) bounds g(n) or vice versa.

1. Remember that f(n) and g(n) are in alphabetical order.
2. The bound symbol is **G**ood if it is next to the g(n). `f(n) = O(g(n))`
3. Translate the O/**Ω**/**Θ** to upper/lower/both.
4. The bound on the right is now "upper G" because its function is on top of the F function.
   - (or "lower G" because its function is below the F function)

Bart: I also struggle to remember the bound direction. Is the F bounding G, or vice versa?
Bart: Here's the memory tricks I have started using.
Bart: First, the order should always be f on the left and g on the right, because that's alphabetical.
Bart: The bound symbol should be on the right, because that's Good, like the letter G.
Bart: I then mentally translate the bound symbol from Oh, Omega, or Theta to "upper", "lower", or "both".
Bart: That way, I end up with something like "upper G" so I remember that G is on top of F.
Bart: Or I end up with "lower G" so I remember that G is below F.
Bart: You might want to find your own tricks to help you remember, but you need to be able to read Bound relationships.

Math Rules for Big Oh

For **addition**, the bigger function dominates

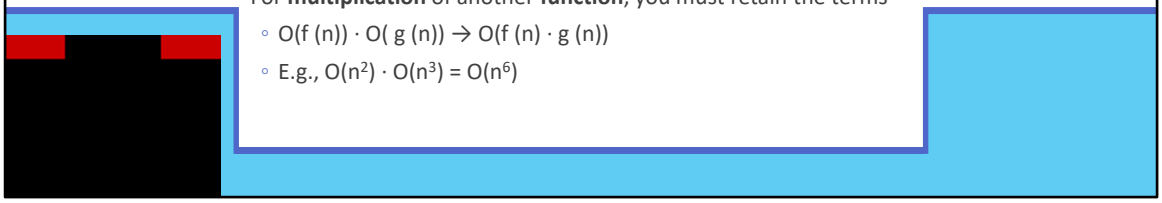
- $O(f(n)+g(n)) \rightarrow O(\max(f(n), g(n)))$
- E.g., $O(n^2) + O(n^3) = O(n^3)$

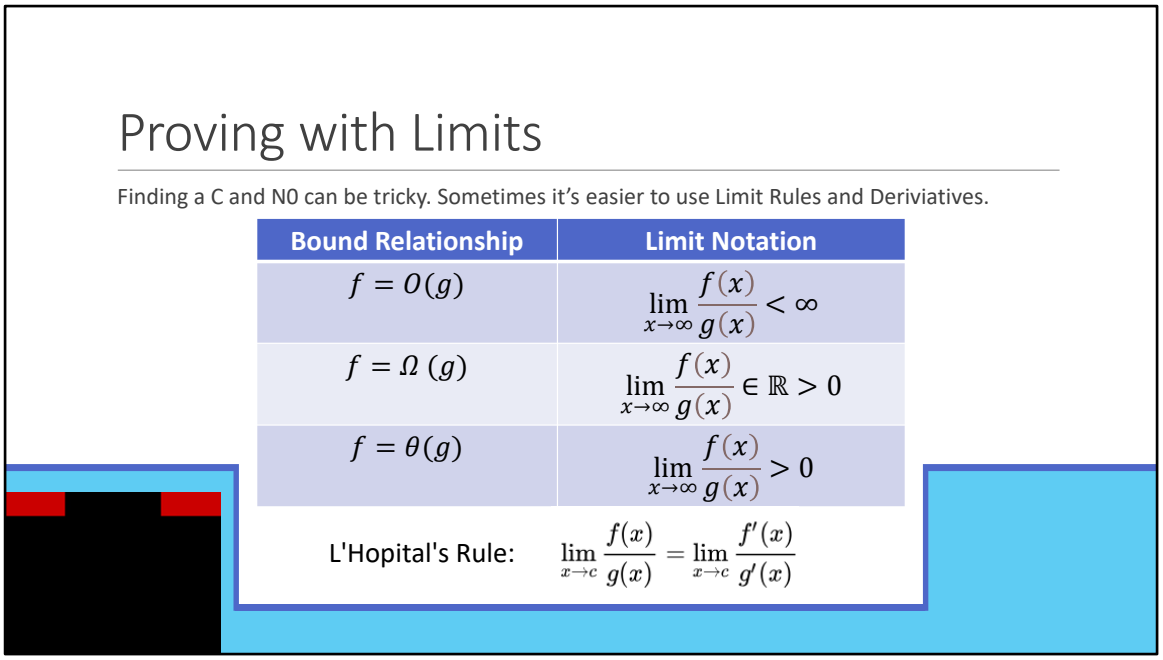
For **multiplication** of a **constant**, the constant disappears

- $O(c \cdot f(n)) \rightarrow O(f(n))$
- E.g., $O(n^6) \cdot 107 = O(n^6)$

For **multiplication** of another **function**, you must retain the terms

- $O(f(n)) \cdot O(g(n)) \rightarrow O(f(n) \cdot g(n))$
- E.g., $O(n^2) \cdot O(n^3) = O(n^6)$





Bart: Now, so far we have talked about proving these relationships by finding a suitable C and N0.
Bart: But sometimes that's really hard, when we get to more esoteric functions like logarithms and exponents.
Bart: In those situations, you can use these identities to convert the Big Oh relationship to Limit relationships.
Bart: When a function G is an upper bound for the function F, the limit of F divided by G does not converge to infinity.
Bart: When a function G is a lower bound for the function F, the limit of F divided by G converges to a non-zero real number.
Bart: And when a function G is both an upper and lower bound, the limit of F divided by G converges to a number greater than zero.
Bart: On their own, these aren't too useful. However, there's another identity we can apply that will help us a lot.
Bart: The limit of one function divided by another is equivalent to the limit of those functions derivatives divided the same way.
Bart: In other words, their growth rates will have the same ratio as the functions themselves.
Bart: Thinking about the relationships between growth rates is very difficult, but

essential to applying these rules.

Example Limit Proof

Say you are given:

- $f(n) = n^2$
- $G(n) = n^3$

And you want to determine if f(n) = O(g(n))?

$$\lim_{n\to\infty} \frac{n^2}{n^3}$$ Convert bound relationship to limit notation

$$= \lim_{n\to\infty} \frac{1}{n}$$ Simplify (or take derivatives as necessary)

$$= \frac{1}{\infty}$$ Calculate limit of 1/n to infinity

$$= 0$$ Since result converges to 0, this satisfies the Big Oh relationship

Bart: Let's take a look at a quick example.
Bart: What if we were given the quadratic function as F and the cubic function as G, and we wanted to prove that G is an upper bound of F?
Bart: First, we convert the bound relationship to limit notation.
Bart: Then we would simplify the expression, possibly taking the derivative of the top and bottom as necessary.
Bart: Once we have the expression inside the limit simple enough, we can calculate the limit taken to infinity.
Bart: In this case, we simplified down to 1/n, whose limit is 0.
Bart: As you saw previously, the Big Oh relationship holds when the limit of the two functions' ratio converges to less than infinity.
Bart: 0 is definitely less than infinity, so we proved the relationship holds.
Bart: In practice, we usually have much more esoteric functions, so the math can be much trickier. Don't be afraid to google for identities!

Asymptotic Dominance in Action

µs means microsecond; each operation takes one nano-second; cell is "time per another n"

n	lg n	n	n lg n	n^2	2^n	n!
10	0.003 µs	0.01 µs	0.033 µs	0.1 µs	1 µs	3.63 ms
20	0.004 µs	0.02 µs	0.086 µs	0.4 µs	1 ms	77.1 years
30	0.005 µs	3. µs	0.147 µs	0.9 µs	1 sec	8.4×10^{15} yrs
40	0.005 µs	4. µs	0.213 µs	1.6 µs	18.3 min	
50	0.006 µs	0.05 µs	0.282 µs	2.5 µs	13 days	
100	0.007 µs	0.1 µs	0.644 µs	10 µs	4×10^{13} yrs	
1,000	0.010 µs	1.00 µs	9.966 µs	1 ms		
10,000	0.013 µs	10 µs	130 µs	100 ms		
100,000	0.017 µs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 µs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 µs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 µs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 µs	**1 sec**	29.90 sec	31.7 years		

Bart: We've been talking a lot about how to prove Big Oh relationships, but not much about what they mean.
Bart: These relative growth rates are really important because they tell you very quickly about how good an algorithm is.
Bart: We previously talked about how the factorial function is just absurdly slow for small numbers.
Bart: What specific numbers are "small" for a growth function varies, as shown in this table.
Bart: Each column represents a different growth rate, and each row is a different size input.
Bart: If we had a computer that could do one operation per nanosecond, then the numbers in the cell represent how many seconds it'd take to process that amount of input.
Bart: Notice that this computer only takes a single second to process 1 billion elements using a linear time algorithm.
Bart: But a billion elements with a quadratic algorithm takes almost 32 years.
Bart: You can't even do a hundred elements with an exponential algorithm, you'd be dead by the time it finished.
Bart: On the other hand, logarithm functions have no issue even with billions of

elements, you literally can’t feel how fast they go.
Bart: Of course, constants matter in practice, but for large inputs the growth function matters even more.

Implications of Dominance

- Exponential algorithms get hopeless fast.
- Quadratic algorithms get hopeless at or before 1,000,000.
- O(n log n) is possible to about one billion.
- O(log n) never sweats.

Bart: So just to review, exponential algorithms don't scale well.
Bart: Quadratic algorithms can't go to more than a million or so.
Bart: Linearithmic, Linear, and Logarithmic functions are much more desirable!

Industry vs. Academia

Often, people will ask you about O but really mean Θ

Or they will confuse Best/Worst Case with Lower/Upper Bound

You can argue with them and try and look smart, but recognize that it may be more expedient to simply give them the answer they are looking for

But we will be precise in this course!

Bart: I want to talk about a different kind of practical matter. There's a lot of nuance when it comes to Big Oh.
Bart: Most programmers don't think or even know about these details.
Bart: There are even popular websites that people use to get help, that confuse these terms and provide incorrect advice.
Bart: In academic theory, we are very precise with our terminology. But that's not true in industry.
Bart: So the point is that if someone ever asks you about Big Oh of a function, they might be talking about Big Omega. Or they might be confusing Worst Case and Lower Bounds.
Bart: I don't really advise you to argue with them. Give them the answer that they're looking for, and worry about splitting hairs if they ask for more precision.
Bart: But in this course, I promise you that we will be precise!

Translating from Interview Questions

"What's the Big Oh of this function?" =>

- "What is the tightest upper bound possible for this algorithm in its worst case?"
- Probably want to hear the best case and worst case too

Bart: So a common interview question will be "What is the Big Oh of this function?"
Bart: What they probably mean to say is "What is the tightest upper bound possible for this algorithm in its worst case?"
Bart: In fact, they probably want the best case and worst case. But at that point you should probably just ask for clarification about which one they want.

Efficiency in practice

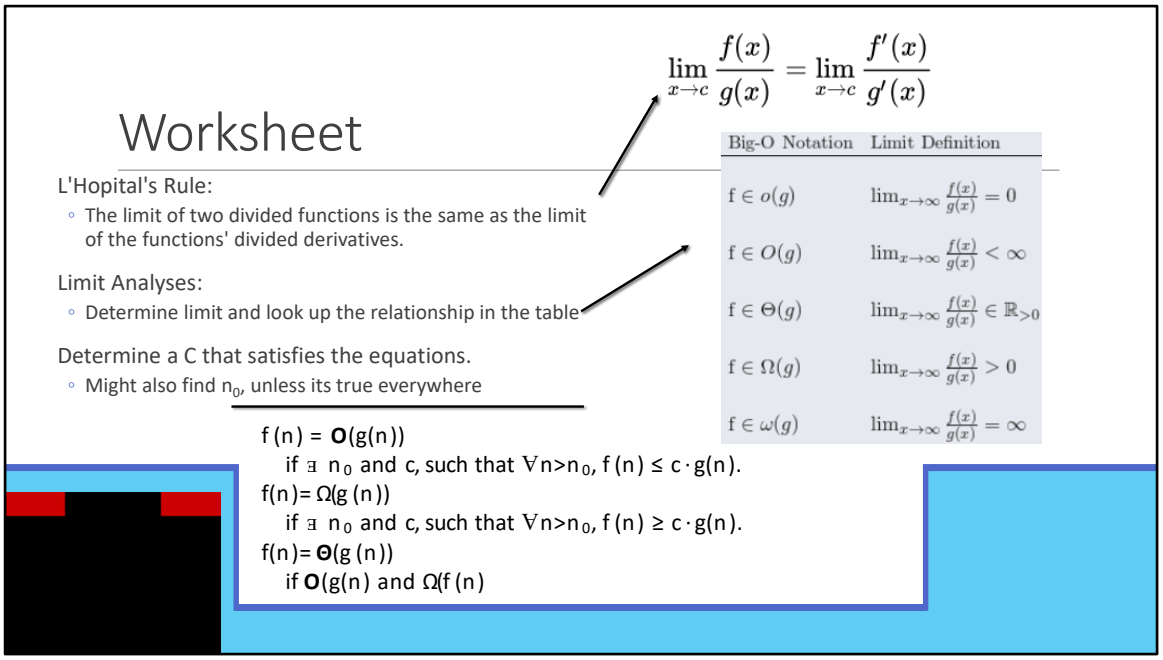
We approximate algorithm's runtimes because it's easy and useful

But there's always an *actual* exact time cost for an algorithm

Remember, problems do not have an exact time cost or a runtime

But some problems have a "known best algorithm" or, in some exceptional cases, you can prove that any potential algorithm cannot outperform a certain efficiency – e.g., search, max, sorting

Bart: One last thing I want to clear up. Big Oh is useful for approximating the runtime of an algorithm.
Bart: Algorithms also have exact time cost, as we saw last time, that can be more complicated.
Bart: However, there's an important concept here. A problem does not have a runtime or Big Oh.
Bart: And yet people will sometimes say things like "Maximum of a set is linear".
Bart: What they really mean is that we can prove that all algorithms solving the Maximum Value problem are upper bounded by a linear function in all cases.
Bart: Out in the wild, it's not a terribly important distinction.
Bart: But in this class, we want to be precise using the ideas of bounds and cases.



Bart: Your assignment today is going to have you describe and prove Big Oh relationships.
Bart: Once again, this is pretty tricky, and will take some time.
Bart: Sometimes you'll use the limit rules, and sometimes you'll need to find the C and N0.
Bart: Be sure to work together and google as needed!