Lecture Notes for CSCI 311: Algorithms

Set 4-Asymptotic Analysis

Professor Talmage

January 30, 2024

---

1 Motivation

We addressed this briefly before, when we were analyzing `InsertionSort`, but the point of algorithm analysis is to understand what constitutes a "fast" or "slow" algorithm. Specifically, we want

- to be able to compare two algorithms for a given problem and quickly know which is more efficient,
- to be able to easily communicate an algorithm's runtime,
- to focus on cost as input size grows, since differences are more pronounced then, and
- broad categories, that ignore the types of differences that come from different implementations and represent the inherent cost of the algorithm idea.

Thought experiment:

- Suppose you have a relative who was half your age when you were 10. When you are 20, how old will they be? When you are 100? This is actually an additive difference, and you are only ever 5 years older, which becomes less significant over time.
- Now, suppose you have a pet deer that can run twice as fast as you can. If you race, how far ahead are they after 5 meters? After 100? 1000? 10000? 100000? (If you can run that far...) The difference keeps growing, being multiplicative, but you're still within an order of magnitude.
- Finally, suppose that you are racing your deer, but while you run on level ground at a constant pace, they are skiing down a hill with continually increasing slope, so their speed keeps increasing. Because even the multiplicative factor by which their speed is greater than yours grows, it will not matter if you were to double your speed–the deer's speed will eventually surpass yours.

The takeaway here is that as n approaches ∞, additive terms become insignificant, constant multiplicative terms make little real difference, but terms which grow with n are worth considering. We use *asymptotic notation* to capture these distinctions.

Definition 1. For a given positive function g, denote the set $O(g(n))$ as

$$O(g(n)) = \{f(n) \mid \text{ there exist constants } c, n_0 > 0 \text{ s.t. for all } n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$

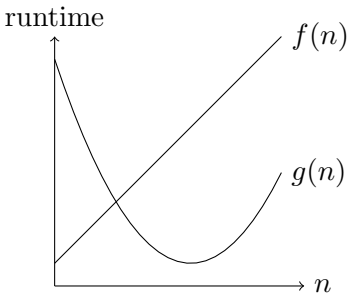
Notation: If $f(n)$ is in the set $O(g(n))$, then we

- say "$f(n)$ is Big-Oh of $g(n)$"
- write $f(n) = O(g(n))$

The intuition for this definition is that eventually $g(n)$ becomes, and stays, at least as large as $f(n)$, ignoring constant factors. $g(n)$ thus provides a sort of upper bound on $f(n)$, so $f(n) = O(g(n))$ is used similarly to $f(n) \leq g(n)$, though $\leq$ is a stronger statement.

2 Examples

Consider the functions $f(n) = n + 1$ and $g(n) = \frac{1}{4}n^2 - 3n + 10$. If we plot these, they look something like this:



Exercise: Does your intuition suggest that $f(n) = O(g(n))$, $g(n) = O(f(n))$, neither, or both?

We can see that $g(n)$ will eventually become, and stay, larger than $f(n)$, so intuition tells us that $f(n) = O(g(n))$. To prove that this is true, we need to find c and n_0 according to the definition of big-Oh. We could solve directly for the intersection points of the two functions and use $c = 1$, since $g(n)$ is strictly larger than $f(n)$ past the second intersection point. Or, since we just need to find any c and n_0 that work, we can make our lives easier by choosing a larger c.

1. Take $c = 4$ to get a clean highest-order term: $cg(n) = n^2 - 12n + 40$
2. Now, if $cg(n) - f(n)$ is positive, then $f(n) < cg(n)$.

   $$cg(n) - f(n) = n^2 - 12n + 40 - (n + 1) = n^2 - 13n + 39$$

   If the positive n^2 term is larger than the negative $-13n$ term, then this whole function will be positive.
3. Restrict n to be at least 13. Then $n * n \geq 13 * n$, so the whole expression will be positive.
4. We have shown that with $c = 4, n_0 = 13$, for all $n \geq n_0, f(n) \leq cg(n)$. Thus, $f(n) = O(g(n))$.

Again, there are many choices of c and n_0 that would have proved the big-Oh relationship. Infinitely many, actually. We just need to find any *one* pair that makes it easy to show the relationship.

Exercise: Show that $2x^2 = O(x^2)$.

Proof. Note that $2x^2 > x^2$ for all $x > 0$, so we cannot simply find a point where g becomes larger than f. Instead, we need to use c to show that g is within a constant factor of f.

Take $c = 2$. Then $cg(x) = 2x^2 = f(x)$ for all x. Take n_0 to be anything convenient, such as 1, and we have $2x^2 \leq 2(x^2) \forall x \geq 1$, so $f(x) = O(g(x))$. □

It is important to remember that while big-Oh is used as $\leq$, the latter can be false, up to constant factors.

Exercise: Show that $2x^2 = O(x^3)$.

Proof. Let $c = n_0 = 1$. Is $2x^2 < x^3$? For all $x \geq 1$? Try $x = 1.5$. The inequality does not hold here.

We have a couple of options to fix our proof:

1. Increase c: If we let $c = 1000000$, is it now true that $2x^2 \leq cx^3$ for all $x \geq 1$? Yes, though we really only need $c = 2$.

2. Increase n_0: Try $c = 1, n_0 = 2$. Is $2x^2 \leq x^3$ for all $x \geq 2$? Yes, because the difference $cg(x) - f(x)$ is $x^2(x-2)$, which is positive for $x \geq 2$.

□

In this example, increasing either c or n_0 alone was sufficient to make the proof work. In general, increasing only one may work, or you may need to increase both c and n_0 together.

Remember, you get to choose values for c and n_0. Use the specifics of f and g to guide you into a choice that makes the math easy. If it gets messy, think about whether a different choice would make things easier.

Exercise: Use big-Oh to relate the functions $\frac{7}{18}n^3 + 5n + \frac{1}{2}$ and $10n^2 + 200$.

We can sometimes show that two functions are not related by big-Oh:

Claim 1. $x^3 \neq O(x^2)$

Proof. Assume in contradiction that there exist $c, n_0 > 0$ s.t. $x^3 \leq cx^2$ for all $x \geq n_0$. Then $x \leq c$ for $x \geq n_0$, but $x \to \infty$ and c is constant, so x will eventually be larger than c, a contradiction. Thus, there is no pair c, n_0 that makes the inequality hold. □

Exercise: Use big-Oh to relate $m(x) = 17x^3 + 8x^2 - 5x + 2$ and $\psi(x) = .04x^{2.75} + x + 40$.

2.1 Common Relations

Exercise: Show that for $f(n) = 10x^2 + 6x + 300$ and $g(n) = x^2$, $f(n) = O(g(n))$.

- Choose $c = 10 + 6 + 300$, so that there is an n^2 term corresponding to each term in f. For $x \geq 1$, these will be larger, so we have a big-Oh relationship.
- This approach will always let us dominate any lower-order polynomial terms, and any coefficients.

Theorem 1. *Any polynomial of degree d is $O\left(n^d\right)$. That is,*

$$f(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = O\left(n^d\right)$$

for any constant a_i's.

Proof. Let $c = \sum_{i=0}^{n} a_i$ and $n_0 = 1$. Then $cn^d = a_d n^d + a_{d-1} d^n + \cdots + a_1 n^d + a_0 n^d$. Since $n^d \geq n^i$ for all $d \geq i \geq 0$ when $n \geq 1$, then $cn^d \geq f(n)$ for all $n \geq n_0$. □

Exercise: Show that $5x^2 + 14x + 3$ and $2x^2 + 4x + 1$ are each big-Oh of each other.

Theorem 2. $n^a = O(n^b)$ *if* $a \leq b$ *for any* $a, b > 0$.

We leave the proof as an exercise, as we have already been using its ideas. These last two theorems give us the most common rule of thumb in asymptotic analysis: We can drop all coefficients and lower-order terms in a polynomial, considering only the highest-degree term. This gives us the ability, which we stated as our goal, to easily communicate and immediately compare two runtime functions. We just need to expand to functions besides polynomials.

Claim 2. $x = O(2^x)$.

Proof. Take $c = 1, n_0 = 1$. At n_0, $x = 2$ and $2^x = 2$. So, at this point, $x \leq c2^x$. We then need to argue that this inequality holds for all larger values of x. Assume that for an arbitrary integer $u \geq 1$, $u \leq 2^u$. Then $u + 1 \leq u + u = 2u \leq 2(2^u) = 2^{u+1}$. Thus, the inequality also holds for $x = u + 1$. By mathematical induction, for all integers $x \geq 1$, $x \leq (1)(2^u)$. □

Aside: Another way we could show our claim is using derivatives to argue that 2^x is growing faster, and thus will be larger, at all points after the functions intersect. While this can work, we must be a little careful, as input sizes are typically integers, while one can only differentiate a continuous function. Thus, technically, one must first expand the runtime function's domain from integers to real numbers, then differentiate. This is usually fine, but is technically a necessary additional step.

Theorem 3. *For constants a and b,*

- $a^n = O(b^n)$ *for any* $0 \leq a \leq b$.
- $n^a = O(b^n)$ *for any* $a > 0, b > 1$.
- $\log^a n = O(n^b)$ *for any* $a \geq 0, b > 1$.
- $\log_a n = O(\log_b n)$ *for* $a, b > 1$.

We will not prove each of these, but the last is worth exploring a little deeper. Recall that $\log_b a$ is the exponent to which we must raise b to get a. That is, $\log_b a = x$ if and only if $b^x = a$. Thus, the logarithm is the inverse of exponentiation. Specifically, $a = b^{\log_b a}$ and $\log_b b^a = a$. Now, recall the base-change rule for logarithms: $\log_b a = \frac{\log_c a}{\log_c b}$. Since b and c are constants, $\log_c b$ is a constant factor difference between $\log_b a$ and $\log_c a$. Thus, changing the base of a logarithm only changes the function by a constant factor, which we can ignore asymptotically. From this point forward in the course, unless a particular logarithm base will make our work easier, we will typically neither write nor care what the base is, as it is only a constant-factor difference. When we find that we need a base, we will typically assume base 2.

Aside: There are several other logarithmic identities which may be worth recalling:

Theorem 4. *For all real $a, b, c > 0$ with $b, c \neq 1$, and all real n:*

- $a = b^{\log_b a}$
- $\log_c(ab) = \log_c a + \log_c b$
- $\log_b(a^n) = n \log_b a$
- $\log_b(1/a) = -\log_b a$
- $\log_b a = \frac{\log_c a}{\log_c b}$
- $\log_b a = \frac{1}{\log_a b}$
- $a^{\log_b c} = c^{\log_b a}$

2.2 Other Asymptotic Relationships

Big-Oh is (by far) the most commonly-used asymptotic relationship, but there are others that express the other possible relationships between two functions. One way to see these is the following:

$$\begin{array}{ccccc} \omega & & \Theta & & o \\ & \Omega & & O & \end{array}$$

Here, the upper row are stronger relationships, and the symbols are in order from f larger than g to f smaller than g. We will define and practice each of these new symbols.

Definition 2. $f(n) = \Omega(g(n))$ if there exist constants $c, n_0 > 0$ s.t. $f(n) \geq cg(n)$ for all $n \geq n_0$.

This is the reverse of big-Oh–the only difference is the direction of the inequality. Here, we are saying that $f(n)$ asymptotically grows at least as fast as $g(n)$, up to constant factors.

Example: $\log^3 n = \Omega(\log n)$

Proof. Let $c = 1$, since there are no coefficients to worry about. We just need to find a point where $\log^3 n$ becomes (and thence stays) larger than $\log n$. Since $\log^3 n = (\log n)(\log^2 n)$, we just need to find a point where $\log^2 n$ becomes greater than 1, which happens when $\log n > 1$, which is true as soon as n is greater than the logarithm base. Using base 2, we can set $n_0 = 2$, and for all $n \geq 2$, $\log n \geq 1$, so $\log^3 n \geq \log n$, and we have the claim. □

Exercise: Show that for $f(n) = 3^n$ and $g(n) = 2^n$, $f(n) = \Omega(g(n))$.

Theorem 5. *$f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.*

Exercise: Prove this claim, by solving $f(n) \leq cg(n), \forall n \geq n_0$ for $g(n)$, and vice versa for the converse.

If two functions are both upper and lower bounds for each other (which can happen for non-equal functions because we are ignoring constant factors and lower-order terms), we say we have a *tight bound*. In essence, our two runtimes are indistinguishable for sufficiently large n. Any function can have many upper (or lower) bounds, close or far, but all tight bounds of a given function are asymptotically tight with each other, since big-O and big-Ω are transitive.

Definition 3. $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
Alternately,

$$\Theta(g(n)) = \{f(n) \mid \text{ there exist constants } c_1, c_2, n_0 > 0 \text{ s.t. } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0.\}$$

Exercise: Show that $\frac{1}{3}n^2 + 8n + 6 = \Theta(n^2)$.

Usage

- We use upper bounds for algorithms: "`InsertionSort` is $O(n^2)$." (worst case)
- Lower bounds are used both
  - For algorithms: "`InsertionSort` is $\Omega(n^2)$ in the worst case."
  - For problems: "Sorting is $\Omega(n \log n)$." This means that *any* algorithm which solves the sorting problem must take at least $n \log n$ time, up to constant factors, in the asymptote.
- In equations, we can use asymptotics for *anonymous functions*:
  - Example: $n^2 + 3n + 6 = n^2 + \Theta(n)$
  - This is most often used for hiding lower order terms that are less important to the overall runtime. We will see more of this when we study recurrences, though we will have to be careful with them there.

- For any constant c, $c\Theta(f(n)) = \Theta(f(n))$ (same for O, Ω)
- $g(n) * \Theta(f(n)) = \Theta(f(n)g(n))$ (same for O, Ω)
- $O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$
- $\Omega(f(n) + g(n)) = \Omega(\max\{f(n), g(n)\})$
  - For these last two, "max" is to be interpreted asymptotically–keep the function which upper bounds the other.

Strict Bounds These next bounds are called *strict*, since they imply that the two functions are not asymptotically equivalent. That is, the difference between them is greater than any constant factor.

Definition 4.

- $f(n) = o(g(n))$ if for all $c > 0$, there exists $n_0 > 0$ s.t. $f(n) < cg(n)$ for all $n \geq n_0$.
- $f(n) = \omega(g(n))$ if for all $c > 0$, there exists $n_0 > 0$ s.t. $f(n) > cg(n)$ for all $n \geq n_0$.

Exercise: How many differences do you see compared to the definitions for O and Ω? What effect do you think they might have?

- Strict inequalities
- Must hold for *every* constant c, not just one.
- Each c can have a different n_0.

Exercise: Show the following relationships:

- $x^2 = o(x^4)$ (choose n_0 s.t. $n_0^2 > c$)
- $n = \omega(\log n)$ (since $n/\log n$ increases without bound, there exists n_0 s.t. $n \log n = c$.)

An equivalent formulation of the little-o and little-ω definitions is that $f(n) = o(g(n))$ if $f(n) = O(g(n))$ but $f(n) \neq \Theta(g(n))$, and similarly $f(n) = \omega(g(n))$ if $f(n) = \Omega(g(n))$ but $f(n) \neq \Theta(g(n))$. It thus follows that $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$. We leave the proof as an exercise.

2.3 Limit Versions

There is one more alternate, equivalent way to define asymptotic relationships. If you need to calculate the relationships between two functions and the math for c, n_0 proofs is not working out, these can be useful. We will not use them much in this course, because they tend to lead to poor solutions which punt to calculus without working out the complete proof. Thus, I do not generally want to see these on homework unless you can argue why no other method was practical.

Definition 5. Given two positive functions $f(n)$ and $g(n)$, the following cases apply:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = o(g(n)) \\ c \in \mathbb{R}^+, \text{ constant} & f(n) = \Theta(g(n)) \\ \infty & f(n) = \omega(g(n)) \end{cases}$$

If either of the first two cases applies, then $f(n) = O(g(n))$. If either of the last two cases applies, then $f(n) = \Omega(g(n))$.

2.4 Intuition

We can think of asymptotics as inequality symbols, though it is important to remember that these are not entirely accurate.

$f(n) =? g(n)$	Inequality
o	$<$
O	$\leq$
Θ	$=$
Ω	$\geq$
ω	$>$

The vast majority of the asymptotic comparisons we will encounter fall somewhere in the following chain, or are combinations of these pieces, where $a > 1$ and $b > 2$:

$$O(1) = o(\log n) = o(\log^a n) = o(n) = o(n^a) = o(2^n) = o(b^n)$$

To deal with combinations, we can remove common terms, e.g. to compare $n \log n$ to n, we remove the common n and note that what is left is $\log n$ versus $O(1)$, and $\log n$ is asymptotically larger.

Exercise: Choose the asymptotically larger of each pair, without proof, then prove your claim:

1. n^2 vs. n^4
2. n vs. $\log^2 n$
3. $n^3 \log n$ vs. $n^2 \log^3 n$
4. $n^5 \log^3 n$ vs. 4^n
5. $1.5^{n/2}$ vs. $\log(n^n)$
6. $\log^{10}(n^3)$ vs. $n2^n$