14.7 Asymptotic Notation

Asymptotic notation is a shorthand used to give a quick measure of the behavior of a function $f(n)$ as n grows large. For example, the asymptotic notation $\sim$ of Definition 14.4.2 is a binary relation indicating that two functions grow at the *same* rate. There is also a binary relation "little oh" indicating that one function grows at a significantly *slower* rate than another and "Big Oh" indicating that one function grows not much more rapidly than another.

14.7.1 Little O

Definition 14.7.1. For functions $f, g : \mathbb{R} \to \mathbb{R}$, with g nonnegative, we say f is *asymptotically smaller* than g, in symbols,

$$f(x) = o(g(x)),$$

iff

$$\lim_{x\to\infty} f(x)/g(x) = 0.$$

For example, $1000x^{1.9} = o(x^2)$ because $1000x^{1.9}/x^2 = 1000/x^{0.1}$ and since $x^{0.1}$ goes to infinity with x and 1000 is constant, we have $\lim_{x\to\infty} 1000x^{1.9}/x^2 = 0$. This argument generalizes directly to yield

Lemma 14.7.2. *$x^a = o(x^b)$ for all nonnegative constants $a < b$.*

Using the familiar fact that $\log x < x$ for all $x > 1$, we can prove

Lemma 14.7.3. *$\log x = o(x^\epsilon)$ for all $\epsilon > 0$.*

Proof. Choose $\epsilon > \delta > 0$ and let $x = z^\delta$ in the inequality $\log x < x$. This implies

$$\log z < z^\delta/\delta = o(z^\epsilon) \qquad \text{by Lemma 14.7.2.} \tag{14.28}$$

■

Corollary 14.7.4. *$x^b = o(a^x)$ for any $a, b \in \mathbb{R}$ with $a > 1$.*

Lemma 14.7.3 and Corollary 14.7.4 can also be proved using l'Hôpital's Rule or the Maclaurin Series for $\log x$ and e^x. Proofs can be found in most calculus texts.

14.7.2 Big O

"Big Oh" is the most frequently used asymptotic notation. It is used to give an upper bound on the growth of a function, such as the running time of an algorithm. There is a standard definition of Big Oh given below in 14.7.9, but we'll begin with an alternative definition that makes apparent several basic properties of Big Oh.

Definition 14.7.5. Given functions $f, g : \mathbb{R} \to \mathbb{R}$ with g nonnegative, we say that

$$f = O(g)$$

iff

$$\limsup_{x\to\infty} |f(x)| / g(x) < \infty.$$

Here we're using the technical notion of *limit superior*[6] instead of just limit. But because limits and lim sup's are the same when limits exist, this formulation makes it easy to check basic properties of Big Oh. We'll take the following Lemma for granted.

Lemma 14.7.6. *If a function $f : \mathbb{R} \to \mathbb{R}$ has a finite or infinite limit as its argument approaches infinity, then its limit and limit superior are the same.*

Now Definition 14.7.5 immediately implies:

Lemma 14.7.7. *If $f = o(g)$ or $f \sim g$, then $f = O(g)$.*

Proof. $\lim f/g = 0$ or $\lim f/g = 1$ implies $\lim f/g < \infty$, so by Lemma 14.7.6, $\limsup f/g < \infty$. ∎

Note that the converse of Lemma 14.7.7 is not true. For example, $2x = O(x)$, but $2x \not\sim x$ and $2x \neq o(x)$.

We also have:

Lemma 14.7.8. *If $f = o(g)$, then it is* not *true that $g = O(f)$.*

Proof.

$$\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{1}{\lim_{x\to\infty} f(x)/g(x)} = \frac{1}{0} = \infty,$$

so by Lemma 14.7.6, $g \neq O(f)$. ∎

[6] The precise definition of lim sup is

$$\limsup_{x\to\infty} h(x) ::= \lim_{x\to\infty} \text{lub}_{y \geq x} h(y),$$

where "lub" abbreviates "least upper bound."

We need lim sup’s in Definition 14.7.5 to cover cases when limits don’t exist. For example, if $f(x)/g(x)$ oscillates between 3 and 5 as x grows, then $\lim_{x\to\infty} f(x)/g(x)$ does not exist, but $f = O(g)$ because $\limsup_{x\to\infty} f(x)/g(x) = 5$.

An equivalent, more usual formulation of big O does not mention lim sup’s:

Definition 14.7.9. Given functions $f, g : \mathbb{R} \to \mathbb{R}$ with g nonnegative, we say

$$f = O(g)$$

iff there exists a constant $c \geq 0$ and an x_0 such that for all $x \geq x_0$, $|f(x)| \leq cg(x)$.

This definition is rather complicated, but the idea is simple: $f(x) = O(g(x))$ means $f(x)$ is less than or equal to $g(x)$, except that we’re willing to ignore a constant factor, namely c and to allow exceptions for small x, namely, $x < x_0$. So in the case that $f(x)/g(x)$ oscillates between 3 and 5, $f = O(g)$ according to Definition 14.7.9 because $f \leq 5g$.

Proposition 14.7.10. $100x^2 = O(x^2)$.

Proof. Choose $c = 100$ and $x_0 = 1$. Then the proposition holds, since for all $x \geq 1$, $\left|100x^2\right| \leq 100x^2$. ■

Proposition 14.7.11. $x^2 + 100x + 10 = O(x^2)$.

Proof. $(x^2+100x+10)/x^2 = 1+100/x+10/x^2$ and so its limit as x approaches infinity is $1+0+0 = 1$. So in fact, $x^2+100x+10 \sim x^2$, and therefore $x^2+100x+10 = O(x^2)$. Indeed, it’s conversely true that $x^2 = O(x^2 + 100x + 10)$. ■

Proposition 14.7.11 generalizes to an arbitrary polynomial:

Proposition 14.7.12. $a_k x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0 = O(x^k)$.

We’ll omit the routine proof.

Big O notation is especially useful when describing the running time of an algorithm. For example, the usual algorithm for multiplying $n \times n$ matrices uses a number of operations proportional to n^3 in the worst case. This fact can be expressed concisely by saying that the running time is $O(n^3)$. So this asymptotic notation allows the speed of the algorithm to be discussed without reference to constant factors or lower-order terms that might be machine specific. It turns out that there is another matrix multiplication procedure that uses $O(n^{2.55})$ operations. The fact that this procedure is asymptotically faster indicates that it involves new ideas that go beyond a simply more efficient implementation of the $O(n^3)$ method.

Of course the asymptotically faster procedure will also definitely be much more efficient on large enough matrices, but being asymptotically faster does not mean that it is a better choice. The $O(n^{2.55})$-operation multiplication procedure is almost never used in practice because it only becomes more efficient than the usual $O(n^3)$ procedure on matrices of impractical size.[7]

14.7.3 Theta

Sometimes we want to specify that a running time $T(n)$ is precisely quadratic up to constant factors (both upper bound *and* lower bound). We could do this by saying that $T(n) = O(n^2)$ and $n^2 = O(T(n))$, but rather than say both, mathematicians have devised yet another symbol Θ to do the job.

Definition 14.7.13.

$$f = \Theta(g) \quad \text{iff} \quad f = O(g) \text{ and } g = O(f).$$

The statement $f = \Theta(g)$ can be paraphrased intuitively as "f and g are equal to within a constant factor."

The Theta notation allows us to highlight growth rates and suppress distracting factors and low-order terms. For example, if the running time of an algorithm is

$$T(n) = 10n^3 - 20n^2 + 1,$$

then we can more simply write

$$T(n) = \Theta(n^3).$$

In this case, we would say that T *is of order* n^3 or that $T(n)$ *grows cubically*, which is often the main thing we really want to know. Another such example is

$$\pi^2 3^{x-7} + \frac{(2.7x^{113} + x^9 - 86)^4}{\sqrt{x}} - 1.08^{3x} = \Theta(3^x).$$

Just knowing that the running time of an algorithm is $\Theta(n^3)$, for example, is useful, because if n doubles we can predict that the running time will *by and large*[8] increase by a factor of at most 8 for large n. In this way, Theta notation preserves information about the scalability of an algorithm or system. Scalability is, of course, a big issue in the design of algorithms and systems.

[7]It is even conceivable that there is an $O(n^2)$ matrix multiplication procedure, but none is known.

[8]Since $\Theta(n^3)$ only implies that the running time $T(n)$ is between cn^3 and dn^3 for constants $0 < c < d$, the time $T(2n)$ could regularly exceed $T(n)$ by a factor as large as $8d/c$. The factor is sure to be close to 8 for all large n only if $T(n) \sim n^3$.

14.7.4 Pitfalls with Asymptotic Notation

There is a long list of ways to make mistakes with asymptotic notation. This section presents some of the ways that big O notation can lead to trouble. With minimal effort, you can cause just as much chaos with the other symbols.

The Exponential Fiasco

Sometimes relationships involving big O are not so obvious. For example, one might guess that $4^x = O(2^x)$ since 4 is only a constant factor larger than 2. This reasoning is incorrect, however; 4^x actually grows as the square of 2^x.

Constant Confusion

Every constant is $O(1)$. For example, $17 = O(1)$. This is true because if we let $f(x) = 17$ and $g(x) = 1$, then there exists a $c > 0$ and an x_0 such that $|f(x)| \leq cg(x)$. In particular, we could choose $c = 17$ and $x_0 = 1$, since $|17| \leq 17 \cdot 1$ for all $x \geq 1$. We can construct a false theorem that exploits this fact.

False Theorem 14.7.14.

$$\sum_{i=1}^{n} i = O(n)$$

Bogus proof. Define $f(n) = \sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n$. Since we have shown that every constant i is $O(1)$, $f(n) = O(1) + O(1) + \cdots + O(1) = O(n)$. ■

Of course in reality $\sum_{i=1}^{n} i = n(n+1)/2 \neq O(n)$.

The error stems from confusion over what is meant in the statement $i = O(1)$. For any *constant* $i \in \mathbb{N}$ it is true that $i = O(1)$. More precisely, if f is any constant function, then $f = O(1)$. But in this False Theorem, i is not constant—it ranges over a set of values $0, 1, \ldots, n$ that depends on n.

And anyway, we should not be adding $O(1)$'s as though they were numbers. We never even defined what $O(g)$ means by itself; it should only be used in the context "$f = O(g)$" to describe a relation between functions f and g.

Equality Blunder

The notation $f = O(g)$ is too firmly entrenched to avoid, but the use of "=" is regrettable. For example, if $f = O(g)$, it seems quite reasonable to write $O(g) = f$. But doing so might tempt us to the following blunder: because $2n = O(n)$, we can say $O(n) = 2n$. But $n = O(n)$, so we conclude that $n = O(n) = 2n$, and therefore $n = 2n$. To avoid such nonsense, we will never write "$O(f) = g$."

Similarly, you will often see statements like

$$H_n = \ln(n) + \gamma + O\left(\frac{1}{n}\right)$$

or

$$n! = (1 + o(1))\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

In such cases, the true meaning is

$$H_n = \ln(n) + \gamma + f(n)$$

for some $f(n)$ where $f(n) = O(1/n)$, and

$$n! = (1 + g(n))\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

where $g(n) = o(1)$. These last transgressions are OK as long as you (and your reader) know what you mean.

Operator Application Blunder

It's tempting to assume that familiar operations preserve asymptotic relations, but it ain't necessarily so. For example, $f \sim g$ in general does not even imply that $3^f = \Theta(3^g)$. On the other hand, some operations preserve and even strengthen asymptotic relations, for example,

$$f = \Theta(g) \text{ IMPLIES } \ln f \sim \ln g.$$

See Problem 14.27.

14.7.5 Omega (Optional)

Sometimes people incorrectly use Big Oh in the context of a lower bound. For example, they might say, "The running time $T(n)$ is at least $O(n^2)$." This is another blunder! Big Oh can only be used for *upper* bounds. The proper way to express the lower bound would be

$$n^2 = O(T(n)).$$

The lower bound can also be described with another special notation "big Omega."

Definition 14.7.15. Given functions $f, g : \mathbb{R} \to \mathbb{R}$ with f nonnegative, define

$$f = \Omega(g)$$

to mean

$$g = O(f).$$

For example, $x^2 = \Omega(x)$, $2^x = \Omega(x^2)$ and $x/100 = \Omega(100x + \sqrt{x})$.

So if the running time of your algorithm on inputs of size n is $T(n)$, and you want to say it is at least quadratic, say

$$T(n) = \Omega(n^2).$$

There is a similar "little omega" notation for lower bounds corresponding to little o:

Definition 14.7.16. For functions $f, g : \mathbb{R} \to \mathbb{R}$ with f nonnegative, define

$$f = \omega(g)$$

to mean

$$g = o(f).$$

For example, $x^{1.5} = \omega(x)$ and $\sqrt{x} = \omega(\ln^2(x))$.

The little omega symbol is not as widely used as the other asymptotic symbols we defined.

Problems for Section 14.1

Class Problems

Problem 14.1.

We begin with two large glasses. The first glass contains a pint of water, and the second contains a pint of wine. We pour 1/3 of a pint from the first glass into the second, stir up the wine/water mixture in the second glass, and then pour 1/3 of a pint of the mix back into the first glass and repeat this pouring back-and-forth process a total of n times.

(a) Describe a closed-form formula for the amount of wine in the first glass after n back-and-forth pourings.

(b) What is the limit of the amount of wine in each glass as n approaches infinity?

Problem 14.2.