
3 Growth of Functions

The order of growth of the running time of an algorithm, defined in Chapter 2, gives a simple characterization of the algorithm's efficiency and also allows us to compare the relative performance of alternative algorithms. Once the input size n becomes large enough, merge sort, with its $\Theta(n \lg n)$ worst-case running time, beats insertion sort, whose worst-case running time is $\Theta(n^2)$. Although we can sometimes determine the exact running time of an algorithm, as we did for insertion sort in Chapter 2, the extra precision is not usually worth the effort of computing it. For large enough inputs, the multiplicative constants and lower-order terms of an exact running time are dominated by the effects of the input size itself.

When we look at input sizes large enough to make only the order of growth of the running time relevant, we are studying the *asymptotic* efficiency of algorithms. That is, we are concerned with how the running time of an algorithm increases with the size of the input *in the limit*, as the size of the input increases without bound. Usually, an algorithm that is asymptotically more efficient will be the best choice for all but very small inputs.

This chapter gives several standard methods for simplifying the asymptotic analysis of algorithms. The next section begins by defining several types of “asymptotic notation,” of which we have already seen an example in Θ -notation. We then present several notational conventions used throughout this book, and finally we review the behavior of functions that commonly arise in the analysis of algorithms.

3.1 Asymptotic notation

The notations we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domains are the set of natural numbers $\mathbb{N} = \{0, 1, 2, \dots\}$. Such notations are convenient for describing the worst-case running-time function $T(n)$, which usually is defined only on integer input sizes. We sometimes find it convenient, however, to *abuse* asymptotic notation in a va-

riety of ways. For example, we might extend the notation to the domain of real numbers or, alternatively, restrict it to a subset of the natural numbers. We should make sure, however, to understand the precise meaning of the notation so that when we abuse, we do not *misuse* it. This section defines the basic asymptotic notations and also introduces some common abuses.

Asymptotic notation, functions, and running times

We will use asymptotic notation primarily to describe the running times of algorithms, as when we wrote that insertion sort's worst-case running time is $\Theta(n^2)$. Asymptotic notation actually applies to functions, however. Recall that we characterized insertion sort's worst-case running time as $an^2 + bn + c$, for some constants a , b , and c . By writing that insertion sort's running time is $\Theta(n^2)$, we abstracted away some details of this function. Because asymptotic notation applies to functions, what we were writing as $\Theta(n^2)$ was the function $an^2 + bn + c$, which in that case happened to characterize the worst-case running time of insertion sort.

In this book, the functions to which we apply asymptotic notation will usually characterize the running times of algorithms. But asymptotic notation can apply to functions that characterize some other aspect of algorithms (the amount of space they use, for example), or even to functions that have nothing whatsoever to do with algorithms.

Even when we use asymptotic notation to apply to the running time of an algorithm, we need to understand *which* running time we mean. Sometimes we are interested in the worst-case running time. Often, however, we wish to characterize the running time no matter what the input. In other words, we often wish to make a blanket statement that covers all inputs, not just the worst case. We shall see asymptotic notations that are well suited to characterizing running times no matter what the input.

Θ -notation

In Chapter 2, we found that the worst-case running time of insertion sort is $T(n) = \Theta(n^2)$. Let us define what this notation means. For a given function $g(n)$, we denote by $\Theta(g(n))$ the *set of functions*

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that} \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}^1$$

¹Within set notation, a colon means “such that.”

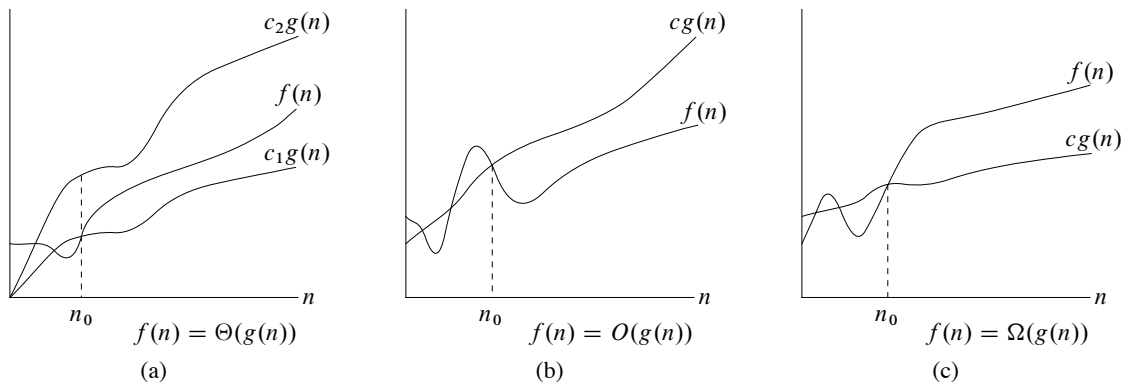


Figure 3.1 Graphic examples of the Θ , O , and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ -notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0 , c_1 , and c_2 such that at and to the right of n_0 , the value of $f(n)$ always lies between $c_1g(n)$ and $c_2g(n)$ inclusive. **(b)** O -notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0 , the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω -notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0 , the value of $f(n)$ always lies on or above $cg(n)$.

A function $f(n)$ belongs to the set $\Theta(g(n))$ if there exist positive constants c_1 and c_2 such that it can be “sandwiched” between $c_1g(n)$ and $c_2g(n)$, for sufficiently large n . Because $\Theta(g(n))$ is a set, we could write “ $f(n) \in \Theta(g(n))$ ” to indicate that $f(n)$ is a member of $\Theta(g(n))$. Instead, we will usually write “ $f(n) = \Theta(g(n))$ ” to express the same notion. You might be confused because we abuse equality in this way, but we shall see later in this section that doing so has its advantages.

Figure 3.1(a) gives an intuitive picture of functions $f(n)$ and $g(n)$, where $f(n) = \Theta(g(n))$. For all values of n at and to the right of n_0 , the value of $f(n)$ lies at or above $c_1g(n)$ and at or below $c_2g(n)$. In other words, for all $n \geq n_0$, the function $f(n)$ is equal to $g(n)$ to within a constant factor. We say that $g(n)$ is an **asymptotically tight bound** for $f(n)$.

The definition of $\Theta(g(n))$ requires that every member $f(n) \in \Theta(g(n))$ be **asymptotically nonnegative**, that is, that $f(n)$ be nonnegative whenever n is sufficiently large. (An **asymptotically positive** function is one that is positive for all sufficiently large n .) Consequently, the function $g(n)$ itself must be asymptotically nonnegative, or else the set $\Theta(g(n))$ is empty. We shall therefore assume that every function used within Θ -notation is asymptotically nonnegative. This assumption holds for the other asymptotic notations defined in this chapter as well.

In Chapter 2, we introduced an informal notion of Θ -notation that amounted to throwing away lower-order terms and ignoring the leading coefficient of the highest-order term. Let us briefly justify this intuition by using the formal definition to show that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. To do so, we must determine positive constants c_1 , c_2 , and n_0 such that

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

for all $n \geq n_0$. Dividing by n^2 yields

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2.$$

We can make the right-hand inequality hold for any value of $n \geq 1$ by choosing any constant $c_2 \geq 1/2$. Likewise, we can make the left-hand inequality hold for any value of $n \geq 7$ by choosing any constant $c_1 \leq 1/14$. Thus, by choosing $c_1 = 1/14$, $c_2 = 1/2$, and $n_0 = 7$, we can verify that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. Certainly, other choices for the constants exist, but the important thing is that *some* choice exists. Note that these constants depend on the function $\frac{1}{2}n^2 - 3n$; a different function belonging to $\Theta(n^2)$ would usually require different constants.

We can also use the formal definition to verify that $6n^3 \neq \Theta(n^2)$. Suppose for the purpose of contradiction that c_2 and n_0 exist such that $6n^3 \leq c_2 n^2$ for all $n \geq n_0$. But then dividing by n^2 yields $n \leq c_2/6$, which cannot possibly hold for arbitrarily large n , since c_2 is constant.

Intuitively, the lower-order terms of an asymptotically positive function can be ignored in determining asymptotically tight bounds because they are insignificant for large n . When n is large, even a tiny fraction of the highest-order term suffices to dominate the lower-order terms. Thus, setting c_1 to a value that is slightly smaller than the coefficient of the highest-order term and setting c_2 to a value that is slightly larger permits the inequalities in the definition of Θ -notation to be satisfied. The coefficient of the highest-order term can likewise be ignored, since it only changes c_1 and c_2 by a constant factor equal to the coefficient.

As an example, consider any quadratic function $f(n) = an^2 + bn + c$, where a , b , and c are constants and $a > 0$. Throwing away the lower-order terms and ignoring the constant yields $f(n) = \Theta(n^2)$. Formally, to show the same thing, we take the constants $c_1 = a/4$, $c_2 = 7a/4$, and $n_0 = 2 \cdot \max(|b|/a, \sqrt{|c|/a})$. You may verify that $0 \leq c_1 n^2 \leq an^2 + bn + c \leq c_2 n^2$ for all $n \geq n_0$. In general, for any polynomial $p(n) = \sum_{i=0}^d a_i n^i$, where the a_i are constants and $a_d > 0$, we have $p(n) = \Theta(n^d)$ (see Problem 3-1).

Since any constant is a degree-0 polynomial, we can express any constant function as $\Theta(n^0)$, or $\Theta(1)$. This latter notation is a minor abuse, however, because the

O-notation

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$$

We write $f(n) = O(g(n))$ to indicate that a function $f(n)$ is a member of the set $O(g(n))$. Note that $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$, since Θ -notation is a stronger notion than O -notation. Written set-theoretically, we have $\Theta(g(n)) \subseteq O(g(n))$. Thus, our proof that any quadratic function $an^2 + bn + c$, where $a > 0$, is in $\Theta(n^2)$ also shows that any such quadratic function is in $O(n^2)$. What may be more surprising is that when $a > 0$, any *linear* function $an + b$ is in $O(n^2)$, which is easily verified by taking $c = a + |b|$ and $n_0 = \max(1, -b/a)$.

Using O -notation, we can often describe the running time of an algorithm merely by inspecting the algorithm's overall structure. For example, the doubly nested loop structure of the insertion sort algorithm from Chapter 2 immediately yields an $O(n^2)$ upper bound on the worst-case running time: the cost of each iteration of the inner loop is bounded from above by $O(1)$ (constant), the indices i

²The real problem is that our ordinary notation for functions does not distinguish functions from values. In λ -calculus, the parameters to a function are clearly specified: the function n^2 could be written as $\lambda n.n^2$, or even $\lambda r.r^2$. Adopting a more rigorous notation, however, would complicate algebraic manipulations, and so we choose to tolerate the abuse.

Since O -notation describes an upper bound, when we use it to bound the worst-case running time of an algorithm, we have a bound on the running time of the algorithm on every input—the blanket statement we discussed earlier. Thus, the $O(n^2)$ bound on worst-case running time of insertion sort also applies to its running time on every input. The $\Theta(n^2)$ bound on the worst-case running time of insertion sort, however, does not imply a $\Theta(n^2)$ bound on the running time of insertion sort on *every* input. For example, we saw in Chapter 2 that when the input is already sorted, insertion sort runs in $\Theta(n)$ time.

Technically, it is an abuse to say that the running time of insertion sort is $O(n^2)$, since for a given n , the actual running time varies, depending on the particular input of size n . When we say “the running time is $O(n^2)$,” we mean that there is a function $f(n)$ that is $O(n^2)$ such that for any value of n , no matter what particular input of size n is chosen, the running time on that input is bounded from above by the value $f(n)$. Equivalently, we mean that the worst-case running time is $O(n^2)$.

Just as O -notation provides an asymptotic *upper* bound on a function, Ω -notation provides an *asymptotic lower bound*. For a given function $g(n)$, we denote by $\Omega(g(n))$ (pronounced “big-omega of g of n ” or sometimes just “omega of g of n ”) the set of functions

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$$

Figure 3.1(c) shows the intuition behind Ω -notation. For all values n at or to the right of n_0 , the value of $f(n)$ is on or above $cg(n)$.

From the definitions of the asymptotic notations we have seen thus far, it is easy to prove the following important theorem (see Exercise 3.1-5).

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. ■

As an example of the application of this theorem, our proof that $an^2 + bn + c = \Theta(n^2)$ for any constants a , b , and c , where $a > 0$, immediately implies that $an^2 + bn + c = \Omega(n^2)$ and $an^2 + bn + c = O(n^2)$. In practice, rather than using Theorem 3.1 to obtain asymptotic upper and lower bounds from asymptotically tight bounds, as we did for this example, we usually use it to prove asymptotically tight bounds from asymptotic upper and lower bounds.

When we say that the *running time* (no modifier) of an algorithm is $\Omega(g(n))$, we mean that *no matter what particular input of size n is chosen for each value of n* , the running time on that input is at least a constant times $g(n)$, for sufficiently large n . Equivalently, we are giving a lower bound on the best-case running time of an algorithm. For example, the best-case running time of insertion sort is $\Omega(n)$, which implies that the running time of insertion sort is $\Omega(n)$.

The running time of insertion sort therefore belongs to both $\Omega(n)$ and $O(n^2)$, since it falls anywhere between a linear function of n and a quadratic function of n . Moreover, these bounds are asymptotically as tight as possible: for instance, the running time of insertion sort is not $\Omega(n^2)$, since there exists an input for which insertion sort runs in $\Theta(n)$ time (e.g., when the input is already sorted). It is not contradictory, however, to say that the *worst-case* running time of insertion sort is $\Omega(n^2)$, since there exists an input that causes the algorithm to take $\Omega(n^2)$ time.

Asymptotic notation in equations and inequalities

We have already seen how asymptotic notation can be used within mathematical formulas. For example, in introducing O -notation, we wrote “ $n = O(n^2)$.” We might also write $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$. How do we interpret such formulas?

When the asymptotic notation stands alone (that is, not within a larger formula) on the right-hand side of an equation (or inequality), as in $n = O(n^2)$, we have already defined the equal sign to mean set membership: $n \in O(n^2)$. In general, however, when asymptotic notation appears in a formula, we interpret it as standing for some anonymous function that we do not care to name. For example, the formula $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ means that $2n^2 + 3n + 1 = 2n^2 + f(n)$, where $f(n)$ is some function in the set $\Theta(n)$. In this case, we let $f(n) = 3n + 1$, which indeed is in $\Theta(n)$.

Using asymptotic notation in this manner can help eliminate inessential detail and clutter in an equation. For example, in Chapter 2 we expressed the worst-case running time of merge sort as the recurrence

$$T(n) = 2T(n/2) + \Theta(n) .$$

If we are interested only in the asymptotic behavior of $T(n)$, there is no point in specifying all the lower-order terms exactly; they are all understood to be included in the anonymous function denoted by the term $\Theta(n)$.

The number of anonymous functions in an expression is understood to be equal to the number of times the asymptotic notation appears. For example, in the expression

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

there is only a single anonymous function (a function of i). This expression is thus *not* the same as $O(1) + O(2) + \cdots + O(n)$, which doesn't really have a clean interpretation.

In some cases, asymptotic notation appears on the left-hand side of an equation, as in

$$2n^2 + \Theta(n) = \Theta(n^2).$$

We interpret such equations using the following rule: *No matter how the anonymous functions are chosen on the left of the equal sign, there is a way to choose the anonymous functions on the right of the equal sign to make the equation valid.* Thus, our example means that for *any* function $f(n) \in \Theta(n)$, there is *some* function $g(n) \in \Theta(n^2)$ such that $2n^2 + f(n) = g(n)$ for all n . In other words, the right-hand side of an equation provides a coarser level of detail than the left-hand side.

We can chain together a number of such relationships, as in

$$\begin{aligned} 2n^2 + 3n + 1 &= 2n^2 + \Theta(n) \\ &= \Theta(n^2). \end{aligned}$$

We can interpret each equation separately by the rules above. The first equation says that there is *some* function $f(n) \in \Theta(n)$ such that $2n^2 + 3n + 1 = 2n^2 + f(n)$ for all n . The second equation says that for *any* function $g(n) \in \Theta(n)$ (such as the $f(n)$ just mentioned), there is *some* function $h(n) \in \Theta(n^2)$ such that $2n^2 + g(n) = h(n)$ for all n . Note that this interpretation implies that $2n^2 + 3n + 1 = \Theta(n^2)$, which is what the chaining of equations intuitively gives us.

***o*-notation**

The asymptotic upper bound provided by O -notation may or may not be asymptotically tight. The bound $2n^2 = O(n^2)$ is asymptotically tight, but the bound $2n = O(n^2)$ is not. We use o -notation to denote an upper bound that is not asymptotically tight. We formally define $o(g(n))$ ("little-oh of g of n ") as the set

$$o(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}.$$

For example, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$.

The definitions of O -notation and o -notation are similar. The main difference is that in $f(n) = O(g(n))$, the bound $0 \leq f(n) \leq cg(n)$ holds for *some* constant $c > 0$, but in $f(n) = o(g(n))$, the bound $0 \leq f(n) < cg(n)$ holds for *all* constants $c > 0$. Intuitively, in o -notation, the function $f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity; that is,

Some authors use this limit as a definition of the o -notation; the definition in this book also restricts the anonymous functions to be asymptotically nonnegative.

By analogy, ω -notation is to Ω -notation as o -notation is to O -notation. We use ω -notation to denote a lower bound that is not asymptotically tight. One way to define it is by

Formally, however, we define $\omega(g(n))$ (“little-omega of g of n ”) as the set

For example, $n^2/2 = \omega(n)$, but $n^2/2 \neq \omega(n^2)$. The relation $f(n) = \omega(g(n))$ implies that

if the limit exists. That is, $f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity.

Many of the relational properties of real numbers apply to asymptotic comparisons as well. For the following, assume that $f(n)$ and $g(n)$ are asymptotically positive.

$$\begin{aligned} f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) &\quad \text{imply} \quad f(n) = \Theta(h(n)) , \\ f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) &\quad \text{imply} \quad f(n) = O(h(n)) , \\ f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) &\quad \text{imply} \quad f(n) = \Omega(h(n)) , \\ f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) &\quad \text{imply} \quad f(n) = o(h(n)) , \\ f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) &\quad \text{imply} \quad f(n) = \omega(h(n)) . \end{aligned}$$
$$\begin{aligned} f(n) &= \Theta(f(n)) , \\ f(n) &= O(f(n)) , \\ f(n) &= \Omega(f(n)) . \end{aligned}$$

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)) .$$
$$\begin{aligned} f(n) = O(g(n)) & \text{ if and only if } g(n) = \Omega(f(n)), \\ f(n) = o(g(n)) & \text{ if and only if } g(n) = \omega(f(n)). \end{aligned}$$

$f(n) = O(g(n))$	is like	$a \leq b$,
$f(n) = \Omega(g(n))$	is like	$a \geq b$,
$f(n) = \Theta(g(n))$	is like	$a = b$,
$f(n) = o(g(n))$	is like	$a < b$,
$f(n) = \omega(g(n))$	is like	$a > b$.

One property of real numbers, however, does not carry over to asymptotic notation:

Although any two real numbers can be compared, not all functions are asymptotically comparable. That is, for two functions $f(n)$ and $g(n)$, it may be the case that neither $f(n) = O(g(n))$ nor $f(n) = \Omega(g(n))$ holds. For example, we cannot compare the functions n and $n^{1+\sin n}$ using asymptotic notation, since the value of the exponent in $n^{1+\sin n}$ oscillates between 0 and 2, taking on all values in between.

3.1-1

3.1-2

$$(n+a)^b = \Theta(n^b). \quad (3.2)$$

3.I-3

Explain why the statement, “The running time of algorithm A is at least $O(n^2)$,” is meaningless.

3.I-4

Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

3.I-5

Prove Theorem 3.1.

3.I-6

Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

3.I-7

Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

3.I-8

We can extend our notation to the case of two parameters n and m that can go to infinity independently at different rates. For a given function $g(n, m)$, we denote by $O(g(n, m))$ the set of functions

$$O(g(n, m)) = \{f(n, m) : \text{there exist positive constants } c, n_0, \text{ and } m_0 \\ \text{such that } 0 \leq f(n, m) \leq cg(n, m) \\ \text{for all } n \geq n_0 \text{ or } m \geq m_0\}.$$

Give corresponding definitions for $\Omega(g(n, m))$ and $\Theta(g(n, m))$.

3.2 Standard notations and common functions

This section reviews some standard mathematical functions and notations and explores the relationships among them. It also illustrates the use of the asymptotic notations.

Monotonicity

A function $f(n)$ is **monotonically increasing** if $m \leq n$ implies $f(m) \leq f(n)$. Similarly, it is **monotonically decreasing** if $m \leq n$ implies $f(m) \geq f(n)$. A function $f(n)$ is **strictly increasing** if $m < n$ implies $f(m) < f(n)$ and **strictly decreasing** if $m < n$ implies $f(m) > f(n)$.

Floors and ceilings

For any real number x , we denote the greatest integer less than or equal to x by $\lfloor x \rfloor$ (read “the floor of x ”) and the least integer greater than or equal to x by $\lceil x \rceil$ (read “the ceiling of x ”). For all real x ,

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1. \quad (3.3)$$

For any integer n ,

$$\lceil n/2 \rceil + \lfloor n/2 \rfloor = n,$$

and for any real number $x \geq 0$ and integers $a, b > 0$,

$$\left\lceil \frac{\lfloor x/a \rfloor}{b} \right\rceil = \left\lceil \frac{x}{ab} \right\rceil, \quad (3.4)$$

$$\left\lfloor \frac{\lceil x/a \rceil}{b} \right\rfloor = \left\lfloor \frac{x}{ab} \right\rfloor, \quad (3.5)$$

$$\left\lceil \frac{a}{b} \right\rceil \leq \frac{a + (b - 1)}{b}, \quad (3.6)$$

$$\left\lfloor \frac{a}{b} \right\rfloor \geq \frac{a - (b - 1)}{b}. \quad (3.7)$$

The floor function $f(x) = \lfloor x \rfloor$ is monotonically increasing, as is the ceiling function $f(x) = \lceil x \rceil$.

Modular arithmetic

For any integer a and any positive integer n , the value $a \bmod n$ is the **remainder** (or **residue**) of the quotient a/n :

$$a \bmod n = a - n \lfloor a/n \rfloor. \quad (3.8)$$

It follows that

$$0 \leq a \bmod n < n. \quad (3.9)$$

Given a well-defined notion of the remainder of one integer when divided by another, it is convenient to provide special notation to indicate equality of remainders. If $(a \bmod n) = (b \bmod n)$, we write $a \equiv b \pmod{n}$ and say that a is **equivalent** to b , modulo n . In other words, $a \equiv b \pmod{n}$ if a and b have the same remainder when divided by n . Equivalently, $a \equiv b \pmod{n}$ if and only if n is a divisor of $b - a$. We write $a \not\equiv b \pmod{n}$ if a is not equivalent to b , modulo n .

Polynomials

Given a nonnegative integer d , a **polynomial in n of degree d** is a function $p(n)$ of the form

$$p(n) = \sum_{i=0}^d a_i n^i ,$$

where the constants a_0, a_1, \dots, a_d are the **coefficients** of the polynomial and $a_d \neq 0$. A polynomial is asymptotically positive if and only if $a_d > 0$. For an asymptotically positive polynomial $p(n)$ of degree d , we have $p(n) = \Theta(n^d)$. For any real constant $a \geq 0$, the function n^a is monotonically increasing, and for any real constant $a \leq 0$, the function n^a is monotonically decreasing. We say that a function $f(n)$ is **polynomially bounded** if $f(n) = O(n^k)$ for some constant k .

Exponentials

For all real $a > 0$, m , and n , we have the following identities:

$$\begin{aligned} a^0 &= 1 , \\ a^1 &= a , \\ a^{-1} &= 1/a , \\ (a^m)^n &= a^{mn} , \\ (a^m)^n &= (a^n)^m , \\ a^m a^n &= a^{m+n} . \end{aligned}$$

For all n and $a \geq 1$, the function a^n is monotonically increasing in n . When convenient, we shall assume $0^0 = 1$.

We can relate the rates of growth of polynomials and exponentials by the following fact. For all real constants a and b such that $a > 1$,

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0 , \tag{3.10}$$

from which we can conclude that

$$n^b = o(a^n) .$$

Thus, any exponential function with a base strictly greater than 1 grows faster than any polynomial function.

Using e to denote $2.71828\dots$, the base of the natural logarithm function, we have for all real x ,

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!} , \tag{3.11}$$

$$e^x \geq 1 + x \, , \tag{3.12}$$

When $x \rightarrow 0$, the approximation of e^x by $1 + x$ is quite good:

$$e^x = 1 + x + \Theta(x^2) \, .$$

$$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x. \quad (3.14)$$

We shall use the following notations:

$$\begin{aligned} \lg n &= \log_2 n && \text{(binary logarithm)} , \\ \ln n &= \log_e n && \text{(natural logarithm)} . \\ \lg^k n &= (\lg n)^k && \text{(exponentiation)} , \\ \lg \lg n &= \lg(\lg n) && \text{(composition)} . \end{aligned}$$

An important notational convention we shall adopt is that *logarithm functions will apply only to the next term in the formula*, so that $\lg n + k$ will mean $(\lg n) + k$ and not $\lg(n + k)$. If we hold $b > 1$ constant, then for $n > 0$, the function $\log_b n$ is strictly increasing.

For all real $a > 0, b > 0, c > 0$, and n ,

$$\begin{aligned} a &= b^{\log_b a}, \\ \log_c(ab) &= \log_c a + \log_c b, \\ \log_b a^n &= n \log_b a, \\ \log_b a &= \frac{\log_c a}{\log_c b}, \\ \log_b(1/a) &= -\log_b a, \\ \log_b a &= \frac{1}{\log_a b}, \\ a^{\log_b c} &= c^{\log_b a}, \end{aligned} \tag{3.15}$$

where, in each equation above, logarithm bases are not 1.

By equation (3.15), changing the base of a logarithm from one constant to another changes the value of the logarithm by only a constant factor, and so we shall often use the notation “ $\lg n$ ” when we don’t care about constant factors, such as in O -notation. Computer scientists find 2 to be the most natural base for logarithms because so many algorithms and data structures involve splitting a problem into two parts.

There is a simple series expansion for $\ln(1 + x)$ when $|x| < 1$:

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots .$$

We also have the following inequalities for $x > -1$:

$$\frac{x}{1+x} \leq \ln(1+x) \leq x , \quad (3.17)$$

where equality holds only for $x = 0$.

We say that a function $f(n)$ is **polylogarithmically bounded** if $f(n) = O(\lg^k n)$ for some constant k . We can relate the growth of polynomials and polylogarithms by substituting $\lg n$ for n and 2^a for a in equation (3.10), yielding

$$\lim_{n \rightarrow \infty} \frac{\lg^b n}{(2^a)^{\lg n}} = \lim_{n \rightarrow \infty} \frac{\lg^b n}{n^a} = 0 .$$

From this limit, we can conclude that

$$\lg^b n = o(n^a)$$

for any constant $a > 0$. Thus, any positive polynomial function grows faster than any polylogarithmic function.

Factorials

The notation $n!$ (read “ n factorial”) is defined for integers $n \geq 0$ as

$$n! = \begin{cases} 1 & \text{if } n = 0 , \\ n \cdot (n-1)! & \text{if } n > 0 . \end{cases}$$

Thus, $n! = 1 \cdot 2 \cdot 3 \cdots n$.

A weak upper bound on the factorial function is $n! \leq n^n$, since each of the n terms in the factorial product is at most n . **Stirling’s approximation**,

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right) , \quad (3.18)$$

where e is the base of the natural logarithm, gives us a tighter upper bound, and a lower bound as well. As Exercise 3.2-3 asks you to prove,

$$\begin{aligned} n! &= o(n^n), \\ n! &= \omega(2^n), \\ \lg(n!) &= \Theta(n \lg n), \end{aligned} \quad (3.19)$$

where Stirling's approximation is helpful in proving equation (3.19). The following equation also holds for all $n \geq 1$:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\alpha_n} \quad (3.20)$$

where

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}. \quad (3.21)$$

Functional iteration

We use the notation $f^{(i)}(n)$ to denote the function $f(n)$ iteratively applied i times to an initial value of n . Formally, let $f(n)$ be a function over the reals. For non-negative integers i , we recursively define

$$f^{(i)}(n) = \begin{cases} n & \text{if } i = 0, \\ f(f^{(i-1)}(n)) & \text{if } i > 0. \end{cases}$$

For example, if $f(n) = 2n$, then $f^{(i)}(n) = 2^i n$.

The iterated logarithm function

We use the notation $\lg^* n$ (read “log star of n ”) to denote the iterated logarithm, defined as follows. Let $\lg^{(i)} n$ be as defined above, with $f(n) = \lg n$. Because the logarithm of a nonpositive number is undefined, $\lg^{(i)} n$ is defined only if $\lg^{(i-1)} n > 0$. Be sure to distinguish $\lg^{(i)} n$ (the logarithm function applied i times in succession, starting with argument n) from $\lg^i n$ (the logarithm of n raised to the i th power). Then we define the iterated logarithm function as

$$\lg^* n = \min \{i \geq 0 : \lg^{(i)} n \leq 1\}.$$

The iterated logarithm is a *very* slowly growing function:

$$\begin{aligned} \lg^* 2 &= 1, \\ \lg^* 4 &= 2, \\ \lg^* 16 &= 3, \\ \lg^* 65536 &= 4, \\ \lg^*(2^{65536}) &= 5. \end{aligned}$$

Since the number of atoms in the observable universe is estimated to be about 10^{80} , which is much less than 2^{65536} , we rarely encounter an input size n such that $\lg^* n > 5$.

Fibonacci numbers

We define the *Fibonacci numbers* by the following recurrence:

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_i &= F_{i-1} + F_{i-2} \quad \text{for } i \geq 2. \end{aligned} \tag{3.22}$$

Thus, each Fibonacci number is the sum of the two previous ones, yielding the sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Fibonacci numbers are related to the *golden ratio* ϕ and to its conjugate $\hat{\phi}$, which are the two roots of the equation

$$x^2 = x + 1 \tag{3.23}$$

and are given by the following formulas (see Exercise 3.2-6):

$$\begin{aligned} \phi &= \frac{1 + \sqrt{5}}{2} \\ &= 1.61803 \dots, \\ \hat{\phi} &= \frac{1 - \sqrt{5}}{2} \\ &= -.61803 \dots \end{aligned} \tag{3.24}$$

Specifically, we have

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}},$$

which we can prove by induction (Exercise 3.2-7). Since $|\hat{\phi}| < 1$, we have

$$\begin{aligned} \frac{|\hat{\phi}^i|}{\sqrt{5}} &< \frac{1}{\sqrt{5}} \\ &< \frac{1}{2}, \end{aligned}$$

which implies that

$$F_i = \left\lfloor \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \right\rfloor, \quad (3.25)$$

which is to say that the i th Fibonacci number F_i is equal to $\phi^i / \sqrt{5}$ rounded to the nearest integer. Thus, Fibonacci numbers grow exponentially.

Exercises

3.2-1

Show that if $f(n)$ and $g(n)$ are monotonically increasing functions, then so are the functions $f(n) + g(n)$ and $f(g(n))$, and if $f(n)$ and $g(n)$ are in addition nonnegative, then $f(n) \cdot g(n)$ is monotonically increasing.

3.2-2

Prove equation (3.16).

3.2-3

Prove equation (3.19). Also prove that $n! = \omega(2^n)$ and $n! = o(n^n)$.

3.2-4 ★

Is the function $\lceil \lg n \rceil!$ polynomially bounded? Is the function $\lceil \lg \lg n \rceil!$ polynomially bounded?

3.2-5 ★

Which is asymptotically larger: $\lg(\lg^* n)$ or $\lg^*(\lg n)$?

3.2-6

Show that the golden ratio ϕ and its conjugate $\hat{\phi}$ both satisfy the equation $x^2 = x + 1$.

3.2-7

Prove by induction that the i th Fibonacci number satisfies the equality

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}},$$

where ϕ is the golden ratio and $\hat{\phi}$ is its conjugate.

3.2-8

Show that $k \ln k = \Theta(n)$ implies $k = \Theta(n / \ln n)$.

Problems

3-1 Asymptotic behavior of polynomials

Let

$$p(n) = \sum_{i=0}^d a_i n^i,$$

where $a_d > 0$, be a degree- d polynomial in n , and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties.

- a. If $k \geq d$, then $p(n) = O(n^k)$.
- b. If $k \leq d$, then $p(n) = \Omega(n^k)$.
- c. If $k = d$, then $p(n) = \Theta(n^k)$.
- d. If $k > d$, then $p(n) = o(n^k)$.
- e. If $k < d$, then $p(n) = \omega(n^k)$.

3-2 Relative asymptotic growths

Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω , or Θ of B . Assume that $k \geq 1, \epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with “yes” or “no” written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^ϵ					
b.	n^k	c^n					
c.	\sqrt{n}	$n^{\sin n}$					
d.	2^n	$2^{n/2}$					
e.	$n^{\lg c}$	$c^{\lg n}$					
f.	$\lg(n!)$	$\lg(n^n)$					

3-3 Ordering by asymptotic growth rates

- a. Rank the following functions by order of growth; that is, find an arrangement g_1, g_2, \dots, g_{30} of the functions satisfying $g_1 = \Omega(g_2), g_2 = \Omega(g_3), \dots, g_{29} = \Omega(g_{30})$. Partition your list into equivalence classes such that functions $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$.

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$
$(\frac{3}{2})^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg^* n$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	1
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$\lg^*(\lg n)$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$	2^{n+1}

- b. Give an example of a single nonnegative function $f(n)$ such that for all functions $g_i(n)$ in part (a), $f(n)$ is neither $O(g_i(n))$ nor $\Omega(g_i(n))$.

Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following conjectures.

- a. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.
- b. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.
- c. $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$, where $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n .
- d. $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.
- e. $f(n) = O((f(n))^2)$.
- f. $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.
- g. $f(n) = \Theta(f(n/2))$.
- h. $f(n) + o(f(n)) = \Theta(f(n))$.

Some authors define Ω in a slightly different way than we do; let's use $\tilde{\Omega}$ (read “omega infinity”) for this alternative definition. We say that $f(n) = \tilde{\Omega}(g(n))$ if there exists a positive constant c such that $f(n) \geq cg(n) \geq 0$ for infinitely many integers n .

- a.** Show that for any two functions $f(n)$ and $g(n)$ that are asymptotically nonnegative, either $f(n) = O(g(n))$ or $f(n) = \tilde{\Omega}(g(n))$ or both, whereas this is not true if we use Ω in place of $\tilde{\Omega}$.

- b. Describe the potential advantages and disadvantages of using $\tilde{\Omega}$ instead of Ω to characterize the running times of programs.

Some authors also define O in a slightly different manner; let's use O' for the alternative definition. We say that $f(n) = O'(g(n))$ if and only if $|f(n)| = O(g(n))$.

- c. What happens to each direction of the “if and only if” in Theorem 3.1 if we substitute O' for O but still use Ω ?

Some authors define \tilde{O} (read “soft-oh”) to mean O with logarithmic factors ignored:

$$\tilde{O}(g(n)) = \{f(n) : \text{there exist positive constants } c, k, \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \lg^k(n) \text{ for all } n \geq n_0\}.$$

- d. Define $\tilde{\Omega}$ and $\tilde{\Theta}$ in a similar manner. Prove the corresponding analog to Theorem 3.1.

3-6 Iterated functions

We can apply the iteration operator $*$ used in the \lg^* function to any monotonically increasing function $f(n)$ over the reals. For a given constant $c \in \mathbb{R}$, we define the iterated function f_c^* by

$$f_c^*(n) = \min \{i \geq 0 : f^{(i)}(n) \leq c\},$$

which need not be well defined in all cases. In other words, the quantity $f_c^*(n)$ is the number of iterated applications of the function f required to reduce its argument down to c or less.

For each of the following functions $f(n)$ and constants c , give as tight a bound as possible on $f_c^*(n)$.

	$f(n)$	c	$f_c^*(n)$
a.	$n - 1$	0	
b.	$\lg n$	1	
c.	$n/2$	1	
d.	$n/2$	2	
e.	\sqrt{n}	2	
f.	\sqrt{n}	1	
g.	$n^{1/3}$	2	
h.	$n / \lg n$	2	

Chapter notes

Knuth [209] traces the origin of the O -notation to a number-theory text by P. Bachmann in 1892. The o -notation was invented by E. Landau in 1909 for his discussion of the distribution of prime numbers. The Ω and Θ notations were advocated by Knuth [213] to correct the popular, but technically sloppy, practice in the literature of using O -notation for both upper and lower bounds. Many people continue to use the O -notation where the Θ -notation is more technically precise. Further discussion of the history and development of asymptotic notations appears in works by Knuth [209, 213] and Brassard and Bratley [54].

Not all authors define the asymptotic notations in the same way, although the various definitions agree in most common situations. Some of the alternative definitions encompass functions that are not asymptotically nonnegative, as long as their absolute values are appropriately bounded.

Equation (3.20) is due to Robbins [297]. Other properties of elementary mathematical functions can be found in any good mathematical reference, such as Abramowitz and Stegun [1] or Zwillinger [362], or in a calculus book, such as Apostol [18] or Thomas et al. [334]. Knuth [209] and Graham, Knuth, and Patashnik [152] contain a wealth of material on discrete mathematics as used in computer science.