
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*

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\} .^1$$

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

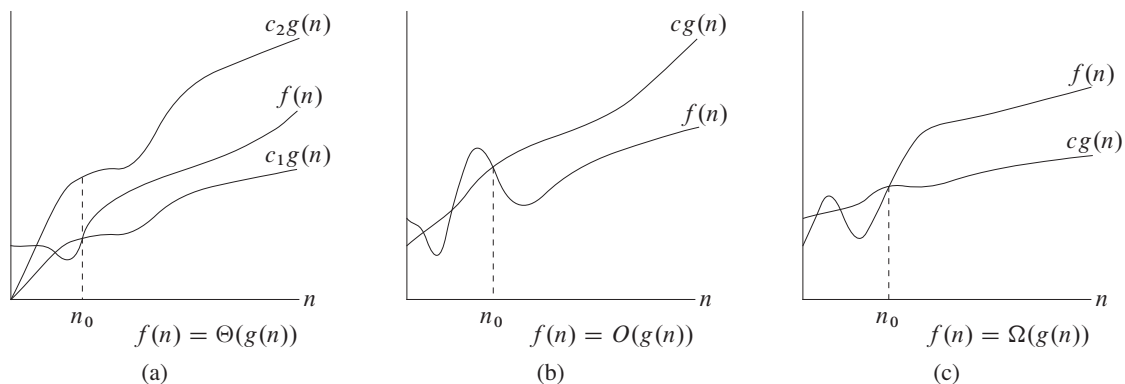


Figure 3.1 Graphic examples of the Θ , O , and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ -notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0 , c_1 , 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

expression does not indicate what variable is tending to infinity.² We shall often use the notation $\Theta(1)$ to mean either a constant or a constant function with respect to some variable.

***O*-notation**

The Θ -notation asymptotically bounds a function from above and below. When we have only an *asymptotic upper bound*, we use *O*-notation. For a given function $g(n)$, we denote by $O(g(n))$ (pronounced “big-oh of g of n ” or sometimes just “oh of g of n ”) the set of functions

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

We use *O*-notation to give an upper bound on a function, to within a constant factor. Figure 3.1(b) shows the intuition behind *O*-notation. For all values n at and to the right of n_0 , the value of the function $f(n)$ is on or below $cg(n)$.

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 $n_0 = \max(1, -b/a)$.

If you have seen *O*-notation before, you might find it strange that we should write, for example, $n = O(n^2)$. In the literature, we sometimes find *O*-notation informally describing asymptotically tight bounds, that is, what we have defined using Θ -notation. In this book, however, when we write $f(n) = O(g(n))$, we are merely claiming that some constant multiple of $g(n)$ is an asymptotic upper bound on $f(n)$, with no claim about how tight an upper bound it is. Distinguishing asymptotic upper bounds from asymptotically tight bounds is standard in the algorithms literature.

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.

and j are both at most n , and the inner loop is executed at most once for each of the n^2 pairs of values for i and j .

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 $O(n^2)$.

Ω -notation

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).

Theorem 3.1

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,

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0. \quad (3.1)$$

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.

ω -notation

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

$f(n) \in \omega(g(n))$ if and only if $g(n) \in o(f(n))$.

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

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

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

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty,$$

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

Comparing functions

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.

Transitivity:

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

Reflexivity:

$$\begin{aligned} f(n) &= \Theta(f(n)), \\ f(n) &= O(f(n)), \\ f(n) &= \Omega(f(n)). \end{aligned}$$

Symmetry:

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)) .$$

Transpose symmetry:

$$f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)) ,$$

$$f(n) = o(g(n)) \text{ if and only if } g(n) = \omega(f(n)) .$$

Because these properties hold for asymptotic notations, we can draw an analogy between the asymptotic comparison of two functions f and g and the comparison of two real numbers a and b :

$$f(n) = O(g(n)) \quad \text{is like} \quad a \leq b ,$$

$$f(n) = \Omega(g(n)) \quad \text{is like} \quad a \geq b ,$$

$$f(n) = \Theta(g(n)) \quad \text{is like} \quad a = b ,$$

$$f(n) = o(g(n)) \quad \text{is like} \quad a < b ,$$

$$f(n) = \omega(g(n)) \quad \text{is like} \quad a > b .$$

We say that $f(n)$ is *asymptotically smaller* than $g(n)$ if $f(n) = o(g(n))$, and $f(n)$ is *asymptotically larger* than $g(n)$ if $f(n) = \omega(g(n))$.

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

Trichotomy: For any two real numbers a and b , exactly one of the following must hold: $a < b$, $a = b$, or $a > b$.

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.

Exercises**3.1-1**

Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ -notation, prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

3.1-2

Show that for any real constants a and b , where $b > 0$,

$$(n + a)^b = \Theta(n^b) . \tag{3.2}$$

3.1-3

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

3.1-4

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

3.1-5

Prove Theorem 3.1.

3.1-6

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 $\Omega(g(n))$.

3.1-7

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

3.1-8

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 $f(m) > f(n)$.