

2:5 EXPONENTIATION REVISITED

The algorithm EXPONENT, verified by induction in the preceding section, correctly computes x^n . We discuss briefly the resources needed for implementation. There are four memory locations required, for x , i , ans , and n . It is possible to construct an exponentiation algorithm similar to EXPONENT that does not require separate memory locations for both n and i ; however, the resulting procedure is not as clear as the one we have presented.

Time is the other resource to assess, but what does that mean? If we run an algorithm on a big, fast computer, then we require less time than on a small, slow machine. If we run an algorithm on a time-sharing minicomputer, the time required depends upon the number of users and what they are computing. Maybe if we program the algorithm in FORTRAN, it will run faster than in Pascal. In this course we do not want to be concerned with specific machines (hardware) or specific languages (software). Instead we concentrate on a theoretical (but useful) measure of time.

The fundamental operations that occur in our algorithms are assignment statements ($:=$), comparisons (If $j < n$, then \dots), additions and subtractions ($+$ and $-$), and multiplications and divisions ($*$ and $/$). In a more advanced course you may study the time required for each of these operations. We'll assume that assignments happen instantaneously, that comparisons, additions, and subtractions each require a modest amount of time, and that multiplications and divisions are the most time-consuming operations. These assumptions certainly hold true when human beings make these calculations! We also assume that one multiplication and one division require the same amount of time. In fact, that is essentially the case for computations on computers.

Now look at Algorithm EXPONENT. We focus on step 4, since it is the only step that involves (time-consuming) multiplications. That is, we estimate the time needed to run this algorithm by counting the number of multiplications. Step 4 is executed once for each integer between 0 and $(n - 1)$ inclusive. Thus exactly n multiplications are executed. At first blush you might think that any algorithm that computes x^n must require this many multiplications, but that is not the case.

Example 5.1. To form x^4 using EXPONENT requires four multiplications. Alternatively, we could form x^2 with one multiplication and then multiply x^2 by itself to obtain x^4 with just two multiplications. We can denote this by

$$x^4 = 1 \cdot x \cdot x \cdot x \cdot x = (x \cdot x) \cdot (x \cdot x) = x^2 \cdot x^2.$$

There are four multiplications in the original algorithm. If you look at the second expression, you see three multiplications; however, the quantities within the two pairs of parentheses are identical. Consequently, we can compute x^4 with two multiplications. Similarly, we can find x^5 with just three multiplications as indi-

cated by

$$x^5 = [(x \cdot x) \cdot (x \cdot x)] \cdot x = x^4 \cdot x.$$

For x^6 we require only three multiplications as follows.

$$x^6 = [(x \cdot x) \cdot (x \cdot x)] \cdot (x \cdot x) = x^4 \cdot x^2.$$

In English, it takes one multiplication to form x^2 , one additional multiplication to form x^4 , and one final multiplication to combine these two products.

Question 5.1. Using the ideas suggested by the preceding example, how many multiplications do you need to form x^n for $n = 7, 11, 12$, and 16 ?

Answering the previous question suggests that efficient evaluation of x^n is related to the binary expansion of the integer n . Thus, for example, since $n = 25 = 16 + 8 + 1$ we could write

$$x^{25} = x^{16} \cdot x^8 \cdot x,$$

use three multiplications to obtain x^8 , one additional multiplication to obtain x^{16} and two more multiplications to combine the three factors. Thus it seems natural to suspect that there is an efficient algorithm to produce x^n that contains an algorithm to find the binary representation of the integer n . We use the Algorithm DtoB whose validity we have verified at the end of the previous section. For convenience we repeat the algorithm here and trace another instance.

Algorithm DtoB

- STEP 1. Set $j := 0$
- STEP 2. Divide m by 2 to obtain the quotient q and the remainder r ; place r in the j th column of the answer (reading from the right)
- STEP 3. If $q = 0$, then stop.
- STEP 4. Set $m := q$
- STEP 5. Set $j := j + 1$
- STEP 6. Go to step 2

Example 5.2. If we use the algorithm DtoB to compute the binary representation of $m = 25$, we get the following table of the values (Table 2.5).

To jazz this algorithm up so it will produce the value of x^m , we need to add three additional steps. First we need

- STEP 0. Input x, m , set $\text{ans} := 1$

2 ARITHMETIC

Table 2.5

Variables	j	m	q	r
Values after step 2	0	25	12	1
	1	12	6	0
	2	6	3	0
	3	3	1	1
	4	1	0	1

to initialize the algorithm. Next, where necessary, we need to multiply the intermediate result into the answer. We insert

STEP 2.5. If $r = 1$, set $\text{ans} := \text{ans} * x$

Finally, we need to insert a step that doubles the exponent on x .

STEP 5.5. Set $x := x * x$

To see how this works, we trace the new algorithm leaving x unspecified.

Example 5.3. The trace of the algorithm to find x^{25} is shown in Table 2.6.

Table 2.6

Variables	j	m	q	r	x	ans
Values after 2.5	0	25	12	1	x	x
	1	12	6	0	x^2	x
	2	6	3	0	x^4	x
	3	3	1	1	x^8	x^9
	4	1	0	1	x^{16}	x^{25}

Notice that four multiplications are required to compute the powers of x corresponding with each power of 2. Three multiplications are required to multiply the various factors together, and there is an additional cost of five divisions to form the binary representation.

Question 5.2. Trace the result of applying the above algorithm to find x^{37} . How many multiplications and divisions does the algorithm make? Do the same for x^{52} .

The straightforward algorithm to produce x^m required m multiplications. This new algorithm based on binary representation seems to do much better. In the next section we shall study just how good this algorithm is and introduce the mathematics and jargon needed to discuss this question.

EXERCISES FOR SECTION 5

1. Modify the algorithm EXPONENT so that upon input x and an integer $n \geq 0$, x^n is computed using only $n - 1$ multiplications. (Be sure to cover the case when $n = 0$.)
2. Modify the algorithm EXPONENT so that it uses only three memory locations for x , n , and ans .
3. Among the integers n with $16 < n < 33$, which integer requires the most multiplications to form x^n using the new algorithm from this section?
4. For each of the following integers n , find a factorization of x^n that will allow its computation by few multiplications (a) $n = 28$, (b) $n = 48$, (c) $n = 53$, and (d) $n = 56$.
5. Trace the result of applying the algorithm presented in this section to the problem of finding 2^{10} .
6. Notice that $x^{18} = x^{16} \cdot x^2 = (x^6)^3$. Which factorization provides the more efficient evaluation of x^{18} ?
7. Notice that $x^6 = (x \cdot x \cdot x)^2$ and that $x^9 = (x \cdot x \cdot x)^3$. Do these factorizations lead to more efficient computations than using the methods of this section?
8. Look back at your algorithm in Exercise 2.4 designed to evaluate nx using only addition. Find a way, using the binary expansion of n , to calculate nx using fewer additions.

2:6 HOW GOOD IS FAST EXPONENTIATION?

In Example 5.3 we saw that computing x^{25} based on binary representation required 12 multiplications and divisions while the traditional method would require 25 multiplications. In this section we investigate just how fast this binary exponentiation method is. We list the algorithm once again, named in anticipation of the analysis of its running time:

Algorithm FASTEXP

- STEP 0. Input x , m , set $\text{ans} := 1$
- STEP 2. Divide m by 2 to obtain quotient q and remainder r
- STEP 2.5. If $r = 1$, set $\text{ans} := \text{ans} * x$
- STEP 3. If $q = 0$, then stop.
- STEP 4. Set $m := q$
- STEP 5.5. Set $x := x * x$
- STEP 6. Go to step 2

Question 6.1. Why have we omitted steps 1 and 5?

2 ARITHMETIC

To compare the time of FASTEXP with that of EXPONENT, we count the number of multiplications and divisions. Thus given an integer m , how many multiplications and divisions will FASTEXP require to form x^m ? Notice that every time step 2 is executed, there will be exactly one division. Similarly, every time step 5.5 is executed, there will be exactly one multiplication. Now look at step 2.5. Sometimes this step results in a multiplication and sometimes it doesn't. We have two options. We can either think hard and try to figure out exactly how many times step 2.5 requires a multiplication. Or we can be blasé and say that in the most time-consuming case step 2.5 will require a multiplication every time it is executed. The most time-consuming case is also called the **worst case**. Thus for the worst-case analysis we count one multiplication every time that step 2.5 is encountered.

For this particular algorithm we can carry out both the precise and the blasé or worst-case analyses. For more complex procedures the worst-case analysis is commonly used and gives an upper bound on the time of the exact counting analysis.

The Worst-Case Analysis. The number of multiplications and divisions required to implement FASTEXP is no more than the number of times steps 2, 2.5, and 5.5 are encountered. Since we never execute steps 2.5 and 5.5 without having first executed step 2, we know that the number of multiplications and divisions is no more than three times the number of times step 2 is encountered. So the analysis depends on the number of times 2 can be divided into m . This is essentially the logarithm of m .

Logarithms. Given integers p and q with $2^p = q$, then p is said to be the **logarithm** to the base 2 of the number q . We shorten this to $p = \log(q)$. Here is the defining relationship in symbols:

$$p = \log(q) \text{ if and only if } 2^p = q.$$

Note that while base 10 is common in high school algebra and base e is typically used in calculus, in discrete mathematics and computer science logs are always assumed to be base 2 unless otherwise specified.

Example 6.1. If $p = 3$, then $2^3 = 8$. Consequently, $\log(8) = 3$. Similarly, $\log(32) = 5$ and $\log(1) = 0$.

Question 6.2. Calculate the following:

$$\begin{array}{cccc} \log(2^2) & \log(2^3) & \log(2^5) & \log(2^{10}) \\ 2^{\log(2)} & 2^{\log(4)} & 2^{\log(6)} & 2^{\log(8)} \end{array}$$

Question 6.3. Explain why $\log(2^p) = p$ and $2^{\log(q)} = q$.

Remember that if n is a positive integer, then $2^{-n} = 1/(2^n)$. Thus if $q = 1/(2^n)$, then $\log(q) = -n$.

We can mimic the definition of logarithm given above for some numbers that are not integers. Suppose that y is a rational number; that is, $y = a/b$, where a and b are integers. Then

$$\log(x) = y,$$

where

$$x = 2^y = 2^{a/b} = (2^a)^{1/b}.$$

In English, 2 to the y is the b th root of 2 to the a . For example, $2^{2/3}$ is the cube root of 4 while $2^{3/2}$ is the square root of 8. Notice that $x = 2^y$ is always a positive number for all rational numbers y . Thus the domain of the function $f(x) = \log(x)$ contains only positive numbers.

Here are two rules that help us work with exponents:

$$s^n \cdot s^m = s^{n+m}, \tag{A}$$

and

$$(s^p)^q = s^{pq}. \tag{B}$$

What do these rules tell us about logarithms? They are equivalent to the following properties, which are convenient for manipulating logarithms.

$$\log(ab) = \log(a) + \log(b), \tag{A'}$$

and

$$\log(a^b) = b \log(a). \tag{B'}$$

Here's why (A') is true in general. Suppose that $n = \log(a)$ and $m = \log(b)$. Then $a = 2^n$ and $b = 2^m$. Thus

$$\begin{aligned} ab &= 2^n \cdot 2^m \\ &= 2^{n+m}. \end{aligned} \tag{A.}$$

This means that

$$\begin{aligned} \log(ab) &= n + m \\ &= \log(a) + \log(b). \end{aligned} \quad \square$$

Property (B') follows similarly from (B). (See Exercise 2.)

2 ARITHMETIC

It would take us too far afield to attempt to define 2^y rigorously for arbitrary real numbers y or $\log(x)$ for arbitrary positive real numbers x . Luckily, it is not necessary. What we are really interested in is the integers that are near to $\log(x)$.

Floor Function. Given a real number x , the **floor function** of x , denoted by $\lfloor x \rfloor$, is defined to be the largest integer that is less than or equal to x . For example, $\lfloor 3 \rfloor = 3$, $\lfloor 3.11 \rfloor = 3$, $\lfloor 15.773 \rfloor = 15$, and $\lfloor -4.15 \rfloor = -5$.

Ceiling Function. Given a real number x , the **ceiling function** of x , denoted by $\lceil x \rceil$, is defined to be the smallest integer that is greater than or equal to x . For example, $\lceil 3 \rceil = 3$, $\lceil 3.11 \rceil = 4$, $\lceil 15.773 \rceil = 16$, and $\lceil -4.15 \rceil = -4$.

Note that for any number x , $\lfloor x \rfloor \leq x \leq \lceil x \rceil$.

Question 6.4. Find $\lfloor \frac{17}{3} \rfloor$, $\lceil \frac{25}{7} \rceil$, $\lfloor \log(8) \rfloor$, $\lceil \log(13) \rceil$, $\lfloor -\frac{14}{9} \rfloor$, $\lceil \log(25) \rceil$, and $\lfloor \log(13.73) \rfloor$.

We now have the vocabulary to answer the question of how many times we encounter step 2 in the execution of FASTEXP. Essentially, we want to know how many binary digits there are in the representation of the number m . For example, the two binary numbers with exactly two digits are 10 and 11; these represent the decimal numbers 2 and 3, respectively. Similarly, the decimal numbers 4, 5, 6, and 7 are all represented by three-digit binary numbers. In general, the decimal number m will be represented by a binary number with exactly r digits if and only if

$$2^{r-1} \leq m < 2^r.$$

Since $0 < c < d$ if and only if $\log(c) < \log(d)$, we can take logs of the previous inequality to get

$$\log(2^{r-1}) \leq \log(m) < \log(2^r)$$

which simplifies to

$$r - 1 \leq \log(m) < r.$$

The floor function allows the convenient representation

$$r - 1 = \lfloor \log(m) \rfloor \quad \text{or} \quad r = 1 + \lfloor \log(m) \rfloor.$$

What we have learned is summarized in Figure 2.6.

Thus in using FASTEXP to find x^m , step 2 will be executed exactly $1 + \lfloor \log(m) \rfloor$ times. So in the worst case the algorithm will require $3(1 + \lfloor \log(m) \rfloor) \leq 3 + 3 \log(m)$ multiplications and divisions.

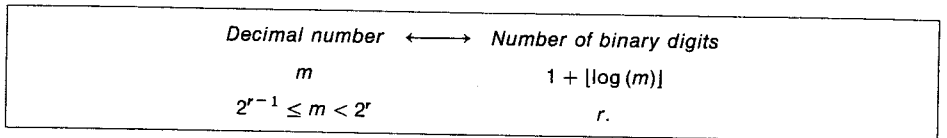


Figure 2.6

There are two differences between the worst-case analysis and the exact count. First FASTEXP terminates at step 3 when $q = 0$. Thus step 5.5 is executed once less than step 2. The second difference occurs because the multiplication in step 2.5 is not executed as often as step 2 is encountered. A little thought will convince you that the number of times this multiplication is made equals the number of 1s in the binary representation of the number m . Thus the exact number of multiplications and divisions required by FASTEXP equals

$$2(1 + \lfloor \log(m) \rfloor) - 1 + \#(1 \text{ bits in } m). \quad (C)$$

At times this exact count may be useful, but on the whole the upper bound given by the worst-case analysis is easier to work with.

Now let us contrast these two algorithms. EXPONENT requires n multiplications to compute x^n . Such an algorithm is called **linear**. The name is appropriate because if one plots the number of multiplications as a function of the exponent, the result is a straight line. Any such linear algorithm has the property that if one doubles the problem size, then the number of steps required (and thus the amount of time required) approximately doubles.

Example 6.2. Suppose that a particular algorithm reads in an integer n and requires $3n + 7$ steps. If $n = 100$, the number of steps is 307. If n is doubled to 200, then the number of steps equals 607. Notice that this is almost double the original 307 steps.

That doubling the input size to a linear algorithm does not exactly double the number of steps is a consequence of the fact that not all straight lines go through the origin. Thus how much the doubling rule is off depends on the y -intercept. In general, if the size of n is large compared to the y -intercept, then the doubling rule will be fairly accurate.

FASTEXP requires no more than $3 \log(n) + 3$ multiplications and divisions to compute x^n . [Notice that $3 \log(n) + 3$ is more convenient to work with than the more precise result in (C).] Such an algorithm is called **logarithmic**. If we plot the number of steps as a function of the input size, we get a logarithmic curve. A logarithmic algorithm has the significant property that one must square the size of the input before the number of steps (and thus the time) approximately doubles.

2 ARITHMETIC

Example 6.3. Suppose that a particular algorithm has an integer n as input and requires no more than $9 \lfloor \log(n) \rfloor$ steps. As in the preceding example if we input $n = 100$, the number of steps is no more than $9 \lfloor \log(100) \rfloor$. Since $2^6 = 64$ and $2^7 = 128$, $\lfloor \log(100) \rfloor = 6$. Thus the number of steps is no more than $9 \cdot 6 = 54$. If we double the input size to $n = 200$, the number of steps is no more than $9 \cdot 7$, since $\lfloor \log(200) \rfloor = 7$. If we square the input size to $n = 10,000$, the number of steps is now no more than $9 \lfloor \log(10,000) \rfloor = 9 \cdot 13 = 117$.

The contrast of the two preceding examples illustrates why logarithmic algorithms are much preferred to linear algorithms. In the next section we shall examine the logarithm function and others in more detail.

EXERCISES FOR SECTION 6

1. Calculate the following. Then match up the answers from the left-hand column that agree with one from the right-hand column.

$\log(2 \cdot 2)$	$\log(2) + \log(4)$
$\log(2 \cdot 4)$	$\log(4) + \log(8)$
$\log(2 \cdot 8)$	$4 \log(16)$
$\log(4 \cdot 8)$	$2 \log(2)$
$\log(2^2)$	$2 \log(8)$
$\log(8^2)$	$\log(2) + \log(8)$
$\log(16^4)$	

2. Explain why $\log(a^b) = b \log(a)$.
3. Find $\lfloor \log(n) \rfloor$ if $n =$ (a) 10, (b) 100, (c) 1000, and (d) 10,000.
4. Decide whether each of the following is true or false.

(a) $\log(2^{56}) = 56$.	(b) $2^{\log(3)} = \log(3)$.
(c) $\log(2 \cdot 3) = \log(3)$.	(d) $\log(\frac{1}{2}) = -1$.
(e) $\lfloor \log(17) \rfloor = 4$.	(f) $\log(2^3) = 4$.
(g) $\log(-2) = -1$.	(h) $\log(\frac{3}{8}) = \log(3)/3$.
(i) $\log(\frac{3}{4}) = -2 \log(3)$.	

 Correct those that are false.
5. Show that $\log(10^t) < 4t$ for all positive integers t .
6. Find the smallest integer k such that $\log(100t) < kt$ for all positive integers t .
7. Find (a) $\lfloor \frac{19}{11} \rfloor$, (b) $\lceil \frac{19}{11} \rceil$, (c) $\lfloor \frac{23}{12} \rfloor$, (d) $\lceil \log(73) \rceil$, (e) $\lfloor \log(73) \rfloor$, (f) $\lfloor \frac{114}{19} \rfloor$, (g) $\lceil \log(4^9) \rceil$, and (h) $\lfloor \log((2^3)^4) \rfloor$.
8. Suppose that $f(n) = 7n + 11$. Find the quotient $f(2n)/f(n)$ if (a) $n = 100$, (b) $n = 200$, and (c) $n = 1000$.

9. Suppose that $f(n) = 3n^2 + 4n + 5$. Find the quotient $f(2n)/f(n)$ if (a) $n = 100$, (b) $n = 200$, and (c) $n = 1000$.
10. Suppose that $f(n) = 4\lceil \log(n) \rceil + 13$. Find the quotient $f(n^2)/f(n)$ if (a) $n = 100$, (b) $n = 200$, (c) $n = 10,000$, and (d) $n = 20,000$.
11. (a) Show that if $f(n)$ is a linear function of the form $f(n) = a \cdot n$, where a is a constant, then $f(2n) = 2f(n)$. Find an equation expressing $f(n^2)$ in terms of $f(n)$.
 (b) Suppose that $g(n)$ is a logarithmic function of the form $g(n) = b \log(n)$. Then express both $g(2n)$ and $g(n^2)$ in terms of $g(n)$.
12. Use induction to show that FASTEXP is correct. (*Hint*: Reread the proof that DtoB is correct.)
13. How many multiplications and divisions are performed if FASTEXP is used to compute x^{89} ?
14. For each of the following values of m and r , verify that m has r binary digits, where $r = 1 + \lceil \log(m) \rceil$:
 (a) $m = 2, r = 2$. (b) $m = 3, r = 2$. (c) $m = 4, r = 3$.
 (d) $m = 7, r = 3$. (e) $m = 8, r = 4$. (f) $m = 15, r = 4$.
 (g) $m = 37, r = 6$. (h) $m = 100, r = 7$.
15. Is the following true or false? The number of binary digits in the number m is $r = \lceil \log(m) \rceil$. Explain.
16. Calculate $2^{(3^2)}$ and $(2^3)^2$. Explain, in general, why $a^{(b^c)}$ does not equal $(a^b)^c$.
17. Does $\log(a^b)$ equal $(\log(a))^b$? Explain.
18. The Post Office now charges 25 cents for a letter weighing up to 1 ounce and then 20 cents for each additional partial or whole ounce. If x is the weight of a letter in ounces and $x \geq 1$, then find a formula for the cost of mailing that letter.
19. Suppose that $R(x)$ is the function that takes a real number x and rounds it to the nearest integer. If $x = j + .5$, where j is an integer, then $R(x) = j + 1$; that is, R rounds up. Find a formula for $R(x)$ using the floor and/or ceiling functions.

2:7 HOW LOGARITHMS GROW

In the previous section we distinguished between EXPONENT and FASTEXP by looking at the functions that count the maximum number of divisions and multiplications that each would perform for a given input. This kind of analysis is crucial to any comparison of algorithms.

2 ARITHMETIC

To appreciate fully the advantages of a logarithmic algorithm over a linear algorithm, we must have a thorough understanding of the logarithm function. Our first observation about the logarithm function is that it gets arbitrarily large. By that we mean that given any positive integer M , no matter how large, once n is sufficiently large $f(n) = \log(n)$ will be larger than M . To see this, note that $\log(2^M) = M$ by definition. Thus

$$\text{if } n > 2^M, \text{ then } \log(n) > M.$$

Many functions get arbitrarily large, for example, $f(n) = n$, $g(n) = n^2$ and the square root function $h(n) = \sqrt{n}$. Note that $g(n) = n^2$ grows more rapidly than $f(n) = n$, which grows more rapidly than $h(n) = \sqrt{n}$. How fast does the logarithm function grow in comparison with these functions? Our principal result about the logarithm function is that its growth is very slow, in fact, even slower than the square root function. The remainder of this section is devoted to this property. We begin with the following fact about exponents.

Lemma 7.1. If r is an integer greater than 5, then $2^r > (r + 1)^2$.

Mathematicians call a particular mathematical statement a *lemma* if it doesn't appear very interesting but is useful in proving something else. Notice that the conclusion of the above lemma is false for $r = 1, 2, 3, 4$, and 5 as shown in Table 2.7.

Table 2.7

r	2^r	$(r + 1)^2$
1	2	4
2	4	9
3	8	16
4	16	25
5	32	36
6	64	49
7	128	64
8	256	81

Lemma 7.1 will be proved by induction. The proposition P_r will be that $2^r > (r + 1)^2$. The base case will be $r = 6$, and we shall show that the truth of P_k implies the truth of P_{k+1} .

Proof. We have the base case from Table 2.7. Next we assume that $2^k > (k + 1)^2$ and use this to show that $2^{k+1} > [(k + 1) + 1]^2$. Now

$$\begin{aligned}
2^{k+1} &= 2 \cdot 2^k && \text{by algebra} \\
&> 2(k+1)^2 && \text{by inductive hypothesis} \\
&= 2k^2 + 4k + 2 && \text{by algebra} \\
&= k^2 + 4k + 4 + (k^2 - 2) && \text{by algebra} \\
&= (k+2)^2 + (k^2 - 2) && \text{by algebra} \\
&> (k+2)^2 && \text{by ignoring } (k^2 - 2), \text{ which} \\
&&& \text{is positive for } k > 1. \quad \square
\end{aligned}$$

We use Lemma 7.1 to show that the logarithm function is eventually smaller than the square root function.

Theorem 7.2. If $n \geq 64$, then $\sqrt{n} > \log(n)$.

Proof. Suppose that k is the largest integer with $2^k \leq n$, that is, $k = \lfloor \log(n) \rfloor$. Then by definition

$$2^{k+1} > n \geq 2^k.$$

These two inequalities will produce the proof. First, if

$$2^{k+1} > n,$$

then, by taking logs of both sides, we get

$$k + 1 > \log(n). \quad (\text{A})$$

Second, if $n \geq 64$, then $k \geq 6$ and Lemma 7.1 applies. Thus

$$n \geq 2^k > (k+1)^2.$$

Taking square roots of the two ends, we get

$$\sqrt{n} > k + 1. \quad (\text{B})$$

Combining inequalities (A) and (B) gives $\sqrt{n} > \log(n)$. □

Question 7.1. Using a calculator, find the smallest integer N such that $\sqrt{N} > \log(N)$. Does this contradict Theorem 7.2? (*Comment:* if your calculator does not work in base 2, see Supplementary Exercises 9 and 10.)

Corollary 7.3. If $n \geq 64$, then $n/\log(n) > \sqrt{n}$.

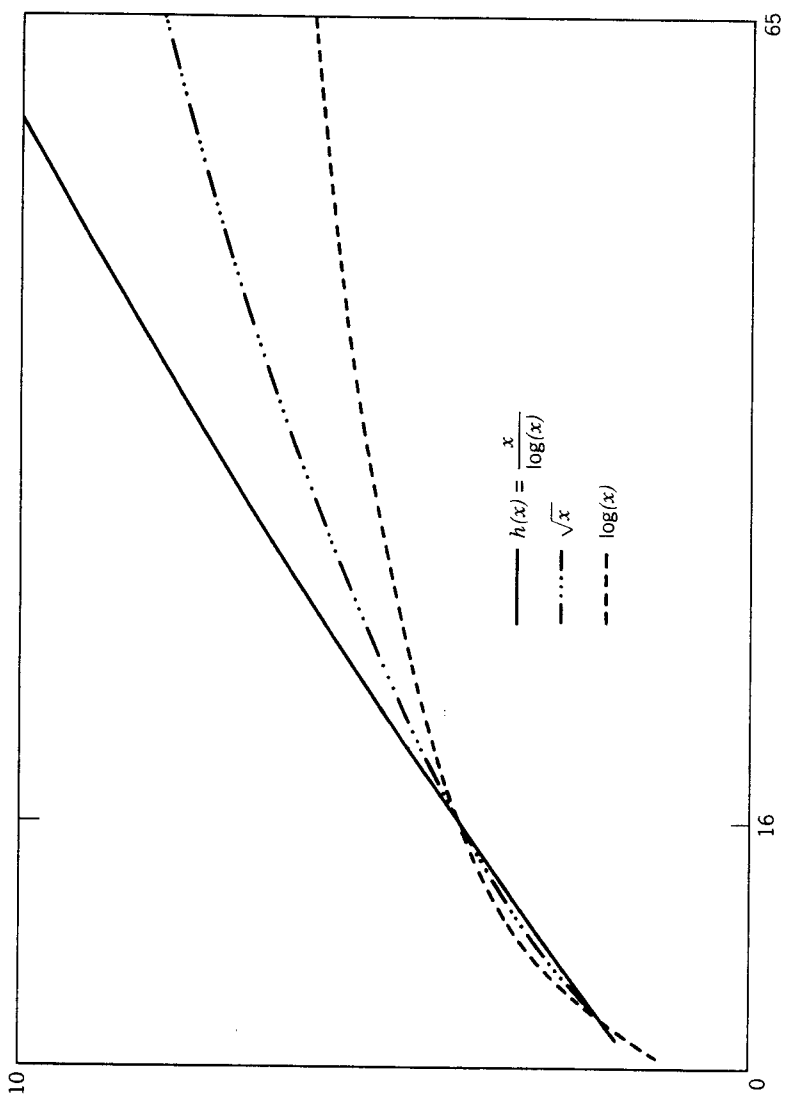


Figure 2.7

Proof. Since $n \geq 64$ implies $\sqrt{n} > \log(n)$, then if $n \geq 64$,

$$\frac{n}{\log(n)} > \frac{n}{\sqrt{n}} = \sqrt{n}. \quad \square$$

Corollary 7.4. The function $h(n) = n/\log(n)$ gets arbitrarily large.

Proof. As above we mean that given any (large) integer M , if n is big enough, then $h(n)$ is bigger than M . Suppose that $M \geq 8$. Then if $n > M^2$, by the previous corollary,

$$h(n) > \sqrt{n} > \sqrt{M^2} = M. \quad \square$$

Mathematicians use the word *corollary* for a statement whose proof almost immediately follows from a previous result. Corollary 7.4 looks inconsequential, but we shall have an important need for it later in this chapter.

We illustrate the growth of these functions in Figure 2.7; however, we shift our point of view on the domain of these functions. Until now we have been interested in counting problems, involving integers and functions evaluated at integers. Thus we write functions like $f(n) = \log(n)$, $g(m) = \sqrt{m}$, and $h(r) = r^2$, where n , m , and r represent integers. Furthermore, we think of our functions as having integer domains, usually the nonnegative or positive integers. On the other hand, most of the functions we are using can be defined for domains of real numbers. For example, $f(x) = \log(x)$ has domain all positive reals, $g(x) = \sqrt{x}$ has domain all nonnegative reals, and $h(x) = x^2$ has domain all real numbers. When we specify a function in terms of the variable x , we mean that x may take on any real value in its domain; in general, the context will make the appropriate domain evident. For graphical illustration we choose to consider the functions $x/\log(x)$, \sqrt{x} and $\log(x)$ as functions of real variables and to graph them as continuous curves as in Figure 2.7, rather than just plot the functions at integer values.

In the next section we develop the standard notation for comparing functions and their growth rates.

EXERCISES FOR SECTION 7

1. Find an integer N such that if $n \geq N$, then $n/\log(n) > 100$.
2. Find the smallest integer m such $2^m > m^2$. Then show that if r is any integer at least as large as m , then $2^r > r^2$.
3. Find the smallest integer q such that $2^q > (q+1)^3$. Use induction to show that if r is any integer with $r \geq q$, then $2^r > (r+1)^3$.

2 ARITHMETIC

- Use the preceding exercise to show that the cube root function is eventually larger than the log function.
- In Question 7.1 you found the least integer N such that $\sqrt{N} > \log(N)$. Prove that if $n \geq N$, then $\sqrt{n} > \log(n)$.
- Show that $\sqrt{n}/\log(n)$ gets arbitrarily large.
- What is the largest value that $2^{\log(n)/\sqrt{n}}$ can achieve for any positive integer n ?
- Suppose that $f(n) = \log(n)$ and $g(n) = \log(f(n))$. The following table lists some values for n , $f(n)$, and $g(n)$. Determine the values that correctly replace the question marks.

n	$f(n)$	$g(n)$
1	0	—
2	1	0
4	2	1
?	4	?
?	?	4
?	?	8

- With $g(n)$ defined as in Exercise 8 comment on the remark, “For practical purposes, $\log(g(n)) < 8$.”
- The function $g(n)$ of Exercise 8 is sometimes written as $g(n) = \mathbf{\log \log}(n)$. Find a simple expression for $\log \log(2^{2^n})$. Explain why $\log \log(n)$ is defined only for $n > 1$. Then explain why $\log \log(n)$ gets arbitrarily large for large values of n .
- Which function is larger: $\log(n)$ or $\log \log(n)$? Explain.
- For what function $m(n)$ is it true that $\log \log(m(n)) = 2 \log \log(n)$?

2:8 THE “BIG OH” NOTATION

In Section 2.7 we began to explore a hierarchy of functions that might appear in the analysis of algorithms. We saw that $\log(n) < \sqrt{n} < n/\log(n) < n$, provided that n is large enough. Consequently, we would prefer an algorithm that used \sqrt{n} steps to one that used $n/\log(n)$ steps on a problem with input size n . Since most algorithms are analyzed on a worst-case basis, we don't want our distinctions to be too finely drawn. Computer scientists and mathematicians have come to use what is called the “big oh” notation when discussing the efficiency of algorithms.

Definition 1. Let $f(n)$ and $g(n)$ be functions whose domains are the positive integers and whose ranges are the positive reals. Then we write

$$f(n) = O(g(n)) \quad \text{or} \quad f = O(g)$$

(read " f of n is big oh of g of n " or " f is big oh of g ") if there is a positive constant C such that

$$f(n) \leq C \cdot g(n)$$

for all positive integers n .

Example 8.1. Suppose that $f(n) = 5n$ and $g(n) = n$. To show that $f = O(g)$, we have to show the existence of the constant C in Definition 1. The best way to show the existence of such a C is to actually produce it. Here the example has been cooked up so that the constant C is staring us in the face. For if we choose $C = 5$, then $f(n)$ actually equals $C \cdot g(n)$. Notice that we could have selected a larger number for C . For instance, if we choose $C = 6$, then $f(n) = 5n < 6n = C \cdot g(n)$. So we write $f(n) = O(n)$.

Example 8.2. Suppose that $f(n) = 5n + 8$. To show that $f(n) = O(n)$, we must produce a constant C such that $f(n) \leq C \cdot n$ for all n . If we try $C = 5$, this doesn't work, since $f(n) = 5n + 8$ is not less than $5n$ [e.g., $f(1) = 13 > 5 = 5 \cdot 1$]. We need C to be at least 13. To see that $C = 13$ will work, note that $8 \leq 8n$. Thus

$$5n + 8 \leq 5n + 8n = 13n.$$

Let's look at a function that is not linear.

Example 8.3. Suppose that $f(n) = n^2$. We show that $f(n)$ is not big oh of n , denoted $f(n) \neq O(n)$. To accomplish this, we must show that there cannot exist a constant C that satisfies the big oh definition. Suppose that there were such a constant C . We would need

$$n^2 \leq C \cdot n$$

for all n . However, if $n > C$,

$$n^2 > C \cdot n.$$

Combining these two inequalities shows that for $n > C$,

$$n^2 > C \cdot n \geq n^2.$$

Thus no C can work.

2 ARITHMETIC

Example 8.4. Suppose that $f(n) = n^2 + 3n - 1$. Then

$$\begin{aligned} f(n) &= n^2 + 3n - 1. \\ &< n^2 + 3n, && \text{since subtraction makes things smaller} \\ &\leq n^2 + 3n^2, && \text{since } n \leq n^2 \text{ for integer } n \\ &= 4n^2. \end{aligned}$$

Thus, letting $C = 4$, we have $f(n) = O(n^2)$.

Question 8.1. Show that $f(n) = 12n^2 - 11$ and $h(n) = 3n^2 + 4n + 11$ are both $O(n^2)$.

These examples illustrate the big oh notation for linear and quadratic functions. Each time we find a simple function that is an upper bound on the original function. We emphasize that it is the growth of the functions that is of interest to us. Any linear function f has the property that $f(2n)$ is almost double $f(n)$. Furthermore, the larger n is, the more exact this rule. In contrast a quadratic function h has the property that $h(2n)$ is almost quadruple $h(n)$. Similarly, the larger n is, the more exact this rule.

We call an algorithm **linear** (or **quadratic**) if there is a function $f(n)$ that counts the number of the most time-consuming steps the algorithm performs, given a problem of size n , and $f(n) = O(n)$ (or $O(n^2)$). Let us see how this idea might help us decide which algorithm to select in a given situation.

Example 8.5. Suppose that we have two algorithms, say L and Q that each correctly solves a particular problem. L is linear and takes 20 minutes on a problem of size 10; Q is quadratic and takes 5 minutes on the same problem of size 10. Suppose that we have a problem of size 100. Which should we use? L ought to take about 200 minutes to solve the problem, and Q ought to take about 500 minutes to solve the problem. Why the difference? Because a tenfold increase in problem size ought to produce a hundredfold increase in running time for a quadratic algorithm while only a tenfold increase in running time for a linear algorithm.

Question 8.2. Suppose that L is a linear algorithm that solves a problem of size 100 in 8 minutes while C is a cubic algorithm that solves a problem of size 100 in 2 minutes. For a problem of size 200 which algorithm would you use? How about a problem of size 1000? {First you will have to decide what we mean by a cubic algorithm.}

It might be that, say, an algorithm we have called linear is faster than linear, that is, $f(n) = O(n)$ and $f(n) = O(g(n))$ for some function $g(n)$ smaller than n . It might be that it is our analysis that is weak and only demonstrates $f(n) = O(n)$. We shall discuss this in more detail in the next section.

Since there are linear, quadratic, and cubic algorithms, you should not be surprised to learn that this hierarchy generalizes to arbitrary exponents.

Example 8.6. If $f(n) = n^{17} + 3n^{15} - 7n^{10} + 20n^6 - 10n$, then we show that $f(n) = O(n^{17})$.

$$f(n) < n^{17} + 3n^{15} + 7n^{10} + 20n^6 + 10n,$$

since for $n > 0$, changing negative coefficients to positive makes the function larger;

$$\leq n^{17} + 3n^{17} + 7n^{17} + 20n^{17} + 10n^{17},$$

since making exponents larger makes the function larger;

$$\begin{aligned} &= (1 + 3 + 7 + 20 + 10)n^{17} \\ &= 41n^{17}. \end{aligned}$$

Question 8.3. Show that $f(n) = 2n^7 - 6n^5 + 10n^2 - 5 = O(n^7)$.

The first theorem of this section says that any polynomial is big oh of its term of highest degree. The proof mimics the previous example. We let a_j denote the coefficient of the term of degree j in the following theorem.

Theorem 8.1. Let $f(n) = a_d \cdot n^d + a_{d-1} \cdot n^{d-1} + \cdots + a_1 \cdot n + a_0$, where d is a positive integer and a_d, a_{d-1}, \dots, a_1 and a_0 are constants. Then $f(n) = O(n^d)$.

Proof. First we change all the coefficients of f to positive numbers. This can only increase the value of $f(n)$ for positive integers n . Next we note that $n^j \leq n^d$ if $j \leq d$. Thus

$$\begin{aligned} f(n) &= a_d \cdot n^d + a_{d-1} \cdot n^{d-1} + \cdots + a_j \cdot n^j + \cdots + a_0 \\ &\leq |a^d| \cdot n^d + \cdots + |a_j| \cdot n^j + \cdots + |a_0| \\ &\leq |a_d| \cdot n^d + \cdots + |a_j| \cdot n^d + \cdots + |a_0| \cdot n^d \\ &= (|a_d| + \cdots + |a_j| + \cdots + |a_0|) \cdot n^d \\ &= C \cdot n^d \end{aligned}$$

provided that C is equal to the sum of the absolute values of the coefficients in the original polynomial. \square

The big oh notation has certain peculiar features. For example, when we write $f = O(g)$ we are not writing down an equation of the usual sort. It does not express a symmetric relation; that is, we do not write $O(g) = f$ nor does it follow that $g = O(f)$, although it might in some instances. Here are some basic properties about big oh that are helpful in manipulations.

2 ARITHMETIC

Theorem 8.2. If $f = O(g)$, then for any constant a ,

$$(i) \ a \cdot f = O(g).$$

If, in addition, $h = O(k)$, then

$$(ii) \ f + h = O(g + k),$$

and

$$(iii) \ f \cdot h = O(g \cdot k).$$

Finally, if $f = O(g)$ and $g = O(h)$, then

$$(iv) \ f = O(h).$$

Example 8.7. Suppose that $f(n) = 5n^3 + 2n$. Then $f(n) = O(n^3)$ by Theorem 8.1. Next consider $w(n) = 15n^3 + 6n = 3f(n)$. By (i) of Theorem 8.2 (as well as by Theorem 8.1), $w(n) = O(n^3)$.

Example 8.8. Suppose that $f(n) = 3n + 7$ and $h(n) = 2n^2 - n + 8$. We know that $f = O(n)$ and that $h = O(n^2)$ by Theorem 8.1. By (ii) of Theorem 8.2 $f + h = O(n + n^2)$. Since $n + n^2 = O(n^2)$, we can use part (iv) of the Theorem 8.2 to obtain $f + h = O(n^2)$. This result also follows from Theorem 8.1, since $f(n) + h(n) = 2n^2 + 2n + 15$.

Example 8.9. Suppose that $f(n) = n^2 + 3n + 7$ and $h(n) = n^3 + 17$. Since $f = O(n^2)$ and $h = O(n^3)$, by part (iii) $f \cdot h = O(n^5)$. Notice that we didn't multiply out $f \cdot h$, but if we had, the result would be a polynomial of degree 5.

Question 8.4. If $f(n) = 3n^5 + 13n^3 - 10$ and $g(n) = 2n^4 + 3n^2$, find $h(n)$ and $k(n)$ such that $f + g = O(h)$ and $f \cdot g = O(k)$. Justify.

Proof of Theorem 8.2. We prove the second assertion leaving the remaining three proofs for Exercise 6. Suppose that C and C' are constants such that $f(n) \leq C \cdot g(n)$ and $h(n) \leq C' \cdot k(n)$. Let D equal the larger of C and C' . Then

$$\begin{aligned} f(n) + h(n) &\leq C \cdot g(n) + C' \cdot k(n) \\ &\leq D \cdot g(n) + D \cdot k(n) \\ &= D \cdot (g(n) + k(n)) \\ &= O(g(n) + k(n)). \end{aligned} \quad \square$$

Here is a summary of what we know about comparisons between functions. First, for all positive n

$$1 \leq \sqrt{n} \leq n \leq n^2 \leq n^3 \leq \dots \leq n^i \leq \dots$$

Thus we also have these big oh results:

$$\begin{aligned} 1 &= O(\sqrt{n}), \\ \sqrt{n} &= O(n), \\ n &= O(n^2), \\ n^2 &= O(n^3), \end{aligned}$$

and in general,

$$n^i = O(n^j) \quad \text{if } i \leq j.$$

You may have noticed that we have been avoiding the log function. Partly, this is because $\log(1) = 0$ and so the log function doesn't have range the positive reals and thus does not fit into the big oh definition. However, we know that if n is large enough ($n \geq 16$), then $\log(n) \leq \sqrt{n}$. Furthermore, as shown in Table 2.8 and Exercise 8, $\log(n)$ is always less than twice \sqrt{n} . Thus we want to say that $\log(n) = O(\sqrt{n})$.

Table 2.8

n	$\log(n)$	\sqrt{n}	$\log(n)/\sqrt{n}$
1	0	1	0
2	1	1.4...	0.7...
4	2	2	1
5	2.32...	2.23...	1.03...
6	2.58...	2.44...	1.05...
7	2.80...	2.6...	1.06...
8	3	2.8...	1.06...
9	3.16...	3	1.05...
10	3.32...	3.16...	1.05...
16	4	4	1
32	5	5.6...	0.8...
64	6	8	0.75

We would also like to say that

$$1 = O(\log(n));$$

2 ARITHMETIC

however, this also doesn't satisfy our present definition of big oh. Since $\log(1) = 0$, there is no constant C such that $1 \leq C \log(n)$ for all n . For $n \geq 2$, it is the case that $1 \leq \log(n)$, and so the constant $C = 1$ will work for n sufficiently large (larger than 1). What this suggests is that we need a stronger definition of big oh.

Definition 2. Let $f(n)$ and $g(n)$ be functions with domain the positive integers. Then we say

$$f(n) = O(g(n)) \quad \text{or} \quad f = O(g)$$

if there are positive constants C and N such that

$$f(n) \leq C \cdot g(n)$$

for all integers $n \geq N$.

With this more general definition we can say that $1 = O(\log(n))$, since $C = 1$ and $N = 2$ work. Also $\log(n) = O(\sqrt{n})$ using $C = 2$. We shall need this stronger definition only when we want to show that an algorithm performs $f(n) = O(\log(n))$ steps on a problem of size n . Such an algorithm is called **logarithmic**.

Definition 2 is the standard definition of the big oh machine. We have emphasized the simpler form because it is almost as widely applicable and it is easier to use. Furthermore, any pair of functions f and g with $f = O(g)$ in the first definition also has $f = O(g)$ in the second definition.

Our hierarchy of functions now looks like

$$1 \leq \log(n) \leq \sqrt{n} \leq n \leq n^2 \leq n^3 \leq \dots \leq n^i \leq \dots$$

This hierarchy of functions is worth remembering. In a typical analysis you will have a (complicated) function $f(n)$ and will need to find $g(n)$, as small and simple as possible, so that $f = O(g)$. The best candidates for g are the functions listed above.

Example 8.10. Suppose that $f(n) = \log(n) \cdot \sqrt{n^3} \cdot \sin(n)$. We want to find g from the list of basic functions above so that $f = O(g)$. First, recall that the value of the sine function never exceeds 1. Next we note that $\sqrt{n^3} = n \cdot \sqrt{n}$. Thus

$$\begin{aligned} f(n) &\leq \log(n) \cdot \sqrt{n^3} \\ &= \log(n) \cdot n \cdot \sqrt{n} \\ &= O(\log(n) \cdot \sqrt{n} \cdot n) \\ &= O(\sqrt{n} \cdot \sqrt{n} \cdot n) = O(n^2). \end{aligned}$$

In many of the examples of this section when $f = O(g)$, it has also been the case that $g = O(f)$. This is a common occurrence in problems about polynomials but will not generally be true. We know from Example 8.3 that $n = O(n^2)$, but $n^2 \neq O(n)$. Also, in the preceding example, although $f(n) = O(n^2)$, it is not true that $n^2 = O(f(n))$, although we shall not go into the tricky details.

EXERCISES FOR SECTION 8

- Find a constant C that demonstrates that $f(n) = O(g(n))$ for each of the following.
 - $f(n) = 17n + 31, g(n) = n.$
 - $f(n) = 21n - 13, g(n) = n.$
 - $f(n) = 12n^2 + 3n + 15, g(n) = n^2.$
 - $f(n) = 3n^2 - 4n + 5, g(n) = n^2.$
 - $f(n) = 2n^2 - n - 1, g(n) = n^2.$
 - $f(n) = 0.2n + 100,000, g(n) = n.$
 - $f(n) = n^3 + 3n^2 + 5n + 11, g(n) = n^3.$
 - $f(n) = n^3 + 3n^2 + 5n + 11, g(n) = n^4.$
- Which of the following is True and which False?
 - $n = O(n),$ (b) $n = O(n^2),$ (c) $n = O(n^3),$ (d) $n^2 = O(n),$ (e) $\log(n) = O(n),$
 - $n = O(\log(n)),$ (g) $n = O(n \cdot \log(n)),$ (h) $n \cdot \log(n) = O(n^2),$ and
 - $1/(n^2 + 1) = O(1/n^2).$

For each true statement, find a C that demonstrates the big oh definition.
- Here is the functional hierarchy: for n sufficiently large.

$$1 \leq \log(n) \leq \sqrt{n} \leq n \leq n^2 \leq \dots \leq n^i \leq \dots$$

Add all the following to this hierarchy. (a) $1/n,$ (b) $1/(n^2),$ (c) $n^{3/2},$ (d) $n/\log(n),$ (e) $n^2/\log(n),$ (f) $n/(\log(n))^2,$ and (g) the cube root of $n, n^{1/3}.$

- Answer the following with explanations for your answers:
 - Is $f(n) = 0.5n \cdot \log(n) + 3n + 15 = O(n^2)?$ $O(n)?$
 - Is $f(n) = (3n^2 + 5n - 13)^2 = O(n^5)?$
 - Is $f(n) = (3 \log(n) + n)^2 = O(n^2)?$
 - Is $f(n) = 1/(n^2 + 1) = O(1)?$
 - Is $4^{\log(n^2)} = O(\log(n) + 1)?$
 - Is $(4^{\log(n^2)})^2 = O(n^3)?$
- Some functions $f(n)$ are listed in column A of the following table. Some functions $g(n)$ are listed in column B. For each f in column A find the smallest g in column B with the property that $f = O(g).$

A	B
$7 + 32n + 14n^2$	1
$17 + \log(n^4)$	$\log(n)$
$\log(n^n)$	\sqrt{n}
$(\log(n^n))^2$	n
$\sin(n^2)$	n^2
$2^n/(n+1)^3$	n^3
$(n^2 + n \cdot \log(n))^2$	n^5
$2^n/n^n$	2^n
$(\log(n))^{0.5}$	n^n
$\sqrt{n} \cdot \log(n)$	$n \cdot (\log(n))$
$\sqrt{n^3}$	$1/n$
$\sqrt{n^3 + 3n^2}$	
$4^{\log(n)}$	
$\log(3^{n^2})$	
$n^{\log(n)}$	
$(n + \sqrt{n})^2$	
$(n+1)/(n^2+1)$	
$(\log(n))^4/n$	
$5 + (\log(n))^2 - n/2^n + 0.01 + n^3$	
$1 + 4 + 9 + \dots + n^2$	
$1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n}$	
$(\log(n)/n)^n$	

6. Prove parts (i), (iii), and (iv) of Theorem 8.2.
7. Suppose that g is a function such that $g(n) > 0$ for all n . Express $f(n) - g(n)$ as big oh of some combination of $f(n)$ and/or $g(n)$. Then prove your answer. Can you deduce a similar result for $f(n)/g(n)$?
8. Fill in the values of Table 2.8 for $n = 11, 12, \dots, 15$. See Supplementary Exercises 9 and 10 about computing $\log(x)$.
9. Show that n^3 is not $O(n^2)$.
10. Show that $2n^2 + 5n$ is not $O(n)$.
11. Is it true that for every pair of functions f and g either $f = O(g)$ or $g = O(f)$?

2:9 $2^n \neq O(p(n))$:

PROOF BY CONTRADICTION

We have seen that the polynomials form a natural hierarchy of functions against which we can compare the counting functions that really interest us. Is it the case that all our counting functions are $O(p(n))$ for some polynomial $p(n)$? No! The first natural example is the function 2^n that counts the number of subsets of a set with n elements. Here we have a negative (and significant) result.

Theorem 9.1. If r is a constant bigger than 1, then the function r^n is not $O(p(n))$ for any polynomial $p(n)$.

What does it mean to say that a function $f(n)$ is not big oh of the polynomial $p(n)$? Well, if f were $O(p)$, then there would exist constants C and N such that $f(n) \leq C \cdot p(n)$ for all $n \geq N$. If f is not $O(p)$, then there are no such constants. Specifically, there is no constant C such that $f(n) \leq C \cdot p(n)$ for all sufficiently large n . Such a fact can be harder to demonstrate than the fact that one function is big oh of another.

The proof of Theorem 9.1. will employ the technique of proof by contradiction. This technique is frequently used without notice if the statement to be proved is simple enough. We did it, for instance, in Chapter 1 in Theorem 9.1 and in (this chapter's) Example 8.3. The technique is almost as important as Mathematical Induction, but isn't as completely specified. We digress to explain.

Proof by Contradiction. We pause first to think a bit about what a mathematical proof is. Although a proof can be described precisely in logical terms with axioms, truth tables, and rules of inference, we choose to be more informal. The aim of a proof is to establish the validity of some assertion A . It may be a simple statement like "14 is an even integer," or it may be in the form of an implication, like " $n \geq 2$, then $\log(n) \geq 1$," or it may be in another form, like "There is an integer that is divisible by both 3 and 7." In a (usual) direct proof, we begin working with known mathematical truths and proceed logically until we deduce the truth of assertion A . For example, look back at the proof that $\log(a \cdot b) = \log(a) + \log(b)$ in Section 6. This was a straightforward, direct proof.

A proof by contradiction follows a different pattern. Suppose that we want to prove an assertion A . In a **proof by contradiction** we begin by assuming that assertion A is false. Then we argue using that assumption until we come to a contradiction. The contradiction will be the denial of some mathematical fact. For example, we might deduce that $0 = 1$ or that 5 is an even number. But now what? The absolute world of mathematics assumes that every statement is either true or false. Thus our original assertion A is either true or false. We assumed that it was false and deduced a contradiction. Thus it must be that assertion A is true.

To begin a proof of assertion A by contradiction, we must formulate the **negation** of A , that is, we must know what it means for A to be false. If assertion A is simple enough, its negation is easily formed. We emphasize that exactly one of assertion A and its negation is true, but for the moment we don't care which. Instead we concentrate on formulating the negation of a given statement A .

Example 9.1. Recall that an integer greater than 1 is called a prime if it cannot be written as the product of two smaller positive integers. The assertion that "7,891,011 is a prime number" is negated by the assertion that "7,891,011 is not a prime number" or "There are integers c and d with $1 < c \leq d$ and $c \cdot d = 7,891,011$."

2 ARITHMETIC

- Question 9.1.** Write down the negation of each of the following assertions.
- 353 is not a prime number.
 - 238 is an even integer.

In contrast, if assertion A is complicated, care might be required to correctly obtain the negation of A . We illustrate with several examples.

Example 9.2. Consider the assertion that " $n^3 + 3n^2 + 2n$ is always divisible by 3." (This can be proved by induction on n : see Exercise 4.17.) What this statement says precisely is that for every integer n , the quantity $n^3 + 3n^2 + 2n$ is a multiple of 3. The negation of this assertion is that there is at least one integer, say m , with the property that $m^3 + 3m^2 + 2m$ is not a multiple of 3.

In general, an assertion of the form "For every instance, something happens" is negated by "There is at least one instance where that something doesn't happen."

- Question 9.2.** Negate each of the following assertions.
- Every integer greater than one has a prime divisor.
 - Every integer of the form $4n + 1$ is a prime.
 - Every prime greater than 2 is odd.

Example 9.3. The assertion that "For some integer n , $n^2 + n + 1$ is divisible by 3" can be negated by the assertion that "For every integer n , $n^2 + n + 1$ is not a multiple of 3."

In general, an assertion of the form "There exists at least one instance when something happens" is negated by an assertion of the form "for every instance, that something doesn't happen."

- Question 9.3.** Negate each of the following assertions.
- For some integer n , $3n + 1$ is a prime number.
 - For some integer n , $\log(n) > n$.
 - For some integer n , $n^2 > 2^n$.

Many mathematical assertions are of the form "If statement H is true, then statement C is true." We frequently use the shorter form IF H , THEN C . Statement H is called the **hypothesis** and statement C is called the **conclusion**. Some assertions that do not appear to be in this form of hypothesis and conclusion can be rephrased as such. For example, the assertion "For every integer n , $n(n + 1)/2$ is an integer" can be rephrased, "If n is an integer, then $n(n + 1)/2$ is an integer." Here H represents the assertion that n is an integer while C represents the assertion that $n(n + 1)/2$ is an integer.

Example 9.4. The assertion “If n is prime, then $n^2 + 1$ is even” has as its hypothesis that n is prime and as its conclusion that $n^2 + 1$ is even. The negation of this assertion is that there exists a prime n with the property that $n^2 + 1$ is not even.

What is the negation of the assertion

“IF H , THEN C ”?

The negation of this is the assertion that statement H is true and statement C is false. We shorten this to

“ H AND (NOT C).”

Question 9.4. Identify the hypothesis and conclusion of each of the following assertions. Then negate each.

- (a) If n is even, then $n^2 + n + 1$ is prime.
- (b) If $n^2 + n + 1$ is prime, then n is even.
- (c) The integer n^2 is divisible by 4 whenever n is divisible by 6.

If we want to construct a proof by contradiction for the assertion “IF H , THEN C ,” we begin by assuming its negation, namely that both statement H is true and statement C is false, “ H AND (NOT C).” We then work logically until we reach the sought-after contradiction.

To illustrate the technique of proof by contradiction, we present an easy version of Theorem 9.1: If $r > 2$, then $r^n \neq O(1)$.

Proof. We begin by assuming that “ H AND (NOT C),” specifically that $r > 2$ and $r^n = O(1)$. From the (first) definition of big oh there is a constant C such that for all n

$$r^n \leq C \cdot 1.$$

Since $r > 2$,

$$2^n < r^n.$$

Now we combine these two inequalities and choose n to be an integer greater than C . This yields

$$2^n < r^n \leq C < n.$$

This provides a contradiction, since $2^n > n$ for every integer n . □

We now construct a proof by contradiction for Theorem 9.1. To make the arithmetic go a little easier, we shall prove the statement “ 2^n is not $O(p(n))$ ” for

2 ARITHMETIC

any polynomial $p(n)$." This is the same statement as in Theorem 9.1, except that $r = 2$ has been substituted. A similar proof would work for any constant $r > 1$.

Proof of Theorem 9.1. The negation of the assertion to be proved is "There exists a polynomial p with $2^n = O(p)$." Thus we begin by assuming this fact. By Theorem 8.1 there is an integer a such that $p = O(n^a)$, and so by part (iv) of Theorem 8.2 we have that $2^n = O(n^a)$. By definition, there is a constant C with

$$2^n \leq C \cdot n^a$$

for all positive integers n . Taking logarithms of both sides of this inequality, we get

$$\log(2^n) \leq \log(C \cdot n^a)$$

or

$$n \leq \log(C) + \log(n^a) \quad \text{by properties of log,}$$

or

$$n \leq \log(C) + a \cdot \log(n) \quad \text{by properties of log.}$$

We divide by $\log(n)$ to get

$$\frac{n}{\log(n)} \leq \frac{\log(C)}{\log(n)} + a.$$

Since for $n \geq 2$, $\log(n) \geq 1$,

$$\frac{n}{\log(n)} \leq \log(C) + a.$$

The right-hand side of this last inequality is a constant number, and by Corollary 7.4 the left-hand side is a function that grows arbitrarily large. These two statements form a contradiction, as we had hoped. We conclude that 2^n is not $O(p)$ for any polynomial p . \square

In practice, a proof of an "IF H, THEN C" assertion often begins with the logically equivalent "IF (NOT C), THEN (NOT H)." We digress briefly to explain and begin with an analogy.

Example 9.5. Imagine that you are sitting in an ice cream parlour. There is a sign in front of you that says, "Try our scrumptious butterscotch raspberry syrup."

Only on top of creamy French vanilla ice cream.” Before you lose too much concentration, let’s rephrase this sign in a more mathematically structured way. If you want b.r. syrup, then you have to have F.v. ice cream. More simply, IF B R SYRUP, THEN F V ICE CREAM. Here the syrup is analogous to the mathematical statement H and the truth of H corresponds to your having the special syrup. The ice cream is analogous to the mathematical statement C and the truth of C corresponds to your having the vanilla. What about the statement H AND (NOT C)? This translates into b.r. syrup without F.v. ice cream and negates the original sign.

Example 9.5 (continued). Suppose that as you get up to leave the ice cream parlour you notice a sign on the back wall that says. “If you don’t have our French vanilla ice cream, then you can’t have our butterscotch raspberry syrup.” Recall that the sign in the front says IF B R SYRUP, THEN F V ICE CREAM, while this one say IF NO F V ICE CREAM, THEN NO B R SYRUP. A little reflection will convince you that these signs mean the same thing.

As suggested by the previous example, given the assertion IF H , THEN C we can form what is called its **contrapositive** by negating both statements and reversing their order. Thus the contrapositive of the original

$$\text{IF } H, \text{ THEN } C \quad (1)$$

is the assertion

$$\text{IF NOT } C, \text{ THEN NOT } H. \quad (2)$$

As in the ice cream example a statement and its contrapositive are **logically equivalent**, that is, either both are true or both are false.

What other possibilities could there be? Well, it might be that the statement in (1) is true and the statement in (2) is false, or vice versa. Let’s see why neither of these cases can happen. First suppose that the statement of line (2) is true and the statement of line (1) is false, that is,

$$H \text{ AND (NOT } C) \quad (3)$$

is a true statement. Combining (3) and (2), we get

$$H \text{ AND (NOT } H),$$

which is a contradiction. Next suppose that statement (1) is true and statement (2) is false, that is,

$$\text{(NOT } C) \text{ AND NOT(NOT } H)$$

or

$$(\text{NOT } C) \text{ AND } H \quad (4)$$

is a true statement. Combining statements (1) and (4) yields

$$(\text{NOT } C) \text{ AND } C,$$

another contradiction. We conclude that statements (1) and (2) are either both true or both false. In other words, they are logically equivalent.

Example 9.6. Recall that a function is one-to-one if whenever $d \neq d'$, then $f(d) \neq f(d')$. The condition that specifies this one-to-one property is of the form IF H , THEN C , where H is the statement " $d \neq d'$ " and C is the statement " $f(d) \neq f(d')$." Thus this property is the same as the property IF NOT C , THEN NOT H . In other words, a function is one-to-one if whenever $f(d) = f(d')$, then $d = d'$. This second, but equivalent, definition is often easier to check, since working with equalities can be easier than with inequalities.

Given an assertion IF A , THEN B , we can also form its **converse**, namely the assertion IF B , THEN A . In the following example we illustrate that the truth of a particular assertion does not determine the truth of the converse.

Example 9.7. The ice cream parlour does allow a customer to have plain French vanilla ice cream, that is, B is true but A is false.

Question 9.5. Form the converse and the contrapositive of each of the assertions from the preceding question as well as Lemma 7.1 and Theorem 7.2.

Sometimes it is the case that both an assertion and its converse are true, that is, both "IF A , THEN B " and "IF B , THEN A " are true. In that event we say that A is true if and only if B is true, which is abbreviated A IFF B . We shall see examples of this sort of assertion in Chapter 4.

EXERCISES FOR SECTION 9

1. Write out a detailed proof that 4^n is not $O(p)$ for any polynomial p .
2. Regardless of the truth or falsehood of the following, write the negation of the following assertions.
 - (a) 14 is even.
 - (b) 6 is prime.
 - (c) $1 + 2 + \cdots + n = n(n + 1)/2$.

- (d) There is an integer divisible by 3 and by 7.
 - (e) For every integer n , $n(n + 1)/2$ is also an integer.
 - (f) Every set has more 2-subsets than 1-subsets.
 - (g) Every even number has an even number of 1s in its binary representation.
 - (h) If $f(n) = O(g(n))$, then $f(n) \leq g(n)$.
 - (i) There is a set that is larger than its complement.
 - (j) If x^2 is odd, then x^6 is odd.
 - (k) If x^6 is divisible by 8, then x is divisible by 2.
 - (l) 1728 is a sum of four cubes.
 - (m) Every even composite number is the sum of two primes.
3. Determine for which of the following functions $g(n)$ it is true that $2^n = O(g(n))$.
 - (a) $g(n) = n^2$, (b) $g(n) = 2^n$, (c) $g(n) = n^{10}$, (d) $g(n) = 10^n$, (e) $g(n) = 2^{\sqrt{n}}$, (f) $g(n) = 2^{\log(n)}$, and (g) $g(n) = 2^{(n^2)}$.
 4. For each of the following statements identify the hypothesis H and the conclusion C , rewriting the statements if necessary:
 - (a) If $n \geq 2$, then $n \leq n^2 - 2$.
 - (b) A set with n elements has 2^{n-1} even subsets.
 - (c) x^n can be calculated with exactly $\log(n) + 1$ multiplications when n is a power of 2.
 - (d) $1 + 3 + \cdots + (2r - 1) = r^2$.
 - (e) All quadratic polynomials are $O(n^2)$.
 - (f) Every even composite number is the sum of two primes.
 - (g) Every even number has an even number of 1s in its binary representation.
 5. Write the negation, the converse, and the contrapositive of each statement in the preceding problem.
 6. (Another Analogy) Many serious hikers have a cardinal principle that they won't hike if they aren't wearing two pairs of socks. Break this principle up into statements H and C and rephrase it in the IF H , THEN C paradigm. Form the negation.
 7. Form the contrapositive and the converse of the hikers and socks assertion that you created in the preceding exercise.
 8. Each of the following allegations can be put into the form IF H , THEN C . After identifying the statements that are H and C , form the converse and the contrapositive of each.
 - (a) If n is even, then $n/2$ is an integer.
 - (b) If $n = p \cdot q$ (where $1 < p, q < n$), then n is not a prime.
 - (c) If r is odd and s is odd, then $r + s$ is odd.
 - (d) x^2 is divisible by 2, provided that x is divisible by 2.
 - (e) x an odd integer implies $(x + 1)/2$ an odd integer.
 For each statement above, decide whether it is true or false. Do the same with the converses and the contrapositives.

2 ARITHMETIC

9. Identify the IFF statements in the preceding problem, that is, those allegations for which both the statement and the converse are true.
10. Beginning with the assertion, "If $r > 2$, then $r^n = O(1)$," use definition 2 of big oh to reach a contradiction.
11. Prove Theorem 9.1 using definition 2 of big oh.

2:10 GOOD AND BAD ALGORITHMS

We have introduced a variety of algorithms, some correct, some incorrect, some efficient, some inefficient, some transparent, some quite complex. We have also begun to talk about what is known as the space and time complexity of an algorithm, that is, the number of variable memories needed and the time needed to run the algorithm (or at least an upper bound on the number of key operations like multiplication). In this section we shall formulate what is meant by a good algorithm.

Our description of "good" will depend upon the efficiency of the algorithm, but we must always remember that an algorithm must be correct to be "good" in any sense of the word. We can easily write efficient algorithms, like the following.

Algorithm SPEEDY

STEP 1. Stop.

This is the world's shortest algorithm and in that sense the most efficient; however, this algorithm has no relevance to any problem in the real world!

Space resources are important. We need enough space to read in the problem addressed by the algorithm, and we shall try to be conservative with additional memory needed for new variables. For example, in Section 2.1 we saw how we could interchange the values of two variables without using an additional memory location. However, our first priority will be to minimize running time and then secondly to minimize storage. The time requirement is a parameter commonly studied and one that leads to a rich and important theory in computer science.

We have already stated that we do not want to analyze time in terms of specific computers or programming languages. We want a procedure for comparing two algorithms that (correctly) solve the same problem. But we want a fair comparison. For example, if we run algorithm EXPONENT with $x = 2$ and $n = 3$, it certainly would be quicker than algorithm FASTEXP with $x = 10,000,000$ and $n = 123,456,789$, and yet we have claimed that the latter algorithm is faster or more efficient. Or think of just one algorithm. Of course, it runs faster when we enter few and small pieces of data rather than larger ones. For example, we can much more quickly list all subsets of a 2-set than of a 20-set.

Thus we must compare execution times on the same data set or at least on data sets of the same cardinality. We do this by introducing a parameter that

measures the size of the data set. How we measure this size depends upon the problem at hand. For instance, we could describe the size of the data for the algorithms DtoB and BtoD by stating the number of digits in the given decimal or binary number. In SUBSET we could describe the size of the problem by giving the size of the set all of whose subsets we want to list. In EXPONENT and FASTEXP we could specify n , the power to which we are raising x , or the number of bits needed to store n .

Typically, we shall begin by supposing that the data or the input to the algorithm is of size n . This may mean that we have n bits of information or that the integer n is the crucial variable. Then we shall estimate the time of running the algorithm in terms of the variable n . Often we shall denote the time as some function $f(n)$ that counts the number of multiplications or the number of comparisons or the number of some time-consuming operation. This function will be known as the (time) **complexity** of the algorithm. Sometimes we shall be able to determine $f(n)$ explicitly; other times we shall make a worst-case analysis and get an upper bound on $f(n)$. Most frequently, we shall be happy to determine that $f = O(g)$ for some nice function g .

Our goal is simplistic: We want to divide correct algorithms into one pile called “good” and another pile called “bad.” Just as philosophers who study language arrive at the meaning of the word *good* through comparisons, so shall we. Suppose that we have two correct algorithms to solve a particular problem, say A and B . Let $a(n)$ denote the complexity of A and $b(n)$ denote the complexity of B . Assume that we know $a = O(f)$ and $b = O(g)$. [Of course, if we know $a(n)$ and $b(n)$ explicitly we can compare them directly.]

Definition. If an algorithm A has complexity $a(n) = O(f(n))$ and an algorithm B has complexity $b(n) = O(g(n))$, we say that A “**appears to be**” as efficient as B if $f = O(g)$. If, in addition, $g = O(f)$ we say that A “**appears to be**” equivalent to B . Otherwise, if $f = O(g)$ and g is not $O(f)$ we say that A “**appears to be**” more efficient than B .

In practice, we shall replace “appears to be” with “is” in the above definitions. You might think that our definition is overly wishy-washy; however, restraint is forced on us by the big oh comparisons. The problem is that it might be difficult to bound the complexity of one or both of the algorithms in question. For example, we might be able to prove that $a(n) = O(n^3)$ and that $b(n) = O(n^4)$. In that instance we would say that A appears to be more efficient than B , since $n^3 = O(n^4)$ and n^4 is not $O(n^3)$. In reality, it might be the case that $a(n) = n^{1.1/4}$ while $b(n) = n^{5/2}$ in which case B is actually more efficient than A . Thus the quality of our comparison depends on how good a big oh estimate we have.

Example 10.1. Suppose that the time complexity function of algorithm A is given by $a(n) = n^2 - 3n + 6$ and that of algorithm B by $b(n) = n + 2$. Then A is a $O(n^2)$ algorithm while B is a $O(n)$ algorithm. In this case we would naturally say that B is more efficient than A .

2 ARITHMETIC

Thus we measure the efficiency of algorithms using the hierarchy of functions developed in the preceding sections. An algorithm with complexity n^i will be more efficient than an algorithm with complexity n^j if and only if $i < j$. An $n \log(n)$ algorithm is more efficient than a quadratic algorithm but less efficient than a linear algorithm, since for n large enough,

$$n \leq n \log(n) \leq n\sqrt{n} \leq n^2.$$

Let's go back to our original question of what constitutes a good algorithm. Of course, if we have two different algorithms to solve the same problem, then we shall consider one better if it is more efficient in the sense described above. But we make a global judgment now about what is known in computer science circles as a "good" algorithm.

Definition. Suppose that A is a correct algorithm with complexity function $a(n)$. Then A is called **good** (or **polynomial**) if there exists a polynomial $p(n)$ with $a = O(p)$. A is called **bad** if a is not $O(p)$ for any polynomial p . If $a(n) \geq r^n$ for some constant r (with $r > 1$), then A is called **exponential**. A problem that does not have a good algorithmic solution is called **intractable**.

There are some important ideas here. An algorithm will be called good regardless of what polynomial gives the bound on $a(n)$. It might be that the polynomial is huge, and so the algorithm takes a long time to run. That would not be a very "good" situation from the point of view of efficiency, but in theory at least the situation is not as bad as having $a(n) = 2^n$, an exponential function.

Now we see the relevance of Theorem 9.1. It states that $r^n \neq O(p(n))$ for any polynomial $p(n)$. Thus if we find an algorithm with complexity 2^n or 3^n or even 1.000005^n , this is an exponential algorithm, not a polynomial one.

For example, look at the algorithm SUBSET. Our input to the algorithm is a set of n elements, and we want as output all 2^n subsets. The size of the input data is n . We systematically create and list all subsets of the n -set. Since there are 2^n subsets, our list making will need 2^n steps and the complexity will be at least 2^n , an exponential function. Thus SUBSET is an exponential algorithm, and the problem of listing all subsets of an n -set is intractable.

But why do we make such a harsh judgment about an exponential algorithm? As we all know from the news media, exponential growth is considered to be very fast growth into large numbers, perhaps dangerously out of control when related to, for example, population. Is this also the case for algorithms? Does this mean that exponential algorithms are too large or take too long? Surely, modern computers can handle large amounts of calculation extraordinarily quickly.

Let's do some arithmetic that is specific to an IBM PC but that would be approximately the same for any microcomputer. The PC can perform about 17,800 single-digit multiplications in a minute. Suppose that we compare seven algorithms that do multiplications and whose complexities are given by the functions $\log(n)$,

\sqrt{n} , n , n^2 , n^3 , 2^n , and 10^n . What size problems could we reasonably solve on the PC? Table 2.9 shows various values of n , the seven complexity functions, and roughly the amount of time needed to run the algorithms. (We stop filling in the table once the numbers become inhumanly large!)

Table 2.9

n	$\log(n)$	\sqrt{n}	n	n^2	n^3	2^n	10^n
8	0.01 sec	0.01 sec	0.027 sec	0.216 sec	1.73 sec	0.863 sec	3.9 days
16	0.013 sec	0.013 sec	0.054 sec	0.863 sec	13.8 sec	3.7 min	10,689 cent.
24	0.015 sec	0.017 sec	0.081 sec	1.94 sec	46.6 sec	15.7 hr	...
32	0.017 sec	0.019 sec	0.108 sec	3.45 sec	1.84 min	168 days	...
64	0.02 sec	0.027 sec	0.216 sec	13.8 sec	14.7 min	19,717,160 cent.	...
128	0.024 sec	0.038 sec	0.43 sec	55 sec	2 hr		
256	0.027 sec	0.054 sec	0.86 sec	3.7 min	16 hr		

Thus we see that we will be in trouble as soon as we have at least 24 pieces of data on which we must run an exponential algorithm. Of course, we could switch over to minicomputers, which typically run about 100 times as fast, but our problem size still demands too much computing with data of size 32 or larger. Thus there really is a problem when we must do an exponential amount of computing on even a moderate amount of material.

EXERCISES FOR SECTION 10

- Suppose that algorithm A has complexity function $a(n)$ and algorithm B has complexity function $b(n)$. In each of the following cases decide whether the algorithms are equivalent or if not, which is more efficient.
 - $a(n) = 36$. $b(n) = 2n - 10$.
 - $a(n) = n^2$. $b(n) = n$.
 - $a(n) = n^2$. $b(n) = n - 6$.
 - $a(n) = 2n^2$. $b(n) = 3n^2$.
 - $a(n) = n^2 + 2n$. $b(n) = n^2 + n$.
- Explain why $n^n \neq O(r^n)$ for any constant r .
- Find a function that is not $O(n^n)$.
- In Exercise 1.7.6 you designed an algorithm that lists all $n(n-1)/2$ subsets of size 2 of a set A containing n elements. Suppose that you count as one step the formation and output of a single subset. Is your algorithm good or bad?

2 ARITHMETIC

5. Suppose that A contains n objects and we have an algorithm that outputs all elements of the Cartesian product A^n . Counting one step as the formation and output of one element of the Cartesian product, is this algorithm good or exponential?
6. Show that the algorithm to settle the Satisfiability Problem for a Boolean function (see Section 1.10), which consists of trying all possibilities, is bad.

2:11 ANOTHER LOOK BACK

By far the most important ideas in this chapter have been the proof techniques of induction and contradiction. Their use permeates all of mathematics and computer science. Indeed the practitioners of these disciplines regularly apply these methods without acknowledgment. We have met typical instances of proofs using these techniques.

Induction proofs work well on set theory problems, on summation formulas, and on algorithmic problems, that is, in proving that an algorithm does what it is supposed to do in all cases. The Principle of Induction gives us a three-step format that allows us to set up and attack problems in a straightforward way. This doesn't mean that proof by induction will always be easy and automatic. We shall still need to think carefully and creatively about each instance.

Proofs by contradiction arise naturally for statements of the form, "Object A does not have property P ." Often we can begin with the assumption that A has property P ; do some algebra or a logical argument, and arrive at a contradiction. Often some experimentation is needed to find a reasonable argument, leading to, say, $0 = 1$. Probably, the best advice for both kinds of proofs is to study the examples and theorems in the text and to try to imitate these in the exercises.

This chapter also presents the central computer science and mathematical ideas about the analysis of algorithms. We now have a sequence of tasks to perform when we look for an algorithmic solution to a problem. Not only must we come up with the algorithm, but we must also prove that it is correct in all cases and we must analyze its time and space requirements. The worst-case analysis is commonly used, since we can often estimate an upper bound on the maximum number of the most time-consuming steps. Notice that these upper bounds may be crude, too large, and it may be that our algorithm works more efficiently.

Of course, even if we know an exact analysis of the number of steps performed, we have no guarantee that there isn't a faster algorithm. For example, how do we know that computing x^n can't be done by an algorithm faster than FASTEXP, that is, by an algorithm that is faster than logarithmic? We haven't discussed this issue at all. There are other ways to analyze algorithms. For example, we might ask for typical or average-case behavior. Or we might be more demanding and count all kinds of steps: multiplication, division, addition, subtraction, compari-

sons, assignment statements, and so on. Such details are suitable for more advanced courses in theoretical computer science.

Most algorithms presented in the computer science literature include a discussion of worst-case behavior, and this discussion inevitably entails use of the big oh jargon. The big oh definitions are subtle yet important; it is well worth doing lots of exercises on this concept. It is the primary way that computer scientists and mathematicians distinguish between the quality of algorithms. There are a number of important problems (including searching and sorting, which we will consider in Chapter 6, and the fast Fourier transform, which we will not consider) that have naive algorithms that are $O(n^2)$ and better algorithms that are $O(n \log(n))$. The existence of the better algorithms greatly expands the size of the problems that are feasible to solve. One further satisfactory point about the $O(n \log(n))$ sorting algorithms is that we can also prove that every algorithm (of a certain type) that solves a sorting problem must use at least $cn \log(n)$ steps for some constant c . That tells us that we've found essentially the very best algorithm.

In the SUBSET problem and algorithm we realized that every algorithm must form and list all 2^n subsets, and so is necessarily exponential. We have a lower bound on the number of steps needed; often it is difficult to obtain such lower bounds. The distinction between polynomial and exponential algorithms is the most actively researched area within computer science. We shall soon see many problems for which every known algorithm is exponential, but no one has proved that there is no polynomial algorithm. (The Satisfiability Problem of Section 1.10 is one such example.) Thus upper bounds, but not lower bounds, are known, and it is not yet known whether the problems are inherently intractable. This leads to the heart of some fascinating, unsolved problems in computer science.

SUPPLEMENTARY EXERCISES FOR CHAPTER 2

1. Show that if there is a counterexample to Fermat's Last Theorem, then there exists a counterexample in which no pair of the integers x , y , and z has a common factor. Use this fact to show how large z must be in any counterexample.
2. It was claimed that Fermat's Last Theorem is known to be true for $n \leq 125,080$. From the previous problem you know that z can't be 1 (for instance). Suppose that you wanted to write down the digits in this hypothesized counterexample. If you could write 100 digits a minute for the rest of your life (with no resting time), would you be able to transcribe this example?
3. (a) The distinguished mathematician Gauss was a child prodigy. Legend has it that at the age of 10 he was asked to add the integers $81,297 + 81,495 + 81,693 + \cdots + 100,899$. Almost immediately he wrote one number, 9,109,800, on his slate. Was he correct?

2 ARITHMETIC

(b) Let a and b be constants. Show that the sum of the arithmetic progression $a + (a + b) + (a + 2b) + \cdots + (a + nb)$ is given by $(n + 1)(a + nb/2)$.

4. Prove by induction that if b is a number such that $1 \leq b \leq n(n + 1)/2$, then there is a subset of $\{1, 2, \dots, n\}$ whose sum equals b .
5. Prove by induction that if $a \neq b$, then

$$\begin{aligned} a^n + a^{n-1} \cdot b + \cdots + a^j \cdot b^{n-j} + \cdots + b^n \\ = \frac{a^{n+1} - b^{n+1}}{a - b}. \end{aligned}$$

6. Consider the following algorithm:

STEP 1. Input r, s $\{r$ and s positive integers $\}$

STEP 2. While $s > 0$ do

Begin

STEP 3. $r := r - s$

STEP 4. $s := s - 1$

End {step 2}

STEP 5. Output r , then stop.

- (a) Trace through this algorithm when 12 and 3 are input for r and s , respectively.
- (b) Show that this algorithm must terminate no matter what positive integer values of r and s are input.
- (c) What is the output in terms of r and s ?
7. Given the sum $t + 2(t - 1) + \cdots + j(t - j) + \cdots + t$, guess a formula in terms of t for this sum. Use induction to prove that your guess is correct.
8. Determine the sums: $1^2, 2^2 - 1^2, 3^2 - 2^2 + 1^2$, and $4^2 - 3^2 + 2^2 - 1^2$. Then deduce and prove a general formula for the sum

$$n^2 - (n - 1)^2 + (n - 2)^2 - \cdots + (-1)^{n+1} \cdot 1^2.$$

9. Your calculator probably has logarithms to the base 10 or to the base e . Here's how to change from base 10, written $\log_{10}(x)$, to base 2, $\log(x)$:

$$\log(x) = \frac{\log_{10}(x)}{\log_{10}(2)} = 3.3219 \log_{10}(x).$$

Prove the above identity. [*Hint*: Begin with $x = 2^{\log(x)}$.]

10. Find a formula for changing natural logarithms, logs to the base e , to base 2 logarithms. Justify your formula.

11. Count the number of comparisons made in the algorithm MAX of Exercise 4.12 and in BUBBLES in 4.13. Which is more efficient? Count the number of assignment statements in the two algorithms. Which is more efficient from the assignment point of view? Suppose that an assignment can be done in 1 second and a comparison in 2 seconds, then which algorithm is more efficient?
12. Suppose that you have a computer that can only perform addition, subtraction, and multiplication. Write an algorithm that upon input x , a positive real number, calculates and outputs $\lfloor \sqrt{x} \rfloor$.
13. Suppose that you have a computer that can perform addition, subtraction, and multiplication, but not division. Design an algorithm that upon input of real numbers x and y (with $x \neq 0$) calculates and outputs $\lfloor y/x \rfloor$.
14. Show that if $f = O(g)$, then $2^f = O(2^g)$. Is the converse true? Give a proof or counterexample.
15. Here is one of the oldest and most famous results in all of mathematics.

Theorem. There are infinitely many primes.

- (a) What is the negation of this theorem?
 - (b) If the theorem is false, suppose that t equals one plus the product of all the primes. Is t prime?
 - (c) Is the t you formed in part (b) divisible by 2? 3? 5? Any prime?
 - (d) Prove the Theorem by contradiction.
16. Every positive integer can be expressed in the form $3n$, $3n + 1$, or $3n + 2$ for some integer n .
 - (a) Explain why every prime number greater than 3 is of the form $3n + 1$ or $3n + 2$.
 - (b) Explain why a number of the form $3n + 2$ must have a prime divisor of the form $3n + 2$.
 - (c) Prove that there are infinitely many primes of the form $3n + 2$. [*Hint:* Suppose that p_1, p_2, \dots, p_k are all the primes of the form $3n + 2$. Then consider $3(p_1 \cdot p_2 \cdot \dots \cdot p_k) + 2$.]
 17. Here we prove that $(x + 1)^2$ does not equal $x^2 + 2x + 1$. The proof is by contradiction, so we begin by assuming the negation:

$$(x + 1)^2 = x^2 + 2x + 1.$$

We subtract $2x + 1$ from both sides to obtain

$$(x + 1)^2 - (2x + 1) = x^2.$$

Next we subtract $x(2x + 1)$ to get

$$(x + 1)^2 - (2x + 1) - x(2x + 1) = x^2 - x(2x + 1),$$

2 ARITHMETIC

which by factoring becomes

$$(x + 1)^2 - (x + 1)(2x + 1) = x^2 - x(2x + 1).$$

Next we add $(2x + 1)^2/4$ to both sides:

$$\begin{aligned}(x + 1)^2 - (x + 1)(2x + 1) + (2x + 1)^2/4 \\ = x^2 - x(2x + 1) + (2x + 1)^2/4.\end{aligned}$$

Since both sides are perfect squares, we can factor them into

$$\left[(x + 1) - \frac{(2x + 1)}{2} \right]^2 = \left[x - \frac{(2x + 1)}{2} \right]^2.$$

Taking square roots of both sides, we obtain

$$\begin{aligned}(x + 1) - \frac{(2x + 1)}{2} &= x - \frac{(2x + 1)}{2} \\ (x + 1) &= x \\ 1 &= 0.\end{aligned}$$

This contradiction forces us to conclude that $(x + 1)^2$ does not equal $x^2 + 2x + 1$. What is wrong with this proof?

18. Prove Theorem 9.1 for arbitrary $r > 1$.
19. There are functions $f(n)$ such that $f(n) \neq O(p(n))$ for any polynomial $p(n)$ and $f(n) < r^n$ for every positive r and sufficiently large n . Find such a function.
20. Here is another exponentiation algorithm.

```
STEP 0.   Input  $x$  and  $m$  { $m$  a positive integer}
STEP 1-6. Algorithm DtoB {assume that the output is in string  $s$ , a  $(j + 1)$ -
          bit binary number}
STEP 7.   ans :=  $x$ 
STEP 8.   For  $i := (j - 1)$  down to 0 do
          STEP 9.   If the  $i$ th entry of  $s$  is 1,
                    then set ans :=  $x * ans * ans$ 
                    Otherwise, set ans := ans * ans
STEP 10.  Output ans and stop.
```

[COMMENT: "For $i := (j - 1)$ down to 0 do" means that i successively takes on the values $(j - 1), (j - 2), \dots, 0$ and for each value performs step 9.] Run this algorithm on some data to check that it correctly calculates x^m for m , a positive integer. Then compare the efficiency of this algorithm with that of FASTEXP.

ARITHMETIC OF SETS

3:1 INTRODUCTION

College Hall has decided to automate the internal mail system. Although the administrators communicate easily by electronic mail, they also need to circulate documents, such as memos, reports, and minutes of meetings, among their offices. They have hired an outside consultant who suggests that the flow of paper will be improved by the use of robotlike mail carts, called Mailmobiles. These carts can be programmed to travel through office corridors, to pause at designated points (or if the bumper hits anything!), and to stop at a location where they can be reprogrammed for another journey. Although most college administration buildings are irregularly shaped, let's think about a simple floor plan, the rectangular grid shown in Figure 3.1. We have marked the Mail Room with M and the President's Office with P . Each line segment indicates a corridor, and the corridor intersections are possible stopping points for the mailmobile.

Here are some questions that occur to the consultant while planning for the mailmobile.

1. A shortest trip from M to P requires the traversal of 11 corridors. In how many different ways can the mailmobile make a trip of shortest length from M to P ?
2. Is it possible to visit every stopping point exactly once on a trip (necessarily of longer length) from M to P ? If so, in how many different ways can such a trip be planned?
3. Is it possible to travel along every corridor exactly once on a trip from M to P ? If so, in how many different ways can such a trip be planned?

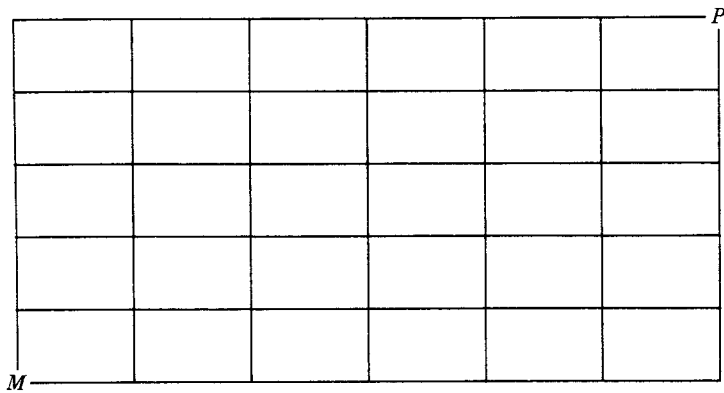


Figure 3.1 A 6×5 rectangular grid.

Question 1.1. Answer the preceding three questions on the 3×2 rectangular grid in Figure 3.2.

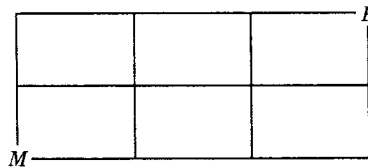


Figure 3.2 A 3×2 rectangular grid.

In this chapter we focus on the first of the three questions posed above. The second and third questions will be studied in depth in Chapter 8.

Let's consider some of the shortest trips between M and P in Figure 3.1. Such a trip is shortest if and only if it covers exactly 11 corridors. Think of Figure 3.1 as being a map, oriented with north at the top of the figure. Then we could describe a shortest trip from M to P , for example, by

E, E, E, E, E, E, N, N, N, N, N,

where N stands for moving north along a corridor and E for moving east. Two other shortest trips are

E, E, E, E, E, N, E, N, N, N, N, and N, N, N, N, E, E, E, E, E, N.

In fact, to move from M to P in a shortest path we must travel six units east and five units north. Furthermore if we write down any sequence of five Ns and six Es, then these give instructions for a shortest path from M to P .

Question 1.2. Why does a sequence of five Ns and six Es always stay inside the rectangle of Figure 3.1? Why does such a sequence describe a path that always reaches P , starting at M ? Describe all sequences of Ns and Es that correspond with a trip from M to P in Figure 3.2.

One way to describe these shortest paths from M to P in Figure 3.1 is as a subset of $\{N, E\}^{11}$ (recall the Cartesian product from Chapter 1). Specifically, the set of all shortest paths corresponds with the subset consisting of all of the 11-tuples that contain exactly five Ns.

Has the introduction of the Ns and Es helped us count the number of shortest paths from M to P or to find these paths? So far, not at all! We have only found another way to look at the problem. We shall construct an algorithm to list all suitable N and E sequences; however, there is a simple formula for the number of shortest paths, to which the work in this chapter will lead.

The diagrams in Figures 3.1 and 3.2, although they may not be representative of building or office floor plans, are ones that arise repeatedly in mathematics and computer science. A grid that has $(m + 1)$ vertical lines and $(n + 1)$ horizontal lines is known as an $m \times n$ **rectangular grid**; a picture of the $m \times n$ grid is given in Figure 3.3. This grid might represent streets in a city (like New York); it might represent the intricate connections on layers of silicon on a computer chip; or it might represent the pixels on a computer monitor. Thus this configuration is studied for a variety of reasons.

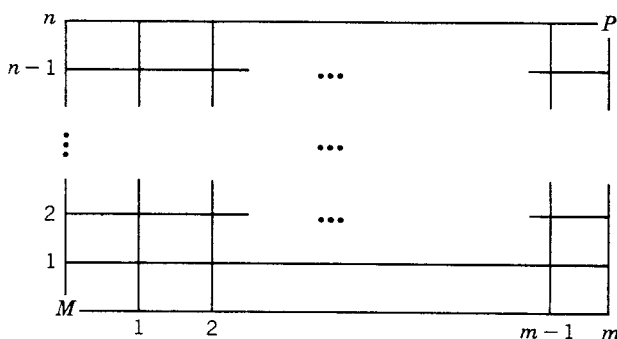


Figure 3.3

Question 1.3. Estimate the number of shortest paths from M to P in Figure 3.1: Choose the interval from the following list that most likely contains the correct number.

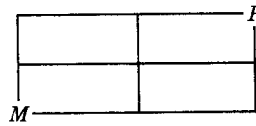
- (a) Less than 25.
- (b) Between 25 and 50.

3 ARITHMETIC OF SETS

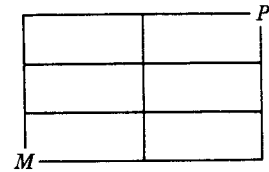
- (c) Between 50 and 100.
- (d) Between 100 and 250.
- (e) Between 250 and 500.
- (f) Between 500 and 1000.
- (g) Between 1000 and 1500.
- (h) More than 1500.

EXERCISES FOR SECTION 1

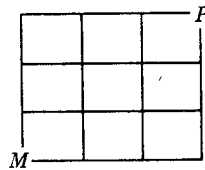
1. Here are four rectangular grids. For each draw all possible shortest paths from M to P .



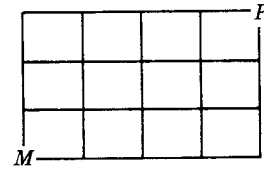
(a) 2×2 grid.



(b) 2×3 grid.



(c) 3×3 grid.



(d) 4×3 grid.

2. For each of the four grids in Exercise 1, describe a shortest path from M to P in terms of Es and Ns.
3. Among the following grids, which has the largest number of shortest paths from M to P ? 1×5 , 2×4 , 3×3 , 4×2 , and 5×1 .
4. Explain why the number of shortest paths from M to P in an $m \times n$ grid is the same as the number of shortest paths from M to P in an $n \times m$ grid.
5. What are the dimensions of the rectangular grid on which the sequence N, N, E, E, N, E, E, N, N, E, N, N, E gives a path from M to P ?
6. Check that the number of sequences of Ns and Es of length six containing exactly two Ns equals the number of six-digit binary numbers with exactly

- two ones. Is the same result true if there are three Ns and if the binary numbers have three ones? Is the same true if the sixes in these statements are all changed to sevens? Explain why.
7. In the four grids of Exercise 1, find a path from M to P that passes through every intersection point exactly once, or else deduce that there is no such path. In which grids is there more than one such path?
 8. Make a conjecture about what values of m and n are such that an $m \times n$ grid contains a path from M to P that passes through every intersection point exactly once.
 9. Explain why in the grids of Exercise 1 it is impossible to travel from M to P traveling along every corridor (or line segment) exactly once. Are there values of m and n for which such a path exists?
 10. Show that an $m \times n$ rectangular grid with M at the lower left corner and P at the upper right corner contains no more than 2^{m+n} shortest paths from M to P . When $n = 1$, determine exactly how many shortest paths there are from M to P .

3:2 BINOMIAL COEFFICIENTS

This section introduces the important factorial function and a counting device from the seventeenth century known as Pascal's triangle. With these we can readily calculate the number of shortest paths from the lower left-hand corner to the upper right-hand corner of any $m \times n$ grid. In addition, the factorial function and Pascal's triangle lead to the study of subsets of sets and related algorithms.

We return to rectangular grids. Figure 3.4 presents the $m \times n$ grid with each point in the grid labeled with its Cartesian coordinates. The point M is placed at the origin $(0, 0)$ and P lies at the point (m, n) .

Next we define a function f whose domain is the set of points of the grid; let $f(i, j)$ equal the number of shortest paths from $M = (0, 0)$ to (i, j) . Although we were originally looking only for the value $f(P)$, we'll find this number using the other function values.

Let's figure out some values of f . We say that $f(0, 0) = 1$ because there is only one shortest way to travel from M to M , that is, by doing nothing! Now $f(1, 0) = f(0, 1) = 1$, and $f(1, 1) = 2$, since we can reach $(1, 1)$ via $(0, 1)$ or $(1, 0)$.

Question 2.1. (a) Show that $f(0, 2) = f(2, 0) = 1$, $f(1, 2) = f(2, 1) = 3$, and $f(2, 2) = 6$.
 (b) Determine the values of $f(0, 3)$, $f(1, 3)$, $f(2, 3)$, $f(3, 2)$, $f(3, 1)$, and $f(3, 0)$.

Question 2.2. Explain why in the $m \times n$ rectangular grid $f(i, 0) = 1$ for $i = 0, 1, \dots, m$ and $f(0, j) = 1$ for $j = 0, 1, \dots, n$.

3 ARITHMETIC OF SETS

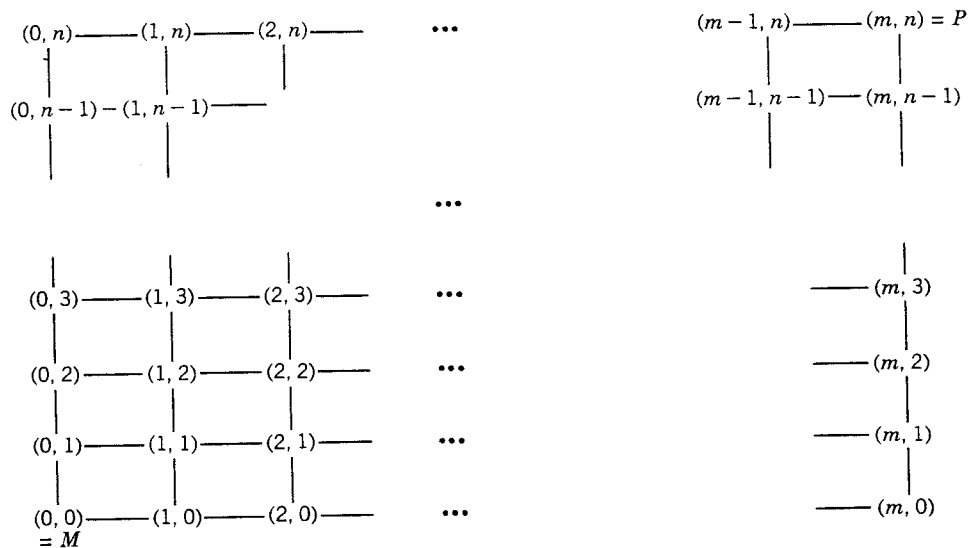


Figure 3.4 An $m \times n$ grid.

Consider a point (i, j) some place out in the middle of the grid shown in Figure 3.5. There are exactly two ways to arrive at (i, j) along a shortest path. Either we approach it from the point to the left $(i - 1, j)$ or from the point below $(i, j - 1)$. Thus the number of shortest paths from $(0, 0)$ to (i, j) is the number of shortest paths to $(i - 1, j)$ plus the number of shortest paths to $(i, j - 1)$, that is,

$$f(i, j) = f(i - 1, j) + f(i, j - 1).$$

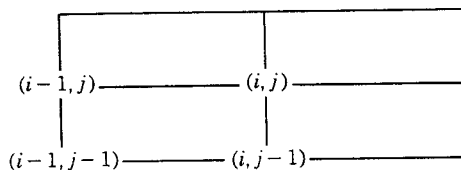


Figure 3.5

This relationship provides a method for calculating all the f values in a given grid. We know from Question 2.2 that $f(i, 0) = f(0, j) = 1$ for all i and j . From these initial values we can fill in the remaining values, moving from the lower left up to

the upper right:

$$f(1, 1) = f(0, 1) + f(1, 0) = 1 + 1 = 2$$

$$f(1, 2) = f(0, 2) + f(1, 1) = 1 + 2 = 3$$

$$f(2, 1) = f(1, 1) + f(2, 0) = 2 + 1 = 3$$

$$f(1, 3) = f(0, 3) + f(1, 2) = 1 + 3 = 4$$

$$f(2, 2) = f(1, 2) + f(2, 1) = 3 + 3 = 6,$$

and so on. Eventually, $f(m, n)$ can be computed as the sum of $f(m - 1, n)$ and $f(m, n - 1)$.

Question 2.3. Show that $f(3, 3) = 20$ and that $f(4, 2) = 15$.

This process would be tedious on the 6×5 grid and hopeless for much larger grids. We haven't yet found the promised simple formula for the number of shortest paths.

We now uncover Pascal's famous triangle. Suppose that we redraw Figure 3.4, omitting the coordinate labels and the line segments and writing instead the values of f at each point [see Figure 3.6(a)]. Next we rotate the figure $135^\circ (= 3\pi/4$ radians) clockwise so that M is at the top and P at the bottom. Then Pascal's triangle emerges as shown in Figure 3.6(b). A larger version appears in Figure 3.7.

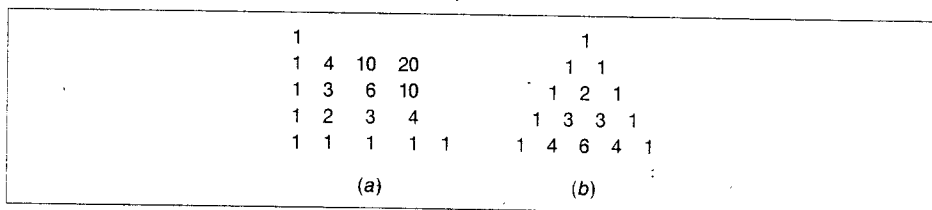


Figure 3.6 Pascal's triangle

The numbers in Pascal's triangle are organized by rows; numbers on the k th horizontal row (starting with $k = 0$) are said to form the k th row of Pascal's triangle. For example, here are the first rows:

the 0th row: 1,

the 1st row: 1 1,

the 2nd row: 1 2 1,

the 3rd row: 1 3 3 1,

3 ARITHMETIC OF SETS

and

the 4th row: 1 4 6 4 1.

Exactly as in the shortest path problem (once we account for the rotation), each row of **Pascal's triangle** begins and ends with a 1, and in the middle the entries are the sum of the two numbers immediately above.

Question 2.4. Calculate the 5th row of Pascal's triangle. Then determine the coordinates of all points in the $m \times n$ grid that end up, after the 135° rotation, on the 5th row of Pascal's triangle. Compare their f values with the entries of the 5th row.

We want to determine explicitly the numbers that lie on the k th row of Pascal's triangle for an arbitrary positive integer k . These in turn will give us the f values needed in the shortest path problem. To do this, we introduce the **factorial function**: For each natural number n , we define $n!$, read " n factorial," by

$$n! = n(n-1)(n-2)\cdots 3 \cdot 2 \cdot 1 \quad \text{if } n \text{ is positive,}$$

and

$$0! = 1.$$

The definition of $0!$ may be surprising, but as you'll see, it's useful to have $0!$ defined in this way. From the definition we see that

$$1! = 1, \quad 2! = 2 \cdot 1 = 2, \quad \text{and } 3! = 3 \cdot 2 \cdot 1 = 6.$$

Question 2.5. Calculate $n!$ for $n = 4, 5, 6, 7$, and 8 . Then find a value of n such that $n!$ is greater than 1,000,000.

Notice that the values of $n!$ grow rapidly as n increases.

Here's a useful property of n factorial that follows immediately from the definition:

$$n! = n[(n-1)!].$$

Now we can solve many mysteries of Pascal's triangle and of shortest paths within a grid. We define the **binomial coefficients** $\binom{k}{i}$, read " k choose i " or " k above i " or " k pick i " for all natural numbers k and i with $0 \leq i \leq k$ by

$$\binom{k}{i} = \frac{k!}{i!(k-i)!}.$$

Theorem 2.1. The n th row of Pascal's triangle consists of the binomial coefficients

$$\binom{n}{0}, \binom{n}{1}, \binom{n}{2}, \dots, \binom{n}{i}, \dots, \binom{n}{n-1}, \binom{n}{n}.$$

Proof. The proof is by induction on n . Some of the rows of the Pascal triangle are listed in Figure 3.7 and for $n < 6$ agree with the results of Example 2.1 and Question 2.6. Thus we assume that the k th row of Pascal's triangle consists of the binomial coefficients $\binom{k}{i}$ for $i = 0, 1, \dots, k$. In Figure 3.8 we display this row and label the unknown values in the $(k + 1)$ st row with variables x_1, x_2, \dots, x_k , whose values we must now determine.

$\binom{k}{0}$	$\binom{k}{1}$	$\binom{k}{2}$	\dots	$\binom{k}{i-1}$	$\binom{k}{i}$	\dots	$\binom{k}{k-1}$	$\binom{k}{k}$
1	x_1	x_2	x_3	\dots	x_i	x_{k-1}	x_k	1

Figure 3.8

We know that the zeroth and $(k + 1)$ st entries of the $(k + 1)$ st row equal 1. We also know that each entry equals the sum of the two numbers above. Thus x_1 must equal the sum

$$\binom{k}{0} + \binom{k}{1} = 1 + k = \binom{k+1}{1}.$$

Question 2.8. Show that $x_2 = \binom{k+1}{2}$.

In general, we see that

$$\begin{aligned} x_i &= \binom{k}{i-1} + \binom{k}{i} = \frac{k!}{(i-1)!(k-i+1)!} + \frac{k!}{i!(k-i)!} \\ &= \frac{k!}{(i-1)!(k-i)!} \cdot \left[\frac{1}{k-i+1} + \frac{1}{i} \right] = \frac{k!}{(i-1)!(k-i)!} \cdot \frac{i + (k-i+1)}{i(k-i+1)} \\ &= \frac{(k+1)!}{i!(k-i+1)!} = \binom{k+1}{i}. \end{aligned}$$

Thus the i th element in the n th row of Pascal's triangle is $\binom{n}{i}$. □

Corollary 2.2. $\binom{k+1}{i} = \binom{k}{i-1} + \binom{k}{i}$ for all $1 \leq i \leq k$.

Proof. This was exactly the central calculation in the induction proof of Theorem 2.1. \square

Why do we want to know all these facts about rows of the Pascal triangle? First of all, they provide the solution to the shortest path problem. We began this section with an effort to calculate $f(P)$, the number of shortest paths from M to P in the 6×5 rectangular grid. The point P lies on the 11th row of Pascal's triangle and is the 6th entry, counting beginning with 0. Thus by Theorem 2.1,

$$f(P) = \binom{11}{6} = \frac{11!}{6! \cdot 5!} = \frac{11 \cdot 10 \cdot 9 \cdot 8 \cdot 7}{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 462.$$

In the case of the general $m \times n$ rectangular grid, the point $P = (m, n)$ lies on the $(m+n)$ th row of the Pascal triangle and is the m th entry. Thus in this case

$$f(P) = \binom{m+n}{m} = \frac{(m+n)!}{m! n!},$$

an explicit formula for the number of shortest paths.

Question 2.9. How many shortest paths are there from $(0, 0)$ to $(4, 3)$ in a rectangular grid?

Look back at Pascal's triangle in Figure 3.7 and notice that there is a great deal of symmetry in the binomial coefficients. For a fixed value of k the numbers $\binom{k}{0}, \binom{k}{1}, \dots, \binom{k}{i}, \dots, \binom{k}{k}$ begin with 1, first increase as i increases, and then decrease through the same values back to 1. In other words, the second half of this sequence is a mirror image of the first half. The largest value occurs in the middle.

Theorem 2.3. $\binom{k}{i} = \binom{k}{k-i}$ for all $i = 0, 1, \dots, k$.

Proof.

$$\begin{aligned} \binom{k}{i} &= \frac{k!}{i!(k-i)!} = \frac{k!}{(k-i)!i!} \\ &= \frac{k!}{(k-i)!(k-(k-i))!} = \binom{k}{k-i}. \end{aligned} \quad \square$$

3 ARITHMETIC OF SETS

The largest value of $\binom{k}{i}$ for fixed k occurs when $i = k/2$ if k is even and when $i = (k - 1)/2$ and $(k + 1)/2$ if k is odd.

Theorem 2.4. Given natural numbers k and i with $i \leq (k - 1)/2$,

$$\binom{k}{i} \leq \binom{k}{i+1}.$$

Proof. Given $i \leq (k - 1)/2$, we multiply both sides by 2 to get $2i \leq k - 1$. Next we add 1 and subtract i to get $i + 1 \leq k - i$. Taking reciprocals, we get

$$\frac{1}{k-i} \leq \frac{1}{i+1}. \quad (*)$$

Now

$$\binom{k}{i} = \frac{k!}{i!(k-i)!} = \frac{1}{(k-i)} \cdot \frac{k!}{i!(k-i-1)!}$$

which after substituting (*) yields

$$\binom{k}{i} \leq \frac{1}{(i+1)} \cdot \frac{k!}{i!(k-i-1)!} = \binom{k}{i+1}. \quad \square$$

Corollary 2.5. For natural numbers k and j with $j \geq (k - 1)/2$

$$\binom{k}{j} \geq \binom{k}{j+1}.$$

Proof. Since $j \geq (k - 1)/2$, $(k - j - 1) \leq k - (k - 1)/2 - 1 = (k - 1)/2$. Applying Theorem 2.4 with $i = k - j - 1$, we have

$$\binom{k}{k-j-1} \leq \binom{k}{k-j}.$$

Applying Theorem 2.3, we get

$$\binom{k}{j+1} \leq \binom{k}{j}. \quad \square$$

Theorem 2.4 and Corollary 2.5 combine to tell us that the largest value among

$$\binom{k}{0}, \binom{k}{1}, \dots, \binom{k}{i}, \dots, \binom{k}{k}$$

occurs at

$$\binom{k}{\lfloor k/2 \rfloor}.$$

Theorems 2.3, 2.4, and Corollary 2.5 are attractive and interesting, and they are also useful. We need these results for the analysis of the complexity of the next major algorithm, which generates all k -subsets of an n -set.

EXERCISES FOR SECTION 2

1. Fill in the f value for every point on the 4×4 rectangular grid.
2. In rotating the rectangular grid into Pascal's triangle, the vertical and horizontal rows of the grid become diagonals in the triangle. Even more important, it is the diagonals from the point $(0, i)$ to the point $(i, 0)$ that become the rows of Pascal's triangle. Determine the coordinates of all the points in the grid that end up on the 3rd row of Pascal's triangle. Then do the same for the 4th row.
3. Find the integer coordinates of all the points in the rectangular grid that end up on the k th row of Pascal's triangle. On what row of Pascal's triangle does the point (i, j) end up?
4. Is the following statement true or false: Every number on the k th row of Pascal's triangle, except for the first and last, is divisible by k . If true, explain why. If false, characterize the rows for which it is a true statement.
5. Calculate the sum of the k th row of the Pascal triangle for $k = 3, 4, 5$, and 6. Formulate a conjecture about the sum of the k th row of the triangle.
6. Calculate the following binomial coefficients:

$$\binom{6}{3}, \quad \binom{7}{3}, \quad \binom{18}{2}, \quad \binom{18}{15}, \quad \binom{13}{3}, \quad \binom{97}{1}, \quad \binom{100}{2}.$$

7. Verify Corollary 2.2 by calculating the following sums and checking that the corollary is correct in these cases:

$$\binom{5}{3} + \binom{5}{4}, \quad \binom{6}{2} + \binom{6}{3}, \quad \binom{8}{4} + \binom{8}{5}.$$

8. Find the largest binomial coefficient of the form $\binom{14}{i}$ and $\binom{15}{j}$.
9. The number 10 equals at least four different binomial coefficients:

$$\binom{10}{1}, \quad \binom{10}{9}, \quad \binom{5}{2}, \quad \text{and} \quad \binom{5}{3}.$$

3 ARITHMETIC OF SETS

Are there any other values of k and j such that $10 = \binom{k}{j}$? Find a number $n > 1$ that equals at least five different binomial coefficients, and then find a number k that equals only the two binomial coefficients $\binom{k}{1}$ and $\binom{k}{k-1}$.

10. Construct an algorithm FACTORIAL that upon input of a positive integer j , calculates and outputs $j!$.
11. How many zeros does $15!$ end in? Find the smallest value of $n!$ that is divisible by 1,000,000.
12. Notice that $2 = 2!$, $3 = 2! + 1!$, $4 = 2 \cdot 2!$, and $5 = 2 \cdot 2! + 1!$. Express 6, 7, 8, 9, and 10 as sums of multiples of factorials in the following form:

$$n = a_j \cdot j! + a_{j-1} \cdot (j-1)! + \cdots + a_1 \cdot 1!$$

where a_j, a_{j-1}, \dots, a_1 are integers satisfying $a_i \leq i$ for $i = 1, 2, \dots, j$. This form is called the **factorial representation** of the integer n . (See Supplementary Exercises 8 and 9.)

13. Determine whether the following are true or false. Give reasons for your answers.

(a) $n! = O((n+1)!)$	(b) $n! + 5 = O(n!)$
(c) $n! = O((n-1)!)$	(d) $n! + (n-1)! = O(n!)$
(e) $1! + 2! + 3! + \cdots + n! = O(n!)$.	
14. If p is a prime, show that each of the following numbers is a composite number. (A number that is not prime is called **composite**.)

$$p! + 2, p! + 3, \dots, p! + p.$$

Describe how to obtain 1000 consecutive composite numbers. If p is not a prime, can you use the same construction to obtain m consecutive composites for any integers p and m ?

15. Prove Theorem 2.3 using the grid path formulation of binomial coefficients.
16. Each row of Pascal's triangle is called **symmetric** because the i th entry equals the $(k-i)$ th. Each row is called **unimodal**, since the numbers increase to a maximum and then decrease. Find an example of a sequence of 10 numbers that is symmetric but not unimodal. Then find an example of a sequence of 10 numbers that is unimodal but not symmetric.
17. Explain why

$$\binom{k}{i} = \binom{k-2}{i} + 2\binom{k-2}{i-1} + \binom{k-2}{i-2}.$$

Then express $\binom{k}{i}$ in terms of binomials coefficients of the form “ $k - 3$ choose something.” Interpret these identities in the grid.

18. How many shortest paths proceed from $(0, 0)$ to $(6, 9)$ in the rectangular grid? How many of these go through the point $(4, 5)$?
19. How many shortest paths proceed from $(0, 0)$ to (m, n) through the point (r, s) if $0 < r < m$ and $0 < s < n$?
20. Suppose that k is odd. Explain why Theorem 2.3 and Corollary 2.4 imply that

$$\binom{k}{\frac{k-1}{2}} = \binom{k}{\frac{k+1}{2}} \geq \binom{k}{i} \quad \text{for all } 0 \leq i \leq k.$$

Then when k is even, explain why for all $0 \leq i \leq k$

$$\binom{k}{k/2} \geq \binom{k}{i}.$$

3:3 SUBSETS OF SETS

This section considers the problems of counting and generating all the j -subsets of an n -set.

Example 3.1. Suppose that we want to know how many 3-subsets there are in a 6-set. One method would be to list them all. We now do this for the 6-set $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$:

$$\begin{array}{lll} \{a_1, a_2, a_3\}, & \{a_1, a_2, a_4\}, & \{a_1, a_2, a_5\} \\ \{a_1, a_2, a_6\}, & \{a_1, a_3, a_4\}, & \{a_1, a_3, a_5\} \\ \{a_1, a_3, a_6\}, & \{a_1, a_4, a_5\}, & \{a_1, a_4, a_6\} \\ \{a_1, a_5, a_6\}, & \{a_2, a_3, a_4\}, & \{a_2, a_3, a_5\} \\ \{a_2, a_3, a_6\}, & \{a_2, a_4, a_5\}, & \{a_2, a_4, a_6\} \\ \{a_2, a_5, a_6\}, & \{a_3, a_4, a_5\}, & \{a_3, a_4, a_6\} \\ \{a_3, a_5, a_6\}, & \{a_4, a_5, a_6\}. & \end{array}$$

Thus there are 20 3-subsets of a 6-set. Notice that $20 = \binom{6}{3}$.

3 ARITHMETIC OF SETS

In this problem we asked for the number of 3-subsets. We would have been content to have the number 20 without the list of subsets.

Exercises 1.7.2 and 1.7.3 asserted that n equals both the number of 1-subsets of an n -set and the number of $(n - 1)$ -subsets of an n -set. Notice that

$$n = \binom{n}{1} = \binom{n}{n-1}.$$

Similarly, the number of 2-subsets and the number of $(n - 2)$ -subsets of an n -set equals $\binom{n}{2}$ as seen in Exercises 1.7.5 and 1.7.7. Thus in these cases the exact subset count is given by a binomial coefficient.

Question 3.1. If $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, check that the number of 4-subsets of A is $\binom{6}{4}$ and the number of 5-subsets is $\binom{6}{5}$.

Specific examples all point toward the truth of the following theorem. The idea of the proof is similar to the proof by induction that an n -set has 2^n subsets, given in Section 2.4.

Theorem 3.1. For $j = 0, 1, \dots, n$, the number of j -subsets of an n -set equals the binomial coefficient $\binom{n}{j}$.

Proof. The proof of this theorem is by induction on n . For $n = 0$ and $j = 0$, there is only one subset of the empty set (itself) and $1 = \binom{0}{0}$. When $n = 1$, there is $\binom{1}{0}$ 0-subset, the empty set, and $\binom{1}{1}$ 1-subset, the entire set.

We assume that the theorem is valid for $n = k$ and try to prove it for $n = k + 1$. Suppose that $A = \{a_1, a_2, \dots, a_{k+1}\}$, and consider all j -subsets of A . We divide these into two piles, those that contain the last element a_{k+1} and those that don't.

Question 3.2. Divide the 3-subsets of the 6-set given in Example 3.1 into two piles depending on whether a_6 is contained in the 3-subset or not.

Let $A' = A - \{a_{k+1}\}$. See Figure 3.9. If S is a j -subset of A that contains a_{k+1} , then $S - \{a_{k+1}\}$ is a $(j - 1)$ -subset of the k -set A' . By the inductive hypothesis we know that there are $\binom{k}{j-1}$ of these $(j - 1)$ -subsets of A' . Thus there are