For example, $f(x) = x^2$ with D equal all real numbers and $g(x) = x^2$ with D equal all reals greater than one are different functions. They have different graphs, for instance. Given a function f with domain D, the range R is determined. However, in a specific instance, it might take some effort to decide what the range is. In the preceding example the range of f is all nonnegative real numbers and the range of g is all reals greater than one. In contrast the target of a function is not uniquely determined. It is often useful to designate a target T that is a large and familiar set containing the range. Thus we stated that the functions f and g above have the real numbers as target. We could also have picked the nonnegative reals as the target.

In this section we shall consider repeatedly the functions that map a binary number to its decimal equivalent and a natural number to its binary equivalent. We don't have formulas for these functions, like $f(x) = x^2$, but we can think about these mappings, and we have algorithms to compute these functions whenever necessary.

Example 9.1. Let B be the set of all binary numbers, or equivalently all finite strings of zeros and ones, and let N be the set of all natural numbers expressed in decimal notation. Then f, g, h, and j given below are functions from B to N. For s in B,

$f(s)$ equals the decimal equivalent of s,

$g(s)$ equals the number of bits in s,

$h(s)$ equals the number of ones in s,

and

$j(s)$ equals the ones bit of s.

For instance, if $s = 110010$, then $f(s) = 50$, $g(s) = 6$, $h(s) = 3$, and $j(s) = 0$. The range of f, g, and h is in each case all of N. To see this, let m be any decimal number in N. If s is m's binary equivalent, then $f(s) = m$. If r is the binary number consisting of m ones, then $g(r) = h(r) = m$. However, the range of j is $\{0, 1\}$.

Here are two mappings from B to N that are not functions. For s in B,

$k(s)$ equals the fifth bit in s, counting from the left,

and

$$l(s) = \begin{cases} 1 \text{ if } s \text{ ends with } 1 \\ 2 \text{ if } s \text{ ends with } 0 \\ 4 \text{ if } s \text{ ends with } 00. \end{cases}$$

The mapping k is not defined on all of B, only on those with five or more digits, and l specifies two different images for strings ending with two zeros.

Question 9.1. Suppose that B and N are as in the preceding example. Define $b: N \to B$ by $b(r) = s$ if s is the binary equivalent of the decimal number r. Explain why the range of b is all of B.

Question 9.2. Which of the following is a function from N to B, where N and B are as defined in Example 9.1? For r in N,

$$f_1(r) \text{ equals the number of digits of } r,$$

$$f_2(r) = \begin{cases} 0 & \text{if } r \text{ is even} \\ 1 & \text{if } r \text{ is odd,} \end{cases}$$

$$f_3(r) \text{ equals the string of } r \text{ ones,}$$

and

$$f_4(r) = \begin{cases} 0 & \text{if 2 divides } r \\ 1 & \text{if 3 divides } r \\ 1 & \text{if neither 2 nor 3 divides } r. \end{cases}$$

For each that is a function, specify its range.

A function is said to be **onto** or an onto function if its range equals its target, $R = T$. Thus functions f, g, and h of Example 9.1 are onto, but j is not. In Figure 1.8 the first function shown is onto whereas the second is not onto. We also say that two functions f and g are **equal** if they have the same domain D and $f(d) = g(d)$ for every d in D.

Sometimes the domain of a function is a set of sets.

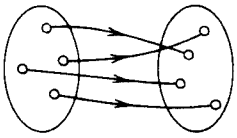
Example 9.2. If $U = \{a_1, a_2, \ldots, a_n\}$, let $P(U)$ be the set of all subsets of U. We know that $P(U)$ contains 2^n subsets. We define the complementation function $c: P(U) \to P(U)$ by $c(A) = A^c$ for every A in $P(U)$. Then c is an onto function, because for every set B in $P(U)$ we have

$$c(B^c) = (B^c)^c = B.$$

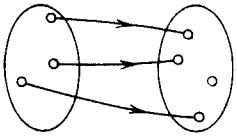
If we define $n: P(U) \to P(U)$ by $n(A) = \varnothing$ for every A in $P(U)$, then n is not onto. It is evident that the functions c and n have the same domain, but they are not equal.

Question 9.3. Which of the functions in Questions 9.1 and 9.2 are onto?

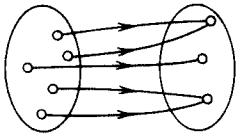
A function is said to be **one-to-one** (or 1-1) if it maps distinct elements of the domain to distinct elements of the range. In other words, if $d \neq d'$, then $f(d) \neq f(d')$. A diagram of different function properties is shown in Figure 1.9.



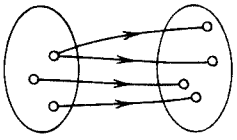
(a) One-to-one and onto function.



(b) One-to-one but not onto function.



(c) Not one-to-one function.



(d) Not a function.

Figure 1.9

Example 9.1 (continued). Here again are the functions defined in Example 9.1. For s in B,

$f(s)$ equals the decimal equivalent of s,

$g(s)$ equals the number of bits in s,

$h(s)$ equals the number of ones in s,

and

$j(s)$ equals the ones bit of s.

The function f is one-to-one because if $s \neq s'$, then their decimal equivalents will be different. However, g is not one-to-one because, for example, $g(101) = 3 = g(111)$. Neither h nor j is one-to-one, since $h(101) = h(110) = 2$, and $j(101) = j(11) = 1$.

Question 9.4. Let the functions b, f_2, and f_3 be as defined in Questions 9.1 and 9.2. Which of these functions is one-to-one and why?

Question 9.5. Let $U = \{a_1, a_2, \ldots, a_n\}$. Is $c: P(U) \to P(U)$ defined by $c(A) = A^c$ a one-to-one function?

Suppose that a function $f: D \to T$ is not one-to-one. Then we know that there is an element t and two elements $d \neq d'$ in D such that $f(d) = f(d') = t$. The next result gives a condition under which a function is surely not one-to-one.

The Pigeonhole Principle. If f is a function with finite domain D and target T, where $|D| > |T|$, then f is not one-to-one. In particular, there is some element t in T that is the image of at least two different elements of D.

Why is this so? The function f is either one-to-one or it isn't. Since $R \subseteq T$ for any function f, it follows that $|R| \leq |T|$. If $|D| > |T|$ and f were one-to-one, then $|R| = |D|$ and so $|R| > |T|$, a contradiction. Thus f cannot be one-to-one.

This principle has far-reaching applications in combinatorics. Its name derives from the following flightful application: If more than n pigeons fly into n pigeonholes, then some pigeonhole must contain at least two pigeons.

Example 9.3. Let S be a set of 11 or more binary numbers. Then at least 2 elements of S must have the same last digit when expressed in decimal notation. The pigeonhole principle shows why: Define f^* to be the function with domain S and target $T = \{0, 1, \ldots, 9\}$, where for s in S, $f^*(s)$ equals the last digit of s when s is expressed in decimal notation. Since $|S| > 10 = |T|$, there must be two numbers in S that map to the same element of T.

For further examples see Exercises 20–23 and Supplementary Exercises 11 and 12.

Often a function maps one set into the same set, that is, $T = D$. This was the case in Example 9.2 and Question 9.5. When $T = D$ and the function is one-to-one and onto, it is called a **permutation**. The examples in Figure 1.10 illustrate per-

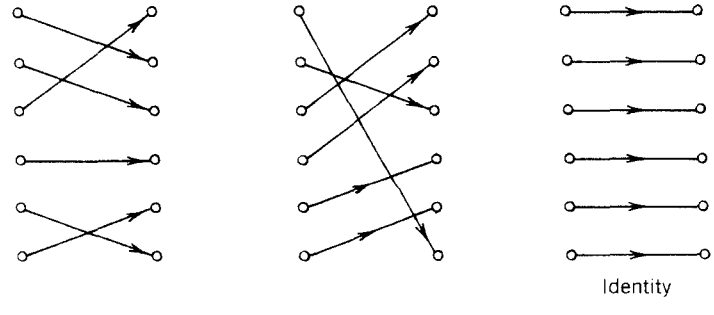


Figure 1.10 Permutations

mutations. One important, but easy, permutation is called the **identity map** or **identity permutation**. We define it by $i:D \to D$, where $i(d) = d$ for every d in D. This map doesn't do much, but it is one-to-one and onto; soon we'll see that it plays an important role.

When $T = D$, the properties of being one-to-one and onto are closely related.

Theorem 9.1. Let D be a finite set and let $f:D \to D$ be a function. Then f is one-to-one if and only if f is onto.

Proof. If D has n elements and f is one-to-one, then R, the range of f, has n elements also. Since $R \subseteq D$ and they have the same (finite) cardinalities, $R = D$ and so f is onto. On the other hand, if f is onto but not one-to-one, then some pair of elements in D gets mapped by f to the same element. Consequently, the remaining $n - 2$ elements of the domain must be mapped to $n - 1$ elements of the range. Then some element must have two images, contradicting the definition of a function. Thus f must be one-to-one. (See Exercise 24 for another look at this idea.) □

Just as we can combine sets to form new sets, so can we combine functions to form new functions. Suppose that $f:D \to T$ and $g:T \to W$ are such that the range of f is contained in the domain of g. Then we define the **composite** of g with f, denoted $g \circ f$, to be the function

$$g \circ f:D \to W, \qquad \text{where } g \circ f(d) = g(f(d))$$

for all d in D. In words this means that for d in D, we first map d to T using f, and then we map the result, $f(d)$, to W using g. This process of combining two functions is known as **composition**. Notice that in the composition $g \circ f$, it is f that gets performed first even though it is g that is on the left and thus read first. This idea is illustrated in Figure 1.11.

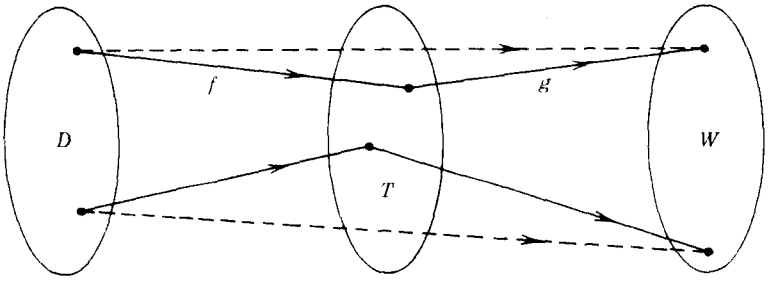


Figure 1.11 $g \circ f$ is shown in dashed lines.

Example 9.4. If $f:B \to N$ is defined as before with $f(s) = m$, where m is the decimal equivalent of s, and if $g:N \to N$ is defined by

$$g(t) = \begin{cases} 0 & \text{if } t \text{ is even} \\ 1 & \text{if } t \text{ is odd,} \end{cases}$$

then we can define $g \circ f:B \to N$ by

$$\begin{aligned} g \circ f(s) &= g(f(s)) \\ &= \begin{cases} 0 & \text{if } f(s) \text{ is even} \\ 1 & \text{if } f(s) \text{ is odd.} \end{cases} \end{aligned}$$

Thus $g \circ f = j$, where j is defined in Example 9.1.

Now suppose that $f:D \to T$ and $g:T \to D$; that is, the domain of g is the target of f, and vice versa. Then

$$g \circ f:D \to D,$$

and the composite function takes us back where we started from.

Sometimes $g \circ f$ does even more than that. If $g \circ f = i$, the identity map $i(d) = d$, then g is called the **inverse** of f. Specifically, if

$$g \circ f(d) = g(f(d)) = d$$

for every d in D, then g undoes the work of f, and g is the inverse of f. Similarly, if we compose the other way around, $f \circ g:T \to T$ and get

$$f \circ g(t) = f(g(t)) = t$$

for all t in T, then f is called the inverse of g. (See Figure 1.12, where f is shown with a solid line, its inverse by a dashed line.)

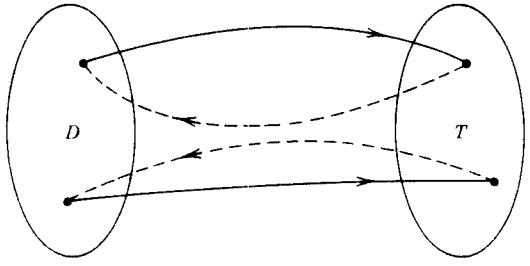


Figure 1.12

Example 9.1 (continued again). Let $f(s) = m$ and $b(m) = s$ be defined as in Example 9.1 and Question 9.1. Then $b \circ f : B \to B$ maps a binary number to its decimal equivalent and back to its binary equivalent. Thus $b \circ f = i$, and b is the inverse of f. Also $f \circ b = i$, since translating decimals to binary and then back to decimals returns the original decimal number. Thus f is also the inverse of b. Note that the two identity functions of this example are different, since they have different domains.

Example 9.4 (continued). Let f be as already given, and $g : N \to B$ be defined by $g(t) = 0$ or 1 according as t is even or odd. Then $g \circ f : B \to B$ and $f \circ g : N \to N$, but neither composite mapping is the identity. For instance, $g \circ f(101) = g(5) = 1$, and $f \circ g(2) = f(0) = 0$.

It is not by chance that in the latest continuation of Example 9.1 f and g were inverses of each other.

Theorem 9.2. Suppose that both $f : D \to T$ and $g : T \to D$ are onto functions. Then f is the inverse of g if and only if g is the inverse of f.

Proof. Suppose that f is the inverse of g, that is, $f \circ g = i$. We must show that g is the inverse of f, that is, we must show that $g \circ f(d) = d$ for all d in D. For any element d in D, since $g : T \to D$ is onto, there is an element t in T such that $g(t) = d$. Then

$$\begin{aligned} g \circ f(d) = g \circ f(g(t)) &= g(f(g(t))) && \text{by definition of } g \circ f \\ &= g(f \circ g(t)) && \text{by definition of } f \circ g \\ &= g(i(t)) && \text{since } f \circ g = i \\ &= g(t) = d. \end{aligned}$$

The remaining proof, that if g is the inverse of f, then f is the inverse of g, goes the same way. □

Question 9.6. Let U be a finite set, $P(U)$ the set of all subsets of U, and $c : P(U) \to P(U)$ be defined by $c(A) = A^c$. Then explain why c is its own inverse, that is, why $c \circ c = i$.

In summary, a function, like an algorithm, has input and output, elements of the domain and range, respectively. So what is the difference between a function and an algorithm? An algorithm may have as input values for several variables, say x, q, and n. If x can be from the set A, q from B, and n from C, then the input to the algorithm can be thought of as one element in the Cartesian product

$A \times B \times C$. Similarly, the output of the algorithm can be thought of as an element from a Cartesian product. In short, every algorithm is a function. Conversely, the functions in this book (though not all functions) are mappings that can be computed by algorithms, sometimes by several different algorithms.

There is, however, a striking difference in the contexts in which these concepts get used. When we think of an algorithm, we are vitally concerned with the mechanism by which a domain element d gets mapped to its corresponding range element $f(d)$. In contrast, when we think of a function it is the correspondence itself that matters, not how $f(d)$ is computed. One of the main goals of discrete mathematics is to supply the tools which enable a rational choice among various algorithms that evaluate a function.

EXERCISES FOR SECTION 9

1. Let N be the set of all nonnegative integers. Which of the following are functions with domain and target N?

(a) $f(n) = n + 1$. **(b)** $f(n) = 2n + 1$.

(c) $f(n) = \frac{n}{2} + 1$. **(d)** $f(n) = n - 1$.

(e) $f(n) = n^2 + 1$. **(f)** $f(n) = \sqrt{n} + 1$.

(g) $f(n) = \frac{1}{n} + 1$. **(h)** $f(n) = n^3$.

(i) $f(n) = 2^n$. **(j)** $f(n)$ equals the remainder when n is divided by 3.

Of those that are functions with domain and target N, find their range and determine whether or not they are onto.

2. Let R be the set of all real numbers. Which of the following are functions with domain and target R?

(a) $f(x) = 2x + 1$. **(b)** $f(x) = \frac{x}{2} + 1$.

(c) $f(x) = x - 1$. **(d)** $f(x) = \sqrt{x} + 1$.

(e) $f(x) = \frac{1}{x} + 1$. **(f)** $f(x) = x^2 - 3x + 2$.

(g) $f(x) = |x|$, where $|x|$ stands for the absolute value of x.

Of those that are functions, find those that are one-to-one.

3. Give an example of a function, with domain and target the positive integers, that is onto and is not the identity map.

4. If two functions are not equal, they are called different. Suppose that $A = \{a_1, a_2, \ldots, a_n\}$. How many different functions are there with domain A and target $\{0, 1\}$? How many of these are onto and how many of these are one-to-one?

5. For each of the following conditions give an example of a function $f: Z \to Z$, where Z is the set of all integers, that satisfies the condition:
   (i) f is onto and one-to-one.
   (ii) f is onto but not one-to-one.
   (iii) f is one-to-one but not onto.
   (iv) f is neither one-to-one nor onto.
   (v) Every integer is the image of exactly two integers.
   (vi) f has an inverse g.
   (vii) f does not have an inverse g.

6. Let $U = \{a_1, a_2, a_3\}$ and let $S = \{a_1\}$. For every set A in U we define the map $f: P(U) \to P(U)$ by $f(A) = A \cap S$ and the map $g: P(U) \to P(U)$ by $g(A) = A \cup S$. Write down the image $f(A)$ and $g(A)$ for every subset A. Is either f or g onto?

7. Let $U = \{a_1, \ldots, a_n\}$, and for S a fixed subset of U define $f(A) = A \cap S$ and $g(A) = A \cup S$ for every subset A of U. For what sets S is f an onto function and for what sets S is g onto? Is either f or g one-to-one?

8. Fix a finite universal set U. The size function $s: P(U) \to N$, where N is the set of all nonnegative integers, is given by $s(A) = |A|$ for all subsets A of U. What is the range of s? Is it one-to-one?

9. The characteristic function of a set S, a fixed subset of the universe U, is given by $\chi_S(x): U \to N$, where

$$\chi_S(x) = \begin{cases} 0 & \text{if } x \text{ is not in } S \\ 1 & \text{if } x \text{ is in } S. \end{cases}$$

   For a fixed subset S of U, let $h: U \to N$ be given by $h(x) = |\{x\} \cap S|$. Explain why χ_S and h are equal functions.

10. Suppose that A and B are finite sets and $A \subseteq B$. Explain why $A = B$ if and only if $|A| = |B|$.

11. Suppose that $f: D \to D$ is a function with the same domain and target. Then we can define $f^2 = f \circ f$ as the composition of f with itself. For each of the following, write down a simple expression for f^2:
   (i) $f: R \to R, f(x) = x$.
   (ii) $f: R \to R, f(x) = x^2$.
   (iii) $f: N \to N, f(i) = i + 1$.
   (iv) $f: N \to N, f(j) = 2j + 1$.

(v) $f: P(U) \to P(U)$, $f(A) = A^c$.
(vi) $f: P(U) \to P(U)$, $f(A) = \varnothing$.

12. If $f: D \to D$, then we can also define $f^3 = f(f^2) = f \circ (f \circ f)$ and $f^4 = f \circ (f^3) = f \circ [f \circ (f \circ f)]$. Determine f^3 and f^4 for each of the functions in Exercise 11.

13. If $f: D \to D$, we define f^n, where ***n is*** a positive integer, by $f^1 = f$, $f^2 = f \circ f$, and, in general, $f^n = f \circ (f^{n-1})$. For each of the functions in Exercise 11 find an expression for f^n in terms of n.

14. Let $A = \{a_1, a_2, \ldots a_n\}$ and form the Cartesian product $A \times A$. Define two projection functions, P_1 and P_2: $A \times A \to A$ by $P_1(a_i, a_j) = a_i$ and $P_2(a_i, a_j) = a_j$. Is either of these functions one-to-one or onto'?

15. Suppose that we define a function b:.4 $\to A \times A$ by $b(a_i) = (a_i, a_1)$. Then is ***b*** the inverse of either of the projection functions P_1 or P_2? *1s* either P_1 or P_2 the inverse of ***b?***

16. Show by example that Theorem 9.1 is false if D is not finite.

17. Suppose that f and g are functions such that f: $D \to T$, $g: T \to D$, and $f \circ g = i$. Prove that f is onto.

18. Suppose that $f: D \to T$ *is* a one-to-one function and its domain D is finite. Then prove that there is a function g that is f's inverse.

19. For each of the following, find the inverse off. Let Z stand for the integers, N the nonnegative integers, R the real numbers, and U = {al, a_2, ... ***an}.***
(i) $f: Z \to Z$, $f(x) = x + 1$.
(ii) $f: R - \{0\} \to R$, $f(x) = \dfrac{1}{x}$.
(iii) $f: \mathbf{N} \to R$, $f(x) = \sqrt{x}$.
(iv) $f: U \to \mathbf{N}$, $f(a_i) = i$.
(v) $f: U \to U$, $f(a_i) = a_{n-i}$.

20. Suppose that $f: D \to T$, where $|D| > |T|$. Can you conclude any of the following? Explain.
(a) There are at least two elements t_1 and t_2 of T that are each the image of two or more domain elements.
(b) Every t in T is the image of at least two domain elements.
(c) There is an element t in T and three distinct elements d_1, d_2, and d_3 in D such that $f(d_1) = f(d_2) = f(d_3) = t$.

21. Explain why a set of 16 numbers selected from {2,. . . . 50} must contain two with a common divisor greater than one.

22. Explain why a subset of 51 numbers taken from { 1,2,. ... 100} must contain two numbers. where one is a divisor of the other.

23. Let f be a function with domain D and target T. If $|D| = d$ and $|T| = n$, then explain why there is an element t of T that is the image of at least d/n elements of D.

24. Here is a stronger version of Theorem 9.1. If $f:D \to T$ is a function, where $|D| = |T|$, then f is one-to-one if and only if f is onto. Prove that this is so and explain why this is more general than Theorem 9.1.

25. Suppose that f and g are functions such that $f:D \to T$ and $g:T \to D$, but one of them is not onto. Then is the conclusion of Theorem 9.2 still true; that is, is it still the case that f is the inverse of g if and only if g is the inverse of f?

1:10 BOOLEAN FUNCTIONS AND BOOLEAN ALGEBRA

We return to the dichotomies mentioned in Section 2 and concentrate on functions with two-element targets: $T = \{0, 1\}$ or {True, False} or {yes, no}, and so on; without loss of generality we assume that $T = \{0, 1\}$. Such functions are central to computer science and related mathematics. In this section we develop algebraic properties of these functions, characterize their fundamental forms, and consider an easily stated, but unresolved, research problem, known as the Satisfiability Problem.

Definition. Let $Z_2 = \{0, 1\}$. A function $f:(Z_2)^n \to Z_2$, with domain a Cartesian product of Z_2 and target Z_2, is called a **Boolean function.**

Example 10.1. Here are three fundamental Boolean functions.

1. **NOT:** $Z_2 \to Z_2$ defined by $\mathrm{NOT}(x) = \begin{cases} 0 & \text{if } x = 1 \\ 1 & \text{if } x = 0. \end{cases}$
   This is usually written $\mathrm{NOT}(x) = \sim x$.

2. **AND:** $(Z_2)^2 \to Z_2$ defined by $\mathrm{AND}(x, y) = \begin{cases} 1 & \text{if } x = y = 1 \\ 0 & \text{otherwise.} \end{cases}$
   This is usually written $\mathrm{AND}(x, y) = x \wedge y$.

3. **OR:** $(Z_2)^2 \to Z_2$ defined by $\mathrm{OR}(x, y) = \begin{cases} 0 & \text{if } x = y = 0 \\ 1 & \text{otherwise.} \end{cases}$
   This is usually written $\mathrm{OR}(x, y) = x \vee y$.

These three functions represent so-called "logical operations." If we associate "False" with 0 and "True" with 1, then $\mathrm{NOT}(x)$ is "True" (or 1) if and only if x is "False" (or 0). Furthermore, $\mathrm{AND}(x, y)$ is "True" precisely when both x and y are "True," and $\mathrm{OR}(x, y)$ is "True" precisely when either x or y is "True" or both are "True."

Example 10.2. The functions of Example 10.1 can be combined to make more complex functions. For example,

$$f(x, y) = \sim(x \wedge y) = \begin{cases} 0 & \text{if } x = y = 1 \\ 1 & \text{otherwise,} \end{cases}$$

and

$$g(x, y, z) = (x \wedge y) \wedge z = \begin{cases} 1 & \text{if } x = y = z = 1 \\ 0 & \text{otherwise.} \end{cases}$$

How can we check that functions like those of Example 10.2 take on the values claimed? Or how can we determine the values of a new function? One foolproof method is to construct a table of all domain values and the resulting function values.

Example 10.3. The values of $f^*(x, y) = (\sim x) \vee (\sim y)$ are listed in Table 1.5.

Table 1.5

x	y	$\sim x$	$\sim y$	$f^*(x, y) = (\sim x) \vee (\sim y)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Notice that the function f^* agrees with the function f of Example 10.2 and so the two functions are equal. In other words, we have shown that

$$\sim(x \wedge y) = (\sim x) \vee (\sim y).$$

Question 10.1. Show that (a) $\sim(x \vee y) = (\sim x) \wedge (\sim y)$ and (b) $(x \wedge y) \wedge z = x \wedge (y \wedge z)$.

The equalities of Example 10.3 and Question 10.1(a) are known as **de Morgan's laws**.

Example 10.4. Another useful function is known as "**exclusive or**," abbreviated XOR. It is a function that is "True" if x or y is "True", but not both. In Boolean

notation

$$\textbf{XOR}:(Z_2)^2 \to Z_2 \text{ is defined by} \qquad \mathrm{XOR}(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases}$$

The function XOR is also written as $\mathrm{XOR}(x,y) = x \oplus y$.

Question 10.2. By checking all domain elements, verify that XOR can also be expressed by

$$\begin{aligned} \mathrm{XOR}(x,y) &= (x \vee y) \wedge \sim(x \wedge y), \text{ or equivalently} \\ &= [x \wedge (\sim y)] \vee [(\sim x) \wedge y]. \end{aligned}$$

Boolean functions model digital networks and electronic circuits well. In these models, the voltage is either high (1) or low (0), and switches are either ON (1) or OFF (0). For example, consider the problem of adding two binary numbers, say $0 + 1$. In a computer this is carried out by a circuit in which high voltage (1) together with low voltage (0) is combined to produce high voltage (1). Such circuit combinations can be imitated by Boolean functions as follows. Define

$$\mathrm{ADD}(x,y) = \begin{cases} 0 & \text{if } x = y = 0 \text{ or } x = y = 1 \\ 1 & \text{otherwise,} \end{cases}$$

and

$$\mathrm{MULT}(x,y) = \begin{cases} 1 & \text{if } x = y = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Notice that MULT is simply the AND function, $x \wedge y$, and ADD is the XOR function, $x \oplus y$. Then for one-digit binary numbers x and y, $\mathrm{MULT}(x,y) = x \cdot y$ and $\mathrm{ADD}(x,y) = x + y$ except that $1 + 1 = 10$ in binary. A "carry" bit is needed to complete the latter addition.

Example 10.5. Here are Boolean functions that produce the sum of 2 two-digit binary numbers. Let x_1, x_0, y_1 and y_0 be in Z_2 so that x_1x_0 and y_1y_0 both represent two-digit binary numbers. Then their sum is $z_2z_1z_0$, where

$$z_0 = \mathrm{ADD}(x_0, y_0) = x_0 \oplus y_0.$$

$$z_1 = \begin{cases} (x_1 \oplus y_1) \oplus 1 & \text{if } x_0 = y_0 = 1 \\ (x_1 \oplus y_1) \oplus 0 & \text{otherwise.} \end{cases}$$

(In other words, the two's digit in $z_2z_1z_0$ is the sum of x_1 and y_1 plus 1 if the 1 is "carried over" from the addition of x_0 and y_0.) There is a carry if and only if $x_0 = y_0 = 1$, and so

$$z_1 = [(x_1 \oplus y_1) \oplus (x_0 \wedge y_0)].$$

Finally, z_2, the four's digit of the sum, is 0 unless there is a carry from the second addition. There will be a carry from the second addition if either $x_1 = y_1 = 1$ or if there is a carry from the first addition and either x_1 or y_1 is 1. In symbols

$$z_2 = [x_1 \wedge y_1] \vee [(x_0 \wedge y_0) \wedge (x_1 \vee y_1)].$$

There are a few more rules of arithmetic concerning Boolean functions that are summarized in the next result. Two other useful rules of arithmetic are de Morgan's laws (see after Question 10.1).

Theorem 10.1. The functions AND and OR satisfy the following properties:

1. $x \wedge y = y \wedge x$	Commutative law
2. $x \vee y = y \vee x$	Commutative law
3. $(x \wedge y) \wedge z = x \wedge (y \wedge z)$	Associative law
4. $(x \vee y) \vee z = x \vee (y \vee z)$	Associative law
5. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	Distributive law
6. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$	Distributive law

Notice that part 3 has been verified in Question 10.1. A consequence of parts 3 and 4 is that we may write $x \wedge y \wedge z$ and $x \vee y \vee z$ (i.e., without parentheses) without ambiguity.

Question 10.3. Verify parts 1 and 4.

Proof of part 5. The straightforward way to check this is to substitute all eight possible choices of 0s and 1s for x, y, and z. For example, $0 \wedge (0 \vee 1) = 0 \wedge 1 = 0 = 0 \vee 0 = (0 \wedge 0) \vee (0 \wedge 1)$.

Here is another type of proof. Think of the $\{\text{True, False}\} \leftrightarrow \{1, 0\}$ correspondence. Then the statement $x \wedge (y \vee z)$ is true if and only if x is true and either y or z is true, that is, if and only if either x and y are true or x and z are true, that is, if and only if $(x \wedge y) \vee (x \wedge z)$ is true. □

Optional Material

You might notice that every Boolean function considered in this section has been expressed in terms of NOT, AND, OR, and/or XOR (and XOR can be expressed

in terms of the first three by Question 10.2). This is the case for all Boolean functions; in addition, the form of the expressions can be specified.

Definition. If $x_1, x_2, \ldots, x_n$ are variables, then for $i = 1, 2, \ldots, n$ both x_i and $\sim x_i$ are called **literals**. An expression of the form $L_1 \vee L_2 \vee \cdots \vee L_j$, where each L_i is a literal for $i = 1, 2, \ldots, j$ is called a **clause**. A function $f\colon (Z_2)^n \to Z_2$ is said to be in **conjunctive normal form** (or CNF) if it is of the form

$$f(x_1, x_2, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_k,$$

where C_i is a clause for $i = 1, 2, \ldots, k$.

Example 10.2 (reexamined.) $f(x, y) = \sim(x \wedge y)$ is not in CNF, but we know that $f(x, y) = f^*(x, y) = (\sim x) \vee (\sim y)$ by Question 10.1, which is in CNF with $f^*(x, y) = C_1$, where $C_1 = L_1 \vee L_2$, $L_1 = \sim x$ and $L_2 = \sim y$. Then $g(x, y) = (x \wedge y) \wedge z$ is in CNF with $C_1 = x$, $C_2 = y$, and $C_3 = z$.

Question 10.4. Which of the functions that determine z_2, z_1, and z_0 in Example 10.5 are in CNF?

Although the function f of Example 10.2 was not in CNF, we found that it was equal to a function, f^*, that was in CNF.

Theorem 10.2. Every Boolean function is equal to a function in CNF.

Proof. Let $f\colon (Z_2)^n \to Z_2$. Imagine writing out the table of all 2^n elements of $(Z_2)^n$ and the corresponding values of f. Suppose that $b = (b_1, b_2, \ldots, b_n)$ in $(Z_2)^n$ is such that $f(b) = 0$. Create a clause C of the form $C = L_1 \vee L_2 \vee \cdots \vee L_n$, where

$$L_i = \begin{cases} x_i & \text{if } b_i = 0 \\ \sim x_i & \text{if } b_i = 1. \end{cases}$$

Notice that when, for $i = 1, 2, \ldots, n$, b_i is substituted for x_i in the clause C, the value is 0. Furthermore, b is the only element of $(Z_2)^n$ which when substituted yields 0; since $C = L_1 \vee L_2 \vee \cdots \vee L_n$, C assumes the value of one when some L_i is one and some L_i is one unless all are zero, precisely the value at b.

Next we form clauses $C_1, C_2, \ldots, C_k$, one for each b in $(Z_2)^n$ for which $f(b) = 0$. Then we let g be defined by

$$g(x_1, x_2, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_k.$$

We claim that $f = g$. Suppose that b in $(Z_2)^n$ is such that $f(b) = 0$. Then there is a clause C_i in g corresponding to b; when b is substituted in C_i, a value of 0 results.

From the formula for g we see that $g(b) = 0$. Suppose that $b = (b_1, b_2, \ldots, b_n)$ is such that $f(b) = 1$. Consider what happens when, for $i = 1, 2, \ldots, n$; b_i is substituted for x_i in any clause C. As we have already seen, a 1 results, since C corresponds to an element $d \neq b$ for which $f(d) = 0$. Since every clause in g equals 1 upon input of b, by definition of g, $g(b) = 1$. □

Example 10.6. We use the technique of the proof of Theorem 10.2 to find a function equal to f, but in CNF, where

$$f(x, y, z) = \sim[(x \wedge y) \vee (x \wedge z)].$$

The elements of $(Z_2)^3$ on which f is 0 are precisely $(1, 1, 0)$, $(1, 0, 1)$ and $(1, 1, 1)$. We form the clauses $C_1 = (\sim x) \vee (\sim y) \vee z$, $C_2 = (\sim x) \vee y \vee (\sim z)$, and $C_3 = (\sim x) \vee (\sim y) \vee (\sim z)$, and let $g = C_1 \wedge C_2 \wedge C_3$.

Two types of Boolean functions merit special names. A Boolean function is called a **tautology** if its range is $\{1\}$; in other words, the function assumes the value "True" upon all input. A Boolean function is called a **contradiction** if its range is $\{0\}$; that is, it is always "False." For example, the function

$$f(x) = x \vee (\sim x)$$

is a tautology whereas

$$g(x) = x \wedge (\sim x)$$

is a contradiction.

Question 10.5. Decide if the following are tautologies, contradictions, or neither.

(a) $(x \vee y) \wedge [(\sim x) \wedge y] \wedge [x \wedge (\sim y)] \wedge [(\sim x) \vee (\sim y)]$.
(b) $(x \wedge y) \vee [(\sim x) \wedge y] \vee [x \wedge (\sim y)] \vee [(\sim x) \wedge (\sim y)]$.
(c) $(x \vee y \vee z) \vee (\sim y)$.

A Boolean function is not a contradiction if upon some input b the value of the function is 1. Then we say that f is **satisfied** by b and f is **satisfiable**. Suppose that we are given a function, expressed as a huge combination of ANDs, ORs, and NOTs, like

$$f(v, w, x, y, z) = ((((\sim x) \wedge y) \vee z) \wedge ((\sim x) \wedge v) \wedge ((\sim y) \vee v) \wedge w).$$

It certainly could be a lot of work to check whether this is a tautology or a contradiction, but suppose that all we want to know is whether there is some value b for which $f(b) = 1$. This is an instance of the following important problem.

The Satisfiability Problem. Given a Boolean function f, is there a b in the domain for which $f(b) = 1$?

How might we proceed to solve the Satisfiability Problem? For a specific function f we could check all domain elements in f. If the domain of f is $(Z_2)^n$, this checking might require all 2^n binary strings before we find, say, that the last string satisfies f or that no string satisfies f. Using the ideas of Chapter 2, we shall make this process precise, as an algorithm, and we shall see that this particular process is a very slow and time-consuming one. But surely there must be some "tricks of the trade," some techniques with which to attack this problem without trying all possibilities. In fact, it is an open research problem whether there is a "fast" and "efficient" process by which to determine whether an arbitrary Boolean function is satisfiable; these terms will be defined precisely in Chapter 2. For the time being we repeat that this simply stated problem is at the heart of some very hard, unresolved research questions that are actively being studied by the research community.

EXERCISES FOR SECTION 10

1. Which of the following tables represent a Boolean function?

Function	
Domain	Value
(0, 0)	1
(0, 1)	1
(1, 0)	1

Function	
Domain	Value
(0, 1)	0
(0, 2)	1
(1, 0)	0
(2, 0)	1

Function	
Domain	Value
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	0

2. Give an example of a Boolean function from $(Z_2)^3$ onto Z_2. How many Boolean functions from $(Z_2)^n$ to Z_2 are onto? How many are one-to-one?
3. Write out a table of values for the following functions. Then identify all pairs of functions that are equal.
   (a) $f_1(x, y) = (\sim x) \vee y$. **(b)** $f_2(x, y) = ((\sim x) \vee y) \wedge x$.
   (c) $f_3(x, y) = y \wedge x \wedge (\sim y)$. **(d)** $f_4(x, y) = \sim(x \vee (\sim x))$.
   (e) $f_5(x, y) = x \wedge (\sim y)$. **(f)** $f_6(x, y) = \sim(x \wedge (\sim y))$.
   (g) $f_7(x, y) = y \vee x$. **(h)** $f_8(x, y) = x \wedge y$.
4. Show that the function $\text{AND}(x, y) = x \wedge y$ is equal to a function using only ORs and NOTs. Show that the function $\text{OR}(x, y) = x \vee y$ is equal to a function using only ANDs and NOTs. Is the function $f(x, y) = \sim(x \wedge y)$ equal to a function that uses only ANDs and ORs, but no NOTs?

5. If x_1x_0 and y_1y_0 are two-digit binary numbers, find the functions that give their product in binary.

$$z_2z_1z_0 = (x_1x_0) \cdot (y_1y_0).$$

6. Verify parts 2 and 6 of Theorem 10.1.

7. For each of the following, find an equal function expressed in CNF.
   (*a*) $f_1(x, y, z) = (\sim x) \vee (y \wedge z)$. (*b*) $f_2(x, y, z) = x \vee ((\sim y) \wedge z)$.
   (*c*) $f_3(x, y, z) = \sim(x \vee y \vee z)$. (*d*) $f_4(x, y, z) = \sim(x \wedge y \wedge z)$.
   (*e*) $f_5(x, y, z) = (x \wedge y) \vee (x \wedge z)$. (*f*) $f_6(x, y, z, w) = (x \wedge y) \vee (z \wedge w)$.

8. A function is said to be in **disjunctive normal form** (or DNF) if it is in the form

$$f(x_1, x_2, \ldots, x_n) = D_1 \vee D_2 \vee \cdots \vee D_k,$$

   where, for $i = 1, 2, \ldots, k$, D_i is of the form $L_1 \wedge L_2 \wedge \cdots \wedge L_j$ with each L_i a literal for $i = 1, 2, \ldots, j$. Find all functions in Exercises 3 and 7 that are in DNF.

9. Find an example of a function that is in neither CNF nor DNF.

10. For each of the following, find an equal function expressed in DNF.
   (*a*) $f_1(x, y, z) = (\sim x) \wedge (y \vee z)$. (*b*) $f_2(x, y, z) = x \wedge [(\sim y) \vee z]$.
   (*c*) $f_3(x, y, z) = \sim(x \wedge y \wedge z)$. (*d*) $f_4(x, y, z) = \sim(x \vee y \vee z)$.
   (*e*) $f_5(x, y, z) = (x \vee y) \wedge (x \vee z)$. (*f*) $f_6(x, y, z, w) = (x \vee y) \wedge (z \vee w)$.

11. Is each of the following satisfiable?
   (*a*) $f(x, y, z) = y \vee \sim[z \vee (\sim x)]$.
   (*b*) $f(x, y, z, w) = [x \vee \sim(y \vee \sim z)] \vee \sim\{x \vee \sim[y \vee \sim(z \wedge w)]\}$.
   (*c*) $f(x, y) = (x \wedge y) \vee ((\sim x) \wedge y)$.
   (*d*) $f(x, y) = x \vee \sim\{x \vee \sim[y \vee (\sim x)]\}$.
   (*e*) $f(x, y) = (x \wedge y) \wedge (x \wedge (\sim y))$.

12. For $n = 2$, 3, and 4 find Boolean functions $f:(Z_2)^n \to Z_2$ that are tautologies.

13. For $n = 2$, 3, and 4 find Boolean functions $f:(Z_2)^n \to Z_2$ that are contradictions.

14. Identify which of the following are tautologies.
   (*a*) $g(x, y) = y \vee \sim\{[y \vee (\sim x)] \wedge (y \vee x)\}$
   (*b*) $g(x, y) = [(\sim x) \vee y] \vee (y \vee x)$.
   (*c*) $g(x, y) = x \vee \sim\{y \vee [(\sim x) \wedge y]\}$.
   (*d*) $g(x, y) = x \vee [\sim(x \vee y)]$.
   (*e*) $g(x, y) = (\sim x) \vee \{y \wedge [(\sim y) \vee (\sim x)]\}$.
   (*f*) $g(x, y) = [x \wedge (\sim y)] \vee [y \vee (\sim x)]$.
   (*g*) $g(x, y, z) = [z \vee (\sim y)] \vee \sim[z \vee \sim(x \wedge y)]$.
   (*h*) $g(x, y, z) = z \vee (\sim x) \vee [(\sim z) \wedge (x \wedge y)]$.

15. A mathematical statement of the form "If A, then B" is said to be true if whenever A is true, then so is B. When A is false, then it doesn't matter whether B is true or false. Thus "If A, then B" is logically equivalent to the statement that either A is false or B is true. We define a new function

$$\textbf{IMPLIES } (x, y):(Z_2)^2 \to Z_2 \text{ defined by IMPLIES } (x, y) = (\sim x) \vee y.$$

Determine the value of $(Z_2)^2$ on which IMPLIES is 1. Interpreting 1 as "True," explain why this is analogous to "If A, then B."

16. A mathematical statement of the form "A **if and only if** B" is true if whenever A is true then so is B and whenever A is false then so is B. Find a Boolean function EQUIVALENT (x, y) that is the appropriate analogue of "A if and only if B."

1:11 A LOOK BACK

We began this chapter with a particular problem (admittedly one of only small import) and introduced some substantial mathematics in order to understand the problem and its solutions. There are a number of ideas that you should be comfortable with before you proceed to the next chapter. Foremost among the important concepts discussed is the notion of algorithm. This course will be oriented toward the solution of problems. Typically, a problem will be to find a mathematical object with a particular property. Often the problem is fairly easy in small instances. However, larger cases may be quite difficult. Such problems frequently have algorithmic solutions and it will be on these that we concentrate. The algorithmic solutions provide us with systematic ways to solve problems, so systematic that we could easily turn the algorithm over to a computer, or more realistically to a computer programmer. Although computers can handle many large numbers quickly, there are limits to computer size and speed. One of the principal themes of the rest of this course will be the analysis of the correctness and efficiency of algorithms and the search for such effective algorithms.

This chapter also contains an introduction to set theory, an important tool because our mathematical objects will usually be described in the language of sets. We need to distinguish between contexts where the order of objects is not important, as with subsets, versus those where order is important, as in Cartesian products. We have also begun to develop counting techniques, the multiplication principle and P.I.E. This material is significant because of its applications to the analysis of algorithms. In particular, these mathematical techniques will be necessary tools in the evaluation of the quality of particular algorithms.

Functions are also important mathematical tools because they describe transformations and relations between sets. Often it will be important to know whether

a particular function is one-to-one or onto or whether it has an inverse. For example, in Chapter 3 we shall study the one-to-one functions called permutations in depth. We have noted that algorithms and functions are similar. They take input and domain elements, respectively, and transform these in a well-defined and unique manner into output and range elements. We shall use algorithms to study functions, and vice versa. In the next chapters two central problems will be to find algorithms that compute functions efficiently and to find and study functions that measure the efficiency of algorithms. The special case of Boolean functions is sufficiently important and applicable to warrant extra study. These functions model well how computers work. They may be thought of as mapping from complicated sets to the values of "True" and "False," and so they also model how logical thinking and proofs work. The Satisfiability Problem is introduced not only because it is an unsolved research problem, but also because it is computationally equivalent to other famous problems, some of which will arise in Chapters 5 and 8.

We have begun to see proofs. Understanding these proofs is crucial to the development of this course. We have left some important facts unproved in this chapter, for example, that the algorithms BtoD, DtoB, and SUBSET work correctly in all cases. One of the main goals of this book is to develop proof techniques and skill in their application to mathematical and algorithmic problems.

The concept of algorithm is pervasive throughout computer science and is increasingly important in abstract mathematics. A beginning computer science student is tempted to attack problems by writing computer code directly, trying out the resulting program on reasonable examples of data and correcting obvious problems as they arise. Instead, especially with complicated problems and programs, it will become essential to plan ahead carefully, to outline the entire attack on the problem. An algorithm is just such a precise outline of the approach to be made on the problem. In mathematical work students are always searching for concrete guidelines for ways to solve specified problems. In the past mathematicians would often prove that a problem could be solved, but then fail to address explicitly how to solve the problem. Now many mathematicians are revising their philosophical ideas about mathematics and are turning to algorithmic approaches to problems. Indeed, some would say that a problem hasn't been solved even in theory unless an algorithmic solution has been found. Thus although the problems that we consider in this book are beginning ones in the study of discrete mathematics and algorithms, the philosophy is central to current approaches and research in these areas.

SUPPLEMENTARY EXERCISES FOR CHAPTER 1

1. Suppose that a computer program contains two Boolean variables, Done and Correct. That is, each of these variables can assume the value either True or False. How many different pairs of values can the two variables (Done, Correct)

assume? Suppose that JEQUALSONE is another Boolean variable. How many different values can the triple (Done, Correct, JEQUALSONE) assume? Suppose that we had n Boolean variables called $X_1, X_2, X_3, \ldots, X_n$. Determine a formula for the number of different values the n-tuple $(X_1, X_2, \ldots, X_n)$ can assume.

2. *Problem.* Given a positive integer n, calculate and print the sum of the integers from 1 to n. Is the following an algorithm that correctly solves this problem?

   Response

   STEP 1. Read in n, a positive integer

   STEP 2. Calculate $n(n + 1)/2$ and print this out

   STEP 3. Stop.

3. Describe an algorithm to change a number written in base four to an equivalent number expressed in decimal.

4. Describe an algorithm to change a number from decimal to base four.

5. Describe an algorithm to convert between binary and base four that does not use decimal representations.

6. A numerical representation system often used in calculators is known as BCD or **binary coded decimal**. In this system each digit in a decimal number is converted to binary and stored in four consecutive bits, lined up in the same order as the original decimal digits. Thus the number 139 is stored as 000100111001. Convert the following numbers into BCD: **(*a*)** 12, **(*b*)** 19, **(*c*)** 25, **(*d*)** 28, and **(*e*)** 77.

7. How many bits does an r-digit decimal number require to be stored in BCD? Given 16 bits with the first bit reserved for sign designation, what is the largest decimal number that can be stored in BCD in the remaining 15 bits?

8. Write an algorithm to convert a decimal integer into BCD, and vice versa.

9. Write an algorithm that inputs $A = \{a_1, a_2, \ldots, a_n\}$ and outputs all elements of the Cartesian product $A \times A$.

10. Suppose that $A = \{a_1, \ldots, a_n\}$ and $T = \{t_1, t_2, \ldots, t_m\}$. How many functions are there with domain A and target T? How many of these functions are one-to-one? (*Hint*: Consider separately the cases when $n < m$, $n = m$, and $n > m$.)

11. How many numbers must you pick from the set $\{1, 2, \ldots, n\}$ so that there must be two with a common divisor greater than one?

12. Explain why a subset of $(n + 1)$ numbers taken from $\{1, 2, \ldots, 2n\}$ must contain two numbers where one divides the other. Is the same true if the subset contains only n numbers?

13. A hiker is lost in the mountains but stumbles into an area where it is known that all inhabitants are either True-tellers or Liars, meaning that an individual either always tells the truth or always lies. She meets a man at a fork in the road and wants to learn the way to the nearest village by asking only one question. Explain why she will not learn the way to the village by asking any of the following questions:

(*a*) Are you a truth-teller?
(*b*) Are you a truth-teller and does the left fork lead to the village?
(*c*) Are you a truth-teller or does the left fork lead to the village?
(*d*) If you are a truth-teller, does the left fork lead to the village?
(*e*) Are you a truth-teller if and only if you are a liar?
(*f*) If the left fork leads to the village, are you a liar?

14. Devise one question with which the hiker, in the predicament of the preceding exercise, can determine the way to the village.

15. Deborah receives $1 million from an anonymous friend. She suspects that either Alice, Bob, or Catherine gave it to her. When she asks each of them, they respond as follows:

(*a*) Alice says, "I didn't do it. Bob is an acquaintance of yours, and Catherine is an especially good friend of yours."
(*b*) Bob says, "I didn't do it. I've never met you before, and I've been out of town for the past month."
(*c*) Catherine says, "I didn't do it. I saw Alice and Bob in the bank on the day you received the check so it must be one of them."

Assuming that the two who didn't give Deborah the money are telling the truth and that the donor is lying, who gave Deborah the money?

16. There is a very simple programming language known as TRIVIAL. In this language only the following six types of instructions are allowed:

(*a*) Input X, a natural number
(*b*) Go to step # —
(*c*) Set $X := X + 1$
(*d*) If $X = 0$, then go to step # —
(*e*) If $X > 0$, then set $X := X - 1$
(*f*) Stop.

Suppose that a program must begin with the instruction Input X and end with the instruction Stop. How many different programs are there, written in the language TRIVIAL if the program is two steps long? Three steps long? 12 steps long? n steps long?

17. Suppose that a TRIVIAL program can begin with one or more statements Input X, where X may be any letter of the alphabet; for example, we may begin with Input A and then follow with Input B. Then we allow statements of the form $A := A + 1$, and if $B > 0$, then set $B := B - 1$, and so on.

provided that A and B are input variables. If a TRIVIAL program begins by reading in two variables A and B, then how many different TRIVIAL programs are there using three steps? Four steps? n steps?

18. Write a program in the language TRIVIAL that upon input of X and Y, positive integers, calculates the sum $X + Y$ and leaves the result stored in X. Note that $X := X + Y$ is not a valid statement in TRIVIAL.

19. Suppose that we allow two output statements: "X is even" and "X is odd." Write a program in TRIVIAL that upon the input of X, a positive integer, determines whether X is even or odd, outputs the correct message, and stops.

20. Open Mathematical Problem: The Busy Beaver N-game. In this version we eliminate the Input X statement and assume instead that every variable begins with the value 0. Then we can do the equivalent of reading in a positive integer i for X by writing i consecutive "$X := X + 1$" statements. The score of a TRIVIAL program is defined to be the sum of the values of all variables when the program stops or 0 if the program never stops. Then the Busy Beaver n-game is the problem of determining a TRIVIAL program with n instructions with the highest possible score among all TRIVIAL programs with n instructions. We call that maximum score $BB(n)$. Thus $BB(1) = 0$, since the only TRIVIAL program with one line is

STEP 1. Stop.

A two-step program could be

STEP 1. Set $X := X + 1$

STEP 2. Stop.

Thus $BB(2) = 1$, since X begins at 0 and is increased only to 1. Explain why $BB(n) \geq (n - 1)$. Then show that $BB(3) = 2$ and $BB(4) = 3$. Find a value of n such that $BB(n) > (n - 1)$; you will need to use at least two variables. Very little is known about the Busy Beaver n-game for even small values of n. It is known that there is no simple function that expresses $BB(n)$ as a function of n. It has been shown that $BB(20) \geq 4^{(4^{(4^4)})}$, but the actual value of $BB(20)$ is unknown.

2

ARITHMETIC

2:1 INTRODUCTION

In Chapters 2 and 4 we take a new look at arithmetic. Some of the problems we shall consider will be more advanced than that which the word arithmetic usually conjures up. However, our principal goal will be algorithmic thinking. We shall be especially concerned with the quality of the methods discussed and the mathematics that is necessary to understand what makes one method better than another. We concentrate on the question of exponentiation, that is, of calculating x^n. Although this problem is more important than the magic trick that formed the theme of the previous chapter, it is the methods rather than the solutions to any particular problem that are worth learning. We introduce the important proof techniques of induction and contradiction to analyze the correctness and efficiency of algorithms.

Here is an appetizer. Suppose that we have two variables, say x and y, that have been assigned the values 5 and 2, respectively. It is a common task to switch the values of the variables, that is, to arrange it so that $x = 2$ and $y = 5$. We want a procedure that will work no matter what the values assigned to x and y are.

Example 1.1. Using the ":=" notation, we can suggest a way to trade the values of the variables assigned to x and y.

STEP 1. $x := y$

STEP 2. $y := x$

STEP 3. Stop.

At first glance this seems reasonable, since step 1 assigns to x whatever is assigned to y and step 2 assigns to y whatever is assigned to x. If we trace what happens to the values assigned to the variables x and y, we get Table 2.1.

Table 2.1

	Value Assigned to x	*Value Assigned to y*
Before step 1	5	2
After step 1	2	2
After step 2	2	2

What happened to the 5? After step 1 it has been forgotten. We wanted to assign the old value of x to the variable y; however, the old value of x has been written over.

Question 1.1. Here is a sequence of instructions that will perform the desired switch of values.

STEP 1. $xold := x$

STEP 2. $yold := y$

STEP 3. $y := xold$

STEP 4. $x := yold$

STEP 5. Stop.

As in the preceding example trace what happens when x is initially assigned the value 5 and y is initially assigned the value 2.

It turns out that only four of the five steps just listed are necessary. Exercise 1 asks you to figure out which step can be safely omitted. Thus there is a four-step algorithm that will switch values between two variables; this algorithm requires the use of a supplementary storage location.

Example 1.2. Here is an algorithm that will switch the values of x and y without the use of an extra storage location. Note that there are more steps and that we are required to do some arithmetic.

STEP 1. $x := x + y$

STEP 2. $y := y - x$

STEP 3. $x := x + y$

STEP 4. $y := -y$

STEP 5. Stop.

As in our previous example we trace what is assigned to each of the variables (Table 2.2).

Table 2.2

	Value of x	*Value of y*
Before step 1.	5	2
After step 1.	7	2
After step 2.	7	−5
After step 3.	2	−5
After step 4.	2	5

We have just seen two algorithmic solutions to the problem of switching the values assigned to two variables. The first requires four steps and a supplementary memory location. The second requires five steps and no extra storage. Furthermore, the steps are arithmetic operations rather than just assignment statements. This is typical. Frequently, a problem will have various algorithmic solutions, and one will be faster while a second might require less space. In general, we shall favor algorithms that have fewer steps over those that require less storage. In Sections 2 and 5 we shall consider exponentiation algorithms that are similarly related.

EXERCISES FOR SECTION 1

1. Which of the following four-step algorithms succeed in trading the values of x and y? For each, trace what happens.

   (a) STEP 1. $yold := y$
   STEP 2. $y := x$
   STEP 3. $x := yold$
   STEP 4. Stop.

   (b) STEP 1. $xold := x$
   STEP 2. $y := xold$
   STEP 3. $x := y$
   STEP 4. Stop.

   (c) STEP 1. $xold := x$
   STEP 2. $yold := y$
   STEP 3. $y := xold$
   STEP 4. Stop.

2. Suppose that $x \geq y \geq 0$. Find an algorithm that interchanges the values of x and y and has the properties that no supplementary storage is required and at no stage is a negative number stored in either location.

3. Find a four-step algorithm that interchanges the values of x and y and does not require a supplementary storage location.

4. Suppose that the three variables x, y, and z are each assigned values. Find an algorithm that will cyclically switch the assigned values so that the old value of x will be assigned to y, the old value of y will be assigned to z, and the old

value of z will be assigned to x. Can this be accomplished with no arithmetic and just one extra storage location?

5. Suppose that you are given n variables labeled $x_1, x_2, \ldots, x_n$ each of which has been assigned a value. Construct an algorithm that will take the value assigned to x_n and store it in x_1, take the old value of x_1 and store it in x_2, take the old value of x_2 and store it in $x_3, \ldots$, and take the old value of x_{n-1} and store it in x_n.
6. Here is an algorithm whose goal is, upon input of numbers x and y, to calculate their sum and store it in x, and calculate their difference and store it in y. Is this algorithm correct?

   STEP 1. Input x and y
   STEP 2. $x := x + y$
   STEP 3. $y := x - y$
   STEP 4. Stop.

   If it is incorrect, rewrite the algorithm correctly.
7. Design a four-step algorithm that upon input of numbers x and y calculates xy and x/y and stores these in the variables x and y, respectively. If possible, use no supplementary storage location.

2:2 EXPONENTIATION, A FIRST LOOK

We begin with the problem of computing the nth power of x, given a real number x and a natural number n. How do computers calculate powers of numbers? The basic arithmetic operations of computers are addition, subtraction, multiplication, and division. Using these operations, we want to develop correct and efficient ways to calculate x^n. There is a straightforward solution to this problem, namely, multiply x by itself $(n - 1)$ times. If n were fixed, then there would be no difficulty in writing down a correct algorithm to solve this problem. For example, if n were always 5 we could calculate

$$\text{answer} := x \cdot x \cdot x \cdot x \cdot x.$$

The situation when n is allowed to vary is slightly more complicated. Now we list an algorithm that finds x^n.

Algorithm EXPONENT

STEP 1. Input x, n {n a natural number}
STEP 2. $i := 0$, ans $:= 1$

STEP 3. While $i < n$ do
Begin
STEP 4. ans := ans $*$ x
STEP 5. $i := i + 1$
End {step 3}
STEP 6. Output ans and stop.

COMMENTS. In Section 1.4 we used "Go to" statements to create loops within algorithms. Here we use the more modern "**While . . . do**" construction, wherein the steps within the loop are executed in order as long as the while condition is satisfied. Note that the loop is indented for ease of reading. When a loop has more than one step, we will signal the beginning of the loop by "Begin" and the end of the loop by "End." The statements within the loop will not be executed if $n = 0$. However, the output of the algorithm will still be correct since $x^0 = 1$ for all x.

This algorithm is sufficiently complicated that you might wonder whether it does what it claims (in all cases).

A Fundamental Problem. How can you be sure that the algorithm listed above (or, more generally, any algorithmic solution to a problem) is correct?

A first response to this problem is to implement the algorithm. If the algorithm is written in a programming language, then the algorithm could be tested by running the program. If, as above, the algorithm is written in a pseudo-code (i.e., in English with a structure similar to a programming language), then the algorithm can be tested by a person tracing the algorithm's commands. In this setting the person acts as a computer. We shall do this for the algorithm listed above, but before we do, it is appropriate to consider what will be learned from such a test.

Presumably, we shall run the algorithm on input for which we already know the answer. If the algorithm's answer is wrong, then we can discard the algorithm. In one sense this is satisfactory, since we know without a doubt that the algorithm is incorrect. If the algorithm's answer agrees with the already known answer, what does that prove? It does show that the algorithm performs correctly on at least one set of input data. If we run the algorithm for a variety of inputs and each run gives a correct answer, then we can increase our confidence in the algorithm. Unless the algorithm only runs on a finite set of inputs, such a strategy cannot demonstrate that the algorithm will always work.

Example 2.1. If $x = 5$ and $n = 3$, then $x^n = 5^3 = 125$. In Table 2.3 we trace EXPONENT with this input.

Table 2.3

Current Step Completed		i	Answer
1		?	?
2		0	1
3		0	1
4		0	5
5		1	5
4		1	25
5		2	25
4		2	125
5		3	125
6	STOP	3	125

Question 2.1. Trace the executive of EXPONENT for $x = 3$ and $n = 4$.

After completing the above question, you probably believe that EXPONENT correctly produces the value of x^n when n is a positive integer. Your belief is based upon the fact that you have witnessed one experiment and performed one additional experiment. This is similar to what you might do in a chemistry class. In the next section we discuss the principle of mathematical induction that will enable us to prove that the algorithm EXPONENT works for all input.

EXERCISES FOR SECTION 2

1. Trace Algorithm EXPONENT for
   (*a*) $x = 17$ and $n = 1$. (*b*) $x = 2$ and $n = 5$.
   (*c*) $x = -2$ and $n = 3$. (*d*) $x = 5$ and $n = 0$.
2. Construct an algorithm that will input a real number x and an integer n, which may be positive, negative, or zero, and output x^n. Trace your algorithm for
   (*a*) $x = 3$ and $n = 3$. (*b*) $x = 3$ and $n = -3$.
   (*c*) $x = 0$ and $n = -5$. (*d*) $x = -1$ and $n = 0$.
3. If Algorithm EXPONENT is run with $x = 5$ and $n = 7$, how many multiplications are performed? If $n = 132$, how many multiplications are done?
4. Suppose that we have a computer that can perform addition but not multiplication. Devise an algorithm that, upon input of a real number x and an integer $n \geq 0$, calculates and outputs the product nx. How many additions does your algorithm use?
5. Construct an algorithm NEWEXP to compute x^n for n, a positive integer. The first step, after x and n are input, should be to compute and store $z = x^2$. At subsequent steps the variable ans (which will contain the answer) is multiplied by z unless that would make ans too large, in which case ans is multiplied by x.

Trace NEWEXP for
(*a*) $x = 3$ and $n = 7$. (*b*) $x = 3$ and $n = 16$.
(*c*) $x = 3$ and $n = 10$.

6. Determine how many multiplications NEWEXP requires to find (*a*) x^6, (*b*) x^{12}, (*c*) x^{13}, and finally (*d*) x^n, where $n = 2t$ for t, any positive integer.
7. Construct an algorithm NEWEREXP whose first two steps, after x and n are input, is to compute and store $z = x^2$ and $w = z^2$. At subsequent steps the variable ans (which will contain the answer) is multiplied by w unless that would make ans too large, in which case ans is multiplied by x until ans is x^n. Trace NEWEREXP for
(*a*) $x = 3$ and $n = 12$. (*b*) $x = 3$ and $n = 16$.
(*c*) $x = 3$ and $n = 19$.
8. Determine how many multiplications NEWEREXP requires to find (*a*) x^{10}, (*b*) x^{16}, (*c*) x^{25}, and finally (*d*) x^n, where $n = 4t$ for t, any positive integer.
9. Write an algorithm that calculates nx (as requested in Exercise 4) using fewer than $n - 1$ additions.

2:3 INDUCTION

Mathematics has distinguished itself from other human endeavors because the truths of mathematics are known with greater certainty than the truths of other subjects. This is because assertions in mathematics must be proved before they are regarded as valid. Although this does not eliminate the possibility of error (mathematicians are, after all, human beings who can and do make mistakes), the necessity of providing proofs greatly diminishes the potential for error.

Example 3.1. The most famous open problem in all of mathematics concerns the positive integers and is very simply stated. It goes as follows: Do there exist positive integers x, y, z, and n with $n > 2$ such that

$$x^n + y^n = z^n?$$

The restriction that $n > 2$ is there because, for instance,

$$\text{if } n = 1, \text{ then } 2^1 + 3^1 = 5^1$$

and

$$\text{if } n = 2, \text{ then } 3^2 + 4^2 = 5^2$$

provide easy affirmative answers to the question. The assertion that for n bigger than 2, there do not exist integers with the required relationship is known as

"**Fermat's Last Theorem**." The name is misleading, since the assertion is still unproved; however, Fermat scribbled in the margin of a book that he had found a proof but had no room to write it down. This note was found after his death! It has been proved that no such integers exist if $n = 3$, that is, there do not exist integers x, y, and z with $x^3 + y^3 = z^3$. Indeed, Fermat's Last Theorem has been proved to be true for all exponents of reasonable size, in fact for $n \leq 125{,}080$. The philosophical question

"Is Fermat's Last Theorem true?"

suggests itself. Certainly, mathematicians would be surprised if it turned out to be false. However, no mathematician would consider the assertion of Fermat's Last Theorem to be a mathematical truth until a valid proof is found, no matter how much numerical evidence is marshaled in support of it.

The complex programs and structures of computer science have certain features analogous to the state of our knowledge concerning Fermat's Last Theorem. It is a common occurrence for programs (or algorithms) to work correctly on some inputs without working correctly on all inputs. Even if a program always has worked correctly, that is no guarantee that it always will work correctly. Because of this, computer scientists frequently require proofs that their assertions and programs are correct. How does one construct a proof that an algorithm (or a program) is correct? The most common technique is mathematical induction.

Principle of Mathematical Induction. Suppose that we wish to prove that a certain assertion or proposition is true. Further suppose that the statement of the proposition explicitly depends on a positive integer, say n. We denote the proposition emphasizing the dependence on the integer n by P_n. Our goal will be to prove that P_n is true for all values of n.

Frequently, it will be easy to prove that the proposition P_n is true for certain (usually small) values of n. The first step in applying an induction proof is to verify the proposition directly for one value of n. This is called checking the **base case**. This one value will usually be $n = 1$ or $n = 0$. The next step in applying an induction proof is analogous to climbing a ladder. We must demonstrate that whenever the proposition P_k is true, then so is the proposition P_{k+1}.

Here is a formal statement of the Principle of Mathematical Induction:

Let P_n be a proposition that depends upon the integer n. Then P_n is true for all positive n provided that

(i) P_1 is true,

and

(ii) if P_k is true, then so is P_{k+1}.

Soon you will see examples of proofs using induction. Before looking at these, let's see why the principle is valid. Note that when we say the principle is valid, we mean that if assertions (i) and (ii) are both verified, then the proposition P_n is proved true for all n.

To begin with, P_1 is true by (i). Since P_1 is true, setting $k = 1$ in (ii) shows that P_2 is true. Then we can repeat (ii) with $k = 2$. Since we've just demonstrated that P_2 is true, we get the result that P_3 is true, and so on. Is P_{17} true? We won't do all the details, but we could work our way up to 17 using (ii) repeatedly, building upon the known results P_j for smaller values of j. In general, we can work our way up to the truth of P_n for any integer n.

The way that we go about establishing an assertion by induction is quite algorithmic. In fact, here are the key steps.

Algorithm INDUCTION

STEP 1. (The base case). Verify that P_1 is true.

STEP 2. (The **inductive hypothesis**). Assume that P_k is true for an arbitrary value of k.

STEP 3. (The **inductive step**). Verify that P_{k+1} is true, using the assumption that P_k is true.

We illustrate proofs by induction with three typical examples.

Example 3.2. The following is an important identity that will reappear several times in this book.

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}. \tag{A}$$

Suppose that we wish to prove this using induction. The statement already has an explicit dependence on the positive integer n: P_n is the statement that equation (A) is true.

First we check the base case when $n = 1$. For $n = 1$ the left-hand side of equation (A) is 1, while the right-hand side equals $1(1 + 1)/2 = 1$. (Although we've done the base case, we'll get a better feel for the problem if we check at least one more case. If the equation were not true, it could be a real time-sink to try to prove it!) For $n = 2$ the left-hand side of (A) equals $1 + 2 = 3$, while the right-hand side equals $2(2 + 1)/2 = 3$. For $n = 3$ we check that the two sides of (A) both equal 6.

Now we come to the inductive hypothesis (step 2). We assume that

$$1 + 2 + \cdots + k = \frac{k(k+1)}{2}. \tag{B}$$

We want to prove that

$$1 + 2 + \cdots + k + (k + 1) = \frac{(k + 1)(k + 2)}{2}. \tag{B'}$$

Note that equation (B') agrees with (A) after substituting $k + 1$ for n. How can we proceed? We want the sum of $k + 1$ integers and what we have to build upon is the sum of k integers. So we use associativity to obtain

$$\begin{aligned}
&(1 + 2 + \cdots + k) + (k + 1) \\
&\quad = \frac{k(k + 1)}{2} + (k + 1) && \text{by inductive hypothesis} \\
&\quad = (k + 1)\left(\frac{k}{2} + 1\right) && \text{by factoring} \\
&\quad = \frac{(k + 1)(k + 2)}{2} && \text{by algebra.}
\end{aligned}$$

□

Question 3.1. Prove by induction that $2 + 4 + \cdots + 2n = n(n + 1)$. (*Hint:* Mimic the preceding example.)

Example 3.3. The following formula gives the sum of a (finite) **geometric series**:

$$1 + x + x^2 + \cdots + x^n = \begin{cases} \dfrac{1 - x^{n+1}}{1 - x} & \text{if } x \neq 1 \\ (n + 1) & \text{if } x = 1. \end{cases}$$

It is a polynomial identity that holds true for every real number x and for every integer $n \geq 0$. This can be proved by induction on n. Instead we prove the slightly more complex formula for the sum of an **alternating geometric series** in the next example. You will be asked to imitate the latter proof in Question 3.3 to verify Example 3.3.

Example 3.4. Here we use induction to verify that for every integer $n > 0$ the following polynomial identity is valid.

$$\begin{aligned}
&1 - x + x^2 - x^3 + \cdots + (-x)^n \\
&\quad = \begin{cases} \dfrac{1 - (-x)^{n+1}}{1 + x} & \text{if } x \neq -1 \\ n + 1 & \text{if } x = -1. \end{cases}
\end{aligned} \tag{C}$$

By a **polynomial identity** we mean an equation that is valid for every substitution of a real number for the variable x.

If $x = -1$, the left-hand side of (C) consists of $n + 1$ terms, each of which is 1. Thus the sum is $n + 1$. If $x \neq -1$, then the identity has an explicit dependence on the positive integer n: P_n is the statement that equation (C) is true for all positive integers n when x is any real number other than -1. First we check the base case when $n = 1$. For $n = 1$, the left-hand side of (C) equals $1 - x$. With $n = 1$, the right-hand side of (C) is

$$\frac{1 - (-x)^2}{1 + x} = \frac{1 - x^2}{1 + x} = 1 - x.$$

Question 3.2. Check that (C) holds for both $n = 2$ and $n = 3$.

Now we come to the inductive hypothesis (step 2). We assume that

$$1 - x + \cdots + (-x)^k = \frac{1 - x^{k+1}}{1 + x}. \tag{D}$$

We want to prove that

$$1 - x + \cdots + (-x)^{k+1} = \frac{1 - (-x)^{k+2}}{1 + x}. \tag{D'}$$

Note that equation (D') agrees with (C) by substituting $k + 1$ for n. We use associativity to obtain

$$\begin{aligned}
&1 - x + \cdots + (-x)^k + (-x)^{k+1} \\
&\quad = \frac{1 - (-x)^{k+1}}{1 + x} + (-x)^{k+1} && \text{by inductive hypothesis} \\
&\quad = \frac{1 - (-x)^{k+1} + (-x)^{k+1} + x(-x)^{k+1}}{1 + x} && \text{by making common denominators} \\
&\quad = \frac{1 + x(-x)^{k+1}}{1 + x} && \text{by algebra} \\
&\quad = \frac{1 - (-x)(-x)^{k+1}}{1 + x} && \text{by algebra} \\
&\quad = \frac{1 - (-x)^{k+2}}{1 + x} && \text{by algebra.}
\end{aligned}$$

□

Question 3.3. Prove by induction on n that the sum of a geometric series is as given in Example 3.3.

Example 3.3 (another look). Here is another way to verify this identity. We have established that the identity of line (C) is valid for all x. Thus we may substitute the expression $-x$ in every instance of the variable x and the identity remains valid. Upon substitution we get

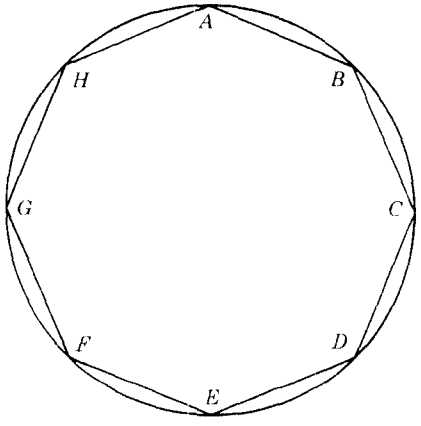
$$1 - (-x) + (-x)^2 - (-x)^3 + \cdots + [-(-x)]^n = \begin{cases} \dfrac{1 - [-(-x)]^{n+1}}{1 + (-x)} & \text{if } -x \neq -1 \\ n + 1 & \text{if } -x = -1 \end{cases}$$

This simplifies to

$$1 + x + x^2 + x^3 + \cdots + x^n = \begin{cases} \dfrac{1 - x^{n+1}}{1 - x} & \text{if } x \neq 1 \\ n + 1 & \text{if } x = 1. \end{cases}$$

□

Example 3.5. Choose $n + 2$ distinct points from the circumference of a circle. If consecutive points along the circle are joined by line segments creating a polygon with $n + 2$ sides, then the sum of the interior angles of the resulting polygon equals $180n$ degrees (see Figure 2.1). Even though the assertion to be proved depends on n in an obvious way, we still need to be careful with the statement P_n. Specifically, we insist that the assertion holds no matter which $n + 2$ points are selected.



$n + 2 = 8$, so $n = 6$.

Figure 2.1 The sum of angles $A + B + \cdots + H = 180 \cdot 6^\circ = 1080^\circ$.

The base case, when $n = 1$, asserts that no matter how three points are selected from the circumference of a circle, then the resulting triangle contains three interior angles that total 180°, a result known from plane geometry.

The inductive hypothesis asserts that no matter how $k + 2$ points are selected from the circumference of a circle, then the resulting polygon contains $k + 2$ interior angles that total $180k$ degrees. Suppose that we are given $k + 3$ [note that $k + 3 = (k + 1) + 2$] points chosen from the circumference of a circle. The inscribed $k + 3$ sided polygon P is shown in Figure 2.2(a). To use the inductive hypothesis, we need to ignore one of the selected points and consider the resulting $k + 2$ sided polygon P' shown in Figure 2.2(b).

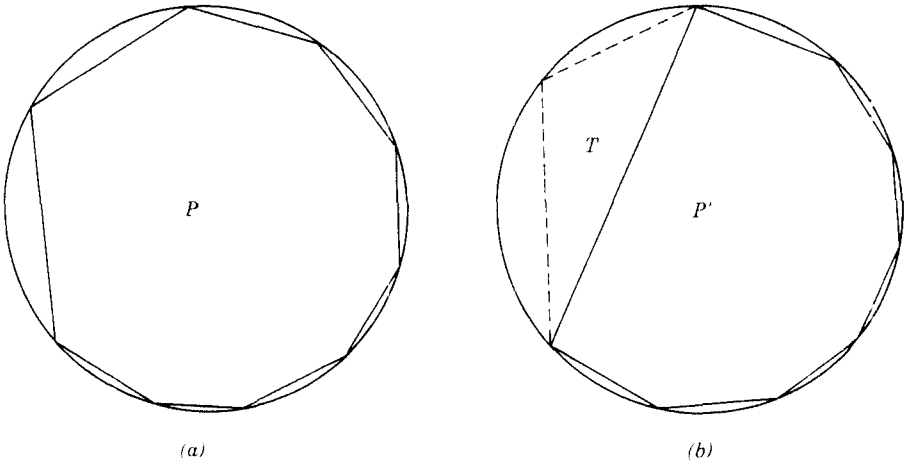


Figure 2.2

P can be obtained from P' by attaching the triangle T as shown. The sum of the interior angles of P equals the sum of the interior angles of P' together with the interior angles of T. Thus

$$\begin{aligned} \text{angles of } P &= \text{angles of } P' + \text{angles of } T \\ &= 180k + 180 \\ &= 180(k + 1). \end{aligned}$$

□

Example 3.2 (revisited). There are numerous proofs of equation (A) that do not require the technique of mathematical induction. We present here two especially beautiful ones for your pleasure.

First a geometric proof: We shall represent the left-hand side of (A) as the area of a plane figure. For example, if $n = 2$ we think of the sum $1 + 2$ as the area of two rows of unit squares one with one square and one with two squares as shown in Figure 2.3.

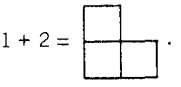


Figure 2.3

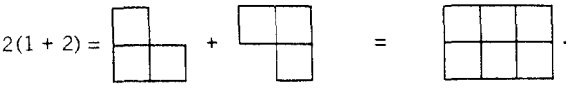


Figure 2.4

If we double the area under consideration, we get the following figure (Figure 2.4). The figure on the right is a 2×3 rectangle whose area is 6. In general, $1 + 2 + \cdots + n$ is represented by n rows of unit squares. If we double this value and piece the two areas together as shown in Figure 2.5, we get an $n \times (n + 1)$ rectangle whose area is $n(n + 1)$. □

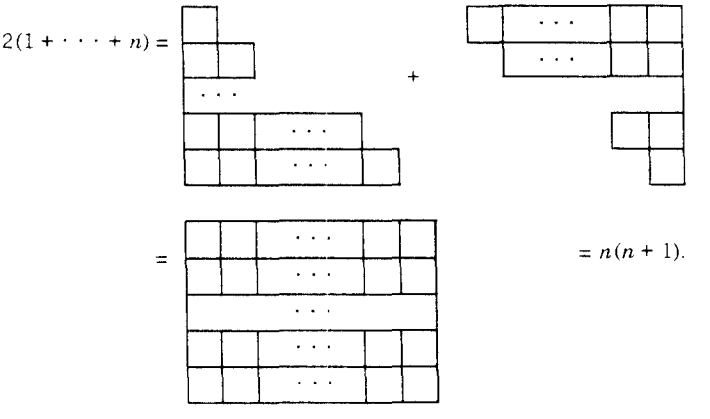


Figure 2.5

The third proof is even simpler although it is divided into two cases. The first case is when n is even, that is, $n = 2r$ for some integer r. We begin with the left-hand side of equation (A), where $2r$ has been substituted for n. Next we add the

largest and smallest terms, then the second largest and second smallest, and so on, as shown in the following equation.

$$
\begin{aligned}
&1 + 2 + 3 + \cdots + (2r - 2) + (2r - 1) + 2r \\
&\quad = (1 + 2r) + (2 + 2r - 1) + (3 + 2r - 2) + \cdots && \text{by regrouping} \\
&\quad = (2r + 1) + (2r + 1) + (2r + 1) + \cdots && \text{by arithmetic} \\
&\quad = r(2r + 1) = \frac{(2r)(2r + 1)}{2} && \text{by algebra} \\
&\quad = \frac{n(n + 1)}{2} && \text{by substitution.} \quad \square
\end{aligned}
$$

Exercise 4 asks you to complete this proof for the case when n is odd.

EXERCISES FOR SECTION 3

1. Give a noninductive proof that $2 + 4 + \cdots + 2r = r(r + 1)$.
2. Show that $1 + 3 + 5 + \cdots + (2r - 1) = r^2$ in two different ways, one of which must be induction.
3. Prove that $1 + 4 + 7 + \cdots + (3n - 2) = (3n^2 - n)/2$.
4. Complete the third proof of equation (A) for the case $n = 2r + 1$. You can do this by temporarily ignoring either the last term or the middle term.
5. Use induction to show that

$$1 + 5 + 9 + \cdots + (4n + 1) = (n + 1)(2n + 1).$$

6. Here is another argument that proves the result in Exercise 5.

$$
\begin{aligned}
1 + 5 + 9 + \cdots + (4n + 1) &= (4 \cdot 0 + 1) + (4 \cdot 1 + 1) + (4 \cdot 2 + 1) + \cdots \\
&\quad + (4 \cdot n + 1) \\
&= (4 \cdot 0 + 4 \cdot 1 + 4 \cdot 2 + \cdots + 4 \cdot n) \\
&\quad + (1 + 1 + \cdots + 1) \\
&= 4 \cdot (0 + 1 + 2 + \cdots + n) + (n + 1) \cdot 1.
\end{aligned}
$$

   Use known results to simplify the right-hand side and to deduce that the sum is the same as is given in Exercise 5.

7. Prove, by any method you like, that
   (a) $2 + 6 + 10 + \cdots + (4n + 2) = 2n^2 + 4n + 2$.
   (b) $2 + 5 + 8 + \cdots + (3r - 1) = (3r^2 + r)/2$.

8. Express the following sums as simply as possible:
   (*a*) $1 + 3 + 9 + \cdots + 3^n$.
   (*b*) $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n}$.
   (*c*) $1 - 4 + 16 - 64 + \cdots + (-1)^n \cdot 4^n$.
9. Find a formula for the following sums:
   (*a*) $1 + x^2 + x^4 + \cdots + x^{2n}$.
   (*b*) $1 + \frac{1}{x} + \frac{1}{x^2} + \cdots + \frac{1}{x^n}$.
10. Use induction to show that $n^2 - n$ is always even.
11. Use induction to show that $n^4 + n^3 - 2n^2$ is always even.
12. Use induction to show that $n^4 + n^3 - 2n^2$ is always divisible by 4.
13. Find a formula for the sum $m + (m + 1) + (m + 2) + \cdots + n$; your answer will be in terms of the variables m and n. Use induction to show that your formula is correct.
14. Suppose that n distinct points are selected from the circumference of a circle. Let C_n denote the maximum number of line segments joining two distinct points that can be drawn so that no two segments intersect. Find a formula for C_n and verify your guess by induction.

2:4 THREE INDUCTIVE PROOFS

It would be difficult to overemphasize the importance of mathematical induction to the mathematician and computer scientist. Believing that the best way to learn is by studying and doing, we offer some more examples and problems.

Example 4.1. Look back at the assertions and algorithms about the number of subsets of a set given in Chapter 1. We claimed that if A is a set with n elements, then there exist 2^n distinct subsets of the set A. We can prove this assertion now by induction.

Precisely, we formulate P_n to be the statement that a set with n elements has exactly 2^n subsets. First we check the base case with $n = 1$, P_1. But we have already checked that P_1, P_2, and P_3 are true in Chapter 1. We saw there that a 1-set has two subsets, a 2-set has four subsets, and a 3-set has eight subsets. Thus we have accomplished step 1.

Now we get to steps 2 and 3, which are, of course, the heart of the matter. We assume the truth of the statement P_k: specifically that any set with k elements has 2^k subsets. Then we move to step 3 and examine P_{k+1}.

Suppose that A is a set with $k + 1$ elements. We want to show that there exist exactly 2^{k+1} subsets of A. What we are allowed to use is the assumption that P_k is true. The trick here is to overlook one of the elements of the set A in order to obtain a set with k elements. Let x be an element of A and define B to be $A - \{x\}$. We illustrate in Table 2.4 where, with $A = \{x, y, z, w\}$, we list the subsets of B and the subsets of A that contain x.

Table 2.4

$A = \{x, y, z, w\}$ Subsets of A That Contain x	$B = A - \{x\} = \{y, z, w\}$ Subsets of B
$\{x\}$	$\varnothing$
$\{x, y\}$	$\{y\}$
$\{x, z\}$	$\{z\}$
$\{x, w\}$	$\{w\}$
$\{x, y, z\}$	$\{y, z\}$
$\{x, y, w\}$	$\{y, w\}$
$\{x, z, w\}$	$\{z, w\}$
$\{x, y, z, w\}$	$\{y, z, w\}$

In the specific example above $k = 3$. B has 2^3 subsets, each of which is a subset of A. A has 2^3 subsets containing the specific element x. Each of these subsets could be obtained from a subset of B by adding the element x. Each of the 2^4 subsets of A can be constructed in this manner.

The general argument is the same. If A has $k + 1$ elements, B has k elements. Thus we know by step 2, the inductive hypothesis, that there exist 2^k subsets of B. Each of these subsets of B is also a subset of A. Thus we know that there exist 2^k subsets of A, none of which contains the element x. From each of these subsets that don't contain x we can create a new subset of A that does contain x. Specifically, if $S \subseteq B$, define S' to be $S \cup \{x\}$. Now S' is a subset of A. Furthermore, since S' contains x, it is not a subset of B.

Finally, every subset of A either contains x or it doesn't. If a subset, say T, of A does not contain x, then T is a subset of B. If, on the other hand, T does contain x, then $T - \{x\}$ is a subset of B. Either way T has been accounted for. Then we see that

$$
\begin{aligned}
\#(\text{subsets of } A) &= \#(\text{subsets containing } x) + \#(\text{subsets not containing } x) \\
&= 2^k + 2^k && \text{by inductive hypothesis} \\
&= (1 + 1) \cdot 2^k && \text{by factoring} \\
&= 2 \cdot 2k && \text{by algebra} \\
&= 2^{k+1} && \text{by laws of exponents.}
\end{aligned}
$$

□

We have proved the following theorem using the Principle of Induction.

Theorem 4.1. If A is a set containing n objects, then A has 2^n subsets.

The proof of Theorem 4.1 suggests an algorithm for listing all subsets of a given set. Suppose that we want to list all subsets of the set $A = \{a_1, a_2, \ldots, a_{10}\}$. Then the idea is, picking $x = a_{10}$, to list all subsets of $B = \{a_1, a_2, \ldots, a_9\}$ and then to repeat each subset with element a_{10} added in. How do we get all subsets of B? We could list all of the subsets of $\{a_1, \ldots, a_8\}$ *and* . . . . Wait a minute. Let's go forward rather than backward. We know that the set $\{a_1\}$ has two subsets, the empty subset and $\{a_1\}$. From these subsets we can get all subsets of $\{a_1, a_2\}$ by repeating those just listed and adding a_2 to get the additional sets $\{a_2\}$ and $\{a_1, a_2\}$. This procedure should sound familiar. Reread algorithm SUBSET in Chapter 1 and see that the algorithm uses exactly this idea of adding in elements a_j to previously formed subsets.

QUESTION 4.1. A set is said to be **even** if it has an even number of elements. Note that the empty set has zero elements and is thus an even set. If A is a set with n elements, guess a formula for the number of even subsets of A. Prove your formula by induction. (*Hint:* How many odd subsets does A have'?)

Example 4.2. We now return to the algorithm presented at the beginning of Section 2 and use induction to prove that the algorithm does compute x". Here is the algorithm listed again with a comment between step 5 and step 6. (To avoid two different uses of the integer n the algorithm now calculates x'.)

Algorithm EXPONENT

STEP 1. Input x, r [r a natural number]
STEP 2. $i := 0$, ans $:= 1$
STEP 3. While $i < r$ do
Begin
STEP 4. ans : = ans * x
STEP 5. $i := i + 1$
{Comment: Right now ans has the value x'.]
End { step 3}
STEP 6. Output ans and stop.

This example is more complicated than previous ones because the proposition we need to verify, the P_n, is not explicitly presented. What we will do is use induction to show that the comment inserted between step 5 and step 6 is true.

Before we do this, note that if the comment is always true, then it will be true the last time it is encountered. The variable i is assigned the value O at step 2, and

this value is incremented by 1 each time through the loop. This continues until $i = r$, when the algorithm, upon returning to step 3, discovers that the condition "While $i < r$" is no longer true. Hence the algorithm proceeds to step 6 in which it outputs the value of ans. If the comment is true at $i = r$, the value of ans equals x^r, which is what the algorithm was supposed to produce.

The P_n then is the statement that the nth time the comment is encountered it is true. First we verify the base case. If $n = 1$, then we have just finished step 5 for the first time. At step 2 the value of 0 was assigned to the variable i. This remains unchanged until step 5 when the value assigned to i was increased by 1. Thus the first time the comment is encountered the value of 1 is assigned to the variable i. Similarly, at step 2 the value of 1 was assigned to the variable ans. This remains unchanged until step 4 when the value of ans is multiplied by x. Thus the first time the comment is encountered the value of $x = x^1$ is assigned to the variable ans and thus P_1 is true.

Now for the inductive hypothesis. We assume that the comment is true the kth time it is encountered. To accomplish the inductive step, we must use this assumption and show that the $(k + 1)$st time the comment is encountered it is still true. The value assigned to i at the $(k + 1)$st encounter with the comment is the value assigned at the kth encounter plus one. This value is $k + 1$. The value assigned to ans at the $(k + 1)$st encounter equals the value assigned at the kth encounter times x. The value assigned to ans at the kth encounter equals x^k by the inductive hypothesis. Thus

$$\begin{aligned} \text{ans } \{\text{after } k+1 \text{ encounters}\} &= x \cdot \text{ans } \{\text{after } k \text{ encounters}\} \\ &= x \cdot x^k \\ &= x^{k+1}. \end{aligned}$$

□

Question 4.2. Consider the following algorithm:

Algorithm SUM

STEP 1. Input r, set ans $:= 0$
STEP 2. For $j = 1$ to r do
STEP 3. Set ans $:=$ ans $+ j$ {*Comment:* Right now ans has the value $j(j + 1)/2$.}
STEP 4. Output ans and stop.

COMMENT. Step 2 "**For** $j = 1$ to r **do** Step 3" is similar to the "While $\cdots$ do" loop of Example 4.2, only more compactly written. It means that first we set $j = 1$ and execute step 3; then we set $j = 2$ and carry out step 3, . . . ; until finally we set $j = r$ and execute step 3.

Use induction to show that this algorithm outputs $r(r + 1)/2$.

Example 4.3. We now verify that the algorithm DtoB, which finds the binary representation of a number, is correct. For convenience we list the algorithm once again.

Algorithm DtoB

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

The proof will be by induction. Thus we need a statement P_n with which to work. Let n denote the number of bits in the binary representation of the integer m. The statement P_n will be that the algorithm correctly finds the binary representation of all integers whose representation has exactly n bits.

First we check the base case P_1. There is just one positive integer whose binary representation has exactly one bit, namely $m = 1$. In this case at step 2, q will equal 0, r will equal 1, and the base case holds. Back in Chapter 1 you undoubtedly checked many other cases, so the result seems reasonable.

Now for the inductive hypothesis. We assume that whenever m is an integer whose binary representation has k bits, then DtoB correctly finds these k bits. The inductive step says that we must prove the same for an integer with $(k + 1)$ bits: Suppose that m is such an integer. Take m and step through the algorithm until we get to step 6 for the first time. If m is even, r will be 0 and q will be $m/2$. If m is odd, r will be 1 and q will be $(m - 1)/2$. Consider the number q. We assert that it has one fewer binary digits than m.

Question 4.3. How many bits are there in the binary representation of (a) 14, (b) 7, (c) 13, and (d) 6? How can you get the binary representation of 14 from that of 7? How can you get the binary representation of 13 from that of 6?

We may assume (by the inductive hypothesis) that the algorithm will correctly produce the binary representation for q, since it has k bits. If we begin working the algorithm on q, we shall get exactly the same sequence of remainders beginning with j equal to 0 that we would have obtained from m beginning at the second encounter of step 2 with $j = 1$.

If m is even, $m = 2q$. When a number in binary is multiplied by 2, its digits are just shifted one space to the left and a zero, the first remainder, is attached at the end. Consequently, the algorithm will produce the correct binary representation for m. If m is odd, then $m = 2q + 1$. Here the binary representation for m can be

obtained from the binary representation for q by shifting each digit one space to the left and atttaching a 1 as the last digit. If m is odd, the first remainder is one, so the algorithm will once again produce the correct binary representation. □

EXERCISES FOR SECTION 4

1. Compare the idea for an algorithm contained in Example 4.1 with that of Algorithm SUBSET in Section 1.7. In what ways do they agree and in what ways do they differ?
2. Prove by induction that the number of 2-subsets of an n-set A equals $n(n-1)/2$. [*Hint:* Let x be any object of A and $B = A - \{x\}$. Then a 2-subset of A is either a 2-subset of B or a 1-subset of B. Count the number of subsets in each case.]
3. Prove by induction that the number of 3-subsets of an n-set equals $n(n-1)(n-2)/6$. (*Hint:* Do Exercise 2 first.)
4. Consider the following algorithm:

   Algorithm ODDSUM

   STEP 1. Input n, set ans $:= 0$
   STEP 2. For $j = 1$ to n do
   Begin
   STEP 3. Set $t := 2 * j - 1$
   STEP 4. Set ans $:=$ ans $+ t$
   {*Comment:* Right now ans has the value j^2.}
   End
   STEP 5. Output ans and stop.

   Use induction to show that this algorithm outputs n^2.
5. Consider the following algorithm:

   Algorithm FOURSUM

   STEP 1. Input n, set ans $:= 0$
   STEP 2. For $j = 1$ to n do
   Begin
   STEP 3. Set $t := 4 * j - 3$
   STEP 4. Set ans $:=$ ans $+ t$
   End
   STEP 5. Output ans and stop.

   Use induction to show that this algorithm outputs $n(2n-1)$.

6. How many binary digits do each of the following pairs of numbers have?
(i) 6 and 3. **(ii)** 10 and 5.
(iii) 12 and 6. **(iv)** 5 and 2.
(v) 7 and 3. **(vi)** 11 and 5.

7. What is the relationship between the pairs of decimal numbers whose binary representations follow?
(i) 10 and 100. **(ii)** 11 and 110.
(iii) 101 and 1010. **(iv)** 10 and 101.
(v) 11 and 111. **(vi)** 101 and 1011.

8. Suppose that m is a decimal number with binary representation s. Describe the binary representation of the numbers $n = 4m$, $p = 4m + 1$, $q = 4m + 2$, and $r = 4m + 3$ in terms of s.

9. Reread Algorithm SUBSET in Section 1.7. Let P_k be the statement that the kth time the comment after step 3 is encountered, it is correct. Prove that P_k is true for all positive integers k.

10. Design an algorithm to list all even subsets of an n-set and prove by induction that the algorithm is correct.

11. Consider the following algorithm.

Algorithm SQUARESUM

STEP 1. Input n, set ans $:= 0$
STEP 2. For $j = 1$ to n do
Begin
STEP 3. Set $k := j * j$
STEP 4. Set ans $:=$ ans $+ k$
{*Comment:* Right now ans has the value $j(j + 1)(2j + 1)/6$.}
End {step 2}
STEP 5. Output ans and stop.

Use induction to show that this algorithm outputs $n(n + 1)(2n + 1)/6$.

12. Consider the following algorithm.

Algorithm MAX

STEP 1. Input n, a positive integer, and $x_1, \ldots, x_j, \ldots, x_n$, real numbers.
STEP 2. Set max $:= x_1$
STEP 3. For $j = 2$ to n do
STEP 4. If $x_j >$ max, then max $:= x_j$
STEP 5. Output max and stop.

Use induction to show that this algorithm outputs the maximum of the numbers $x_1, x_2, \ldots, x_n$.

13. Consider the following algorithm.

Algorithm BUBBLES

STEP 1. Input n, a positive integer, and $x_1, \ldots, x_j, \ldots, x_n$, real numbers
STEP 2. For $j = 1$ to $n - 1$ do
  STEP 3. If $x_j > x_{j+1}$, then do
    Begin {Trade the values of x_j and x_{j+1}.}
    STEP 4. Set temp $:= x_j$
    STEP 5. Set $x_j := x_{j+1}$
    STEP 6. Set x_{j+1} := temp
    End {step 3}
STEP 7. Output x_n and stop.

Figure out what this algorithm does. Prove your guess by induction.

14. Find a formula for the sum $1 - 2 + 3 - 4 + \cdots + (-1)^n n$. Then prove that your formula is correct.

15. Here is a general statement of the Multiplication Principle: Suppose a counting procedure can be divided into n independent and successive stages. If there are c_1 outcomes for the first stage, and for each of these there are c_2 outcomes for the second stage, and for each of these initial outcomes there are c_3 outcomes for the third stage, and . . . , and finally for each of these there are c_n outcomes for the final stage, then the total number of possible outcomes equals $c_1 \cdot c_2 \cdot \cdots \cdot c_n$. Prove this principle by induction on n.

16. Here is a short algorithm.

STEP 1. Input n (n a positive integer)
STEP 2. Set $i := n$, set ans $:= 0$
STEP 3. While $i > 0$ do
  Begin
  STEP 4. ans $:=$ ans $+ i$
  STEP 5. $i := i - 1$
  End
STEP 6. Output ans and stop.

(*a*) If the input is 4, what answer does this algorithm produce?
(*b*) Explain why this algorithm will always stop regardless of what positive integer n is used as the input.
(*c*) In general, what is the answer (in terms of n) produced by this algorithm? Express this answer as simply as possible.

17. Use induction to prove that 3 divides $n^3 + 3n^2 + 2n$ for every nonnegative integer n.