4

NUMBER THEORY

4:1 GREATEST COMMON DIVISORS

In this chapter seemingly elementary questions from integer arithmetic lead to surprising and elegant ‘mathematics. We shall look at divisibility properties of integers, the greatest common divisor of two integers, and the Fibonacci numbers. These topics have aesthetic appeal and are applicable, as ‘we shall see, in cryptography.

Here are two problems on which we spent many (dull?) hours in elementary school. Recall that a fraction a/b is **simplified** (or reduced) if a and b have no common factor greater than 1.

Problem 1. *Is* the fraction a/b simplified? If not, simplify it.

Problem 2. Compute $a/b + c/d$ and leave the answer simplified.

Question 1.1. Simplify, if possible, the following: $\frac{3}{12}$, $\frac{13}{121}$, $\frac{65}{130}$, $\frac{34{,}567}{891{,}011}$. Add and simplify the following $\frac{1}{3}+\frac{1}{2}$, $\frac{1}{4}+\frac{1}{3}$, $\frac{1}{15}+\frac{1}{65}$.

You might wonder why we did these exercises in elementary school as well as how we did them. Probably being dutiful and bright students, we just did them. But why bother? Certainly, calculators remove the need to simplify fractions.

Try an experiment. Add $\frac{1}{3}$ to itself three times on a calculator. You might get 1 or you might get ,99999999 (depending on your calculator). In either case subtract 1 from your total. Surprisingly enough you won’t get zero (unless your calculator is fancy or broken). There are instances (you will see one in Section 7) when we know quantities to be integers and want to retain the accuracy and precision of integer arithmetic. Most computer languages give us the option of exact arithmetic with integers, provided that the integers are not too large.

How did we do Problems 1 and 2? To find the sum of two fractions, most of us would compute

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

and then simplify this fraction. Both problems require the ability to simplify fractions. As a practical technique, most people would simplify the fraction a/b by searching for integers that are divisors of both a and b. When such an integer, say c, is found, they cancel c from both the numerator and the denominator to obtain the smaller problem of reducing $(a/c)/(b/c)$. This is fine if the numbers a and b are small or have common divisors that are easy to find, for instance, if both a and b are even or both end in O or 5.

A slightly more sophisticated approach is to look for common divisors among the primes, for if two numbers have a common divisor, then they have a common prime divisor. An even better description of how to proceed is *to* find the greatest common divisor of a and b and then cancel that number. Although this is better as a description, if the numbers a and b are at ail large, we might be at a loss in finding the greatest common divisor or, for that matter, any common divisor.

Question 1.2. Find the greatest common divisor of the pairs (a) (65, 130), (b) (48, 88), and (c) (34567, 89101 1).

In this section we work out a straightforward, although slow, procedure for finding the greatest common divisor of two integers. A more efficient algorithm will be presented in a later section.

We begin with some precise definitions pertaining to integer arithmetic. If b and c are integers, we say that b **divides** c (b is a **divisor** of c, and c is a **multiple** of b) if c/b *is* an integer. Then as the name implies, the **greatest common divisor** of two positive integers b and c is the largest integer that is a divisor of both b and c. We denote the greatest common divisor of b and c by **gcd (b, c).**

Does every pair have a greatest common divisor? Any pair of positive integers has 1 as a common divisor, and the largest number that could possibly be a common divisor of b and c is the minimum of b and c. Thus the greatest common divisor always exists and lies somewhere between 1 and the minimum of b and c.

Question 1.3. Find b and c (with $b \leq c$) such that (i) gcd $(b, c) = 1$, (ii) $1 <$ gcd $(b, c) < b$, and (iii) gcd $(b, c) = b$. Why is it impossible for gcd (b, c) to be larger than the minimum of b and c?

Our first gcd algorithm, a brute force search, looks for gcd (b, c) starting with the largest possibility, the minimum of b and c, and then checks each smaller integer in turn until a common divisor is found. The first common divisor found will be the greatest. The algorithm must stop, since 1 is a common divisor.

Algorithm GCD1

```
STEP 1. Input $b, c$; set $g :=$ minimum of $b$ and $c$
STEP 2. While $g > 1$ do
            Begin
            STEP 3. If $b/g$ and $c/g$ are both integers, then output $g$ and stop.
            STEP 4. Set $g := g - 1$
            End
STEP 5. Output gcd $= 1$ and stop.
```

Question 1.4. Carry out GCD1 on the pairs (3,4), (3, 12), and (6, 20).

We judge the efficiency of this algorithm by the number of divisions (which occur only in step 3). The exact number will depend upon b and c, and so we carry out a worst-case analysis to obtain an upper bound. Our input to GCD 1 is two integers b and c; suppose that $b \le c$. We measure the size of the input by c and let the complexity function $f(c)$ count the maximum number of divisions carried out for any pair of numbers $b \le c$. Two divisions are performed every time step 3 is encountered. Step 3 will be executed with $g = b$, then $g = (b - 1)$, then $g = (b - 2)$, and so on, until g has decreased down to the real gtd. Thus step 3 will happen most often when the gcd is 1. In this event we would encounter step 3 a total of $b - 1$ times, performing $2(b - 1)$ divisions. Then

$$f(C) \le 2(b - 1) < 2(c - 1) < 2c \qquad \text{so } f(c) = o(c).$$

We see that the number of divisions in GCD1 is linear in the size of the input, and thus it seems to be an efficient algorithm.

Question 1.5. Find two positive integers b and c such that when GCD1 is applied to them we find the following.
(a) The number of divisions is exactly $2(b - 1)$.
(b) The number of divisions is less than $2(b - 1)$.
(c) The number of divisions is as small as possible.

With GCD1 we can respond precisely to Problems 1 and 2. With a more efficient gcd algorithm, we could upgrade our responses by replacing GCD 1. Here is a solution to Problem 1.

Algorithm SIMPLIFY

```
STEP 1. Input $a$ and $b$ {The fraction $a/b$ is to be simplified.}
STEP 2. Use GCD1 and set g:= gcd (a, $b$)
STEP 3. Set $a' := a/g$ and $b' := b/g$
STEP 4. Output the fraction $a'/b'$ and stop.
```

Question 1.6. Write an algorithm ADDFRACT1 that solves Problem 2. Upon the input of fractions a/b and c/d, it should calculate their sum and output that sum as a simplified fraction. You may use the algorithm SIMPLIFY within ADDFRACT1.

Question 1.7. Count the number of multiplications and divisions performed by SIMPLIFY and by ADDFRACT1, including those in GCD1.

Previously, we have called linear algorithms fast and claimed that they were more efficient than, say, quadratic algorithms. Although GCD 1 performs at most O(c) divisions, it seems slow and inefficient on hand calculations. In fact, it is not the approach that many humans would take to find the gcd of two integers, and it doesn't use any properties of integers that might speed up the process. In the next sections we shall reexamine the complexit y of GCD 1 and the way we perform complexity analyses. We shall find that GCD 1 is not an efficient algorithm, but we shall develop a good gcd algorithm, one that performs $O(\log(c))$ divisions in the worst case upon input of integers b and c with $b \leq c$.

EXERCISES FOR SECTION 1

L Simplify the following fractions: *(a)* $\frac{138}{240}$, *(b)* $\frac{75}{615}$, *(c)* $\frac{357}{189}$, and (d) $\frac{164}{644}$.

2. Combine the following into one simplified fraction: (a) $\frac{138}{15} - \frac{138}{16}$ and *(b)* $\frac{1}{15}$ +&.

3. If both a/b and c/d are simplified, is $(ad + bc)/(bd)$ simplified?

4. If a/b is simplified, is a^2/b^2 simplified?

5. If a^2/b^2 is simplified, is a/b simplified?

6. Suppose that we find the lowest common denominator of $a/b + c/d$ to be e, and with this denominator we get $a/b + c/d = f/e$ for some integer f. Is f/e always a simplified fraction?

7. Trace GCD1 on the following pairs: (a) *(4, 7)*, **(b)** *(4, 6)*, *(c) (8, 10)*, **(d)** *(8, 12)*, *(e) (15,35)*, and *(f)* (18,42).

8. Algorithm GCD 1 begins with g equal to the minimum of b and c and then decreases g, searching for a common divisor of b and c. Design an algorithm that instead begins with g = 1 and then increases g until the gcd is found. How does the efficiency of this algorithm compare with that of GCD1?

9. Suppose that $a, b,$ and c are three positive integers with $a \leq b \leq c$. We define **gcd (a, *b*, c)** to be the largest integer that divides all three numbers, $a, b,$ and c. Explain why gcd $(a, b, c) \leq a$. Design an algorithm that upon the input of $a, b,$ and c finds gcd (a, b, c). Find gcd (24, 68, 128), gcd (28,70, 98), and gcd (1 12,148, 192).

10. Find pairs (b, c) such that when GCD1 is applied, the number of divisions is exactly (a) 12, **(*b*)** 16, and (c) $b/2$.

11. Given two integers b and c, the **least common multiple** of b and c, denoted by **lcm (*b*, c), is** the smallest integer that is a multiple of both b and c. Find a pair of integers b and c with $b \leq c$ such that **(i)** lcm $(b, c) = bc$ and **(ii)** lcm $(b, c) = c$. Then explain why in all cases $c \leq$ lcm $(b, c) \leq bc$.

12. Find the following lcm (2, 3), lcm (3,4), and lcm (6, 8). Then add and simplify the fractions: $\frac{1}{2} + \frac{1}{3}, \frac{1}{3} + \frac{1}{4}$, and $\frac{1}{6} + \frac{3}{8}$.

13. Calculate the following
(u) gcd (5, 7) and lcm (5, 7).
(*b*) gcd (4,9) and lcm (4, 9).
(c) gcd (6, 10) and lcm (6, 10).
(*d*) gcd (6,9) and lcm (6, 9).
(e) gcd (8, 12) and lcm (8, 12).
(*f*) gcd (5, 10) and lcm (5, 10).

14. Here is a proof that lcm (b, c). gcd $(b, c) = bc$. *Give* reasons for each step. {Let g = gcd (b, c), b' = b/g, $c' = c/g$, and m = lcm (b, c).}
1. be/g is a multiple of b and a multiple of c
2. lcm $(b, c) \leq be/g$
3. gcd $(b, c) \cdot$ lcm $(b, c) \leq bc$
4. be/m divides both b and c
5. gcd $(b, c) \geq be/m$
6. gcd (b, c)" lcm $(b, c) \geq bc$
7. gcd $(b, c) \cdot$ lcm (b, c) = bc.

15. Given the following pairs of integers b and c, find g = gcd (b, c), $b' = b/g$, $c' = c/g$, and lcm (b, c). Then check that lcm $(b, c) = b'c'g$. ***(a)*** 3 and 4, **(b)** 6 and 8, (c) 4 and 6, **(*d*)** 3 and 9, and (e) 8 and 20.

16. Prove that lcm $(b, c) = b'c'g$, where b', c', and g are as defined in Exercise 15.

17. Find pairs (b, c) such that gcd (b, c) equals **(a)** *3,* **(b)** *8, (c) b/2,* **(d)** *b/3,* and (e) $\sqrt{b}$. Find pairs (b, c) such that lcm (b, c) equals ***(a)*** *14,* **(b)** *29, (c) 2b,* **(*d*)** *3b,* and **(*e*)** b^2.

18. What can be said about the relation between gcd (a, b) and gcd (at, bt) where t is any positive integer?

19. Prove that if a and b are positive integers and x and y are nonzero integers such that $ax + by = 1$, then

$$\gcd(a, b) = \gcd(a, y) = \gcd(x, b) = \gcd(x, y) = 1.$$

Show that exactly one of the numbers x and y must be negative. [We can define gcd (c, d), where one or both of c and d are negative with exactly the same definition as for positive integers.]

20. If $a, b, x,$ and y are nonzero integers such that $ax + by = 2$, is it true that gcd $(a, b) = 2$?

21. Prove that if gcd $(a, b) = 1$ and if c divides b, then gcd $(a, c) = 1$.

22. Suppose that $a = qb + r$, where $a, b, q,$ and r are integers. Is it true that gcd $(a, b) =$ gcd (a, r)? Is gcd $(a, r) =$ gcd (b, r)? Explain your answers.

23. Here is the idea for another algorithm to add the fractions a/b and c/d. Set $g :=$ gcd (b, d), $b' := b/g$, $d' := d/g$, and $m :=$ lcm (b, d). First calculate m by $m = bd/g$. Then $a/b = ad'/m$ and $c/d = cb'/m$ (Why?) and $a/b + c/d = (ad' + cb')/m$. Finally, simplify this last fraction. Implement these ideas as an algorithm ADDFRACT2. How many variables does ADDFRACT2 use? Count the number of multiplications and divisions performed, including those of GCD 1.

24. Compare the algorithms ADDFRACT1 and ADDFRACT2 with respect to number of variables used and number of multiplications and divisions performed. Which uses less space and which is quicker?

4:2 ANOTHER LOOK AT COMPLEXITIES

We want to reexamine the complexity of algorithms, especially those from number theory. In a formal analysis of an algorithm the size of the input should be measured by the number of bits (zeros and ones) needed to represent the input. For number theory algorithms whose input is typically one or more positive integers, the size of the input should be the total number of zeros and ones needed to represent the input integers in binary notation. As before, we count the number of time-consuming operations performed in the worst case of the algorithm (usually multiplications and divisions for number theory algorithms) and express the resulting upper bound as a function of the number of input bits. In this section we discuss the effects of this change of perspective on complexity analysis.

Why the change? There is a certain (bureaucratic-style) inefficiency built into our previous approach to the analysis of algorithms. We measured how efficient an algorithm was by estimating the number of steps it required as a function of the input size. The problem with this is that if we are careless about measuring the size of the input, that is, if we let it be artificially large, then the algorithm might appear to take a correspondingly small number of steps. This is just what happened in our study of GCD1 and the exponentiation algorithms of Chapter 2. Measuring input size in terms of bits leads to complexities that reflect actual running times.

Changing the input measure, to bit size, is not hard. Suppose that an integer n is the input to an algorithm. As we saw in Section 2.6 the number of bits needed to represent n is precisely

$$B = \lfloor \log(n) \rfloor + 1.$$

This formula gives the translation from n to B, and it implies the following useful relationships.

$$\begin{aligned} \log(n) < B &\le \log(n) + 1 \\ &\le 2\log(n) \qquad \text{for } n \ge 2. \end{aligned} \tag{1}$$

Example 2.1. Suppose that algorithm A performs at most Clog(n) time-consuming operations upon input of an integer n for some constant C. Then what can be said about the complexity function as a function of B, the number of bits needed to represent n? By (1)

$$C\log(n) \le CB = O(B).$$

Thus in terms of the variable B, the number of time-consuming operations is a linear function.

Look back in Section 2.6 at the complexity analysis of FASTEXP. There we found that no more than 3 log(n) + 3 multiplications and divisions are needed to compute x^n. Using (1), we see that

$$3\log(n) + 3 \le 3B + 3 = O(B).$$

In terms of input bits FASTEXP is a linear algorithm and so deserving of its name.

Question 2.1. Suppose that algorithms R, S, and T each have an integer n as input, and their complexity functions are, respectively, $(\log(n))^2$, $\log(n^2)$, and $\log(\log(n))$. Find an upper bound on their complexity functions in terms of B, the number of bits needed to represent n.

Example 2.2. Suppose that algorithm A performs at most C n time-consuming operations upon input of an integer n for some constant C. Then what can be said about the complexity function as a function of B, the number of bits needed to represent n?

$$\begin{aligned} Cn &= C\,2^{\log(n)} && \text{by properties of log} \\ &\le C\,2^{B} && \text{using (1)} \\ &= O(2^{B}). \end{aligned}$$

Thus in terms of the variable B, the number of time-consuming operations is big oh of an exponential function. Furthermore, if there are instances when .4' uses all C n operations, then

$$\begin{aligned} Cn &= C\,2^{\log(n)} && \text{by properties of log} \\ &\ge c\,2^{(B/2)} && \text{using (1)} \\ &= C(\sqrt{2})^{B} \\ &\ge C(1.414)^{B}. \end{aligned}$$

Thus A' is an exponential algorithm.

The analysis in Example 2.2 shows why both the algorithms GCD 1 and EXPONENT of Chapter 2 are bad algorithms. Since GCD1 has integers b and c input, the number of bits needed to express b and c in binary is given by

$$\begin{aligned} \log(c) \le B &= \lfloor \log(b) \rfloor + 1 + \lfloor \log(c) \rfloor + 1 \\ &\le 2\log(b) + 2\log(C) \qquad \text{for } b \ge 2 \\ &\le 4\log(c). \end{aligned} \tag{2}$$

We know that GCD1 performs at most 2C divisions. From Example 2.2 we know that $2C \le 2(2^{B})$, giving an exponential upper bound. In addition, when $b = c - 1$, gcd $(b, c) = 1$ (see Exercise 2). In that case GCD1 performs exactly $2(b - 1) = 2C - 4$ divisions.

$$\begin{aligned} 2C - 4 &= 2(2^{\log(c)}) - 4 && \text{by properties of log} \\ &\ge 2(2^{(B/4)} - 2) && \text{from (2)} \\ &\ge 2(2^{(B/4-1)}) && \text{when } B \ge 8 \\ &= 2^{(B/4)} \\ &= (2^{(1/4)})^{B} \\ &> (1.189)^{B}. \end{aligned}$$

Thus in the worst case GCD1 performs an exponential number of divisions in terms of the input bit size.

Question 2.2. In Section 2.5 it was observed that EXPONENT always performs n multiplications. If B is the number of bits needed to represent n in binary, explain why EXPONENT is an exponential algorithm.

Since GCD 1 is now recognized to be bad, it is clear why we continue to search for a faster algorithm. From now on we shall measure the input size by the number of bits needed. This approach is standard in the study of algorithms using Turing Machines.

EXERCISES FOR SECTION 2

1. Comment on the following statement: "Most of the time $\lfloor \log(n) \rfloor = \lceil \log(n) \rceil - 1$."
2. Explain why gcd $(c - 1, c) = 1$ for all integers $c > 1$.
3. Let $B = \lfloor \log(n) \rfloor + 1$. For each function f listed in the table find the smallest function g such that $f(n) \le g(n)$.

$f(n)$	$g(B)$
$2\log(n) - n$	$\sqrt{B}$
$\sqrt{\log(n)}$	B
$(\log(n))^2 + 2\log(n) + 1$	$2B$
$2^{\log(n)}$	B^2
$3^{\log(n)}$	$10B^2$
$\sqrt{n}$	$\sqrt{2}^B$
$3n + 3$	2^B
$n\log(n)$	$2^{(B+3)}$
n^2	$B(2^B)$
$n^3 - n$	2^{2B}
2^n	6^B
	8^B
	$2^{(B^2)}$
	$2^{(2^B)}$

4. Let the input to algorithm A be an integer n. Thus the number of bits needed is $B = \lfloor \log(n) \rfloor + 1$. Suppose that the complexity function for algorithm A is $a(n) = g(B)$.

(a) Show that if $g(B) = O(p(B))$, then $a(n) = O(p(n))$.
(b) Show that if $a(n)$ # $O(p(n))$ for any polynomial p, then $g(B)$ # $O(q(B))$ for any polynomial q.
(c) If $a(n) = O(p(n))$ for some polynomial p, is it true that g(n) = $O(q(B))$ for some polynomial q?

5. In the algorithms SUBSET, JSET, and PERM we measured the input by the integer variable n. If we translate now to the number of bits input, $B = \lfloor \log(n) \rfloor + 1$, do these algorithms remain exponential in the variable **B** using the worst-case analysis? (See Exercise 4.)
6. Suppose that the input to an algorithm A is an integer n and suppose the size of the input is measured by the number of decimal digits needed to express n. Would this change of measure of input size change whether or not A is a good algorithm?

4:3 THE EUCLIDEAN ALGORITHM

We have developed the simplistic (but bad) algorithm GCD1 to determine gcd (b, c). Fortunately, there is a much more efficient algorithm that appeared in 300 B.C. in Euclid's *Elements.* This Euclidean algorithm is probably the oldest algorithm still in use today. The Babylonians wrote down some precise arithmetic procedures about 1500 years before Euclid, but these have all been replaced by more efficient methods. The amazing fact about the Euclidean algorithm is that, except for minor variations, it is the best (most efficient) algorithm for calculating the greatest common divisor of two integers. In this section we'll learn the algorithm and in subsequent sections the mathematics needed to determine its complexity.

Here is the idea behind the algorithm. Suppose that we are given positive integers $b \leq c$ and want to calculate gcd (b, c). If d divides both b and c [i.e., d is a candidate for gcd (b, c)], then d divides $c - b$. Indeed if d divides $c - b$ and b, then it divides c also. What is the advantage of working with b and $c - b$ instead of b and c? Very simply, $c - b$ *is* smaller than c.

Question 3.1. Find gcd (18, 30), gcd (18, 48), and gcd (18, 66).

If $c - b$ is better than c, then $c - 2b$ should be better still. While we're at it, there is $c - 3b$, $c - 4b$, and so on, to consider. Indeed why not subtract off as many bs as possible subject to the condition that the remaining value is not negative?

Question 3.2. For each pair (b, c), find the maximum integer q such that $c - qb \geq 0$. (a) (24, 36), (b) (36, 120), and (c) (34, 170).

This question illustrates the general rule that the right number of bs to subtract from c is the floor function of the quotient c/b. Thus we divide c by b to obtain an integer quotient q_1 and a remainder r_1, where

$$q_1 = \left\lfloor \frac{c}{b} \right\rfloor \qquad \text{and} \qquad \frac{c}{b} = q_1 + \frac{r_1}{b}.$$

We rewrite the previous equation in the form

$$c = q_1 b + r_1, \tag{A}$$

and note that the remainder r_1 must satisfy $0 \le r_1 < b$. We call q_1 the **quotient** and r_1 the **remainder** of the division c/b.

Here is an important fact about the numbers in (A).

Lemma 3.1. If b, c, q, and r are integers such that $c = qb + r$, then $\gcd(b, c) = \gcd(b, r)$.

Proof. Since an integer that divides b and c also divides b and r, $\gcd(b, c)$ divides both b and r and so is at most $\gcd(b, r)$. Thus

$$\gcd(b, c) \le \gcd(b, r).$$

An integer that divides b and r also divides c. Thus

$$\gcd(b, r) \le \gcd(b, c),$$

and the lemma follows. $\square$

Applying the lemma to line (A) gives $\gcd(b, c) = \gcd(r_1, b)$.

Question 3.3. For each of the following pairs of numbers, determine q_1 and r_1. Check that (A) holds and that $0 \le r_1 < b$. Finally, compute $\gcd(b, c)$ and $\gcd(r_1, b)$. (a) (3, 12), (b) (13, 121), (c) (233, 377), and (d) (34567, 891011).

Notice that if in (A) $r_1 = 0$, then $c = q_1 b$ and $\gcd(b, c) = b$. But if $r_1 > 0$, then we don't have the gcd at our fingertips and consequently must do more work. The problem is simpler now because we have smaller numbers. This technique of replacing c by a smaller number, the remainder, worked once. Let's do it again. Thus we divide b by r_1, a number smaller than b, to obtain a new integer quotient q_2 and a new remainder r_2:

$$b = q_2 r_1 + r_2 \qquad \text{with } 0 \le r_2 < r_1.$$

If $r_2 = 0$, then r_1 divides b and so $r_1 = \gcd(r_1, b) = \gcd(b, c)$ by Lemma 3.1. More generally (even when $r_2 \neq 0$), we have by Lemma 3.1 that

$$\gcd(b, c) = \gcd(r_1, b) = \gcd(r_2, r_1).$$

Next we divide r_1 by r_2, then r_2 by r_3, and keep dividing each remainder by the next until we reach a remainder of zero. Here is the sequence of divisions spelled out precisely; for future reference we call these the **Euclidean equations.** Note that every variable assumes only integer values.

The Euclidean Equations

$$\begin{array}{rll}
c = q_1 b + r_1 & & \text{with } 0 \le r_1 < b \\
b = q_2 r_1 + r_2 & & \text{with } 0 \le r_2 < r_1 \\
r_1 = q_3 r_2 + r_3 & & \text{with } 0 \le r_3 < r_2 \\
\cdots & & \\
r_{i-2} = q_i r_{i-1} + r_i & & \text{with } 0 \le r_i < r_{i-1} \\
\cdots & & \\
r_{k-3} = q_{k-1} r_{k-2} + r_{k-1} & & \text{with } 0 \le r_{k-1} < r_{k-2} \\
r_{k-2} = q_k r_{k-1} + 0 & & \text{with } r_k = 0.
\end{array}$$

The claim made by Euclid is that r_{k-1}, the last nonzero remainder, equals $\gcd(b, c)$. Before we verify this, how do we know that this algorithm stops? That is, how do we know that eventually we shall find a remainder of zero? Notice that the remainders satisfy

$$b > r_1 > r_2 > r_3 > \cdots > r_{i-1} > r_i > \cdots > r_{k-1},$$

and all the remainders are nonnegative integers. Eventually, a remainder must equal zero, certainly after no more than b remainders.

Example 3.1. Let's carry out the Euclidean algorithm on the numbers 26 and 32:

$$\begin{aligned}
32 &= 1 \cdot 26 + 6 \\
26 &= 4 \cdot 6 + 2 \\
6 &= 3 \cdot 2 + 0.
\end{aligned}$$

We know that $\gcd(26, 32) = 2$, the last nonzero remainder.

Next we try 233 and 377:

$$\begin{aligned}
377 &= 1 \cdot 233 + 144\\
233 &= 1 \cdot 144 + 89\\
144 &= 1 \cdot 89 + 55\\
89 &= 1 \cdot 55 + 34\\
55 &= 1 \cdot 34 + 21\\
34 &= 1 \cdot 21 + 13\\
21 &= 1 \cdot 13 + 8\\
13 &= 1 \cdot 8 + 5\\
8 &= 1 \cdot 5 + 3\\
5 &= 1 \cdot 3 + 2\\
3 &= 1 \cdot 2 + 1\\
2 &= 2 \cdot 1 + 0
\end{aligned}$$

(That took a while!) This calculation implies that gcd (233, 377)= 1. To check this, note that 233 is a prime while 377 = 13.29.

Question 3.4. Use the Euclidean algorithm *to* calculate the following (a) gcd (12, 20), (b) gcd (5, 15), (c) gcd (377,610), and (d) gcd (34567,89101 1). Check that the gcd divides each remainder in the Euclidean equations. In each instance count the number of divisions needed to find the gtd.

In our development of the Euclidean algorithm the concurrent explanation can readily be turned into a proof that the algorithm is correct. We now give such a proof.

Theorem 3.2. Given positive integers *b* and c, the last nonzero remainder produced by the Euclidean algorithm equals gcd *(b, c).*

Proof. Suppose that *b* and c produce the Euclidean equations as listed above. We must prove that $r_{k-1} = \gcd(b, c)$. The last equation tells us that r_{k-1} is a divisor of r_{k-2}, since r_{k-2}/r_{k-1} *is* the integer q_k. Thus

$$r_{k-1} = \gcd(r_{k-2}, r_{k-1}). \tag{B}$$

Applying Lemma 3.1 to the next to last equation, we get

$$\gcd(r_{k-3}, r_{k-2}) = \gcd(r_{k-2}, r_{k-1}) = r_{k-1} \qquad \text{by (B).}$$

Continuing and repeatedly applying Lemma 3.1, we get

$$\begin{aligned} \gcd(b, c) = \gcd(b, r_1) &\\ &= \gcd(r_1, r_2) \\ &= \gcd(r_2, r_3) \\ &\ \ \cdots \\ &= \gcd(r_{k-2}, r_{k-1}) \\ &= r_{k-1} \qquad \text{by (B).} \end{aligned}$$ ❑

Corollary 3.3. If g = gcd *(b, c)*, then there are integers x and y such that g = *xb* + *yc.*

Proof. Look at the Euclidean equations. Notice that r_1 can be expressed as $r_1 = c - q_1 b$. If g = r_1, then we have demonstrated this result. If not, we can use the second Euclidean equation to express

$$\begin{aligned} r_2 &= b - q_2 r_1 \\ &= b - q_2(c - q_1 b) && \text{by substitution} \\ &= (1 + q_1 q_2)b - q_2 c && \text{simplifying and factoring.} \end{aligned}$$

We continue this process until we reach

$$g = r_{k-1} = r_{k-3} - q_{k-1} r_{k-2}$$

and can substitute in expressions for r_{k-3} and r_{k-2} found earlier, to express g = r_{k-1} in the form *xb* + *yc.* ❑

We say that the resulting equation expresses the gcd as a **linear combination** of *b* and c. This result will be useful in Section 7; other applications are explored in the exercises.

Example 3.1 (continued). We found that gcd (26, 32) = 2. Now we use the Euclidean equations to express 2 as a linear combination of 26 and 32. From the first Euclidean equation we have

$$6 = 1 \cdot 32 - 1 \cdot 26.$$

From the second equation

$$2 = 1 \cdot 26 - 4 \cdot 6.$$

We substitute the first equation into the second to get

$$2 = 1 \cdot 26 - 4(1 \cdot 32 - 1 \cdot 26) = 5 \cdot 26 - 4 \cdot 32.$$

The same procedure applied to 233 and 377 yields

$$1 = (-144) \cdot 233 + 89.377,$$

but we spare you the 11 equations needed to derive this. Notice that once derived, it is easy to check that the values of x and y work.

Now we write the Euclidean algorithm in pseudocode. Note that the Euclidean equations all are in the same form.

Algorithm EUCLID

```
STEP 1. Input band $c$ $\{0 < b \leq c\}$; set $r := b$
STEP 2. While $r > 0$ do
        Begin
        STEP 3. Set $q := \lfloor c/b \rfloor$
        STEP 4. Set $r := c - q * b$
        STEP 5. If $r = O$, then output gcd = $b$
                    else
                set $c := b$ and $b := r$
        End {step 2}
STEP 6. Stop.
```

Question 3.5. Run EUCLID on the following pairs of integers and express the gcd as a linear combination of the pair of numbers. (a) (6,20), (b) (3, 4), and (C) **(55, 89).**

What can we say about the complexity of EUCLID? We begin as we did with GCD1. Let e(c) count the maximum number of divisions and multiplications performed in the algorithm upon input of numbers $b \leq c$. Not surprisingly, there are lots of these operations. One division occurs in step 3 and one multiplication in step 4. Every time we execute step 3 we immediately execute step 4. Thus e(c) = *2m,* where *m is* the number of times that step 3 is executed. Another way to count this is to notice that e(c) equals twice the number of Euclidean equations needed to calculate gcd (b, c). This is so, since we do one division *to* get the quotient and one multiplication to get the remainder in each new equation.

Thus e(c) = *2k,* where *k is* the number of Euclidean equations used upon the pair b and c. We search for an upper bound on k that will give us an upper bound on e(c). Since the remainders in the equations decrease, we know that in the worst case we can have no more than b equations. For this to occur, the remainders must be precisely $(b - 1)$, @ – 2),. . . . 1, and O. Then

$$e(c) \leq 2b \leq 2c = O(c),$$

a complexity result no better than that of GCD 1.

We shall see in the next sections that the remainders cannot behave in such a perverse manner and that EUCLID is considerably more efficient than GCD 1. In fact, we shall see that as a function of the size of the bit input, EUCLID is a linear algorithm.

EXERCISES FOR SECTION 3

1. Use EUCLID to find the gcd of the following pairs: (a) (10, 14), (b) (14, 35), (c) (24, 42), (d) (128, 232), (e) (98, 210).
2. For each of the pairs in Exercise 1, express the greatest common divisor as a linear combination of the given numbers.
3. Suppose that you EUCLID the pair (b, c) and then the pair (tb, tc) for some integer constant t. What is the relationship between the two sets of Euclidean equations? What is the relationship between the pairs of integers x and y that express b and c and tb and tc as linear combinations of their gcd's?
4. Suppose that $a = bc + d$. Which of the following are true and which false? Explain.
   (i) If e divides a and b, then e divides d.
   (ii) If e divides a and c, then e divides d.
   (iii) If e divides a and d, then e divides b.
   (iv) If e divides c and d, then e divides a.
   (v) If e divides b and d, then e divides a.
   (vi) If e divides b and c, then e divides a.
   (vii) gcd (a, c) = gcd (c, d).
   (viii) gcd (a, c) = gcd (b, d).
   (ix) gcd (a, b) = gcd (b, d).
   (x) gcd (a, c) = gcd (b, c).
5. Find a number c such that with $b = 3 <$ c, the remainders in the Euclidean equations are precisely the numbers 2, 1, and O. Is there a number c such that with $b = 4 <$ c the remainders are (all) the numbers 3, 2, 1, and O? Can you find a pair of numbers b and c with $4 < b \leq c$ such that the remainders in the Euclidean algorithm are all the numbers $(b - 1)$, $(b - 2)$,. .. ,1, and O?
6. Suppose that d divides b and c – sb, wheres is an integer such that c – $sb < 0$. Is it still true that d divides c?
7. What is the maximum number of Euclidean equations you can have if ***(a)*** b = 4, ***(b)*** b = 5, and (c) b = **6?**

8. What is the maximum number of Euclidean equations you can have if (a) $c = 7$, (b) $c = 9$, and (c) $c = 10$?
9. Rewrite the Euclidean algorithm so that all qs and rs are stored in arrays as they are calculated. Then extend this algorithm so that it also calculates x and y such that $g = xb + yc$.
10. Construct a modified Euclidean algorithm incorporating the following idea. Given the Euclidean equation $c = qb + r$, if $r < b/2$, set $c := b - r$ and $b := r$. Otherwise, set $c := r$ and $b := b - r$. Show that the gcd of the new b and c is equal to the gcd of the old b and c. Call the resulting algorithm MODEUCLID.
11. Use MODEUCLID to find the gcd of the following pairs: (a) (42, 136), (b) (18, 324), (c) (148,268), (d) (233, 377), and (e) (324,432).
12. Discuss the efficiency of MODEUCLID.
13. For each pair (b, c) below characterize $IC(b, c)$, the set of integer combinations of b and c, defined by

$$IC(b, c) = \{mb + nc: m, n \text{ are integers}\}.$$

In each case determine the smallest positive integer in $IC(b, c)$. [Note that in the definition of $IC(b, c)$ m and n do not have to be positive integers.]

(a) (2, 4)	(b) (6, 8)	(c) (6, 9)	(d) (12, 15)
(e) (9, 14)	(f) (5, 7)	(g) (13, 18)	(h) (21, 54).

14. What is the relationship between the Euclidean equations with input (b, c) and those with input $(c - b, c)$?
15. Prove that given integers b and c, there are integers x and y such that $1 = xb + yc$ if and only if $\gcd(b, c) = 1$.
16. Find integers a and b such that $\gcd(a, b) = 3$. Then explain why for these values of a and b there are no integers x and y such that $2 = ax + by$. Comment on the following statement: "If $h \neq \gcd(a, b)$, then there are no integers x and y such that $h = ax + by$."
17. Is the following true or false? Given integers b and c, there are integers x and y such that $d = xb + yc$ if and only if $\gcd(b, c) = d$. Explain your answer.
18. Write a formal induction proof of Corollary 3.3.

4:4 FIBONACCI NUMBERS

We digress to a seemingly unrelated topic, the Fibonacci numbers, because the mathematics associated with them is interesting and because (surprisingly) they are intimately related with the complexity analysis of the Euclidean algorithm.

Here are the first 16 **Fibonacci numbers:**

0 1 1 2 3 5 8 1 3 2 1 3 4 5 5   8 9 1 4 4 2 3 3 3 7 7 6 1 0 .

The convention is to start numbering at zero, so that we have listed the Oth, the 1st, ..., and the 15th Fibonacci number. We denote the nth Fibonacci number by F_n for each nonnegative integer n.

Question 4.1. Compute $F_{n-1} + F_{n-2}$ for n = 2, 3,4, 5, 6, 11, and 13.

Your answer to the previous question should suggest that there is an easy method for obtaining the Fibonacci numbers. First, $F_0 = O$, $F_1 = 1$, and then for all $n \geq 2$,

$$F_n = F_{n-1} + F_{n-2}.$$

In fact, this is an inductive sort of a definition. Once you know the two base cases, F_0 and F_1, then you can find all the others, one at a time, by adding successive values.

Question 4.2. Calculate F_I~,FI~,F18, F_{19}, and F_{20}. Then compare F_n with 2“. Which seems to be (or is) larger? '

Since the nth Fibonacci number is defined in terms of smaller Fibonacci numbers, it is natural to try to build proofs about these numbers using induction. However, the nth Fibonacci number is not defined solely in terms of its immediate predecessor, but rather in terms of two predecessors. Consequently, we need a strengthening of our induction machine.

Mathematical Induction Revisited. First we repeat the form of induction that we have used so far.

Algorithm INDUCTION

STEP 1. Verify the base case.

STEP 2. Assume that P_k *is* true for an arbitrary value of k.

STEP **3.** Verify that P_{k+1} *is* true, using the assumption that P_k *is* true.

Sometimes the truth of P_{k+1} depends on the truth of more than one of the preceding P_j's or depends on the truth of P_j, where $j < k$. There is still hope for the method of induction if we use the following principle, which is known as Complete Induction.

The Principle of Complete Induction. Suppose that P_n *is* a proposition that depends upon the positive integer *n*. Then P_n is true for all $n \geq N$, (where N is some fixed integer) provided that

(i) P_N is true,

and

(ii) if $P_N, P_{N+1}, \ldots,$ and P_k are all true, then so is P_{k+1}.

There are two changes here. First we've introduced an unspecified constant N. In the original version of induction we always mentioned the base case P_1 although we admitted that the base case might start off at P_0. Lemma 7.1 from Chapter 2 was only true for $n > 5$; we used this to show that $\sqrt{n}$ is bigger than log *(n)* if *n is* bigger than 64. For situations like this we would like to have the flexibility to begin proofs by induction at different starting points. The variable N allows us this flexibility and tells us what the starting point for the proposition P_n should be. It also tells us the first value of *n* for which we should check the base case, namely $n = N$.

The second difference between induction and complete induction is in the inductive hypothesis. In this second version we assume that $P_N, P_{N+1}, \ldots,$ and P_k are all true. Since we assume more, it should be easier to use this form.

Question 4.3. Look back at the informal explanation of the Principle of Induction in Section 2.3. Write out a similar argument to explain why the Principle of Complete Induction is valid.

Here is the algorithmic version of complete induction.

Algorithm INDUCTION

STEP 1. (The base cases). Verify that $P_N, P_{N+1}, \ldots . P_{N+j}$ *are* valid for some constants *N* and *j* (depending upon the problem).

STEP 2. (The inductive hypothesis). Assume that $P_N, P_{N+1}, \ldots .$ and P_k are all true for an arbitrary value of *k*.

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

There is one more change, introduced in this algorithm, namely the constant *j* in Step 1. At times we shall need to check more than one base case, depending on the proof we construct, to show that P_{k+1} is true. The value of *j* depends upon the number of $P_N, P_{N+1}, \ldots . P_k$ that we refer back to in our verification of P_{k+1}.

We shall point out explicitly the values of j in each case, but as a rule of thumb you should get in the habit of checking at least two base cases.

Here is an initial example of the use of complete induction.

Example 4.1. Theorem. Every integer $n > 1$ has a prime divisor.

Proof. The statement gives the starting point of the proposition N = 2. This statement is true for $n = 2$, since 2 is a prime number and divides itself. Similarly, the statement is valid for $n = 3$, since 3 is a prime. We check that it is also true fern $= 4 = 2 \cdot 2$.

The inductive hypothesis tells us to assume the truth of the statement for $n = 2, 3, \ldots, k$, for some arbitrary value of k. To accomplish the inductive step, we must prove the result that the integer $(k + 1)$ has a prime divisor. Now either $(k + 1)$ is a prime or it isn't. If $(k + 1)$ is a prime, then it has itself as a prime divisor. If $(k + 1)$ is not a prime, then $k + 1 = bc$, where b and c satisfy $1 < b < (k + 1)$ and $1 < c < (k + 1)$. Consider b. By the inductive hypothesis, since $1 < b < (k + 1)$, we may assume that b has a prime divisor, say p. Then

$$\frac{k+1}{p} = \frac{bc}{p} = \frac{b}{p} c.$$

We see that $(k + 1)/p$ *is* an integer, since it is expressed as the product of two integers. This means that p is a divisor of $(k + 1)$ and so $(k + 1)$ has a prime divisor, namely p. ❑

Note that we could not have used the standard form of induction in this problem because the truth of the assertion P_{k+1} depends not on the truth of P_k, but on the truth of P_b, where *b is less* than k. Since our proof depends upon only one earlier case, P_b with $1 < b < (k + 1)$, our base case needed only one value, namely N = 2 and $j = 0$, although to get a feel for the problem we checked three base cases.

We now use complete induction to establish some facts about the Fibonacci numbers.

Example 4.2. From the examples calculated in Question 4.2, the following was observed. **Theorem.** $F_n < 2^n$.

We prove this for all nonnegative integers n using complete induction. For the base cases we notice that $F_0 = 0 < 1 = 2^0$, and that $F_1 = 1 < 2 = 2^1$. We require base cases with two consecutive integers because in our proof we use the fact that F_{k+1} can be written in terms of its two immediate predecessors. (Thus j = 1 in this example). We shall use complete induction and so assume that $F_i < 2^i$ for all $0 \leq i \leq k$. Then we must prove that $F_{k+1} < 2^{k+1}$. But we know exactly what F_{k+1}

equals:

$$\begin{aligned}
F_{k+1} &= F_k + F_{k-1} && \text{by definition} \\
&< 2^k + 2^{k-1} && \text{by the inductive hypothesis} \\
&\cdot\ 2^{k-1} \cdot 2 + 2^{k-1} && \text{by algebra} \\
&= 2^{k-1} \cdot (2 + 1) && \text{by factoring} \\
&< 2^{k-1} \cdot 4 && \text{since } (2 + 1) < 4 \\
&= 2^{k-1} \cdot 2^2 && \\
&= 2^{k+1} && \text{by laws of exponents.}
\end{aligned}$$

❑

Example 4.2 might cause us to ask whether $F_n = O(p)$ for some polynomial p; however, this is not the case. The result of the next question shows that the Fibonacci numbers grow exponentially.

Question 4.4. Find an integer N such that $F_N > (\frac{3}{2})^N$. Prove by induction that $F_n > (\frac{3}{2})^n$ for all $n \geq N$.

Before we do more magic, rabbit-out-of-the-hat tricks with the Fibonacci numbers, let's learn where they come from and why. The Fibonacci numbers first appeared in the book *Liber Abaci* published in 1202 by Leonardo of Piss (also known as Leonardo Fibonacci, since he was the son of Bonacci). Although Leonardo was mainly interested in their mathematical properties, he also noted the following application.

Example 4.3. A pair of rabbits requires one month to mature to the age when it can reproduce. Suppose that the rabbits then mate and produce another pair in every subsequent month, and that the pair of offspring is always conveniently one male and one female, who then form a new breeding pair. If in the first month we have one pair of rabbits, how many pairs do we have at the beginning of the nth month? For simplicity, we assume no death or loss of fertility. We call the resulting number R_n.

At the beginning of the first month we have one pair, so $R_1 = 1$. At the beginning of the second month we still have one pair, but during the second month they produce a pair of bunnies. Thus $R_2 = 1$ and $R_3 = 2$. During the third month the original pair produces another pair of bunnies, but the new pair of bunnies doesn't reproduce yet. So $R_4 = 3$. Then $R_5 = 5$.

We might as well argue the general case That is, let's determine R_n in terms of previous values of R. At the beginning of the nth month we have all the rabbit pairs that we had at the beginning of the $(n-1)$st month, R.-~, plus some new bunny pairs. The number of new bunny pairs is the number of rabbit pairs that are at least one month old. The rabbit pairs that are this old are precisely those that were around in the $(n-2)$nd month, R_{n-2}. In symbols then

$$R_n = R_{n-1} + R_{n-2}.$$

Now we see that the R_n are exactly the same as the Fibonacci numbers and that $R_n = F_n$ for all positive n.

Fibonacci numbers arise in other natural settings. For example, the spacing of leaves on some plants and some arrangements of flower petals and seeds are closely related to the Fibonacci numbers. Mollusk shells spiral in curves derived from Fibonacci numbers. Ratios of successive Fibonacci numbers, like $\frac{5}{3}$, $\frac{8}{5}$, and $\frac{13}{8}$ are considered aesthetically pleasing. The squares in Figure 4.1 each have sides equal to a Fibonacci number. They combine to make rectangles with sides in ratios of 34 to 21, 21 to 13, 13 to 8, 8 to 5, 5 to 3, 3 to 2, and 2 to 1.

In fact, for large values of n, F_n/F_{n-1} gets arbitrarily close to a constant

$$\phi = \frac{1+\sqrt{5}}{2},$$

known as the golden ratio; ϕ is approximately equal to 1.618. The Fibonacci numbers are even thought to be useful in predicting highs and lows on the stock market. These numbers have so many interesting and varied properties that there is a mathematics research journal, the *Fibonacci Quarterly*, dedicated to results about Fibonacci numbers.

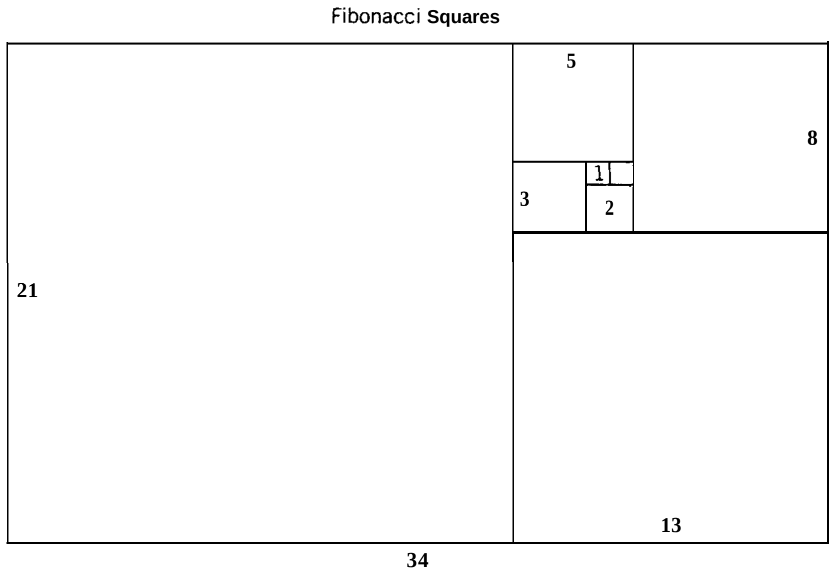

Figure 4.1.

The defining property is useful for proving results about Fibonacci numbers by induction. But one thing seems missing from our knowledge. Is there a formula for F_n? Or to calculate, say, F_{17} must we determine all the smaller Fibonacci numbers? Yes and maybe no, respectively. We shall write down a formula for F_n, but we stress that in most situations the inductive definition that $F_n = F_{n-1} + F_{n-2}$ is the most helpful fact to know. In Chapter 7 we shall do a more systematic study of sequences of numbers and their formulas.

Question 4.5. Show that $\phi = (1 + \sqrt{5})/2$ has the property that its reciprocal is itself minus one. Find all solutions to the equation $x - 1 = 1/x$.

The two solutions to the equation in Question 4.5 are $\phi = (1 + \sqrt{5})/2$ and the closely related $\phi' = (1 - \sqrt{5})/2$. These can be found by rewriting x — 1 = I/x as x’ - x — 1 = O and then solving using the quadratic formula. The relationship between ϕ and ϕ' and the Fibonacci numbers is given in our next result.

Theorem 4.1. For nonnegative integers $n, F_n = (\phi^n - \phi'^n)/\sqrt{5}$.

Proof. The proof will be by complete induction. First we check the base cases. As above we need to verify the truth of the theorem for two consecutive integers, since we shall use the crucial fact that $F_{k+1} = F_k + F_{k-1}$. First we substitute n = O, to obtain

$$\frac{\phi^0 - \phi'^0}{\sqrt{5}} = \frac{1-1}{\sqrt{5}} = 0 = F_0.$$

Next, for $n = 1$ we get

$$\frac{\phi^1 - \phi'^1}{\sqrt{5}} = \frac{(1+\sqrt{5})/2 - (1-\sqrt{5})/2}{\sqrt{5}} = 1 = F_1.$$

Using complete induction, we assume that the given formula is correct for $F_0, F_1, \ldots . F_{k-1}$ and F_k. We must prove that the formula is correct for F_{k+1}. We write 'k+ 1using smaller values:

$$\begin{aligned} F_{k+1} &= F_k + F_{k-1} \\ &= \frac{\phi^k - \phi'^k}{\sqrt{5}} + \frac{\phi^{k-1} - \phi'^{k-1}}{\sqrt{5}} \\ &= \frac{\phi^{k-1}(\phi + 1)}{\sqrt{5}} - \frac{\phi'^{k-1}(\phi' + 1)}{\sqrt{5}}. \end{aligned}$$

Since ϕ is a root of the equation $X^2 - x - 1 = 0$, we get $\phi^2 = \phi + 1$. Similarly, $\phi'^2 = \phi' + 1$. We substitute these into the equation above to get

$$F_{k+1} = \frac{\phi^{k-1}\phi^2}{\sqrt{5}} - \frac{\phi'^{k-1}\phi'^2}{\sqrt{5}}$$
$$= \frac{\phi^{k+1} - \phi'^{k+1}}{\sqrt{5}},$$

and that's exactly what we wanted to show. □

Question 4.6. Check the formula for F_2 given in Theorem 4.1.

Corollary 4.2. *F_n is* approximately equal to $\phi^n/\sqrt{5}$. Specifically,

$$\frac{\phi^n}{\sqrt{5}} - 1 < F_n < \frac{\phi^n}{\sqrt{5}} + 1$$

Proof We begin by noting that ϕ' is approximately equal to – 0.618. What we need is not its exact value but the fact that its absolute value is less than 1. Consequently, ϕ'^n will be less than 1 in absolute value for all positive integers n and $\phi'^n/\sqrt{5}$ will be less than 1 in absolute value for all nonnegative integers n. Thus

$$F_n = \frac{\phi^n}{\sqrt{5}} - \frac{\phi'^n}{\sqrt{5}} < \frac{\phi^n}{\sqrt{5}} + 1.$$ □

The other inequality is proved similarly; see Exercise 17.

We now have two ways to calculate F_n for any fixed n. One involves many additions:

$$F_2 = F_1 + F_0 = 1 + 0 = 1$$
$$F_3 = F_2 + F_1 = 1 + 1 = 2$$
$$\cdots$$
$$F_n = F_{n-1} + F_{n-2}.$$

Thus F_n could be calculated with $(n - 1)$ additions.

Question 4.7. To calculate F_n by adding as shown above appears to require that we store all of $F_0, F_1, \ldots, F_{n-1}$. Is it possible to calculate F_n by addition without storing all the previous values in n different memory locations? What is the minimum number of memory locations that you need to calculate F_n in this way?

The second way we now have to calculate F_n *is* using the formula proved in Theorem 4.1. This requires two exponentiations, one division and one subtraction as well as an approximation of the square root of 5. There are a variety of additional methods known for calculating F_n, including an addition method that is analogous to the FASTEXP algorithm developed in Chapter 2, that is, one that does not require the determination of all intermediate Fibonacci numbers. (See Exercises 13 and 14.)

For years applications of Fibonacci numbers have been found throughout mathematics. For example, a very famous open problem posed by David Hilbert in 1900, known as Hilbert's tenth problem, was finally solved in 1970 when the mathematicians Martin Davis, Yuri Matiasevic, Hilary Putnam, and Julia Robinson thought to examine the Fibonacci numbers carefully. Applications of Fibonacci numbers are also pervasive in computer science. Efficient ways to approximate the local maximum and the local minimum of a function or to merge files can use Fibonacci numbers. In Chapter 8 we shall study problems concerning shortest paths. Recent results have shown "Fibonacci heaps" and "Fibonacci trees" (whatever they are!) to be crucial in developing fast algorithms to solve these problems.

Our interest is to turn now to the complexity analysis of the Euclidean algorithm, where we shall encounter Fibonacci numbers.

EXERCISES FOR SECTION 4

1. Find a sequence of numbers G_n that satisfies the equation $G_n = G_{n-1} + G_{n-2}$ for all *n* but differs from the sequence of Fibonacci numbers.

2. Let $H_0 = 0$ and $H_1 = 1$.Fern > 1 define H_n by $H_n = H_{n-1} + 2H_{n-2}$. List the first 11 terms of the *H* sequence. What happens to the quotient H_n/H_{n-1} as *n* gets big? Prove that $H_n = [2" + (-1)^{n-1}]/3$.

3. Show that $F_1 + F_2 + \cdots + F_n = F_{n+2} - 1$.

4. Show that $F_1 + F_3 + \cdots + F_{2n-1} = F_{2n}$.

5. Show that $F_0 + F_2 + \cdots + F_{2n} = F_{2n+1} - 1$.

6. Show that every positive integer can be written as the sum of distinct, positive Fibonacci numbers. Is the choice of numbers for a given sum unique?

7. Suppose that GO = O, $G_1 = 1$, and for all $n \geq 2, G_n = 2G_{n-1} + G_{n-2}$. Find $G_2, G_3, \ldots . G_8$. Determine which of the following assertions are true.
   (a) $G_n = O(n^2)$.
   (b) G" > $2^{"-1}$ if *n* is large enough.
   (c) $G_n < 3$" if *n is* large enough.
   (d) G_n is even if and only if *n* is even.

8. Knowing F_0 and F_1, one might believe for consistency's sake that F_{-1} should be that number with the property that $F_{-1} + F_0 = F_1$. Since $F_0 = O$ and

$F_1 = 1, F_{-1}$ ought to equal 1. Determine $F_{-2}, F_{-3}, \ldots, F_{-7}$. What is F_{-n} in terms of F_n?

9. Show that $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$.

10. Show that $F_1F_2 + F_2F_3 + \cdots + F_{2n-1}F_{2n} = F_{2n}^2$.

11. Show that $F_1F_2 + F_2F_3 + \cdots + F_{2n}F_{2n+1} = F_{2n+1}^2 - 1$.

12. Find all natural numbers n such that $F_n = n$. Prove that you have found all such numbers.

13. *(a)* Show that $F_{2n} = F_n(F_n + 2F_{n-1})$.
(b) Find a similar formula for F_{2n+1} in terms of F_{n+1} and smaller Fibonacci numbers.

14. Using the results of Exercise 13, design an algorithm to determine F_n. Then count the number of multiplications and additions needed in this approach and compare with those discussed at the end of this section.

15. In Example 4.2 and Question 4.4, induction was used to show that $(\frac{3}{2})^n < F_n < 2^n$ provided that n is large enough.
(a) Find a number $b < 2$ so that a similar argument will show that $F_n < b^n$ provided that n is large enough. What is the smallest b your argument will support?
(b) Find a number $c > \frac{3}{2}$ so that $c^n < F_n$ provided that n is large enough. What is the largest c your argument will support?

16. Are the following equations true or false?

$$F_n = \lceil \phi^n/\sqrt{5} \rceil.$$
$$F_n = \lfloor \phi^n/\sqrt{5} \rfloor.$$

17. Finish the proof of Corollary 4.2 by showing that $F_n > \phi/\sqrt{5} - 1$.

18. Find all pairs of numbers (x, y) such that $x + y = 1$ and $xy = -1$.

4:5 THE COMPLEXITY OF THE EUCLIDEAN ALGORITHM

The question before us is to determine the complexity of the Euclidean algorithm when, given $b \leq c$, it computes gcd (b, c). To begin with, we let $e(c)$ denote the maximum number of multiplications and divisions performed in EUCLID when c is the larger of the two input integers. Later we shall convert $e(c)$ to a function of the number of bits needed to represent b and c in binary. We shall show that $e(c) = O(\log(c))$; that is, for all pairs of integers $b \leq c$, the Euclidean algorithm requires at most $O(\log(c))$ divisions and multiplications. Consequently, if B denotes the number of bits of input needed to encode the integers b and c, then EUCLID will be linear in B.

Look back in Section 3 and notice that we determined that e(c) = $2k$, where k is the number of Euclidean equations produced by b and c. Thus we shall search for an upper bound fork in terms of b and c. Before we find such an upper bound, we investigate some pairs for which EUCLID seems to take a long time. Example 3.1, Question 3.4, and some of the exercises point toward the Fibonacci numbers. So let's see what happens if we let b and c be consecutive Fibonacci numbers.

Lemma 5.1. In the Euclidean equations if $c = F_n$ and $b = F_{n-1}$ with $n \geq 4$, then $r_1 = F_{n-2}$.

Proof. The defining relation for the nth Fibonacci number is

$$F_n = F_{n-1} + F_{n-2}.$$

Since $0 \leq F_{n-2} < F_{n-1}$ for $n \geq 4$, this gives the first Euclidean equation with $q_1 = 1$. Thus $r_1 = F_{n-2}$. ❑

Theorem 5.2. If EUCLID is run with c = F_{k+2} and $b = F_{k+1}$ with $k \geq 1$, then there are exactly k Euclidean equations.

Proof. The proof is by (ordinary) induction on k. If $k = 1$, then c = $F_{k+2} = F_3 = 2$ and $b = F_{k+1} = F_2 = 1$. Given c = 2 and $b = 1$, EUCLID produces just one equation, specifically,

$$2 = 2 \cdot 1 + 0.$$

Now we assume that if c = F_{k+2} and $b = F_{k+1}$ with $k \geq 1$, then there are exactly k equations, and try to show that if c = F_{k+3} and b = F_{k+2}, then there are exactly $(k+1)$ equations. Now, if c = F_{k+3} and $b = F_{k+2}$, $k + 3 > 4$ and by Lemma 5.1 our first Euclidean equation is

$$F_{k+3} = F_{k+2} + F_{k+1}. \tag{A}$$

Next we divide F_{k+2} by F_{k+1}, but this is the same as if we began the Euclidean algorithm with c = F_{k+2} and $b = F_{k+1}$. By the inductive hypothesis we get k Euclidean equations from this starting point. Hence, in total, (A) is the first of $(k + 1)$ Euclidean equations. ❑

Question 5.1. Construct the Euclidean equations for (i) c = F_8 and $b = F_7$ and (ii) c = F_{10} and $b = F_9$.

Question 5.2. For O < $b \le c \le 5$, what is the maximum number of equations and for what integers does that maximum occur?

What we have done is analyze the complexity of EUCLID for a restricted set of inputs. Specifically, if $b = F_{k+1}$ and $c = F_{k+2}$, then e(c) = $2k$. Indeed a much stronger statement is true. The smallest integer c that produces as many as k Euclidean equations is c = F_{k+2} and given this c, the only integer b that will (together with c) produce as many as k Euclidean equations is $b = F_{k+1}$. *So* the Fibonacci numbers provide the worst possible input to EUCLID. We shall not prove this theorem, due to Lame in about 1845 (but you may prove it in Supplementary Exercise 14). However, we shall show that the worst-case behavior is of the same order of magnitude as that on the Fibonacci numbers.

We begin by investigating the relationship between c = F_{k+2} and k, where $k \ge 1$. Thus c ≥ 2. Corollary 4.2 and some algebra imply that

$$\phi^{(k+2)} < (\mathbf{c} + 1)\sqrt{5}.$$

Taking logs (and noting that $\sqrt{5} < 4$), we see that

$$(k+ 2)\log(\phi) < \log(c + 1)+ \log(\$)< \log(c + 1)+2.$$

Next we solve for k by dividing by log(ϕ) and subtracting 2 to get

$$k < -2 + \frac{\log(c+1)+2}{\log(\phi)}$$

If we estimate log (ϕ) by log (1.618) > $\frac{1}{2}$ we get

$$\begin{aligned} k < -2 + 2(\log(c+1)+2) &= 2\log(c+1)+2 \\ &\le 2\log(2c)+2 && \text{since } c \ge 2 \\ &= 2\log(c)+4 \le 6\log(c) && \text{since } c \ge 2. \end{aligned}$$

In short, we have that when c = F_{k+2} and $b = F_{k+1}$, the number of Euclidean equations that occur, *k, is* O(log (c)). In Exercise 11 we ask you to use Corollary 4.2 to bound k from below.

We shall now show that for any inputs b and c, the number of Euclidean equations is no larger than logarithmic in c. To accomplish this, we need a closer look at the Euclidean equations.

Theorem 5.3. If e(c) counts the maximum number of multiplications and divisions in the Euclidean algorithm with input $b \le c$, then

$$\mathbf{e(c)} = O(\log(c)).$$

Proof. Suppose that we consider the first two Euclidean equations

$$c = q_1 b + r_1$$

and

$$b = q_2 r_1 + r_2.$$

We know that $b > r_1 > r_2$. From these inequalities r_2 might be as large as $b - 2$. In fact, we shall obtain the much better estimate that $r_2 < b/2$. This estimate is better in the sense that it allows us to conclude that r_2 *is* smaller than we otherwise knew. If r_2 *is* smaller, then we expect to use fewer Euclidean equations.

Example 5.1. In the Euclidean equations of Example 3.1,

$$\begin{aligned} b = 233 \text{ and } r_2 &= 89 < b/2, \\ r_4 &= 34 < r_2/2, \\ r_6 &= 13 < r_4/2, \\ r_8 &= 5 < r_6/2, \\ r_{10} &= 2 < r_8/2, \end{aligned}$$

and

$$r_{12} = 0 < r_{10}/2.$$

Question 5.3. Show that the Euclidean equations with $b = 77$ and $c = 185$ have $r_2 < \frac{77}{2}$.

To show that r_2 is less than $b/2$ in general, we look first at r_1. If $r_1 \leq b/2$, then $r_2 < r_1$ implies that $r_2 < b/2$. If, on the other hand, $r_1 > b/2$, then $2r_1 > b$. Thus in the second Euclidean equation $q_2 = 1$ and we have that

$$b = r_1 + r_2.$$

By solving for r_2, *we* get

$$\begin{aligned} r_2 &= b - r_1 \\ &< b - \frac{b}{2} = \frac{b}{2} \qquad \text{since } r_1 > \frac{b}{2}. \end{aligned}$$

So in either case we have r_2 less than $b/2$. By doing the same thing to the next two Euclidean equations, we can show that

$$r_4 < \frac{r_2}{2} < \frac{b}{4}.$$

Let's be careful at this next step so that the pattern becomes clear. As above we argue that

$$r_6 < \frac{r_4}{2} < \frac{b}{8}.$$

Thus, in general, we have

$$r_{2t} < \frac{b}{2^t}. \tag{B}$$

How long can this go on? If $b/(2^t) \leq 1$, then by the preceding equation the remainder r_{2t} equals zero. But $b/(2^t) \leq 1$ implies that log(b) $\leq t$. Thus once $t \geq \log(b)$ or equivalently once $t = \lceil \log(b) \rceil$, then

$$\frac{b}{2^t} \leq \frac{b}{2^{\log(b)}} = \frac{b}{b} = 1,$$

and from (B) we get $r_{2t} = O$. Thus k, the number of Euclidean equations, is at most $2t$, where $t = \lceil \log(b) \rceil$. Thus

$$\mathbf{e(c)} = 2k \leq 2(2t) = 2(2\lceil \log(b) \rceil) \leq 4\lceil \log(c) \rceil = O(\log(c)).$$

Question 5.4. Look at the Euclidean equations from Example 3.1. For each integer t compute r_{2t+2}/r_{2t}.

In conclusion we have that the Euclidean algorithm is a good algorithm. Look back in Section 2 at the inequalities of line (2):

$$\log(c) \leq B = \lfloor \log(b) \rfloor + 1 + \lfloor \log(c) \rfloor + 1.$$

Since it requires B bits to input b and c in binary, the number of multiplications and divisions is bounded by

$$\mathbf{e(c)} = O(\log(c)) = O(B).$$

Thus EUCLID is a linear algorithm.

EXERCISES FOR SECTION 5

1. Show that if $c = F_{k+2}$ and $b > F_{k+1}$ in the Euclidean equations, then $r_1 < F_k$.

2. In the Euclidean equations, if c = F_{k+2} and $b < F_{k+1}$, *is* $r_1 < F_k$?
3. Construct the Euclidean equations if c = F_9 and $b = F$,.
4. If c = F_{k+3} and $b = F_{k+1}$, what can you say about the number of Euclidean equations?
5. Suppose that GO = 4, $G_1 = 7$, and for $n > 1$, $G_n = G_{n-1} + G_{n-2}$. Exhibit G_n fern= 2,3,...,8.
6. Exhibit the Euclidean equations with c = G_8 and $b = G_7$. (G_n *is* defined in Exercise 5.)
7. How many Euclidean equations are there if c = G_{k+2} and $b = G_{k+1}$? (G_n is defined as above.)
8. In the Euclidean equations we know that $r_4 < b/4$. Is $r_3 < b/4$?
9. Choose a value oft so that in the Euclidean equations $r_8 < r_4/t$.
10. Suppose that C_n is a sequence of nonnegative integers with the property that $C_n < C_{n-1}/4$. If $C_1 = M$, for what value z can you guarantee that C_z = O?
11. Use Corollary 4.2 to find a constant D such that if $c = F_{k+2}$, then $k > D$ log(c).
12. The complexity of EUCLID was shown to be O(B), where B equals the number of bits needed to represent the integers b and c. Thus the number of multiplications and divisions performed is at most $sB + t$ for some integers s and t. Find integers s and t that give an upper bound on the number of these arithmetic operations that is as small as possible based on the analysis in this section.

4:6 CONGRUENCES AND EQUIVALENCE RELATIONS

Integer arithmetic is fundamental to the mathematical field of number theory and to the computer science field of cryptography. The particular kind of arithmetic used in these fields is known as **modular** or **congruence arithmetic.** In this section we introduce the basics of arithmetic **modulo *n*** and develop simultaneously the concept of an equivalence relation. In the next section we apply this work to encryption schemes.

Definition. If n is a positive integer and a and b any two integers, we define

$$a \equiv b \pmod{n}$$

(read **"*a* is congruent to *b* modulo *n*"**) if $(a - b)$ *is* divisible by n. We let $[a]$ denote the set of all integers congruent to a modulo n,

$$[a] = \{x: a \equiv x \pmod{n)}\}.$$

This is called the equivalence class containing a.

Example 6.1. Let $n = 12$. Then 1-13 (mod 12), $1 \equiv 25$ (mod 12),$13 \equiv 25$ (mod 12), 1- –11 (mod 12), and

$$\begin{aligned} [1] &= \{\ldots, -23, -11, 1, 13, 25, \ldots\} \\ &= \{1 + 12k : k \text{ an integer}\}. \end{aligned}$$

If "#" means "not congruent to," then $1 \not\equiv 0$ (mod 12), $1 \equiv 1$ (mod 12), $1 \not\equiv 2$ (mod 12), and $1 \not\equiv i$ (mod 12) for $i = 3$, **4,...,12.**

We are used to working "modulo 12," since that is how our clocks and some of our measurements work, If it is 11 A.M. and I have an appointment in 3 hours, then since $11 + 3 = 14 \equiv 2$ (mod 12), the appointment is for 2 P.M.

Question 6.1, Determine which of the following are true and which false: (a) 2 ≡ 3 (mod 12), (b) 2 ≡ 4 (mod 12), (c) 2-10 (mod 12), (d) 2 ≡ 14 (mod 12), (e) 2 ≡ – 10 (mod 12), and (f) – 10- – 22 (mod 12). Describe all integers x that are congruent modulo 2 to O. Working modulo 3, list six elements of [1] and then describe the entire set precisely.

Question 6.2. Let n be a positive integer and i an integer such that O $\leq i < n$. What is the least integer j $> i$ such that $i \equiv j$ (mod n)? What is the largest negative number m such that $i \equiv m$ (mod n)?

Congruences modulo n behave like equalities.

Lemma 6.1. Let n be a positive integer and a, b, and c arbitrary integers. Then
(i) $a \equiv a$ (mod n),
(ii) If $a \equiv b$ (mod n), then $b \equiv a$ (mod n), and
(iii) If $a \equiv b$ (mod n) and $b \equiv c$ (mod n), then $a \equiv c$ (mod n).

Proof. We prove part (iii). If a $\equiv b$ (mod n), then n divides (a – b) and so there is an integer i such that a – b = in. If $b \equiv c$ (mod n), then there is an integer j such that $b - c = jn$. Thus

$$a - c = (a - b) + (b - c) = in + jn = (i + j)n,$$

and (a – c) is divisible by n, that is, a ≡ c (mod n). □

Question 6.3. Prove parts (i) and (ii) of Lemma 6.1.

Here's a vocabulary to highlight the similarities between relationships like "–," "= (mod n)," and others. We say that a relation " -" is defined on a set S

(finite or infinite) if for each pair of elements (a, b) in S, $a \sim b$ is either true or false. Colloquially, if $a - b$ is true, then we say a is related to b. A more formal way to describe a relation on S is to say that $\sim$ corresponds to a function

$$T: S \times S \rightarrow \{\text{True, False}\}$$

such that $T(a, b)$ = True if and only if $a - b$ is true (or equivalently a is related to b).

Example 6.2. Let S be the set Z of all integers. Then equality gives us a relation on Z by defining $i \sim j$ to be true for integers i and j if and only if $i = j$. Similarly, for a fixed positive integer n, congruence modulo n *is* a relation on Z if we define $i - j$ to mean that $i \equiv j \pmod{n}$.

Question 6.4. Which of the following defines a relation on the given set S? (a) S = Z and $\sim$ stands for $\leq$; (b) S = all subsets of Z and $\sim$ stands for $\subseteq$; and (c) S = all real numbers and $r \sim s$ means that $(r - s)$ *is* even.

Definition. A relation $\sim$ defined on a set S is said to be an equivalence relation if it satisfies the following three properties. If a, b, and c are arbitrary elements of S, then

(i) $a \sim a$	**(reflexive property)**
(ii) If $a \sim b$, then $b \sim a$	**(symmetric property)**
(iii) If a $\sim b$ and $b \sim c$, then $a \sim c$	**(transitive property).**

Lemma 6.1. says that the relation "congruence modulo n" defined on Z is an equivalence relation.

Example 6.3. Let $U = \{1, 2, \ldots n\}$ and let P be the set of all subsets of U. If $A, B, C \subseteq U$, then (i) $A \subseteq A$ is true for every subset A, and (iii) if $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$, but it is not true that (ii) if $A \subseteq B$, then $B \subseteq A$. Thus the relation of containment is not an equivalence relation on P.

Example 6.1 (continued). Working modulo 12, we saw that 1 is not congruent modulo 12 to 0,2,3,. ... or 11. Let us look at the equivalence classes [0], [1], [2], . . . [11].

$$[0]=\{\ldots -24, -12, 0, 12, 24, 36\ldots\}$$
$$[1]=\{\ldots -23, -11, 1, 13, 25, 37\ldots\}$$
$$[2]=\{\ldots -22, -10, 2, 14, 26, 38 \ldots\}$$
$$\ldots$$
$$[11]=\{\ldots -25, -13, -1, 11, 23, 35\ldots\}.$$

Notice that no two of these sets intersect and that every integer is in precisely one of these sets.

Definition. If $\sim$ is an equivalence relation on the set S, then we define for a in S, the **equivalence class containing** a to be

$$[a] = \{x \text{ in } S: a \sim x\}.$$

Then just as in the case of congruence of integers modulo n, the collection of all distinct equivalence classes of S divides up S into disjoint subsets. Such a division is called a **partition of S.**

Lemma 6.2. If $\sim$ is an equivalence relation on a set S, then
(i) a is in [a] for alla in S,
(ii) [a] = [b] if and only if $a \sim b$, and
(iii) if *[a] # [b],* then *[a] n [b]*= $\varnothing$·

Proof. (i) *a is* in *[a],* since $a \sim a$ by the reflexive property of equivalence relations.

(ii) If *[a] = [b],* then *a* in *[a]* implies that a is in $[b]$ and so b $\sim a$ by definition. By the symmetric property, $a \sim b$. Conversely, suppose that $a \sim b$ and let x be in *[a].* Then $a \sim x$ and x $\sim a$, and by the transitive property x $\sim b$ and so $b \sim x$. Thus x is in *[b]* and [u] $\subseteq$ *[b].* The proof that *[b]* $\subseteq$ *[a]* is carried out in the same way.

(iii) Suppose that [a] *n [b]* # $\varnothing$ *so* that there is an element x in [a] *n [b].* Then $a \sim x$ and b-x. By the symmetric property x $\sim b$ and by the transitive property a $\sim b$. Using part (ii), we have that [a] = *[b].* We have proved the contrapositive of (iii). □

Fix a positive integer n. When working with the integers modulo n, there are many ways to express the same equivalence class. For example, [0] = *[n]* = *[– n]* = $[17n]$. It is often convenient to represent an equivalence class $[i]$ using the least nonnegative integer to which i is congruent modulo n. We can find that integer by dividing i by *n:*

$$i = qn + r \qquad \text{with } 0 \leq r < n$$

and so *[i]* = *[r].*

Definition. If $i = qn$ + r with O $\leq r < n$, then *r is* called the **least nonnegative residue** of i modulo n.

This process also shows that every integer i is in one of the classes [0], [1],. . . . *[n* – 1] modulo n. Furthermore, no two of the equivalence classes [0], [1],...,

$[n-1]$ are equal, for if $[i] = [j]$, then $i \equiv j \pmod{n}$ by Lemma 6.2 (part ii). But since $0 \leq i, j < n$, then n cannot divide $(i - j)$ unless $i = j$. Thus it is not the case that $[i] = [j]$ when $0 \leq i < j < n$. The equivalence classes $[0], [1], \ldots, [n-1]$ are a complete (and useful) set of equivalence classes of the integers modulo n.

Example 6.1 (continued). When working modulo 12, we use the equivalence classes [0], [1],. . . . [11].

We can also do arithmetic with the equivalence classes modulo n: addition, subtraction, multiplication, exponentiation, and sometimes division.

Definition. **The equivalence** classes $\{[0], [1], \ldots, [n-1]\}$ are called the integers **modulo** n and are denoted by $\mathbf{Z}_n$.

Lemma 6.3. **If** $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then
(i) $a + c \equiv b + d \pmod{n}$,
(ii) $a - c \equiv b - d \pmod{n}$, and
(iii) $ac \equiv bd \pmod{n}$.

Proof. We prove part (iii). **Since** $a \equiv b \pmod{n}$, there is an integer i such that $a - b = in$ or equivalently $a = b + in$. Since $c \equiv d \pmod{n}$, there is an integer j such that $c = d + jn$. Thus

$$\begin{aligned} uc &= (b + in)(d + jn) \\ &= bd + bjn + din + ijn^2 \\ &= bd + (bj + di + ijn)n. \end{aligned}$$

Thus $(ac - bd)$ is divisible by n, and $ac \equiv bd \pmod{n}$. □

Question 6.5. Verify the first two parts of Lemma 6.3.

Thus we can define arithmetic modulo n on equivalence classes as follows.

$$[a] + [b] = [a + b],\ [a] - [b] = [a - b],$$
$$[a] \cdot [b] = [ah], \text{ and } [a]^k = [a^k] \text{ fork a positive integer.}$$

These definitions look sensible, but there are some important points to be checked. We must check that addition and multiplication are **"well defined** by these equations. (Subtraction is just addition of negative numbers, and exponentiation is just repeated multiplication, so we concentrate on the other two operations.) What this means is that if x is any element of [a] and y is any element of

$[b]$, then

$$[x+y]=[a+b] \qquad \text{and} \qquad [xy]=[ah].$$

We now show that addition of equivalence classes modulo n is well defined. If x is in $[a]$ and y is in $[b]$, then

$$a \equiv x \pmod{n} \qquad \text{and} \qquad b \equiv y \pmod{n}.$$

By Lemma 6.3

$$a+b \equiv x+y \pmod{n} \qquad \text{and} \qquad [x+y]=[a+b].$$

Question 6.6. Show that multiplication of equivalence classes is well defined.

Example 6.1 (once more). Working with the integers modulo 12, we want to add and multiply in the following way:

$$[1]+[1]=[2],\ [1]+[2]=[3],\ [8]+[9]=[17]=[5],$$
$$[-5]+[10]=[5],\ [3]\cdot[0]=[0],\ \text{and}\ [5].\,[6]=[30]=[6].$$

But is this consistent? We know that [1] = [25], since $1 \equiv 25 \pmod{12}$. Thus it should be the case that

$$[1]+[25]=[1]+[1]=[2],$$

Fortunately, [1] + [25] = [26] = [2], since $2 \equiv 26 \pmod{12}$. Similarly, [17] = [5] and [– 6] = [6]. Thus

$$[17]\ .\ [-6]=[-102]=[6]=[5]\cdot[6].$$

When can we do division or cancel modulo n?

Lemma 6.4. If $ab \equiv cd \pmod{n}$ and $a \equiv c \pmod{n}$, then $b \equiv d \pmod{n}$ provided that $\gcd(a, n) = 1$.

Proof. By assumption there are integers i and j such that

$$ab-cd=in \qquad \text{and} \qquad a-c=jn.$$

Substituting $c = a - jn$ in the first equation yields

$$ab-(a-jn)d=in.$$

Thus

$$ab - ad = in - jdn$$

and

$$a(b - d) = (i - jd)n. \tag{A}$$

Since n divides the right-hand side of (A), n also divides the left-hand side, $a(b - d)$. Since a and n have gcd 1 and thus no factors in common, n must divide $(b - d)$. Thus $b \equiv d \pmod{n}$. ❑

Question 6.7. Pick five distinct integers a, b, c, d, and n such that gcd $(a, n) = 1$, $a \equiv c \pmod{n}$, and $ab \equiv cd \pmod{n}$. Verify that $b \equiv d \pmod{n}$. Then find integers a, b, c, d, and n such that gcd (a, n) # 1, $a \equiv c \pmod{n}$, $ab \equiv cd \pmod{n}$, but $b \not\equiv d \pmod{n}$.

Now what would it mean to say that we can do division with the integers modulo n? Division by a number x is the same as multiplying by l/x, and I/x has the property that $x(1/x) = 1$.

Definition. Given [a] in the integers modulo n, we say that $[a]$ has a multiplicative inverse if there is another equivalence class $[b]$ such that

$$[a] . [b] = [1].$$

Thus $[b]$ *is* playing the role of" $1/[a]$" and is called the multiplicative inverse of [a]. If $[a] \cdot [b] = [1]$, then $[b] \cdot [a] = [1]$ and so $[a]$ is also the multiplicative inverse of $[b]$.

Similarly, if a and b are two integers with $O < a, b < n$ such that $ab \equiv 1$ (mod n), then we say that a and b are each other's multiplicative inverses.

Corollary 6.5. Let n be a positive integer. Then the equivalence class $[a]$ has a multiplicative inverse if and only if gcd $(a, n) = 1$.

Proof. If 1 = gcd (a, n), Corollary 3.3 says there are integers x and y such that

$$1 = xa + yn.$$

Thus

$$1 \equiv \text{xa} \pmod{n},$$

and so

$$[1] = [xa] = [x] \cdot [a].$$

Thus $[x]$ is the multiplicative inverse of $[a]$.

Conversely, if

$$[a] \cdot [x] = 1,$$

then

$$ax = 1 + kn \qquad \text{for some integer } k.$$

Thus any common divisor of a and n must also divide 1, and so $\gcd(a, n) = 1$. ❑

Two integers a and b are said to be relatively prime if $\gcd(a, b) = 1$. Thus an integer a has a multiplicative inverse modulo n if and only if a and n are relatively prime.

Example 6.1 (again). In Z_{12}, only 1, 5, 7, and 11 are relatively prime to 12. Here are their multiplicative inverses:

$$[1] \cdot [1] = [1], \quad [5] \cdot [5] = [25] = [1],$$
$$[7] \cdot [7] = [49] = [1], \quad \text{and} \quad [11] \cdot [11] = [121] = [1].$$

In other words, each of 1, 5, 7, and 11 is its own multiplicative inverse. Here is a brute force check that [2] does not have a multiplicative inverse:

$$\begin{array}{lll} [2] \cdot [0] = [0], & [2] \cdot [1] = [2], & [2] \cdot [2] = [4], \\ [2] \cdot [3] = [6], & [2] \cdot [4] = [8], & [2] \cdot [5] = [10], \\ [2] \cdot [6] = [0], & [2] \cdot [7] = [2], & [2] \cdot [8] = [4], \\ [2] \cdot [9] = [6], & [2] \cdot [10] = [8], & [2] \cdot [11] = [10]. \end{array}$$

Question 6.8. Find multiplicative inverses for all elements of Z_5 and of Z_{10} that have inverses. Which elements of Z_{18} have multiplicative inverses?

Finding inverses will be important in the application presented in Section 7, as will a variation on the next theorem, known as Fermat's little theorem. This one he really did prove.

Theorem 6.6. If p is a prime number and gcd $(a, p) = 1$, then

$$a^{p-1} \equiv 1 \pmod p.$$

Proof. Notice that for any integer i, gcd (i, p) is either 1 or p, the only divisors of p. Thus gcd $(i, p) = 1$ if and only if p does not divide i, that is, if and only if $i \not\equiv 0 \pmod p$. Thus by assumption $a \not\equiv 0 \pmod p$. Consider the equivalence classes

$$[a], [2a], [3a], \ldots, [(p-1)a]. \tag{B}$$

We claim that none of these is [0] and that no two of them are equal.

First if it were the case that

$$[is] = [0] \qquad \text{where } 1 \le i < p,$$

then

$$ia \equiv 0 \pmod p.$$

Thus p divides ia, and since gcd $(a, p) = 1$, p divides i, a contradiction, since $1 \le i \le (p-1)$. Thus none of the equivalence classes in (B) equals [0].

Next suppose that

$$[is] = [ja] \qquad \text{where } 1 \le i, j < p.$$

Then

$$\begin{aligned} ia &\equiv ja \pmod p && \text{by Lemma 6.2 (part ii), and} \\ i &\equiv j \pmod p && \text{by Lemma 6.4,} \end{aligned}$$

a contradiction, since both i and j are positive integers less than p. Thus the $(p-1)$ equivalence classes listed in (B) are the same as the equivalence classes [1], [2], [3],..., $[p-1]$, although probably listed in a different order. Then

$$\begin{aligned} [a]\cdot[2a] \cdots [(p-1)a] &= [1]\cdot[2]\cdots[p-1] \\ [a(2a)\cdots((p-1)a)] &= [(p-1)!] && \text{multiplying equivalence classes} \\ a(2a)\cdots((p-1)a) &\equiv (p-1)! \pmod p && \text{by Lemma 6.2 (part ii)} \\ a^{p-1}(p-1)! &\equiv (p-1)! \pmod p && \text{simplifying} \\ a^{p-1} &\equiv 1 \pmod p && \text{by Lemma 6.4} \end{aligned}$$

since gcd $(p, (p-1)!) = 1$. □

Question 6.9. Pick a prime p and an integer b such that gcd $(b, p) = 1$, write down the equivalence classes $[b]$, $[2b]$,..., $[(P-1)b]$ modulo p, and verify that they are the same as the classes [1], [2],. . . . $[p-1]$. Check that $b^{p} 1 \equiv 1 \pmod{p}$. Find c, an integer with gcd (c, p) # 1, and show that $c^{p-1} \not\equiv 1 \pmod{p}$.

This has been a brief introduction to arithmetic modulo n and to the ideas of equivalence relations. We shall use this in an application to cryptography (the art of secret messages) in the next section.

EXERCISES FOR SECTION 6

1. In each of the following find the least nonnegative integer i such that
   (a) 4^{30}- i (mod 19).
   (b) 2^{3}'- i (mod 377).
   (c) 2' $\equiv$ 1 (mod 17).
   (d) $2^{11}3^{13} \equiv i \pmod{7}$.
   (*Hint:* After each multiplication replace the result by its least nonnegative residue modulo n.)
2. Explain why it is always true that $n^5 \equiv n \pmod{10}$ or, in other words, why n^5 and n always have the same last digit.
3. Prove that for every integer n, either $n^2 \equiv O \pmod{4}$ or $n^2 \equiv 1 \pmod{4}$. Use this to show that there are no integers x and y such that $X^2 + y^2 = 1987$.
4. The set {0, 1,. . . , n — 1} is called a complete residue system modulo n because every integer is congruent modulo n to exactly one of these numbers.
   (a) Find a complete residue system modulo n in which all numbers are negative.
   (b) Find a complete residue system modulo n in which all numbers have absolute value at most $n/2$.
5. Is either the relation "<" or "s" an equivalence relation on the integers?
6. Which of the following define an equivalence relation on the integers? Explain your answer.
   (a) $a \sim b$ if a divides b.
   (b) $a \sim b$ if $a < b$.
   (c) $a \sim b$ if $|a| \leq |b|$.
   (d) $a \sim b$ if a and b begin with the same (decimal) digit.
   (e) a $\sim b$ if when a and b are expressed as a = 2is and $b = 2^j t$ with i and j nonnegative integers and s and t odd integers, then s = t.
   (f) $a \sim b$ if when a and b are expressed as in part (e), then i = j.

7. Prove that the following is an equivalence relation defined on the integers: $a \sim b$ if a and b have the same number of prime divisors, counting multiplicity (e.g., $18 = 2 \cdot 3^2$ has three prime divisors). For this equivalence relation are addition and multiplication of equivalence classes well defined by $[a] + [b] = [a+ b]$ and $[a] \cdot [b] = [ah]$? Explain.

8. Give an example of a relation on a set that has the following properties.
   (a) Reflexive and symmetric, but not transitive.
   (*b*) Reflexive and transitive, but not symmetric.
   (*c*) Symmetric and transitive, but not reflexive.

9. (a) Write down the elements of Z_2. Then write down an addition and multiplication table for 2_2; that is, write down all possible sums $[a] + [b]$ and all possible products [u] $\cdot$ $[b]$.
   (b) Do the same for Z_3.
   (c) Do the same for Z_4.

10, Rewrite the Euclidean equations using congruences.

11. Prove that if $[i]$ and [j] are equivalence classes modulo n such that [i] = [j], then gcd (i, n) = gcd (j, n).

12. A relation $\sim$ on a set S is called a total (or linear) ordering if
   (i) for all a and b in S, exactly one of the following holds:
   $a \sim b$, $a = b$, or b-a, and
   (ii) for elements a, b, and c in S, if $a \sim b$, and $b \sim c$, then $a \sim c$.
   Do either $<$ or $\leq$ define a total ordering on the integers? Explain.

13. Give an example of an equivalence relation on the integers that is not a total ordering. Explain which properties of a total order hold for your example and which don't.

14. Explain why, in general, if $\sim$ is a total ordering on a set S, then $\sim$ is not an equivalence relation on S. Conversely, explain why if $\sim$ is an equivalence relation on S, then - is not a total ordering.

15. Prove that if p is a prime number, then for every integer n,
$$n^p \equiv n \pmod{p}.$$

16. Investigate whether or not the following is true: If gcd (u, n) = 1, then $a^{n-1} \equiv 1 \pmod{n}$.

17. Suppose that p is a prime number and gcd (a, p) = 1. Then explain why $[u^{p-}2_{]}$ is the multiplicative inverse of [a] modulo p. For each $i = 2, 3, \ldots, p-1$, find an expression for the multiplicative inverse of $[a^i]$ modulo p.

18. We define an equivalence relation on ordered pairs of integers, Z x (Z – {O}), (i.e., on all ordered pairs with the second entry nonzero) by $(a, b) \sim (c, d)$ if $ad = be$.

(i) Prove that this is an equivalence relation.
(ii) Describe the equivalence classes [(1, 2)], [(1, l)], [(4, 2)], and [(2, 3)].
(iii) In generaI, describe [(r, s)] and compare this with the rational number *r/s*.
(iv) Which equivalence classes [(i-, s)] have multiplicative inverses? If *[(r, s)]* has a multiplicative inverse, what is it?

4:7 AN APPLICATION PUBLIC KEY ENCRYPTION SCHEMES

Although our discussion of the greatest common divisor problem has been couched in modern terminology, most of what we have presented in this chapter is ancient. It was developed without any thought of computing machines like those we now possess and with no anticipation of future applications. It is a truism in mathematics that the purest (i.e., most theoretical and seemingly least applicable) ideas from one generation of mathematicians frequently become indispensable took of the applied mathematicians of subsequent generations. The application of the Euclidean algorithm and related number theory that we are about to present exemplifies this phenomenon.

The problem that we confront is that Bob wants to send Alice a message, the content of which is to remain a secret from Eve. The message will be a sequence of integers, $M_1, M_2, \ldots M_k$ with $O< M_i < N$ for $i=l,2,\ldots k$, where N is a number chosen with which to work modulo N. We'll see how Alice chooses N later. Such a representation of a message by numbers is no restriction. For example, this book has been prepared electronically, and each character of the keyboard of the computer terminal has associated with it a unique decimal number, in this instance called its ASCII code. For use in this chapter we include the ASCII code for capital letters in Table 4.1.

Table 4.1

Letter:	A,	B,	C,	D,	E,	F,	G,	H,	I,	J,	K,	L,	M,	N	O,	P,
Code:	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
	Q,	R,	S,	T,	U,	V,	W,	X,	Y,	Z,	blank					
	81	82	83	84	85	86	87	88	89	90	32					

Thus the assumption that Bob's message is a sequence of decimal numbers is not restrictive. Sending a message might be by telephone or electronic mail or almost anything else. Crucial to the model is that Eve has the technology to intercept the message. This turns out to be surprisingly realistic. Thus if Bob just sends Alice the ASCII code of the real message, we assume that Eve can intercept and correctly interpret its content.

So what can Bob do? Very simply he must devise a way to disguise his message, to encrypt it so that after Eve, or anyone else, intercepts the disguised message, she will not be able to figure it out. Of course, if Bob does too good a job encrypting the message, maybe Alice won't be able to figure it out either. So Bob and Alice agree on an encryption scheme. When Bob has a message to send Alice, he pulls out his encryption book (or maybe calls his encryption computer program) and encrypts his message. Alice, having the appropriate decryption book (or computer program), can unscramble or decrypt the message. This is fine unless Eve obtains a copy of Bob's encryption procedure. Then it may be that the message is no longer secure. (If Eve obtains Alice's decrypting procedure, presumably Eve can decrypt any message that Alice can decrypt.) We shall describe here a method that will tell Bob exactly how to encrypt his message. It will tell Alice exactly how to decrypt the received message. Finally, (and this is truly magical), even if Eve knows Bob's method of encryption, Eve will not be able to decipher the message.

There is a family of related methods that will accomplish the above goals. These are known as public key encryption schemes, and they use so-called trapdoor functions. (The analogy is that encrypting information like opening a trapdoor from above is easy, but decrypting like opening a trapdoor from below when one is stuck in the trap is hard.) The scheme we present uses the Euclidean algorithm and modular arithmetic and is known as the RSA **scheme** for Rivest, Shamir, and Adleman, the inventors of the scheme. There are other schemes based on a wide variety of mathematical ideas, and there is a great deal of research being done on the question of just how secure these trapdoor schemes are.

Suppose that Bob wants to send a message with j letters, including blanks between words. Using Table 4.1, this becomes a decimal number with $2j$ digits when we replace each letter by the corresponding two digits of the ASCII code. If Bob simply transmits the ASCII code equivalent, Eve will be able to look up the ASCII code in a table and understand the message.

Example 7.1. The message "HELLO" becomes 7269767679.

Question 7.1. Translate the message "HOWDY" into its ASCII code equivalent. Decipher the message 83858270327383328580.

Actually, most messages, like those just mentioned, will turn into numbers that are far too large to work with. Thus we agree in advance to break the $2j$ digits of the message up into blocks of length B and then send k messages $M_1, M_2, \ldots M_k$, each of length at most B.

Example 7.1 (continued). Let $B = 4$. Then we send the encryption of HELLO as three messages: 7269, 7676, and 7932, with a blank added at the end to fill out the last block.

Question 7.2. With $B = 4$, the largest code that can be sent using capital letters is 9090. What letters produce this code? What is the smallest possible decimal number that we can transmit with $B = 4$?

Now it's time for Alice to get sneaky. She picks an integer N and announces that all work will be done modulo N. In particular, the transmitted messages will lie between O and N. (Then a convenient choice of block length B is one less than the number of decimal digits in N.) The sneaky part is that Alice picks N to be the product of two nearby, large prime numbers p and q. So $N = pq$ with $p \neq q$. **Now** it is easy and quick to multiply two prime numbers or any two numbers, even if they are very large. What is very difficult to do, given an integer N, is to determine its prime factors.

(Note: All steps in this process are summarized at the end of the section in the algorithm RSA.)

Exercise 14 gives a simple algorithm DIVISORSEARCH that searches for the divisors of an integer N. (A more sophisticated algorithm for finding prime divisors is presented in Supplementary Exercise 4.) DIVISORSEARCH finds divisors by checking whether the integers 2,3,. ... up to $\sqrt{N}$ divide N. If N = pq, then either p or q must be at most $\lfloor\sqrt{N}\rfloor$. If, say, p is discovered to be a divisor, then q *is* found as N/p.

So why not use this algorithm? DIVISORSEARCH is slow. (In Exercise 15 you are asked to verify that DIVISORSEARCH is exponential.) Faster ones have been derived, using very deep mathematics, but all the known algorithms for factoring a number have nonpolynomial running time. For example, whereas we can easily multiply together two 30-digit numbers to get a 60-digit number, if we are given N with 60 digits it takes much longer to unscramble it into its prime factors. If N is either a prime or the product of two large primes that are near one another, then an algorithm as in Supplementary Exercise 4 would have to run about a year before this fact is discovered. That's no problem for this application Alice will choose new values of p and q with N = pq every 6 months before Eve is able to find (or to run a computer program to find) the factors of N.

Question 7.3. Each of the following are of the form pq for primes p and q. Try to factor each 323,4087, and 8633.

However, Alice has more tricks up her sleeve. After selecting N = pq, she selects an integer e > 1, known as the **exponent,** with the property that gcd (e, $(p - 1)(q - 1)$) = 1. Remember that the gcd of two numbers is easy and quick to calculate. Alice can just try random numbers between O and N and run the Euclidean algorithm on them to find an e relatively prime to $(p - 1)(q - 1)$.

Question 7.4. Let N = **7 11 = 77.** Then search through 2,3,4,. . . to find four numbers e that are relatively prime to $6 \cdot 10 = 60$.

Example 7.2. Suppose that N = 9991 = 97 · 103, where 97 and 103 are both prime. (These are not particularly large prime numbers but will keep us occupied with calculations by hand.) Let us check that e = 11 meets the requirements for an exponent by calculating gcd (1 I, 96. 102) = gcd(11, 9792).

$$\begin{aligned} 9792 &= 890 \cdot 11 + 2 \\ 11 &= 5 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 + 0. \end{aligned}$$

Once Alice has determined e, she will perform one more calculation, described later, but then she may destroy the factors of N or else she must guard them closely as they are the key to the security of this system.

However, Alice can be quite open with the numbers e and N. In fact, she sends them to Bob without any secrecy. Maybe she even lists them in a phone book or publishes them in the newspaper. These numbers will tell Bob (and for that matter anyone who cares to send a message to Alice) how to securely encrypt their message. The procedure goes as follows.

Here's what Bob does to encrypt the message. First the original message is turned into ASCII code using Table 4.1 and then the resulting huge number is broken into blocks of length B. Each block is one of the messages $M_1, M_2, \ldots, M_k$ to be sent. Next Bob must check that each message M_i, for $i = 1,2,\ldots, k$, *is* relatively prime to N. If not, in the gcd calculation he will discover that their common divisor is either p or q. In that case he announces to Alice and to the world that he has found a factor of N and it is time to change their protocol (i.e., to change the values of e and N). However, most messages and numbers are relatively prime to N as shown in Exercise 9.

Then for each message M_i, $i = 1,2,\ldots, k$, Bob calculates R_i, where

$$R_i \equiv M_i^e \pmod{N} \qquad \text{and} \qquad 0 < R_i < N.$$

Precisely, he can divide by N and find the remainder R_i

$$M_i^e = QN + R_i \qquad \text{with } 0 < R_i < N.$$

Then he will transmit the encrypted message $R_1, R_2, \ldots, R_k$.

Now M_i^e will often be a large number, but one that can be determined quickly using FASTEXP however, there are additional ways in modular arithmetic to keep the numbers relatively small. Recall that by Lemma 6.3, if $a \equiv b \pmod{N}$, then $a^e \equiv b^e \pmod{N}$. Thus when we need M_i, M_i^2, M_i^4, and so on for FASTEXP, we can repeatedly replace the numbers by their least nonnegative residues modulo N, as shown in the next example. This replacing process is also known as reducing **modulo** N.

Examples 7.1 and 7.2 (continued). With N = 9991 and e = 11, we begin the encryption of the message “HELLO” from its ASCII code 726976767932:

$$M_1 = 7269 \qquad M_2 = 7676 \qquad M_3 = 7932.$$

First we check that gcd (7269, 9991) = 1. Then since $M_1^{11} = M_1^8 M_1^2 M_1$, *we* calculate

$$\begin{aligned}
M_1^2 & \mathit{-52838361-5953} \pmod{9991} \\
M_1^4 & \equiv (5953)^2 \pmod{9991} \\
& -35438209 \pmod{9991} \\
& \equiv 132 \pmod{9991} \\
M_1^8 & \equiv (132)^2 \pmod{9991} \\
& \equiv 17424 \pmod{9991} \\
& \equiv 7433 \pmod{9991}.
\end{aligned}$$

Thus the first message R_1 that we want to send is

$$\begin{aligned}
M_1^{11} & \equiv 7433 \cdot 5953 \cdot 7269 \pmod{9991} \\
& \equiv 44248649 \cdot 7269 \pmod{9991} \\
& \equiv 8501 \cdot 7269 \pmod{9991} \\
& -61793769 \pmod{9991} \\
& \equiv 9425 \pmod{9991}.
\end{aligned}$$

Question 7.5. Show that $M_2 = 7676$ and $M_3 = 7932$ are relatively prime to 9991 and then determine either R_2 or R_3 for these messages.

This procedure to encrypt a message could be implemented easily in a computer program; see algorithm RSA.

There are two questions that require an answer. First, how is Alice to recover the content of the original message? Second, assuming that Eve receives the encrypted message and possesses the numbers e and N, why can’t she discover the hidden message?

We answer the second question first. We assume that Eve intercepts the message *RI*, R_2, ... R_k with O < $R_i < N$ for i = 1,2, . . . ,k. How might she find $M_1, M_2, \ldots, M_k$? Why can’t she just take the eth root of R_i to get M_i? Or why not try all possible messages *M,* raise each to the eth power and reduce modulo N until the correct messages are found?

Examples 7.1 and 7.2 (continued). The 1 lth root of 9425 is 2.2977 ..., and so this is not much help. The problem is that we took the 1 lth power of 7269 modulo 9991 and now we would need the 1 lth root modulo 9991, whatever that means.

The straightforward approach of trying everything would tell us to calculate I'(mod 9991), 2^e(mod 9991), 3^e(mod 9991), and so on. But checking with the ASCII code Table 4.1 we can be more clever. The encrypted word is a four-digit number of the form:

$$3232, \qquad 32wz\ (65 \leq wz \leq 90),\ wz32\ (65 \leq wz \leq 90),$$

or

$$uvwz\ (65 \leq uv, wz \leq 90),$$

a total of 729 possible ASCII codes. Now for any integer i, i^{11} requires five multiplications: three to form i^2, i^4, and i^8, and two more to combine these into i^{11}. If at each stage one reduces modulo 9991, then five more divisions and five more multiplications are needed. Thus in the worst case, after 10,935 multiplications and divisions Eve can uncover which message M_i produced the transmitted message R_i. In Chapter 2, Table 2.9, we assumed that a personal computer can perform 17,800 single-digit multiplications or divisions in a minute. Since multiplying two 4-digit numbers requires at most 16 single-digit multiplications (plus some additions), it will take Eve 10,935" 16/17,800 or about 10 minutes to recover the messages (once she's written the appropriate computer program on her PC). That's not so bad, but as we mentioned earlier, in this example we are really working with small numbers compared with those used in real life.

More generally, one block of a transmitted message may have $B < N$ decimal digits, not just 4. Let's figure out how long it will take Eve to decrypt a B-digit number if she tries all possibilities. If we allow the ASCII code for all characters, not just for capital letters, then the B-digit number will lie between O and $10^{B+1} - 1$. Suppose that the modulus N is roughly 10^{B+1} and so exponentiation by e, where 1< e < N, might use as many as $O(\log(N))$ multiplications and divisions. Thus a systematic search will involve

$$O(\log(10^{B+1})10^{B+1}) = O((B+1)10^{B+1})$$

operations, clearly an exponential amount of work for Eve.

Question 7.6. Suppose that the modulus is $N = 10^{B+1}$ and the B-digit numbers can be any number between O and N and also e = 11. Then using the figures from Examples 7.1 and 7.2, find the minimum value of B such that Eve must calculate for a month before she can figure out all possible messages.

There is an interesting sidelight to the above phenomenon. When we say that Eve has an exponential amount of work to do, in general, we are stating an empirical fact about the worst-case scenario. At present there is no theorem that says that Eve will need to examine all or even a large fraction of all numbers. Thus it is conceivable that a clever idea would enable Eve to break this encryption scheme with an efficient decryption scheme. There are other variations on this scheme with the same uncertainty, namely that there is no theorem that says decryption must be exponential in the worst case, and yet no one has determined how to "crack" these schemes with polynomial-time algorithms. This state of uncertainty has prompted a great deal of research on the mathematics behind public key encryption. The state of the art seems to be that encryptors have the upper hand at the moment; however, the decrypters have made some progress that has resulted in the encryptors having to work harder.

If Eve has such a difficult time decrypting the message, then how can Alice successfully decrypt the message? Remember that Alice calculated one additional piece of information about e and $N = pq$ before she destroyed or hid the values of p and q. Since e and $(p-l)(q-1)$ have gcd one, she used Corollaries 6.5 and 3.3 and secretly found the multiplicative inverse d of e modulo $(p-1)(q-1)$, that is,

$$ed \equiv 1 \pmod{(p-1)(q-1)} \qquad \text{with } O < d < (p-1)(q-1).$$

The pair (d, N) *is* called the decrypting key.

Example 7.1 and 7.2 (yet again). With e = 11 and $N = 9991$, we find the multiplicative inverse d using the Euclidean algorithm. In one part of Example 7.2 we checked that gcd (11, 9792) = 1 and we use these Euclidean equations as in Corollary 3.3:

$$\begin{aligned} 1 &= 11 - 5 \cdot 2 \\ &= 11 - 5 \cdot (9792 - 890 \cdot 11) \\ &= 4451 \cdot 11 - 5 9792. \end{aligned}$$

Thus 4451 is the multiplicative inverse of 11 modulo 9991.

Decryption now is easy for Alice because she knows a theorem that implies that

$$R_i^d \equiv M_i \pmod{N}$$

for $i = 1,2,. \ldots k$. Thus all she has to do is to calculate R_i^d, replace it by the least nonnegative residue modulo N and that's the message M_i. And again she pulls out a computer program that can quickly perform this exponentiation.

Examples 7.1 and 7.2 (concluded). The message 7269 was encrypted as 9425. Here is a summary of the calculation of $(9425)^{445}1$:

$$9425^{4451} = 9425^{4096} 9425^{256} 9425^{64} 9425^{32} 9425^{2} 9425.$$

With 12 multiplications we find

$$\begin{aligned}
9425^{2} &\equiv 644 \pmod{9991}\\
9425^{32} &\equiv 1975 \pmod{9991}\\
9425^{64} &\equiv 4135 \pmod{9991}\\
9425^{256} &\equiv 5202 \pmod{9991}\\
9425^{4096} &\text{-}1225 \pmod{9991}.
\end{aligned}$$

Then with 5 more multiplications we find

$$\begin{aligned}
9425^{4451}\text{-}1225 \cdot 5202.4135 \cdot 1975.644\text{”}9425 \pmod{9991}\\
\equiv 7269 \pmod{9991},
\end{aligned}$$

just as we claimed.

Question 7.7. Suppose that N = 15 = 3 · 5 and let e = 7. Find *d* such that $ed \equiv$ 1 (mod 2” 4). Then encrypt each of the messages 2 and 7 using the exponent e and then decrypt them using *d*.

Why does Alice's decryption scheme work? The reason is the following theorem, notice its similarity with Fermat's little theorem, Theorem 6.6. They are both cases of a more general result due to Euler; see Supplementary Exercises 23 and 24.

Theorem 7.1. If *p* and *q* are distinct primes, $n = pq$, and gcd (a, *n) = 1,* then

$$a^{(p-1)(q-1)} \equiv 1 \pmod{n}.$$

Why does this explain Alice's decryption procedure? Alice knows that

$$R_i \equiv M_i^e \pmod{N}.$$

Thus

$$\begin{aligned}
R_i^d &\equiv (M_i^e)^d \pmod{N}\\
&\equiv M_i^{ed} \pmod{N}\\
&\equiv M_i^{1+k(p-1)(q-1)} \pmod{N}
\end{aligned}$$

for some integer *k*, since $ed \equiv 1 \pmod{(p-1)(q-1)}$

$$\begin{aligned}
&\equiv M_i(M_i^{(p-1)(q-}\text{‘))’} \pmod{N}\\
&\equiv M_i 1^k \pmod{N}
\end{aligned}$$

by Theorem 7.1, since gcd $(M_i, N) = 1$

$$= M_i.$$

Remember that Bob checked that gcd $(M_i, N) = 1$ for $i = 1,2,\ldots, k$ and if not, announced the need for a change of modulus N and exponent e. Now we see that it is vital that M_i and N be relatively prime for the decryption scheme to work.

Proof of Theorem 7.1. (Notice the similarities between this proof and that of Theorem 6.6.) Let Z_n be the integers modulo n:

$$Z_n = \{[0], [1], [2], \ldots, [n-1]\}$$

and let A be the subset defined by

$$A = \{[x] \text{ in } Z_n : \gcd(x, n) = 1\}.$$

First we count the number of elements in A by specifying and counting the elements in $Z_n - A$. Now for any [i] in Z_n, gcd (i, n) is 1, p, q, or pg. The only element [x] for which gcd $(x, n) = pq$ is x = O. Which elements [x] have gcd $(x, n) = p$? Exactly $x = p, 2p, \ldots, (q-1)p$, and that's all since $n = pq$. Which ones have gcd $(x, n) = q$? Exactly $x = q, 2q, \ldots, (p-1)q$. Notice that no two of these numbers x are equal. For example, if $ip = jq$ with $1 \le i \le (q-1)$, $1 \le j \le (p-1)$, then p divides jq. Since p and q are distinct primes, p divides j, a contradiction since $j \le (p-1)$.

Thus the equivalence classes $[p], \ldots, [(q-1)p]$, $[q], \ldots, [(p-1)q]$ are all distinct, and so there are $1 + (q-1) + (p-1)$ elements in $Z_n - A$. Then A contains

$$n - 1 - (p-1) - (q-1) = pq - p - q + 1 = (p-1)(q-1)$$

elements. We list the elements of A as

$$A = \{[r_1], [r_2], ., [r_s]\},$$

where $0 < r_1 < r_2 < \cdots < r_s < n$, $s = (p-1)(q-1)$.

Lemma 7.2. If gcd (a, n) = gcd $(b, n) = 1$, then gcd $(ah, n) = 1$.

Proof (of lemma). We prove the contrapositive. Suppose that

$$\gcd(ab, n) = d > 1.$$

By Example 4.1, d has a prime divisor, say p. Thus p divides both n and ab and thus at least one of a and b. So either a or b share a common prime divisor with n contradicting our hypothesis. n

We now use this lemma. Take any number a such that gcd (a, n) = 1 and look at the equivalence classes

$$S = \{[ar_1], [ar_2], \ldots . [ar_s]\}.$$

By Lemma 7.2 $\gcd(ar_i, n) = 1$ for $i = 1, 2, \ldots, s$. In addition,

$$\gcd(r_1 r_2 \ldots r_s, n) = 1. \qquad (*)$$

We claim that the equivalence classes of S are the same as those of A, only perhaps listed in a different order. Since $\gcd(ar_i, n) = 1$, $[ar_i]$ *is* in A and so $[ar_i] = [r_j]$ for some value of j. Furthermore, no two of the classes in S are equal if

$$[ar_i] = [ar_k]$$

for some values of i and k with $r_i \# r_k$, then

$$\begin{aligned} ar_i &\equiv ar_k \pmod{n} \\ r_i &\equiv r_k \pmod{n} && \text{by Lemma 6.4} \\ r_i &= r_k && \text{since } r_i, r_k < n, \end{aligned}$$

a contradiction. Thus S = A, and

$$[r_1] \cdot [r_2] \cdot \cdots \cdot [r_s] = [ar_1] \cdot [ar_2] \cdot \cdots \cdot [ar_s]$$

since multiplication is well defined. Thus,

$$\begin{aligned} r_1 r_2 \cdots r_s &\equiv ar_1 ar_2 \cdots ar_s \pmod{n} && \text{by Lemma 6.2} \\ &\equiv a^s r_1 r_2 \cdots r_s \pmod{n} \\ 1 &\equiv a^s \pmod{n} && \text{by Lemma 6.4 and } (*) \\ &\equiv a^{(p-1)(q-1)} \pmod{n}. \end{aligned}$$

□

Why can't Eve find the decrypting pair *(d, N)?* Precisely because *d is* the multiplicative inverse of e modulo $(p-1)(q-1)$, and she doesn't know the values of p and q. As we saw before, factoring N to obtain p and q would require an exponential amount of work for her, unless she can think of something new and clever. Perhaps Eve's best bet is to study number theory and cryptography and to search for an efficient decrypting algorithm. However, she should be aware that

Alice could do the same. Alice might even someday come up with a provably secure system, that is, a system for which one can prove there is no polynomial-time algorithm to decrypt messages.

Question 7.8. For either R_2 or R_3, calculated in Question 7.5, check that $R_i^{4451} = M_i$.

We conclude with a summary of the steps needed in the RSA encryption and decryption scheme.

Algorithm RSA

STEP 1. (Numerical calculations by receiver)
- (a) Pick primes p and q, and let $N = pq$.
- (b) Find e such that $\gcd(e, (p-1)(q-1)) = 1$.
- (c) Find d such that $ed \equiv 1 \pmod{(p-1)(q-1)}$ with $0 < d < (p-1)(q-1)$.
- (d) Throw away p and q.
- *(e)* Announce N and e to the world.

STEP 2. (Encryption)
- (a) Translate the message into ASCII code using Table 4.1.
- (b) Pick an integer B less than the number of digits in N.
- (c) Break the ASCII coded message into blocks of B digits each; call these $M_1, M_2, \ldots, M_k$.
- (d) For $i = 1, 2, \ldots, k$ make sure that $\gcd(M_i, N) = 1$; if not, announce that the code is "broken" and return to step 1.
- (e) For $i = 1, 2, \ldots, k$ let $R_i \equiv M_i^e \pmod{N}$ with $0 < R_i < N$.
- (f) Transmit the encrypted messages $R_1, R_2, \ldots, R_k$.

STEP 3. (Decryption)
- (a) For $i = 1, 2, \ldots, k$ calculate $M_i \equiv R_i^d \pmod{N}$ with $0 < M_i < N$.
- (b) For $i = 1, 2, \ldots, k$ translate M_i from ASCII code using Table 4.1.

EXERCISES FOR SECTION 7

1. Using Table 4.1 give the ASCII code for the following: *(a)* RIGHT ON, ***(b)*** THE TRUTH, (c) ENCRYPT ME, and *(d)* FOREVER.

2. What do the following ASCII codes stand for in English?
(a) 7279 3272 8577.
(b) 7079 8287 6582 6832.
(c) 7879 3287 6589.
(d) 8479 3266 6932 7982 3278 7984 3284 7932 6669,

3. Determine which of the following are the product of two distinct primes: (a) 801, (*b*) 803, (c) 807, (d) 809, (e) 161, (*f*) 1631, and (g) 17,947.
4. For each of the following values of N = pq (from Question 7.3), find an integer e such that gcd (e, $(p-1)(q-1)$) = 1 and find the multiplicative inverse of e (*a*) 323, (*b*) 4087, and (c) 8633.
5. If N = 77, e = 7, and B = 4, explain why Bob cannot send the message PEACE to Alice.
6. Using blocks of four digits (B = 4), N = 8633 = 89.97, and e = 5 encrypt the message CHEERS.
7. Using N = 95 and e = 29, decrypt the message (with B = 2)

   53 29 02 51 29.

8. Let p be an odd prime and e an integer such that gcd (e, $p - 1$) = 1. Suppose that a message M *is* encrypted as C, where

$$C \equiv M^e \pmod{p} \qquad \text{where } 0 \leq C < p.$$

   If d *is* the multiplicative inverse of e modulo p, then prove that

$$C^d \equiv M \pmod{p}.$$

9. Show that the number of numbers i such that $0 \leq i < n = pq$ and gcd $(i, n) \neq 1$ is $q + p - 1$. Then deduce that the probability of picking such an i *is*

$$\frac{1}{p} + \frac{1}{q} - \frac{1}{pq}.$$

   If $p, q > 10^{30}$ then show that the probability of choosing, at random, an integer not relatively prime to n *is* less than 10^{-29}.
10. Write down in pseudocode an algorithm ENCRYPT that upon input of a message $M_1, M_2, \ldots, M_k$ and N and e, encrypts the message using the RSA scheme.
11. Write down in pseudocode an algorithm DECRYPT that upon input of a received message $R_1, R_2, \ldots, R_k$, two primes p and q (where N = pq) and the exponent e, decrypts this message.
12. Determine the number of multiplications and divisions performed in the worst case of ENCRYPT and DECRYPT. (You may count each multiplication and division as one, regardless of the number of digits.)
13. Prove the converse of Lemma 7.2.
14. Here is an algorithm to find divisors of an integer N.

Algorithm DIVISORSEARCH

```
STEP 1. Input $N$
STEP 2. For $i := 2$ to $\lfloor\sqrt{N}\rfloor$ do
        STEP 3. If $i$ divides N, then output "$i$ is a divisor of N"
STEP 4. If no divisors have been output, then output "N is a prime"
STEP 5. Stop.
```

Explain why this algorithm correctly determines when N is a prime. Explain why, if N is not a prime, this algorithm finds all, except possibly one, prime divisors of N.

15. Explain why, in the worst case, there is a number $r > 1$ such that the algorithm DIVISORSEARCH performs at least r^D divisions, where D equals the number of bits needed to express N in binary. Find as large a value of r as is possible with your argument.

16. Modify the algorithm DIVISORSEARCH so that its output includes all prime divisors of N. How many divisions does this perform in the worst case?

4:8 THE DIVIDENDS

The overall aim of this chapter has been to introduce the counting and algorithmic ideas of discrete mathematics within number theory. In addition, this chapter introduced specific results from number theory with indications of their applicability in mathematics and computer science.

The chapter has focused on algorithms to determine the greatest common divisor of two integers. In the text and exercises we found straightforward algorithms to solve the gcd problem and then developed the less obvious Euclidean algorithm. From the point of view of bit input, the straightforward algorithms are bad and exponential, but EUCLID is a good and linear algorithm. The worst-case complexity analysis of the latter algorithm is different from that of previous algorithms in that it comes in two stages. First we show that if we use the Euclidean algorithm on two successive Fibonacci numbers, then the number of multiplications and divisions is logarithmic in the input numbers. Next we show that in the worst case of the Euclidean algorithm with arbitrary input $b \leq c$, $O(\log(c))$ operations are performed. Thus the Fibonacci numbers exhibit this worst-case behavior and so the worst-case analysis really does reflect what may happen. What is also true, but we do not prove it, is that the Fibonacci numbers are actually the worst-case input for the Euclidean algorithm. In conclusion, we observe that $O(\log(c)) = O(B)$, where B is the number of bits needed for the input.

Two important general ideas were introduced in this chapter. The first is Complete Induction, which gives us more flexibility at the cost of more checking of base cases. Also we presented the idea that the size of the input to an algorithm ought to be measured in terms of bits. Thus an integer n requires B [roughly log (n)] bits, and it is in terms of this parameter B that we should be determining and analyzing the complexity functions of algorithms. With this perspective we look back to EXPONENT and FASTEXP of Chapter 2 and see that they are exponential and linear algorithms, respectively.

A substantial amount of elementary number theory appears in Section 6. Modular arithmetic and equivalence relations are central to much of mathematics and computer science. For example, the theory of groups and rings involves generalizations of Z_n, the integers modulo n. Many computer languages come with the ability to do arithmetic modulo n; this arithmetic is important in, for instance, random number generation. Equivalence relations will be crucial in further courses in theoretical computer science and mathematics. Thus the lemmas, theorems, and corollaries of Section 6 are worth studying because they will come up again both in applications and in other branches of mathematics and computer science.

There is a variety of different encryption schemes in use today; each uses different aspects of number theory. The approach we pursue relies on the Euclidean algorithm and Fermat's little theorem, but its effectiveness comes from the fact that it is apparently difficult to factor a number into its prime factors. In fact, it has recently been shown that the difficulty of "cracking" a variation of the RSA scheme is computationally equivalent to factoring a number n into two primes. However, there is no known theorem that says it is hard to decrypt a message sent using the RSA scheme or that it is hard to factor a number. A closely related algorithmic problem is that of determining whether a given number is prime. Recent fast, so-called random algorithms have been developed that can test whether "most" numbers are prime, and there is a primality-testing algorithm that has been shown to run in polynomial time on all integers, provided that a famous open problem, the extended Riemann hypothesis, is true. No one has proved the latter result, but most mathematicians believe it is true. Thus if you are tempted to set up an encryption service along the lines of this chapter, take heed. It may be that soon a mathematical or algorithmic breakthrough will occur and destroy the effectiveness of the RSA encryption scheme.

Number theory is an excellent training ground for logical analysis and deduction. It is accessible: Small examples can be explored numerically, general patterns deduced, and proofs constructed by induction and contradiction. The Fibonacci numbers are a sample of the kinds of intriguing problems in the field. Others include prime numbers, modular arithmetic, and solutions of equations. Number theory also gives an introduction to the mathematical discipline of abstract algebra and the computer science discipline of arithmetic and algebraic computations. Especially if the ideas in this chapter interest you, these are fields worthy of further study.

SUPPLEMENTARY EXERCISES FOR CHAPTER 4

1. Design a gcd algorithm called GCD2 that is based on the following idea. If 2 divides b and c, then 2 is a factor of gcd (b, c). Furthermore, we may carry out the division and consider the smaller problem of finding the gcd *(b/2*, c/2). If 2 does not divide b or c, try 3, ... try j. Note that the maximum value of j that you need to check is no more than b or $\sqrt{c}$. Why?
2. Use GCD2 to find the gcd of the following pairs: *(a)* (8, 12), **(*b*)** (24, 32), and (c) (72, 96).
3. In the worst case how many divisions will GCD2 need?
4. Design an algorithm that upon input m will find all prime numbers between 1 and m. (*Hint:* Use the idea behind GCD2. This idea is attributed to the Greek mathematician Eratosthenes of the third century B.C. The method is known as the Sieve of Eratosthenes: First cross out all multiples of 2 except for 2 itself. Next cross out all multiples of 3 except for 3 itself, ... and so on. How far do you have to keep going with this crossing out process?)
5. Use your Sieve of Eratosthenes algorithm to find all prime numbers between 1 and 200.
6. Let U be the set of positive integers less than 49. Set $A = \{x \in U: x \text{ is divisible by } 2\}$, $B = \{x \in U: x \text{ is divisible by } 3\}$, and $C = \{x \in U: x \text{ is divisible by } 5\}$. Find $|A|, |B|$, and $|C|$. Find $|A \cup B \cup C|$. (*Hint:* Look at PIE from Chapter 1.) Use the results of this problem to calculate the number of primes less than 50.
7. Two couples are camping in Hawaii with a pet parrot. They collect a pile of macadamia nuts, but during the night one woman gets up, divides the pile of nuts into four equal piles and finds one nut left over, which she gives to the parrot to keep it quiet. She hides one pile, combines the other three piles into one, and goes back to sleep. Then her husband wakes up, looks suspiciously at the pile, divides the (remaining) nuts equally into four with one extra nut for the parrot, hides one pile, and goes back to sleep. The same thing happens two more times; each time the remaining nuts divide evenly into four equal piles with one nut left over, which is given to the parrot, and one pile is hidden. In the morning the four graciously divide the remaining nuts into four equal piles and find they have one macadamia nut left over for the parrot. What is the minimum number of nuts that they could have had at the start of the evening'?
8. Look back at the definition of $IC(b,c)$ in Exercise 3.13. Prove that $gcd\,(b, c) = \min\,\{IC(b, c)\}$.
9. Design an algorithm EXTENDEDEUCLID that first finds the gcd of b and c as in the algorithm EUCLID and then expresses the gcd as a linear combination of b and c. The algorithm should use only a constant number of variables, say 10 at most.

10. *(a)* Suppose that n is even. Then the following sum equals a Fibonacci number. Which one is it?

$$\binom{n}{0}+\binom{n-1}{1}+\binom{n-2}{2}+\binom{n-3}{3}+\cdots+\binom{n/2}{n/2}.$$

(b) Find a similar sum of binomial coefficients that equals a Fibonacci number when n *is* odd.

11. Prove the results you obtained in Exercise 10. *Hint:* Use induction and the fact that

$$\binom{n-k}{k}=\binom{(n-1)-k}{k}+\binom{(n-2)-(k-1)}{k-1}.$$

12. If the Euclidean algorithm is applied to c = F_{k+4} and $b = F_{k+1}$, what can you say about the number of Euclidean equations?

13. Suppose that the algorithm EUCLID is modified so that in step 3 the variable q is set equal to the nearest integer to *c/b.* Run some examples of this algorithm, including some Fibonacci numbers. Then analyze the complexity of the algorithm in terms of c. Is this version more efficient than EUCLID?

14. Prove Lamé's Theorem: In the Euclidean algorithm the smallest values of c that produces k Euclidean equations is c = F_{k+2}.

15. Prove that if as $\equiv$ 1 (mod n) and $a^t \equiv$ 1 (mod n), then $a^{\gcd(s,t)} \equiv$ 1 (mod *n).*

16. For ordinary integers, xy = O if and only if either x or y equals O. Give examples to show that this is false in Z_n, that is, for equivalence classes modulo n it is not true that

[x] [y]= [0] if and only if [x] = [0] or [Y] = [0].

17. Prove the following about equivalence classes modulo n. Given [x] # [0], there is a [y] # [0] such that [x] · [y] = [0] if and only if gcd (x, *n)* # 1.

18. Consider Z_n, the integers modulo n, where $n = s\,t$ with gcd (s, t) = 1. Show that there are at least four different equivalence classes *[i]* modulo n such that $[i]^2$ = [i]. For example, $[0]^2 = [0^2] = [0]$.

19. If $n = pq$, then explain why the following are true:

$$p+q=n-(p-1)(q-1)+1$$
$$p-q=\sqrt{(p+q)^2-4n}.$$

Suppose that an algorithm were discovered that given an integer $n = pq$, a product of two primes, could quickly calculate $(p-1)(q-1)$. Then use the facts that

$$p = \tfrac{1}{2}((p+q)+(p-q))$$
$$q = \tfrac{1}{2}((p+q)-(p-q))$$

and the results of the previous equations to argue that there would be a fast algorithm to factor n into its two prime divisors.

20. We define $\phi(m)$ to be the number of integers i in $\{1, 2, \ldots, m\}$ such that $\gcd(i, m) = 1$. Determine $\phi(6), \phi(7), \phi(9), \phi(10), \phi(p)$, and $\phi(p^2)$, where p is a prime, and $\phi(pq)$, where p and q are distinct primes.

21. Prove Wilson's theorem: If p is a prime, then p divides $((p-1)!+1)$. [*Hint:* Show that $(p-2)! \equiv 1 \pmod{p}$ by pairing numbers with their multiplicative inverses.]

22. Here are some ideas for an alternative proof of Fermat's little theorem, which states that $b^{p-1} \equiv 1 \pmod{p}$ if $\gcd(b, p) = 1$. First show that if strings of beads of length p are formed using b different colors of beads, then the number of such strings that are not all one color is $b^p - b$. (You should assume that there is an unlimited supply of beads of each color.) If the ends of each string are tied together to form a bracelet, explain why the number of different colored bracelets is $(b^p - b)/p$. (For example, the string of red, blue, and green beads forms the same bracelet as the string of blue, green, and red beads.)

23. A theorem due to Euler states that if $\gcd(u, m) = 1$, then

$$s^{o(m)} \equiv 1 \pmod{m},$$

where the function ϕ is defined in Exercise 20. Verify that this theorem is true for m a prime or a product of two primes.

24. Prove Euler's theorem (of Exercise 23) using the following hints: Let $A = \{[s_1], [s_2], \ldots, [s_{\phi(m)}]\}$, where the s_i are all the integers in $\{1, 2, \ldots, m\}$ that are relatively prime to m. Let $S = \{[as_1], [as_2], \ldots, [as_{\phi(m)}]\}$. *Then proceed* as in the proof of Theorem 7.1.