

---

## 8 Number Theory

*Number theory* is the study of the integers. *Why* anyone would want to study the integers may not be obvious. First of all, what’s to know? There’s 0, there’s 1, 2, 3, and so on, and, oh yeah, -1, -2, . . . . Which one don’t you understand? What practical value is there in it?

The mathematician G. H. Hardy delighted at its impracticality. He wrote:

[Number theorists] may be justified in rejoicing that there is one science, at any rate, and that their own, whose very remoteness from ordinary human activities should keep it gentle and clean.

Hardy was especially concerned that number theory not be used in warfare; he was a pacifist. You may applaud his sentiments, but he got it wrong: number theory underlies modern cryptography, which is what makes secure online communication possible. Secure communication is of course crucial in war—leaving poor Hardy spinning in his grave. It’s also central to online commerce. Every time you buy a book from Amazon, use a certificate to access a web page, or use a PayPal account, you are relying on number theoretic algorithms.

Number theory also provides an excellent environment for us to practice and apply the proof techniques that we developed in previous chapters. We’ll work out properties of greatest common divisors (gcd’s) and use them to prove that integers factor uniquely into primes. Then we’ll introduce modular arithmetic and work out enough of its properties to explain the RSA public key crypto-system.

Since we’ll be focusing on properties of the integers, we’ll adopt the default convention in this chapter that *variables range over the set,  $\mathbb{Z}$ , of integers*.

---

### 8.1 Divisibility

The nature of number theory emerges as soon as we consider the *divides* relation.

**Definition 8.1.1.** *a divides b* (notation  $a \mid b$ ) iff there is an integer  $k$  such that

$$ak = b.$$

The divides relation comes up so frequently that multiple synonyms for it are used all the time. The following phrases all say the same thing:

- $a \mid b$ ,
- $a$  divides  $b$ ,
- $a$  is a *divisor* of  $b$ ,
- $a$  is a *factor* of  $b$ ,
- $b$  is *divisible* by  $a$ ,
- $b$  is a *multiple* of  $a$ .

Some immediate consequences of Definition 8.1.1 are that for all  $n$

$$n \mid 0, \quad n \mid n, \quad \text{and} \quad \pm 1 \mid n.$$

Also,

$$0 \mid n \text{ IMPLIES } n = 0.$$

Dividing seems simple enough, but let’s play with this definition. The Pythagoreans, an ancient sect of mathematical mystics, said that a number is *perfect* if it equals the sum of its positive integral divisors, excluding itself. For example,  $6 = 1 + 2 + 3$  and  $28 = 1 + 2 + 4 + 7 + 14$  are perfect numbers. On the other hand, 10 is not perfect because  $1 + 2 + 5 = 8$ , and 12 is not perfect because  $1 + 2 + 3 + 4 + 6 = 16$ . Euclid characterized all the *even* perfect numbers around 300 BC (Problem 8.2). But is there an *odd* perfect number? More than two thousand years later, we still don’t know! All numbers up to about  $10^{300}$  have been ruled out, but no one has proved that there isn’t an odd perfect number waiting just over the horizon.

So a half-page into number theory, we’ve strayed past the outer limits of human knowledge. This is pretty typical; number theory is full of questions that are easy to pose, but incredibly difficult to answer. We’ll mention a few more such questions in later sections.<sup>1</sup>

### 8.1.1 Facts about Divisibility

The following lemma collects some basic facts about divisibility.

#### Lemma 8.1.2.

1. If  $a \mid b$  and  $b \mid c$ , then  $a \mid c$ .

---

<sup>1</sup>*Don’t Panic*—we’re going to stick to some relatively benign parts of number theory. These super-hard unsolved problems rarely get put on problem sets.

2. If  $a \mid b$  and  $a \mid c$ , then  $a \mid sb + tc$  for all  $s$  and  $t$ .
3. For all  $c \neq 0$ ,  $a \mid b$  if and only if  $ca \mid cb$ .

*Proof.* These facts all follow directly from Definition 8.1.1. To illustrate this, we’ll prove just part 2:

Given that  $a \mid b$ , there is some  $k_1 \in \mathbb{Z}$  such that  $ak_1 = b$ . Likewise,  $ak_2 = c$ , so

$$sb + tc = s(k_1a) + t(k_2a) = (sk_1 + tk_2)a.$$

Therefore  $sb + tc = k_3a$  where  $k_3 ::= (sk_1 + tk_2)$ , which means that

$$a \mid sb + tc.$$

■

A number of the form  $sb + tc$  is called an *integer linear combination* of  $b$  and  $c$ , or, since in this chapter we’re only talking about integers, just a *linear combination*. So Lemma 8.1.2.2 can be rephrased as

If  $a$  divides  $b$  and  $c$ , then  $a$  divides every linear combination of  $b$  and  $c$ .

We’ll be making good use of linear combinations, so let’s get the general definition on record:

**Definition 8.1.3.** An integer  $n$  is a *linear combination* of numbers  $b_0, \dots, b_k$  iff

$$n = s_0b_0 + s_1b_1 + \dots + s_kb_k$$

for some integers  $s_0, \dots, s_k$ .

### 8.1.2 When Divisibility Goes Bad

As you learned in elementary school, if one number does *not* evenly divide another, you get a “quotient” and a “remainder” left over. More precisely:

**Theorem 8.1.4.** [Division Theorem]<sup>2</sup> Let  $n$  and  $d$  be integers such that  $d > 0$ . Then there exists a unique pair of integers  $q$  and  $r$ , such that

$$n = q \cdot d + r \text{ AND } 0 \leq r < d. \tag{8.1}$$

<sup>2</sup>This theorem is often called the “Division Algorithm,” but we prefer to call it a theorem since it does not actually describe a division procedure for computing the quotient and remainder.

The number  $q$  is called the *quotient* and the number  $r$  is called the *remainder* of  $n$  divided by  $d$ . We use the notation  $\text{qcnt}(n, d)$  for the quotient and  $\text{rem}(n, d)$  for the remainder. For example,  $\text{qcnt}(2716, 10) = 271$  and  $\text{rem}(2716, 10) = 6$ , since  $2716 = 271 \cdot 10 + 6$ . Similarly,  $\text{rem}(-11, 7) = 3$ , since  $-11 = (-2) \cdot 7 + 3$ .

There is a remainder operator built into many programming languages. For example, “32 % 5” will be familiar as remainder notation to programmers in Java, C, and C++; it evaluates to  $\text{rem}(32, 5) = 2$  in all three languages. On the other hand, these and other languages treat remainders involving negative numbers inconsistently, so don’t be distracted by your programming language’s behavior, and remember to stick to the definition according to the Division Theorem 8.1.4.

The remainder on division by  $n$  is a number in the (integer) *interval* from 0 to  $n - 1$ . Such intervals come up so often that it is useful to have a simple notation for them.

$$\begin{aligned} (k..n) &::= \{i \mid k < i < n\}, \\ (k..n] &::= (k, n) \cup \{n\}, \\ [k..n) &::= \{k\} \cup (k, n), \\ [k..n] &::= \{k\} \cup (k, n) \cup \{n\} = \{i \mid k \leq i \leq n\}. \end{aligned}$$

### 8.1.3 Die Hard

*Die Hard 3* is just a B-grade action movie, but we think it has an inner message: everyone should learn at least a little number theory. In Section 5.4.4, we formalized a state machine for the Die Hard jug-filling problem using 3 and 5 gallon jugs, and also with 3 and 9 gallon jugs, and came to different conclusions about bomb explosions. What’s going on in general? For example, how about getting 4 gallons from 12- and 18-gallon jugs, getting 32 gallons with 899- and 1147-gallon jugs, or getting 3 gallons into a jug using just 21- and 26-gallon jugs?

It would be nice if we could solve all these silly water jug questions at once. This is where number theory comes in handy.

#### A Water Jug Invariant

Suppose that we have water jugs with capacities  $a$  and  $b$  with  $b \geq a$ . Let’s carry out some sample operations of the state machine and see what happens, assuming

the  $b$ -jug is big enough:

$(0, 0) \rightarrow (a, 0)$	fill first jug
$\rightarrow (0, a)$	pour first into second
$\rightarrow (a, a)$	fill first jug
$\rightarrow (2a - b, b)$	pour first into second (assuming $2a \geq b$ )
$\rightarrow (2a - b, 0)$	empty second jug
$\rightarrow (0, 2a - b)$	pour first into second
$\rightarrow (a, 2a - b)$	fill first
$\rightarrow (3a - 2b, b)$	pour first into second (assuming $3a \geq 2b$ )

What leaps out is that at every step, the amount of water in each jug is a linear combination of  $a$  and  $b$ . This is easy to prove by induction on the number of transitions:

**Lemma 8.1.5 (Water Jugs).** *In the Die Hard state machine of Section 5.4.4 with jugs of sizes  $a$  and  $b$ , the amount of water in each jug is always a linear combination of  $a$  and  $b$ .*

*Proof.* The induction hypothesis,  $P(n)$ , is the proposition that after  $n$  transitions, the amount of water in each jug is a linear combination of  $a$  and  $b$ .

**Base case ( $n = 0$ ):**  $P(0)$  is true, because both jugs are initially empty, and  $0 \cdot a + 0 \cdot b = 0$ .

**Inductive step:** Suppose the machine is in state  $(x, y)$  after  $n$  steps, that is, the little jug contains  $x$  gallons and the big one contains  $y$  gallons. There are two cases:

- If we fill a jug from the fountain or empty a jug into the fountain, then that jug is empty or full. The amount in the other jug remains a linear combination of  $a$  and  $b$ . So  $P(n + 1)$  holds.
- Otherwise, we pour water from one jug to another until one is empty or the other is full. By our assumption, the amount  $x$  and  $y$  in each jug is a linear combination of  $a$  and  $b$  before we begin pouring. After pouring, one jug is either empty (contains 0 gallons) or full (contains  $a$  or  $b$  gallons). Thus, the other jug contains either  $x + y$  gallons,  $x + y - a$ , or  $x + y - b$  gallons, all of which are linear combinations of  $a$  and  $b$  since  $x$  and  $y$  are. So  $P(n + 1)$  holds in this case as well.

Since  $P(n + 1)$  holds in any case, this proves the inductive step, completing the proof by induction. ■

So we have established that the jug problem has a preserved invariant, namely, the amount of water in every jug is a linear combination of the capacities of the jugs. Lemma 8.1.5 has an important corollary:

**Corollary.** *In trying to get 4 gallons from 12- and 18-gallon jugs, and likewise to get 32 gallons from 899- and 1147-gallon jugs,*

***Bruce will die!***

*Proof.* By the Water Jugs Lemma 8.1.5, with 12- and 18-gallon jugs, the amount in any jug is a linear combination of 12 and 18. This is always a multiple of 6 by Lemma 8.1.2.2, so Bruce can’t get 4 gallons. Likewise, the amount in any jug using 899- and 1147-gallon jugs is a multiple of 31, so he can’t get 32 either. ■

But the Water Jugs Lemma doesn’t tell the complete story. For example, it leaves open the question of getting 3 gallons into a jug using just 21- and 26-gallon jugs: the only positive factor of both 21 and 26 is 1, and of course 1 divides 3, so the Lemma neither rules out nor confirms the possibility of getting 3 gallons.

A bigger issue is that we’ve just managed to recast a pretty understandable question about water jugs into a technical question about linear combinations. This might not seem like a lot of progress. Fortunately, linear combinations are closely related to something more familiar, greatest common divisors, and will help us solve the general water jug problem.

---

## 8.2 The Greatest Common Divisor

A *common divisor* of  $a$  and  $b$  is a number that divides them both. The *greatest common divisor* of  $a$  and  $b$  is written  $\gcd(a, b)$ . For example,  $\gcd(18, 24) = 6$ .

As long as  $a$  and  $b$  are not both 0, they will have a gcd. The gcd turns out to be very valuable for reasoning about the relationship between  $a$  and  $b$  and for reasoning about integers in general. We’ll be making lots of use of gcd’s in what follows.

Some immediate consequences of the definition of gcd are that for  $n > 0$ ,

$$\gcd(n, n) = n, \quad \gcd(n, 1) = 1, \quad \gcd(n, 0) = n,$$

where the last equality follows from the fact that everything is a divisor of 0.

### 8.2.1 Euclid’s Algorithm

The first thing to figure out is how to find gcd’s. A good way called *Euclid’s algorithm* has been known for several thousand years. It is based on the following elementary observation.

**Lemma 8.2.1.** For  $b \neq 0$ ,

$$\gcd(a, b) = \gcd(b, \text{rem}(a, b)).$$

*Proof.* By the Division Theorem 8.1.4,

$$a = qb + r \tag{8.2}$$

where  $r = \text{rem}(a, b)$ . So  $a$  is a linear combination of  $b$  and  $r$ , which implies that any divisor of  $b$  and  $r$  is a divisor of  $a$  by Lemma 8.1.2.2. Likewise,  $r$  is a linear combination,  $a - qb$ , of  $a$  and  $b$ , so any divisor of  $a$  and  $b$  is a divisor of  $r$ . This means that  $a$  and  $b$  have the same common divisors as  $b$  and  $r$ , and so they have the same *greatest* common divisor. ■

Lemma 8.2.1 is useful for quickly computing the greatest common divisor of two numbers. For example, we could compute the greatest common divisor of 1147 and 899 by repeatedly applying it:

$$\begin{aligned} \gcd(1147, 899) &= \gcd(899, \underbrace{\text{rem}(1147, 899)}_{=248}) \\ &= \gcd(248, \text{rem}(899, 248) = 155) \\ &= \gcd(155, \text{rem}(248, 155) = 93) \\ &= \gcd(93, \text{rem}(155, 93) = 62) \\ &= \gcd(62, \text{rem}(93, 62) = 31) \\ &= \gcd(31, \text{rem}(62, 31) = 0) \\ &= 31 \end{aligned}$$

This calculation that  $\gcd(1147, 899) = 31$  was how we figured out that with water jugs of sizes 1147 and 899, Bruce dies trying to get 32 gallons.

On the other hand, applying Euclid’s algorithm to 26 and 21 gives

$$\gcd(26, 21) = \gcd(21, 5) = \gcd(5, 1) = 1,$$

so we can’t use the reasoning above to rule out Bruce getting 3 gallons into the big jug. As a matter of fact, because the gcd here is 1, Bruce *will* be able to get any number of gallons into the big jug up to its capacity. To explain this, we will need a little more number theory.

### Euclid’s Algorithm as a State Machine

Euclid’s algorithm can easily be formalized as a state machine. The set of states is  $\mathbb{N}^2$  and there is one transition rule:

$$(x, y) \longrightarrow (y, \text{rem}(x, y)), \tag{8.3}$$

for  $y > 0$ . By Lemma 8.2.1, the gcd stays the same from one state to the next. That means the predicate

$$\text{gcd}(x, y) = \text{gcd}(a, b)$$

is a preserved invariant on the states  $(x, y)$ . This preserved invariant is, of course, true in the start state  $(a, b)$ . So by the Invariant Principle, if  $y$  ever becomes 0, the invariant will be true and so

$$x = \text{gcd}(x, 0) = \text{gcd}(a, b).$$

Namely, the value of  $x$  will be the desired gcd.

What’s more,  $x$ , and therefore also  $y$ , gets to be 0 pretty fast. To see why, note that starting from  $(x, y)$ , two transitions leads to a state whose the first coordinate is  $\text{rem}(x, y)$ , which is at most half the size of  $x$ .<sup>3</sup> Since  $x$  starts off equal to  $a$  and gets halved or smaller every two steps, it will reach its minimum value—which is  $\text{gcd}(a, b)$ —after at most  $2 \log a$  transitions. After that, the algorithm takes at most one more transition to terminate. In other words, Euclid’s algorithm terminates after at most  $1 + 2 \log a$  transitions.<sup>4</sup>

### 8.2.2 The Pulverizer

We will get a lot of mileage out of the following key fact:

**Theorem 8.2.2.** *The greatest common divisor of  $a$  and  $b$  is a linear combination of  $a$  and  $b$ . That is,*

$$\text{gcd}(a, b) = sa + tb,$$

for some integers  $s$  and  $t$ .

We already know from Lemma 8.1.2.2 that every linear combination of  $a$  and  $b$  is divisible by any common factor of  $a$  and  $b$ , so it is certainly divisible by the greatest

---

<sup>3</sup>In other words,

$$\text{rem}(x, y) \leq x/2 \quad \text{for } 0 < y \leq x. \tag{8.4}$$

This is immediate if  $y \leq x/2$ , since the remainder of  $x$  divided by  $y$  is less than  $y$  by definition. On the other hand, if  $y > x/2$ , then  $\text{rem}(x, y) = x - y < x/2$ .

<sup>4</sup>A tighter analysis shows that at most  $\log_\varphi(a)$  transitions are possible where  $\varphi$  is the golden ratio  $(1 + \sqrt{5})/2$ , see Problem 8.14.

of these common divisors. Since any constant multiple of a linear combination is also a linear combination, Theorem 8.2.2 implies that any multiple of the gcd is a linear combination, giving:

**Corollary 8.2.3.** *An integer is a linear combination of  $a$  and  $b$  iff it is a multiple of  $\gcd(a, b)$ .*

We’ll prove Theorem 8.2.2 directly by explaining how to find  $s$  and  $t$ . This job is tackled by a mathematical tool that dates back to sixth-century India, where it was called *kuttak*, which means “The Pulverizer.” Today, the Pulverizer is more commonly known as “the extended Euclidean gcd algorithm,” because it is so close to Euclid’s algorithm.

For example, following Euclid’s algorithm, we can compute the gcd of 259 and 70 as follows:

$$\begin{aligned} \gcd(259, 70) &= \gcd(70, 49) && \text{since } \text{rem}(259, 70) = 49 \\ &= \gcd(49, 21) && \text{since } \text{rem}(70, 49) = 21 \\ &= \gcd(21, 7) && \text{since } \text{rem}(49, 21) = 7 \\ &= \gcd(7, 0) && \text{since } \text{rem}(21, 7) = 0 \\ &= 7. \end{aligned}$$

The Pulverizer goes through the same steps, but requires some extra bookkeeping along the way: as we compute  $\gcd(a, b)$ , we keep track of how to write each of the remainders (49, 21, and 7, in the example) as a linear combination of  $a$  and  $b$ . This is worthwhile, because our objective is to write the last nonzero remainder, which is the GCD, as such a linear combination. For our example, here is this extra bookkeeping:

$x$	$y$	$(\text{rem}(x, y))$	$= x - q \cdot y$
259	70	49	$= a - 3 \cdot b$
70	49	21	$= b - 1 \cdot 49$ $= b - 1 \cdot (a - 3 \cdot b)$ $= -1 \cdot a + 4 \cdot b$
49	21	7	$= 49 - 2 \cdot 21$ $= (a - 3 \cdot b) - 2 \cdot (-1 \cdot a + 4 \cdot b)$ $= \boxed{3 \cdot a - 11 \cdot b}$
21	7	0	

We began by initializing two variables,  $x = a$  and  $y = b$ . In the first two columns above, we carried out Euclid’s algorithm. At each step, we computed  $\text{rem}(x, y)$  which equals  $x - \text{qcnt}(x, y) \cdot y$ . Then, in this linear combination of  $x$  and  $y$ , we

replaced  $x$  and  $y$  by equivalent linear combinations of  $a$  and  $b$ , which we already had computed. After simplifying, we were left with a linear combination of  $a$  and  $b$  equal to  $\text{rem}(x, y)$ , as desired. The final solution is boxed.

This should make it pretty clear how and why the Pulverizer works. If you have doubts, it may help to work through Problem 8.13, where the Pulverizer is formalized as a state machine and then verified using an invariant that is an extension of the one used for Euclid’s algorithm.

Since the Pulverizer requires only a little more computation than Euclid’s algorithm, you can “pulverize” very large numbers very quickly by using this algorithm. As we will soon see, its speed makes the Pulverizer a very useful tool in the field of cryptography.

Now we can restate the Water Jugs Lemma 8.1.5 in terms of the greatest common divisor:

**Corollary 8.2.4.** *Suppose that we have water jugs with capacities  $a$  and  $b$ . Then the amount of water in each jug is always a multiple of  $\text{gcd}(a, b)$ .*

For example, there is no way to form 4 gallons using 3- and 6-gallon jugs, because 4 is not a multiple of  $\text{gcd}(3, 6) = 3$ .

### 8.2.3 One Solution for All Water Jug Problems

Corollary 8.2.3 says that 3 can be written as a linear combination of 21 and 26, since 3 is a multiple of  $\text{gcd}(21, 26) = 1$ . So the Pulverizer will give us integers  $s$  and  $t$  such that

$$3 = s \cdot 21 + t \cdot 26 \tag{8.5}$$

The coefficient  $s$  could be either positive or negative. However, we can readily transform this linear combination into an equivalent linear combination

$$3 = s' \cdot 21 + t' \cdot 26 \tag{8.6}$$

where the coefficient  $s'$  is positive. The trick is to notice that if in equation (8.5) we increase  $s$  by 26 and decrease  $t$  by 21, then the value of the expression  $s \cdot 21 + t \cdot 26$  is unchanged overall. Thus, by repeatedly increasing the value of  $s$  (by 26 at a time) and decreasing the value of  $t$  (by 21 at a time), we get a linear combination  $s' \cdot 21 + t' \cdot 26 = 3$  where the coefficient  $s'$  is positive. (Of course  $t'$  must then be negative; otherwise, this expression would be much greater than 3.)

Now we can form 3 gallons using jugs with capacities 21 and 26: We simply repeat the following steps  $s'$  times:

1. Fill the 21-gallon jug.

2. Pour all the water in the 21-gallon jug into the 26-gallon jug. If at any time the 26-gallon jug becomes full, empty it out, and continue pouring the 21-gallon jug into the 26-gallon jug.

At the end of this process, we must have emptied the 26-gallon jug exactly  $-t'$  times. Here's why: we've taken  $s' \cdot 21$  gallons of water from the fountain, and we've poured out some multiple of 26 gallons. If we emptied fewer than  $-t'$  times, then by (8.6), the big jug would be left with at least  $3 + 26$  gallons, which is more than it can hold; if we emptied it more times, the big jug would be left containing at most  $3 - 26$  gallons, which is nonsense. But once we have emptied the 26-gallon jug exactly  $-t'$  times, equation (8.6) implies that there are exactly 3 gallons left.

Remarkably, we don't even need to know the coefficients  $s'$  and  $t'$  in order to use this strategy! Instead of repeating the outer loop  $s'$  times, we could just repeat *until we obtain 3 gallons*, since that must happen eventually. Of course, we have to keep track of the amounts in the two jugs so we know when we're done. Here's the solution using this approach starting with empty jugs, that is, at  $(0, 0)$ :

$\xrightarrow{\text{fill 21}}$	$(21, 0)$	$\xrightarrow{\text{pour 21 into 26}}$				$(0, 21)$	
$\xrightarrow{\text{fill 21}}$	$(21, 21)$	$\xrightarrow{\text{pour 21 to 26}}$	$(16, 26)$	$\xrightarrow{\text{empty 26}}$	$(16, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 16)$
$\xrightarrow{\text{fill 21}}$	$(21, 16)$	$\xrightarrow{\text{pour 21 to 26}}$	$(11, 26)$	$\xrightarrow{\text{empty 26}}$	$(11, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 11)$
$\xrightarrow{\text{fill 21}}$	$(21, 11)$	$\xrightarrow{\text{pour 21 to 26}}$	$(6, 26)$	$\xrightarrow{\text{empty 26}}$	$(6, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 6)$
$\xrightarrow{\text{fill 21}}$	$(21, 6)$	$\xrightarrow{\text{pour 21 to 26}}$	$(1, 26)$	$\xrightarrow{\text{empty 26}}$	$(1, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 1)$
$\xrightarrow{\text{fill 21}}$	$(21, 1)$	$\xrightarrow{\text{pour 21 to 26}}$				$(0, 22)$	
$\xrightarrow{\text{fill 21}}$	$(21, 22)$	$\xrightarrow{\text{pour 21 to 26}}$	$(17, 26)$	$\xrightarrow{\text{empty 26}}$	$(17, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 17)$
$\xrightarrow{\text{fill 21}}$	$(21, 17)$	$\xrightarrow{\text{pour 21 to 26}}$	$(12, 26)$	$\xrightarrow{\text{empty 26}}$	$(12, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 12)$
$\xrightarrow{\text{fill 21}}$	$(21, 12)$	$\xrightarrow{\text{pour 21 to 26}}$	$(7, 26)$	$\xrightarrow{\text{empty 26}}$	$(7, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 7)$
$\xrightarrow{\text{fill 21}}$	$(21, 7)$	$\xrightarrow{\text{pour 21 to 26}}$	$(2, 26)$	$\xrightarrow{\text{empty 26}}$	$(2, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 2)$
$\xrightarrow{\text{fill 21}}$	$(21, 2)$	$\xrightarrow{\text{pour 21 to 26}}$				$(0, 23)$	
$\xrightarrow{\text{fill 21}}$	$(21, 23)$	$\xrightarrow{\text{pour 21 to 26}}$	$(18, 26)$	$\xrightarrow{\text{empty 26}}$	$(18, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 18)$
$\xrightarrow{\text{fill 21}}$	$(21, 18)$	$\xrightarrow{\text{pour 21 to 26}}$	$(13, 26)$	$\xrightarrow{\text{empty 26}}$	$(13, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 13)$
$\xrightarrow{\text{fill 21}}$	$(21, 13)$	$\xrightarrow{\text{pour 21 to 26}}$	$(8, 26)$	$\xrightarrow{\text{empty 26}}$	$(8, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 8)$
$\xrightarrow{\text{fill 21}}$	$(21, 8)$	$\xrightarrow{\text{pour 21 to 26}}$	$(3, 26)$	$\xrightarrow{\text{empty 26}}$	$(3, 0)$	$\xrightarrow{\text{pour 21 to 26}}$	$(0, 3)$

The same approach works regardless of the jug capacities and even regardless of the amount we're trying to produce! Simply repeat these two steps until the desired amount of water is obtained:

1. Fill the smaller jug.
2. Pour all the water in the smaller jug into the larger jug. If at any time the larger jug becomes full, empty it out, and continue pouring the smaller jug into the larger jug.

By the same reasoning as before, this method eventually generates every multiple—up to the size of the larger jug—of the greatest common divisor of the jug capacities, all the quantities we can possibly produce. No ingenuity is needed at all!

So now we have the complete water jug story:

**Theorem 8.2.5.** *Suppose that we have water jugs with capacities  $a$  and  $b$ . For any  $c \in [0..a]$ , it is possible to get  $c$  gallons in the size  $a$  jug iff  $c$  is a multiple of  $\gcd(a, b)$ .*

---

### 8.3 Prime Mysteries

Some of the greatest mysteries and insights in number theory concern properties of prime numbers:

**Definition 8.3.1.** A *prime* is a number greater than 1 that is divisible only by itself and 1. A number other than 0, 1, and  $-1$  that is not a prime is called *composite*.<sup>5</sup>

Here are three famous mysteries:

**Twin Prime Conjecture** There are infinitely many primes  $p$  such that  $p + 2$  is also a prime.

In 1966, Chen showed that there are infinitely many primes  $p$  such that  $p + 2$  is the product of at most two primes. So the conjecture is known to be *almost* true!

**Conjectured Inefficiency of Factoring** Given the product of two large primes  $n = pq$ , there is no efficient procedure to recover the primes  $p$  and  $q$ . That is, no *polynomial time* procedure (see Section 3.5) is guaranteed to find  $p$  and  $q$  in a number of steps bounded by a polynomial in the length of the binary representation of  $n$  (not  $n$  itself). The length of the binary representation at most  $1 + \log_2 n$ .

---

<sup>5</sup>So 0, 1, and  $-1$  are the only integers that are neither prime nor composite.

The best algorithm known is the “number field sieve,” which runs in time proportional to:

$$e^{1.9(\ln n)^{1/3}(\ln \ln n)^{2/3}}.$$

This number grows more rapidly than any polynomial in  $\log n$  and is infeasible when  $n$  has 300 digits or more.

Efficient factoring is a mystery of particular importance in computer science, as we’ll explain later in this chapter.

**Goldbach’s Conjecture** We’ve already mentioned Goldbach’s Conjecture 1.1.8 several times: every even integer greater than two is equal to the sum of two primes. For example,  $4 = 2 + 2$ ,  $6 = 3 + 3$ ,  $8 = 3 + 5$ , etc.

In 1939, Schnirelman proved that every even number can be written as the sum of not more than 300,000 primes, which was a start. Today, we know that every even number is the sum of at most 6 primes.

Primes show up erratically in the sequence of integers. In fact, their distribution seems almost random:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, \dots$$

One of the great insights about primes is that their density among the integers has a precise limit. Namely, let  $\pi(n)$  denote the number of primes up to  $n$ :

**Definition 8.3.2.**

$$\pi(n) ::= |\{p \in [2..n] \mid p \text{ is prime}\}|.$$

For example,  $\pi(1) = 0$ ,  $\pi(2) = 1$ , and  $\pi(10) = 4$  because 2, 3, 5, and 7 are the primes less than or equal to 10. Step by step,  $\pi$  grows erratically according to the erratic spacing between successive primes, but its overall growth rate is known to smooth out to be the same as the growth of the function  $n / \ln n$ :

**Theorem 8.3.3** (Prime Number Theorem).

$$\lim_{n \rightarrow \infty} \frac{\pi(n)}{n / \ln n} = 1.$$

Thus, primes gradually taper off. As a rule of thumb, about 1 integer out of every  $\ln n$  in the vicinity of  $n$  is a prime.

The Prime Number Theorem was conjectured by Legendre in 1798 and proved a century later by de la Vallée Poussin and Hadamard in 1896. However, after his death, a notebook of Gauss was found to contain the same conjecture, which he

apparently made in 1791 at age 15. (You have to feel sorry for all the otherwise “great” mathematicians who had the misfortune of being contemporaries of Gauss.)

A proof of the Prime Number Theorem is beyond the scope of this text, but there is a manageable proof (see Problem 8.22) of a related result that is sufficient for our applications:

**Theorem 8.3.4** (Chebyshev’s Theorem on Prime Density). *For  $n > 1$ ,*

$$\pi(n) > \frac{n}{3 \ln n}.$$

### A Prime for Google

In late 2004 a billboard appeared in various locations around the country:

$$\left\{ \begin{array}{l} \text{first 10-digit prime found} \\ \text{in consecutive digits of } e \end{array} \right\} \cdot \mathbf{com}$$

Substituting the correct number for the expression in curly-braces produced the URL for a Google employment page. The idea was that Google was interested in hiring the sort of people that could and would solve such a problem.

How hard is this problem? Would you have to look through thousands or millions or billions of digits of  $e$  to find a 10-digit prime? The rule of thumb derived from the Prime Number Theorem says that among 10-digit numbers, about 1 in

$$\ln 10^{10} \approx 23$$

is prime. This suggests that the problem isn’t really so hard! Sure enough, the first 10-digit prime in consecutive digits of  $e$  appears quite early:

$e = 2.718281828459045235360287471352662497757247093699959574966$   
 $9676277240766303535475945713821785251664274274663919320030$   
 $599218174135966290435729003342952605956307381323286279434 \dots$

## 8.4 The Fundamental Theorem of Arithmetic

There is an important fact about primes that you probably already know: every positive integer number has a *unique* prime factorization. So every positive integer can be built up from primes in *exactly one way*. These quirky prime numbers are the building blocks for the integers.

Since the value of a product of numbers is the same if the numbers appear in a different order, there usually isn't a unique way to express a number as a product of primes. For example, there are three ways to write 12 as a product of primes:

$$12 = 2 \cdot 2 \cdot 3 = 2 \cdot 3 \cdot 2 = 3 \cdot 2 \cdot 2.$$

What's unique about the prime factorization of 12 is that any product of primes equal to 12 will have exactly one 3 and two 2's. This means that if we *sort* the primes by size, then the product really will be unique.

Let's state this more carefully. A sequence of numbers is *weakly decreasing* when each number in the sequence is at least as big as the numbers after it. Note that a sequence of just one number as well as a sequence of no numbers—the empty sequence—is weakly decreasing by this definition.

**Theorem 8.4.1.** [Fundamental Theorem of Arithmetic] *Every positive integer is a product of a unique weakly decreasing sequence of primes.*

For example, 75237393 is the product of the weakly decreasing sequence of primes

$$23, 17, 17, 11, 7, 7, 7, 3,$$

and no other weakly decreasing sequence of primes will give 75237393.<sup>6</sup>

Notice that the theorem would be false if 1 were considered a prime; for example, 15 could be written as  $5 \cdot 3$ , or  $5 \cdot 3 \cdot 1$ , or  $5 \cdot 3 \cdot 1 \cdot 1$ , ...

There is a certain wonder in unique factorization, especially in view of the prime number mysteries we've already mentioned. It's a mistake to take it for granted, even if you've known it since you were in a crib. In fact, unique factorization actually fails for many integer-like sets of numbers, such as the complex numbers of the form  $n + m\sqrt{-5}$  for  $m, n \in \mathbb{Z}$  (see Problem 8.25).

The Fundamental Theorem is also called the *Unique Factorization Theorem*, which is a more descriptive and less pretentious, name—but we really want to get your attention to the importance and non-obviousness of unique factorization.

<sup>6</sup>The “product” of just one number is defined to be that number, and the product of no numbers is by convention defined to be 1. So each prime,  $p$ , is uniquely the product of the primes in the length-one sequence consisting solely of  $p$ , and 1, which you will remember is not a prime, is uniquely the product of the empty sequence.

### 8.4.1 Proving Unique Factorization

The Fundamental Theorem is not hard to prove, but we’ll need a couple of preliminary facts.

**Lemma 8.4.2.** *If  $p$  is a prime and  $p \mid ab$ , then  $p \mid a$  or  $p \mid b$ .*

Lemma 8.4.2 follows immediately from Unique Factorization: the primes in the product  $ab$  are exactly the primes from  $a$  and from  $b$ . But proving the lemma this way would be cheating: we’re going to need this lemma to prove Unique Factorization, so it would be circular to assume it. Instead, we’ll use the properties of gcd’s and linear combinations to give an easy, noncircular way to prove Lemma 8.4.2.

*Proof.* One case is if  $\gcd(a, p) = p$ . Then the claim holds, because  $a$  is a multiple of  $p$ .

Otherwise,  $\gcd(a, p) \neq p$ . In this case  $\gcd(a, p)$  must be 1, since 1 and  $p$  are the only positive divisors of  $p$ . Now  $\gcd(a, p)$  is a linear combination of  $a$  and  $p$ , so we have  $1 = sa + tp$  for some  $s, t$ . Then  $b = s(ab) + (tb)p$ , that is,  $b$  is a linear combination of  $ab$  and  $p$ . Since  $p$  divides both  $ab$  and  $p$ , it also divides their linear combination  $b$ . ■

A routine induction argument extends this statement to:

**Lemma 8.4.3.** *Let  $p$  be a prime. If  $p \mid a_1 a_2 \cdots a_n$ , then  $p$  divides some  $a_i$ .*

Now we’re ready to prove the Fundamental Theorem of Arithmetic.

*Proof.* Theorem 2.3.1 showed, using the Well Ordering Principle, that every positive integer can be expressed as a product of primes. So we just have to prove this expression is unique. We will use Well Ordering to prove this too.

The proof is by contradiction: assume, contrary to the claim, that there exist positive integers that can be written as products of primes in more than one way. By the Well Ordering Principle, there is a smallest integer with this property. Call this integer  $n$ , and let

$$\begin{aligned} n &= p_1 \cdot p_2 \cdots p_j, \\ &= q_1 \cdot q_2 \cdots q_k, \end{aligned}$$

where both products are in weakly decreasing order and  $p_1 \leq q_1$ .

If  $q_1 = p_1$ , then  $n/q_1$  would also be the product of different weakly decreasing sequences of primes, namely,

$$\begin{aligned} p_2 \cdots p_j, \\ q_2 \cdots q_k. \end{aligned}$$



**Figure 8.1** Alan Turing

Since  $n/q_1 < n$ , this can't be true, so we conclude that  $p_1 < q_1$ .

Since the  $p_i$ 's are weakly decreasing, all the  $p_i$ 's are less than  $q_1$ . But

$$q_1 \mid n = p_1 \cdot p_2 \cdots p_j,$$

so Lemma 8.4.3 implies that  $q_1$  divides one of the  $p_i$ 's, which contradicts the fact that  $q_1$  is bigger than all them. ■

## 8.5 Alan Turing

The man pictured in Figure 8.1 is Alan Turing, the most important figure in the history of computer science. For decades, his fascinating life story was shrouded by government secrecy, societal taboo, and even his own deceptions.

At age 24, Turing wrote a paper entitled *On Computable Numbers, with an Application to the Entscheidungsproblem*. The crux of the paper was an elegant way to model a computer in mathematical terms. This was a breakthrough, because it allowed the tools of mathematics to be brought to bear on questions of computation. For example, with his model in hand, Turing immediately proved that there exist problems that no computer can solve—no matter how ingenious the programmer. Turing's paper is all the more remarkable because he wrote it in 1936, a full decade

before any electronic computer actually existed.

The word “Entscheidungsproblem” in the title refers to one of the 28 mathematical problems posed by David Hilbert in 1900 as challenges to mathematicians of the 20th century. Turing knocked that one off in the same paper. And perhaps you’ve heard of the “Church-Turing thesis”? Same paper. So Turing was a brilliant guy who generated lots of amazing ideas. But this lecture is about one of Turing’s less-amazing ideas. It involved codes. It involved number theory. And it was sort of stupid.

Let’s look back to the fall of 1937. Nazi Germany was rearming under Adolf Hitler, world-shattering war looked imminent, and—like us —Alan Turing was pondering the usefulness of number theory. He foresaw that preserving military secrets would be vital in the coming conflict and proposed a way *to encrypt communications using number theory*. This is an idea that has ricocheted up to our own time. Today, number theory is the basis for numerous public-key cryptosystems, digital signature schemes, cryptographic hash functions, and electronic payment systems. Furthermore, military funding agencies are among the biggest investors in cryptographic research. Sorry, Hardy!

Soon after devising his code, Turing disappeared from public view, and half a century would pass before the world learned the full story of where he’d gone and what he did there. We’ll come back to Turing’s life in a little while; for now, let’s investigate the code Turing left behind. The details are uncertain, since he never formally published the idea, so we’ll consider a couple of possibilities.

### 8.5.1 Turing’s Code (Version 1.0)

The first challenge is to translate a text message into an integer so we can perform mathematical operations on it. This step is not intended to make a message harder to read, so the details are not too important. Here is one approach: replace each letter of the message with two digits ( $A = 01$ ,  $B = 02$ ,  $C = 03$ , etc.) and string all the digits together to form one huge number. For example, the message “victory” could be translated this way:

$$\begin{array}{cccccc} & v & i & c & t & o & r & y \\ \rightarrow & 22 & 09 & 03 & 20 & 15 & 18 & 25 \end{array}$$

Turing’s code requires the message to be a prime number, so we may need to pad the result with some more digits to make a prime. The Prime Number Theorem indicates that padding with relatively few digits will work. In this case, appending the digits 13 gives the number 2209032015182513, which is prime.

Here is how the encryption process works. In the description below,  $m$  is the unencoded message (which we want to keep secret),  $\hat{m}$  is the encrypted message (which the Nazis may intercept), and  $k$  is the key.

**Beforehand** The sender and receiver agree on a *secret key*, which is a large prime  $k$ .

**Encryption** The sender encrypts the message  $m$  by computing:

$$\widehat{m} = m \cdot k$$

**Decryption** The receiver decrypts  $\widehat{m}$  by computing:

$$\frac{\widehat{m}}{k} = m.$$

For example, suppose that the secret key is the prime number  $k = 22801763489$  and the message  $m$  is “victory.” Then the encrypted message is:

$$\begin{aligned} \widehat{m} &= m \cdot k \\ &= 2209032015182513 \cdot 22801763489 \\ &= 50369825549820718594667857 \end{aligned}$$

There are a couple of basic questions to ask about Turing’s code.

1. How can the sender and receiver ensure that  $m$  and  $k$  are prime numbers, as required?

The general problem of determining whether a large number is prime or composite has been studied for centuries, and tests for primes that worked well in practice were known even in Turing’s time. In the past few decades, very fast primality tests have been found as described in the text box below.

2. Is Turing’s code secure?

The Nazis see only the encrypted message  $\widehat{m} = m \cdot k$ , so recovering the original message  $m$  requires factoring  $\widehat{m}$ . Despite immense efforts, no really efficient factoring algorithm has ever been found. It appears to be a fundamentally difficult problem. So, although a breakthrough someday can’t be ruled out, the conjecture that there is no efficient way to factor is widely accepted. In effect, Turing’s code puts to practical use his discovery that there are limits to the power of computation. Thus, provided  $m$  and  $k$  are sufficiently large, the Nazis seem to be out of luck!

This all sounds promising, but there is a major flaw in Turing’s code.

**Primality Testing**

It's easy to see that an integer  $n$  is prime iff it is not divisible by any number from 2 to  $\lfloor \sqrt{n} \rfloor$  (see Problem 1.9). Of course this naive way to test if  $n$  is prime takes more than  $\sqrt{n}$  steps, which is exponential in the *size* of  $n$  measured by the number of digits in the decimal or binary representation of  $n$ . Through the early 1970's, no prime testing procedure was known that would never blow up like this.

In 1974, Volker Strassen invented a simple, fast *probabilistic* primality test. Strassen's test gives the right answer when applied to any prime number, but has some probability of giving a wrong answer on a nonprime number. However, the probability of a wrong answer on any given number is so tiny that relying on the answer is the best bet you'll ever make.

Still, the theoretical possibility of a wrong answer was intellectually bothersome—even if the probability of being wrong was a lot less than the probability of an undetectable computer hardware error leading to a wrong answer. Finally in 2002, in a breakthrough paper beginning with a quote from Gauss emphasizing the importance and antiquity of primality testing, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena presented an amazing, thirteen line description of a polynomial time primality test.

This definitively places primality testing way below the exponential effort apparently needed for SAT and similar problems. The polynomial bound on the Agrawal *et al.* test had degree 12, and subsequent research has reduced the degree to 5, but this is still too large to be practical, and probabilistic primality tests remain the method used in practice today. It's plausible that the degree bound can be reduced a bit more, but matching the speed of the known probabilistic tests remains a daunting challenge.

### 8.5.2 Breaking Turing’s Code (Version 1.0)

Let’s consider what happens when the sender transmits a *second* message using Turing’s code and the same key. This gives the Nazis two encrypted messages to look at:

$$\widehat{m}_1 = m_1 \cdot k \quad \text{and} \quad \widehat{m}_2 = m_2 \cdot k$$

The greatest common divisor of the two encrypted messages,  $\widehat{m}_1$  and  $\widehat{m}_2$ , is the secret key  $k$ . And, as we’ve seen, the GCD of two numbers can be computed very efficiently. So after the second message is sent, the Nazis can recover the secret key and read *every* message!

A mathematician as brilliant as Turing is not likely to have overlooked such a glaring problem, and we can guess that he had a slightly different system in mind, one based on *modular* arithmetic.

---

MIT OpenCourseWare  
<https://ocw.mit.edu>

6.042J / 18.062J Mathematics for Computer Science  
Spring 2015

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.