4 Number Theory

Number theory is the study of the integers. *Why* anyone would want to study the integers is not immediately 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? Second, what practical value is there in it? The mathematician G. H. Hardy expressed pleasure in its impracticality when 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 specially 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—which may leave poor Hardy spinning in his grave. It's also central to online commerce. Every time you buy a book from Amazon, check your grades on WebSIS, 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 Chapters 2 and 3.

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 of integers*, $\mathbb{Z}$.

4.1 Divisibility

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

$$a \text{ divides } b \quad \text{iff} \quad ak = b \text{ for some } k.$$

The notation, $a \mid b$, is an abbreviation for "a divides b." If $a \mid b$, then we also say that b is a *multiple* of a. A consequence of this definition is that every number divides zero.

This 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. 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.[1] For example, several such problems are shown in the box on the following page. Interestingly, we'll see that computer scientists have found ways to turn some of these difficulties to their advantage.

4.1.1 Facts about Divisibility

The lemma below states some basic facts about divisibility that are *not* difficult to prove:

Lemma 4.1.1. *The following statements about divisibility hold.*

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

Proof. We'll prove only part 2.; the other proofs are similar.

Proof of 2: Assume $a \mid b$ and $b \mid c$. Since $a \mid b$, there exists an integer k_1 such that $ak_1 = b$. Since $b \mid c$, there exists an integer k_2 such that $bk_2 = c$. Substituting ak_1 for b in the second equation gives $(ak_1)k_2 = c$. So $a(k_1k_2) = c$, which implies that $a \mid c$. ■

4.1.2 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 4.1.2 (Division Theorem)**.** [3] *Let n and d be integers such that $d > 0$. Then there exists a unique pair of integers q and r, such that*

$$n = q \cdot d + r \text{ AND } 0 \leq r < d. \tag{4.1}$$

[1] *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.

[3] This theorem is often called the "Division Algorithm," even though it is not what we would call an algorithm. We will take this familiar result for granted without proof.

Famous Conjectures in Number Theory

Fermat's Last Theorem There are no positive integers x, y, and z such that

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

for some integer $n > 2$. In a book he was reading around 1630, Fermat claimed to have a proof but not enough space in the margin to write it down. Wiles finally gave a proof of the theorem in 1994, after seven years of working in secrecy and isolation in his attic. His proof did not fit in any margin.

Goldbach Conjecture Every even integer greater than two is equal to the sum of two primes[2]. For example, $4 = 2 + 2$, $6 = 3 + 3$, $8 = 3 + 5$, etc. The conjecture holds for all numbers up to 10^{16}. 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.

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!

Primality Testing There is an efficient way to determine whether a number is prime. A naive search for factors of an integer n takes a number of steps proportional to $\sqrt{n}$, which is exponential in the *size* of n in decimal or binary notation. All known procedures for prime checking blew up like this on various inputs. Finally in 2002, an amazingly simple, new method was discovered by Agrawal, Kayal, and Saxena, which showed that prime testing only required a polynomial number of steps. Their paper began with a quote from Gauss emphasizing the importance and antiquity of the problem even in his time—two centuries ago. So prime testing is definitely not in the category of infeasible problems requiring an exponentially growing number of steps in bad cases.

Factoring Given the product of two large primes $n = pq$, there is no efficient way to recover the primes p and q. The best known algorithm is the "number field sieve", which runs in time proportional to:

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

This is infeasible when n has 300 digits or more.

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, the expression "32 % 5" evaluates to 2 in Java, C, and C++. However, all these languages treat negative numbers strangely.

4.1.3 Die Hard

Simon: On the fountain, there should be 2 jugs, do you see them? A 5-gallon and a 3-gallon. Fill one of the jugs with exactly 4 gallons of water and place it on the scale and the timer will stop. You must be precise; one ounce more or less will result in detonation. If you're still alive in 5 minutes, we'll speak.
Bruce: Wait, wait a second. I don't get it. Do you get it?
Samuel: No.
Bruce: Get the jugs. Obviously, we can't fill the 3-gallon jug with 4 gallons of water.
Samuel: Obviously.
Bruce: All right. I know, here we go. We fill the 3-gallon jug exactly to the top, right?
Samuel: Uh-huh.
Bruce: Okay, now we pour this 3 gallons into the 5-gallon jug, giving us exactly 3 gallons in the 5-gallon jug, right?
Samuel: Right, then what?
Bruce: All right. We take the 3-gallon jug and fill it a third of the way...
Samuel: No! He said, "Be precise." Exactly 4 gallons.
Bruce: Sh—. Every cop within 50 miles is running his a— off and I'm out here playing kids' games in the park.
Samuel: Hey, you want to focus on the problem at hand?

The preceding script is from the movie *Die Hard 3: With a Vengeance*. In the movie, Samuel L. Jackson and Bruce Willis have to disarm a bomb planted by the diabolical Simon Gruber. Fortunately, they find a solution in the nick of time. (No doubt reading the script helped.) On the surface, *Die Hard 3* is just a B-grade action movie; however, we think the inner message of the film is that everyone should learn at least a little number theory.

Unfortunately, Hollywood never lets go of a gimmick. Although there were no water jug tests in *Die Hard 4: Live Free or Die Hard*, rumor has it that the jugs will return in future sequels:

Die Hard 5: Die Hardest Bruce goes on vacation and—shockingly—happens into a terrorist plot. To save the day, he must make 3 gallons using 21- and 26-gallon jugs.

Die Hard 6: Die of Old Age Bruce must save his assisted living facility from a criminal mastermind by forming 2 gallons with 899- and 1147-gallon jugs.

Die Hard 7: Die Once and For All Bruce has to make 4 gallons using 3- and 6-gallon jugs.

It would be nice if we could solve all these silly water jug questions at once. In particular, how can one form g gallons using jugs with capacities a and b?

That's where number theory comes in handy.

Finding an Invariant Property

Suppose that we have water jugs with capacities a and b with $b \geq a$. The state of the system is described below with a pair of numbers (x, y), where x is the amount of water in the jug with capacity a and y is the amount in the jug with capacity b. Let's carry out sample operations and see what happens, assuming the b-jug is big enough:

$$\begin{aligned}
(0,0) &\to (a,0) && \text{fill first jug}\\
&\to (0,a) && \text{pour first into second}\\
&\to (a,a) && \text{fill first jug}\\
&\to (2a-b,b) && \text{pour first into second (assuming } 2a \geq b)\\
&\to (2a-b,0) && \text{empty second jug}\\
&\to (0,2a-b) && \text{pour first into second}\\
&\to (a,2a-b) && \text{fill first}\\
&\to (3a-2b,b) && \text{pour first into second (assuming } 3a \geq 2b)
\end{aligned}$$

What leaps out is that at every step, the amount of water in each jug is of the form

$$s \cdot a + t \cdot b \tag{4.2}$$

for some integers s and t. An expression of the form (4.2) is called an *integer linear combination* of a and b, but in this chapter we'll just call it a *linear combination*, since we're only talking integers. So we're suggesting:

Lemma 4.1.3. *Suppose that we have water jugs with capacities a and b. Then the amount of water in each jug is always a linear combination of a and b.*

Lemma 4.1.3 is easy to prove by induction on the number of pourings.

Proof. The induction hypothesis, $P(n)$, is the proposition that after n steps, 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. We assume by induction hypothesis that after n steps the amount of water in each jug is a linear combination of a and b. 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 in each jug is a linear combination of a and b before we begin pouring:

$$\begin{aligned} j_1 &= s_1 \cdot a + t_1 \cdot b \\ j_2 &= s_2 \cdot a + t_2 \cdot b \end{aligned}$$

  After pouring, one jug is either empty (contains 0 gallons) or full (contains a or b gallons). Thus, the other jug contains either $j_1 + j_2$ gallons, $j_1 + j_2 - a$, or $j_1 + j_2 - b$ gallons, all of which are linear combinations of a and b. So $P(n+1)$ holds in this case as well.

So in any case, $P(n+1)$ follows, completing the proof by induction. ■

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

Corollary 4.1.4. *Bruce dies.*

Proof. In Die Hard 7, Bruce has water jugs with capacities 3 and 6 and must form 4 gallons of water. However, the amount in each jug is always of the form $3s + 6t$ by Lemma 4.1.3. This is always a multiple of 3 by part 3 of Lemma 4.1.1, so he cannot measure out 4 gallons. ■

But Lemma 4.1.3 isn't very satisfying. We've just managed to recast a pretty understandable question about water jugs into a complicated question about linear combinations. This might not seem like a lot of progress. Fortunately, linear combinations are closely related to something more familiar, namely greatest common divisors, and these will help us solve the water jug problem.

4.2 The Greatest Common Divisor

The *greatest common divisor* of a and b is exactly what you'd guess: the largest number that is a divisor of both a and b. It is denoted by $\gcd(a, b)$. For example, $\gcd(18, 24) = 6$. The greatest common divisor turns out to be a very valuable piece of information about the relationship between a and b and for reasoning about integers in general. So we'll be making lots of arguments about greatest common divisors in what follows.

4.2.1 Linear Combinations and the GCD

The theorem below relates the greatest common divisor to linear combinations. This theorem is *very* useful; take the time to understand it and then remember it!

Theorem 4.2.1. *The greatest common divisor of a and b is equal to the smallest positive linear combination of a and b.*

For example, the greatest common divisor of 52 and 44 is 4. And, sure enough, 4 is a linear combination of 52 and 44:

$$6 \cdot 52 + (-7) \cdot 44 = 4$$

Furthermore, no linear combination of 52 and 44 is equal to a smaller positive integer.

Proof of Theorem 4.2.1. By the Well Ordering Principle, there is a smallest positive linear combination of a and b; call it m. We'll prove that $m = \gcd(a, b)$ by showing both $\gcd(a, b) \le m$ and $m \le \gcd(a, b)$.

First, we show that $\gcd(a, b) \le m$. Now any common divisor of a and b—that is, any c such that $c \mid a$ and $c \mid b$—will divide both sa and tb, and therefore also $sa + tb$ for any s and t. The $\gcd(a, b)$ is by definition a common divisor of a and b, so

$$\gcd(a, b) \mid sa + tb \tag{4.3}$$

for every s and t. In particular, $\gcd(a, b) \mid m$, which implies that $\gcd(a, b) \le m$.

Now, we show that $m \le \gcd(a, b)$. We do this by showing that $m \mid a$. A symmetric argument shows that $m \mid b$, which means that m is a common divisor of a and b. Thus, m must be less than or equal to the *greatest* common divisor of a and b.

All that remains is to show that $m \mid a$. By the Division Algorithm, there exists a quotient q and remainder r such that:

$$a = q \cdot m + r \qquad \text{(where } 0 \le r < m\text{)}$$

Recall that $m = sa + tb$ for some integers s and t. Substituting in for m gives:

$$\begin{aligned} a &= q \cdot (sa + tb) + r, \qquad \text{so} \\ r &= (1 - qs)a + (-qt)b. \end{aligned}$$

We've just expressed r as a linear combination of a and b. However, m is the *smallest positive* linear combination and $0 \leq r < m$. The only possibility is that the remainder r is not positive; that is, $r = 0$. This implies $m \mid a$. ■

Corollary 4.2.2. *An integer is linear combination of a and b iff it is a multiple of* $\gcd(a, b)$.

Proof. By (4.3), every linear combination of a and b is a multiple of $\gcd(a, b)$. Conversely, since $\gcd(a, b)$ is a linear combination of a and b, every multiple of $\gcd(a, b)$ is as well. ■

Now we can restate the water jugs lemma in terms of the greatest common divisor:

Corollary 4.2.3. *Suppose that we have water jugs with capacities a and b. Then the amount of water in each jug is always a multiple of* $\gcd(a, b)$.

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

4.2.2 Properties of the Greatest Common Divisor

We'll often make use of some basic gcd facts:

Lemma 4.2.4. *The following statements about the greatest common divisor hold:*

1. *Every common divisor of a and b divides* $\gcd(a, b)$.
2. $\gcd(ka, kb) = k \cdot \gcd(a, b)$ *for all* $k > 0$.
3. *If* $\gcd(a, b) = 1$ *and* $\gcd(a, c) = 1$, *then* $\gcd(a, bc) = 1$.
4. *If* $a \mid bc$ *and* $\gcd(a, b) = 1$, *then* $a \mid c$.
5. $\gcd(a, b) = \gcd(b, \text{rem}(a, b))$.

Here's the trick to proving these statements: translate the gcd world to the linear combination world using Theorem 4.2.1, argue about linear combinations, and then translate back using Theorem 4.2.1 again.

Proof. We prove only parts 3. and 4.

Proof of 3. The assumptions together with Theorem 4.2.1 imply that there exist integers s, t, u, and v such that:

$$\begin{aligned} sa + tb &= 1 \\ ua + vc &= 1 \end{aligned}$$

Multiplying these two equations gives:

$$(sa + tb)(ua + vc) = 1$$

The left side can be rewritten as $a \cdot (asu + btu + csv) + bc(tv)$. This is a linear combination of a and bc that is equal to 1, so $\gcd(a, bc) = 1$ by Theorem 4.2.1.

Proof of 4. Theorem 4.2.1 says that $\gcd(ac, bc)$ is equal to a linear combination of ac and bc. Now $a \mid ac$ trivially and $a \mid bc$ by assumption. Therefore, a divides *every* linear combination of ac and bc. In particular, a divides $\gcd(ac, bc) =$← $c \cdot \gcd(a, b) = c \cdot 1 = c$. The first equality uses part 2. of this lemma, and the second uses the assumption that $\gcd(a, b) = 1$. ■

4.2.3 Euclid's Algorithm

Part (5) of Lemma 4.2.4 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 part (5):

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

The last equation might look wrong, but 31 is a divisor of both 31 and 0 since every integer divides 0.

This process is called *Euclid’s algorithm* and it was discovered by the Greeks over 3000 years ago. You can prove that the algorithm always eventually terminates by using induction and the fact that the numbers in each step keep getting smaller until the remainder is 0, whereupon you have computed the GCD. In fact, the numbers are getting smaller quickly (by at least a factor of 2 every two steps) and so Euler’s Algorithm is quite fast. The fact that Euclid’s Algorithm actually produces the GCD (and not something different) can also be proved by an inductive invariant argument.

The calculation that $\gcd(1147, 899) = 31$ together with Corollary 4.2.3 implies that there is no way to measure out 2 gallons of water using jugs with capacities 1147 and 899, since we can only obtain multiples of 31 gallons with these jugs. This is good news—Bruce won’t even survive *Die Hard 6*!

But what about Die Hard 5? Is it possible for Bruce to make 3 gallons using 21- and 26-gallon jugs? Using Euclid’s algorithm:

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

Since 3 is a multiple of 1, so we can’t *rule out* the possibility that 3 gallons can be formed. On the other hand, we don’t know if it can be done either. To resolve the matter, we will need more number theory.

4.2.4 One Solution for All Water Jug Problems

Corollary 4.2.2 says that 3 can be written as a linear combination of 21 and 26, since 3 is a multiple of $\gcd(21, 26) = 1$. In other words, there exist integers s and t such that:

$$3 = s \cdot 21 + t \cdot 26$$

We don’t know what the coefficients s and t are, but we do know that they exist.

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

$$3 = s' \cdot 21 + t' \cdot 26 \tag{4.4}$$

where the coefficient s' is positive. The trick is to notice that if we increase s by 26 in the original equation 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. Notice that then t' must 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 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 (4.4), 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 (4.4) 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 that approach gives:

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

The same approach works regardless of the jug capacities and even regardless 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 of the greatest common divisor of the jug capacities—all the quantities we can possibly produce. No ingenuity is needed at all!

4.2.5 The Pulverizer

We have shown that no matter which pair of numbers a and b we are given, there is always a pair of integer coefficients s and t such that

$$\gcd(a,b) = sa + tb.$$

Unfortunately, the proof was *nonconstructive*: it didn't suggest a way for finding such s and t. That job is tackled by a mathematical tool that dates 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.

Euclid's Algorithm for finding the GCD of two numbers relies on repeated application of the equation:

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

For example, we can compute the GCD of 259 and 70 as follows:

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

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	$(\text{rem}(x,y))$	$=\leftarrow x - q \cdot y$
259	70	49	$=\leftarrow 259 - 3 \cdot 70$
70	49	21	$=\leftarrow 70 - 1 \cdot 49$
			$=\leftarrow 70 - 1 \cdot (259 - 3 \cdot 70)$
			$= \quad -1 \cdot 259 + 4 \cdot 70$
49	21	7	$=\leftarrow 49 - 2 \cdot 21$
			$=\leftarrow (259 - 3 \cdot 70) - 2 \cdot (-1 \cdot 259 + 4 \cdot 70)$
			$=\leftarrow \boxed{3 \cdot 259 - 11 \cdot 70}$
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 can be written in the form $x - q \cdot y$. (Remember that the Division Algorithm says $x = q \cdot y + r$, where r is the remainder. We get $r = x - q \cdot y$ by rearranging terms.) Then we replaced x and y in this equation with 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 that was equal to the remainder as desired. The final solution is boxed.

You can prove that the Pulverizer always works and that it terminates by using induction. Indeed, 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.

4.3 The Fundamental Theorem of Arithmetic

We now have almost enough tools to prove something that you probably already know.

Theorem 4.3.1 (Fundamental Theorem of Arithmetic)**.** *Every positive integer n can be written in a unique way as a product of primes:*

$$n = p_1 \cdot p_2 \cdots p_j \qquad (p_1 \le p_2 \le \cdots \le p_j)$$

Notice that the theorem would be false if 1 were considered a prime; for example, 15 could be written as $3 \cdot 5$ or $1 \cdot 3 \cdot 5$ or $1^2 \cdot 3 \cdot 5$. Also, we're relying on a standard convention: the product of an empty set of numbers is defined to be 1, much as the sum of an empty set of numbers is defined to be 0. Without this convention, the theorem would be false for $n = 1$.

There is a certain wonder in the Fundamental Theorem, even if you've known it since you were in a crib. Primes show up erratically in the sequence of integers. In fact, their distribution seems almost random:

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

Basic questions about this sequence have stumped humanity for centuries. And yet we know that every natural number can be built up from primes in *exactly one way*. These quirky numbers are the building blocks for the integers.

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

Lemma 4.3.2. *If p is a prime and $p \mid ab$, then $p \mid a$ or $p \mid b$.*

Proof. The greatest common divisor of a and p must be either 1 or p, since these are the only positive divisors of p. If $\gcd(a, p) = p$, then the claim holds, because a is a multiple of p. Otherwise, $\gcd(a, p) = 1$ and so $p \mid b$ by part (4) of Lemma 4.2.4. ■

The Prime Number Theorem

Let $\pi(x)$ denote the number of primes less than or equal to x. For example, $\pi(10) = 4$ because 2, 3, 5, and 7 are the primes less than or equal to 10. Primes are very irregularly distributed, so the growth of π is similarly erratic. However, the Prime Number Theorem gives an approximate answer:

$$\lim_{x \to \infty} \frac{\pi(x)}{x / \ln x} = 1$$

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

The Prime Number Theorem was conjectured by Legendre in 1798 and proved a century later by de la Vallee 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 sort of have to feel sorry for all the otherwise “great” mathematicians who had the misfortune of being contemporaries of Gauss.)

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

$$\left\{\begin{array}{c}\text{first 10-digit prime found}\\ \text{in consecutive digits of } e\end{array}\right\}\left(\textbf{com}\right.$$

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:

$$\begin{aligned} e = {} & 2.718281828459045235360287471352662497757247093699959574966 \\ & 96762772407663035354759457138217852516642\mathbf{7427466391}9320030 \\ & 5992181741359662904357290033429526059563073813232862794 34\ldots \end{aligned}$$

A routine induction argument extends this statement to:

Lemma 4.3.3. *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 3.1.2 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

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

be two of the (possibly many) ways to write n as a product of primes. Then $p_1 \mid n$ and so $p_1 \mid q_1 q_2 \cdots q_k$. Lemma 4.3.3 implies that p_1 divides one of the primes q_i. But since q_i is a prime, it must be that $p_1 = q_i$. Deleting p_1 from the first product and q_i from the second, we find that n/p_1 is a positive integer *smaller* than n that can also be written as a product of primes in two distinct ways. But this contradicts the definition of n as the smallest such positive integer. ■

4.4 Alan Turing

The man pictured in Figure 4.1 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

Photograph of Alan Turing removed due to copyright restrictions.
Please see: http://en.wikipedia.org/wiki/File:Alan_Turing_photo.jpg

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 obviously 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.

4.4.1 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:

$$\begin{array}{ccccccc} \text{“v} & \text{i} & \text{c} & \text{t} & \text{o} & \text{r} & \text{y”} \\ \rightarrow\leftarrow 22 & 09 & 03 & 20 & 15 & 18 & 25 \end{array}$$

Turing's code requires the message to be a prime number, so we may need to pad the result with a few more digits to make a prime. 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), 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:

$$m^* = m \cdot k$$

Decryption The receiver decrypts m^* by computing:

$$\frac{m^*}{k} = \frac{m \cdot k}{k} = m$$

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

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

There are a couple of questions that one might naturally 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 reasonably good primality tests were known even in Turing's time. In 2002, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena announced a primality test that is guaranteed to work on a number n in about $(\log n)^{12}$ steps, that is, a number of steps bounded by a twelfth degree polynomial in the length (in bits) of the input, n. This definitively places primality testing way below the problems of exponential difficulty. Amazingly, the description of their breakthrough algorithm was only thirteen lines long!

Of course, a twelfth degree polynomial grows pretty fast, so the Agrawal, *et al.* procedure is of no practical use. Still, good ideas have a way of breeding more good ideas, so there's certainly hope that further improvements will lead to a procedure that is useful in practice. But the truth is, there's no practical need to improve it, since very efficient *probabilistic* procedures for prime-testing have been known since the early 1970's. These procedures have some probability of giving a wrong answer, but their probability of being wrong is so tiny that relying on their answers is the best bet you'll ever make.

2. Is Turing's code secure?

   The Nazis see only the encrypted message $m^* = m \cdot k$, so recovering the original message m requires factoring m^*. Despite immense efforts, no really efficient factoring algorithm has ever been found. It appears to be a fundamentally difficult problem, though a breakthrough someday is not impossible. 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.

4.4.2 Breaking Turing's Code

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:

$$m_1^* = m_1 \cdot k \qquad \text{and} \qquad m_2^* = m_2 \cdot k$$

The greatest common divisor of the two encrypted messages, m_1^* and 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!

It is difficult to believe a mathematician as brilliant as Turing could overlook such a glaring problem. One possible explanation is that he had a slightly different system in mind, one based on *modular* arithmetic.

4.5 Modular Arithmetic

On page 1 of his masterpiece on number theory, *Disquisitiones Arithmeticae*, Gauss introduced the notion of "congruence". Now, Gauss is another guy who managed to cough up a half-decent idea every now and then, so let's take a look at this one. Gauss said that a is *congruent* to b *modulo* n iff $n \mid (a - b)$. This is written

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

For example:

$$29 \equiv 15 \pmod{7} \quad \text{because } 7 \mid (29 - 15).$$

There is a close connection between congruences and remainders:

Lemma 4.5.1 (Congruences and Remainders)**.**

$$a \equiv b \pmod{n} \quad \textit{iff} \quad \text{rem}(a, n) = \text{rem}(b, n).$$

Proof. By the Division Theorem, there exist unique pairs of integers q_1, r_1 and q_2, r_2 such that:

$$\begin{aligned} a &= q_1 n + r_1 && \text{where } 0 \leq r_1 < n, \\ b &= q_2 n + r_2 && \text{where } 0 \leq r_2 < n. \end{aligned}$$

Subtracting the second equation from the first gives:

$$a - b = (q_1 - q_2)n + (r_1 - r_2) \qquad \text{where } -n < r_1 - r_2 < n.$$

Now $a \equiv b \pmod{n}$ if and only if n divides the left side. This is true if and only if n divides the right side, which holds if and only if $r_1 - r_2$ is a multiple of n. Given the bounds on $r_1 - r_2$, this happens precisely when $r_1 = r_2$, that is, when $\text{rem}(a, n) = \text{rem}(b, n)$. ■

So we can also see that

$$29 \equiv 15 \pmod{7} \quad \text{because } \text{rem}(29, 7) = 1 = \text{rem}(15, 7).$$

This formulation explains why the congruence relation has properties like an equality relation. Notice that even though (mod 7) appears over on the right side, the $\equiv\leftarrow$ symbol, it isn't any more strongly associated with the 15 than with the 29. It would really be clearer to write 29 $\equiv_{\leftarrow\text{mod } 7}$ 15 for example, but the notation with the modulus at the end is firmly entrenched and we'll stick to it.

We'll make frequent use of the following immediate Corollary of Lemma 4.5.1:

Corollary 4.5.2.
$$a \equiv \text{rem}(a, n) \pmod{n}$$

Still another way to think about congruence modulo n is that it *defines a partition of the integers into n sets so that congruent numbers are all in the same set*. For example, suppose that we're working modulo 3. Then we can partition the integers into 3 sets as follows:

$$\begin{array}{lcccccccl}
\{\leftarrow\ldots, & -6, & -3, & 0, & 3, & 6, & 9, & \ldots & \} \\
\{\leftarrow\ldots, & -5, & -2, & 1, & 4, & 7, & 10, & \ldots & \} \\
\{\leftarrow\ldots, & -4, & -1, & 2, & 5, & 8, & 11, & \ldots & \}\leftarrow
\end{array}$$

according to whether their remainders on division by 3 are 0, 1, or 2. The upshot is that when arithmetic is done modulo n there are really only n different kinds of numbers to worry about, because there are only n possible remainders. In this sense, modular arithmetic is a simplification of ordinary arithmetic and thus is a good reasoning tool.

There are many useful facts about congruences, some of which are listed in the lemma below. The overall theme is that *congruences work a lot like equations*, though there are a couple of exceptions.

Lemma 4.5.3 (Facts About Congruences)**.** *The following hold for* $n \geq 1$*:*

1. $a \equiv a \pmod{n}$
2. $a \equiv b \pmod{n}$ *implies* $b \equiv a \pmod{n}$
3. $a \equiv b \pmod{n}$ *and* $b \equiv c \pmod{n}$ *implies* $a \equiv c \pmod{n}$
4. $a \equiv b \pmod{n}$ *implies* $a + c \equiv b + c \pmod{n}$
5. $a \equiv b \pmod{n}$ *implies* $ac \equiv bc \pmod{n}$
6. $a \equiv b \pmod{n}$ *and* $c \equiv d \pmod{n}$ *imply* $a + c \equiv b + d \pmod{n}$
7. $a \equiv b \pmod{n}$ *and* $c \equiv d \pmod{n}$ *imply* $ac \equiv bd \pmod{n}$

Proof. Parts 1–3. follow immediately from Lemma 4.5.1. Part 4. follows immediately from the definition that $a \equiv b \pmod{n}$ iff $n \mid (a - b)$. Likewise, part 5. follows because if $n \mid (a - b)$ then it divides $(a - b)c = ac - bc$. To prove part 6., assume

$$a \equiv b \pmod{n} \tag{4.5}$$

and

$$c \equiv d \pmod{n}. \tag{4.6}$$

Then

$$\begin{aligned}
a + c &\equiv b + c \pmod{n} && \text{(by part 4. and (4.5)),}\\
c + b &\equiv d + b \pmod{n} && \text{(by part 4. and (4.6)), so}\\
b + c &\equiv b + d \pmod{n} && \text{and therefore}\\
a + c &\equiv b + d \pmod{n} && \text{(by part 3.)}
\end{aligned}$$

Part 7 has a similar proof. ■

4.5.1 Turing's Code (Version 2.0)

In 1940, France had fallen before Hitler's army, and Britain stood alone against the Nazis in western Europe. British resistance depended on a steady flow of supplies brought across the north Atlantic from the United States by convoys of ships. These convoys were engaged in a cat-and-mouse game with German "U-boats"—submarines—which prowled the Atlantic, trying to sink supply ships and starve Britain into submission. The outcome of this struggle pivoted on a balance of information: could the Germans locate convoys better than the Allies could locate U-boats or vice versa?

Germany lost.

But a critical reason behind Germany's loss was made public only in 1974: Germany's naval code, *Enigma*, had been broken by the Polish Cipher Bureau (see `http://en.wikipedia.org/wiki/Polish_Cipher_Bureau`) and the secret had been turned over to the British a few weeks before the Nazi invasion of Poland in 1939. Throughout much of the war, the Allies were able to route convoys around German submarines by listening in to German communications. The British government didn't explain *how* Enigma was broken until 1996. When it was finally released (by the US), the story revealed that Alan Turing had joined the secret British codebreaking effort at Bletchley Park in 1939, where he became the lead developer of methods for rapid, bulk decryption of German Enigma messages. Turing's Enigma deciphering was an invaluable contribution to the Allied victory over Hitler.

Governments are always tight-lipped about cryptography, but the half-century of official silence about Turing's role in breaking Enigma and saving Britain may be related to some disturbing events after the war. More on that later. Let's get back to number theory and consider an alternative interpretation of Turing's code. Perhaps we had the basic idea right (multiply the message by the key), but erred in using *conventional* arithmetic instead of *modular* arithmetic. Maybe this is what Turing meant:

Beforehand The sender and receiver agree on a large prime p, which may be made public. (This will be the modulus for all our arithmetic.) They also agree on a secret key $k \in \{1, 2, \ldots, p-1\}$.

Encryption The message m can be any integer in the set $\{0, 1, 2, \ldots, p-1\}$; in particular, the message is no longer required to be a prime. The sender encrypts the message m to produce m^* by computing:

$$m^* = \text{rem}(mk, p) \tag{4.7}$$

Decryption (Uh-oh.)

The decryption step is a problem. We might hope to decrypt in the same way as before: by dividing the encrypted message m^* by the key k. The difficulty is that m^* is the *remainder* when mk is divided by p. So dividing m^* by k might not even give us an integer!

This decoding difficulty can be overcome with a better understanding of arithmetic modulo a prime.

4.6 Arithmetic with a Prime Modulus

4.6.1 Multiplicative Inverses

The *multiplicative inverse* of a number x is another number x^{-1} such that:

$$x \cdot x^{-1} = 1$$

Generally, multiplicative inverses exist over the real numbers. For example, the multiplicative inverse of 3 is $1/3$ since:

$$3 \cdot \frac{1}{3} = 1$$

The sole exception is that 0 does not have an inverse.

On the other hand, inverses generally do not exist over the integers. For example, 7 can not be multiplied by another integer to give 1.

Surprisingly, multiplicative inverses do exist when we're working *modulo a prime number*. For example, if we're working modulo 5, then 3 is a multiplicative inverse of 7, since:

$$7 \cdot 3 \equiv 1 \pmod{5}$$

(All numbers congruent to 3 modulo 5 are also multiplicative inverses of 7; for example, $7 \cdot 8 \equiv 1 \pmod{5}$ as well.) The only exception is that numbers congruent to 0 modulo 5 (that is, the multiples of 5) do not have inverses, much as 0 does not have an inverse over the real numbers. Let's prove this.

Lemma 4.6.1. *If p is prime and k is not a multiple of p, then k has a multiplicative inverse modulo p.*

Proof. Since p is prime, it has only two divisors: 1 and p. And since k is not a multiple of p, we must have $\gcd(p, k) = 1$. Therefore, there is a linear combination of p and k equal to 1:

$$sp + tk = 1$$

Rearranging terms gives:

$$sp = 1 - tk$$

This implies that $p \mid (1 - tk)$ by the definition of divisibility, and therefore $tk \equiv 1 \pmod{p}$ by the definition of congruence. Thus, t is a multiplicative inverse of k. ■

Multiplicative inverses are the key to decryption in Turing's code. Specifically, we can recover the original message by multiplying the encoded message by the *inverse* of the key:

$$\begin{aligned} m^* \cdot k^{-1} &= \text{rem}(mk, p) \cdot k^{-1} && \text{(the def. (4.7) of } m^*) \\ &\equiv (mk)k^{-1} \pmod{p} && \text{(by Cor. 4.5.2)} \\ &\equiv m \pmod{p}. \end{aligned}$$

This shows that m^*k^{-1} is congruent to the original message m. Since m was in the range $0, 1, \ldots, p - 1$, we can recover it exactly by taking a remainder:

$$m = \text{rem}(m^*k^{-1}, p).$$

So all we need to decrypt the message is to find a value of k^{-1}. From the proof of Lemma 4.6.1, we know that t is such a value, where $sp + tk = 1$. Finding t is easy using the Pulverizer.

4.6.2 Cancellation

Another sense in which real numbers are nice is that one can cancel multiplicative terms. In other words, if we know that $m_1 k = m_2 k$, then we can cancel the k's and conclude that $m_1 = m_2$, provided $k \neq 0$. In general, cancellation is *not* valid in modular arithmetic. For example,

$$2 \cdot 3 \equiv 4 \cdot 3 \pmod{6},$$

but canceling the 3's leads to the *false* conclusion that $2 \equiv 4 \pmod{6}$. The fact that multiplicative terms can not be canceled is the most significant sense in which congruences differ from ordinary equations. However, this difference goes away if we're working modulo a *prime*; then cancellation is valid.

Lemma 4.6.2. *Suppose p is a prime and k is not a multiple of p. Then*

$$ak \equiv bk \pmod{p} \quad \text{IMPLIES} \quad a \equiv b \pmod{p}.$$

Proof. Multiply both sides of the congruence by k^{-1}. ∎

We can use this lemma to get a bit more insight into how Turing's code works. In particular, the encryption operation in Turing's code *permutes the set of possible messages*. This is stated more precisely in the following corollary.

Corollary 4.6.3. *Suppose p is a prime and k is not a multiple of p. Then the sequence:*

$$\text{rem}((1 \cdot k), p), \quad \text{rem}((2 \cdot k), p), \quad \ldots, \quad \text{rem}(((p-1) \cdot k), p)$$

is a permutation[4] *of the sequence:*

$$1, \quad 2, \quad \ldots, \quad (p-1).$$

Proof. The sequence of remainders contains $p - 1$ numbers. Since $i \cdot k$ is not divisible by p for $i = 1, \ldots p - 1$, all these remainders are in the range 1 to $p - 1$ by the definition of remainder. Furthermore, the remainders are all different: no two numbers in the range 1 to $p-1$ are congruent modulo p, and by Lemma 4.6.2, $i \cdot k \equiv j \cdot k \pmod{p}$ if and only if $i \equiv j \pmod{p}$. Thus, the sequence of remainders must contain *all* of the numbers from 1 to $p - 1$ in some order. ∎

[4] A *permutation* of a sequence of elements is a reordering of the elements.

For example, suppose $p = 5$ and $k = 3$. Then the sequence:

$$\underbrace{\text{rem}((1 \cdot 3), 5)}_{=3}, \quad \underbrace{\text{rem}((2 \cdot 3), 5)}_{=1}, \quad \underbrace{\text{rem}((3 \cdot 3), 5)}_{=4}, \quad \underbrace{\text{rem}((4 \cdot 3), 5)}_{=2}$$

is a permutation of 1, 2, 3, 4. As long as the Nazis don't know the secret key k, they don't know how the set of possible messages are permuted by the process of encryption and thus they can't read encoded messages.

4.6.3 Fermat's Little Theorem

An alternative approach to finding the inverse of the secret key k in Turing's code (about equally efficient and probably more memorable) is to rely on Fermat's Little Theorem, which is much easier than his famous Last Theorem.

Theorem 4.6.4 (Fermat's Little Theorem)**.** *Suppose p is a prime and k is not a multiple of p. Then:*

$$k^{p-1} \equiv 1 \pmod{p}$$

Proof. We reason as follows:

$$\begin{aligned}
(p-1)! &::= 1 \cdot 2 \cdots (p-1) \\
&= \text{rem}(k, p) \cdot \text{rem}(2k, p) \cdots \text{rem}((p-1)k, p) && \text{(by Cor 4.6.3)} \\
&\equiv k \cdot 2k \cdots (p-1)k \pmod{p} && \text{(by Cor 4.5.2)} \\
&\equiv (p-1)! \cdot k^{p-1} \pmod{p} && \text{(rearranging terms)}
\end{aligned}$$

Now $(p-1)!$ is not a multiple of p because the prime factorizations of $1, 2, \ldots,$ $(p-1)$ contain only primes smaller than p. So by Lemma 4.6.2, we can cancel $(p-1)!$ from the first and last expressions, which proves the claim. ■

Here is how we can find inverses using Fermat's Theorem. Suppose p is a prime and k is not a multiple of p. Then, by Fermat's Theorem, we know that:

$$k^{p-2} \cdot k \equiv 1 \pmod{p}$$

Therefore, k^{p-2} must be a multiplicative inverse of k. For example, suppose that we want the multiplicative inverse of 6 modulo 17. Then we need to compute $\text{rem}(6^{15}, 17)$, which we can do by successive squaring. All the congruences below hold modulo 17.

$$\begin{aligned}
6^2 &\equiv 36 \equiv 2 \\
6^4 &\equiv (6^2)^2 \equiv 2^2 \equiv 4 \\
6^8 &\equiv (6^4)^2 \equiv 4^2 \equiv 16 \\
6^{15} &\equiv 6^8 \cdot 6^4 \cdot 6^2 \cdot 6 \equiv 16 \cdot 4 \cdot 2 \cdot 6 \equiv 3
\end{aligned}$$

Therefore, $\text{rem}(6^{15}, 17) = 3$. Sure enough, 3 is the multiplicative inverse of 6 modulo 17, since:

$$3 \cdot 6 \equiv 1 \pmod{17}$$

In general, if we were working modulo a prime p, finding a multiplicative inverse by trying every value between 1 and $p - 1$ would require about p operations. However, the approach above requires only about $2 \log p$ operations, which is far better when p is large.

4.6.4 Breaking Turing’s Code—Again

The Germans didn’t bother to encrypt their weather reports with the highly-secure Enigma system. After all, so what if the Allies learned that there was rain off the south coast of Iceland? But, amazingly, this practice provided the British with a critical edge in the Atlantic naval battle during 1941.

The problem was that some of those weather reports had originally been transmitted using Enigma from U-boats out in the Atlantic. Thus, the British obtained both unencrypted reports and the same reports encrypted with Enigma. By comparing the two, the British were able to determine which key the Germans were using that day and could read all other Enigma-encoded traffic. Today, this would be called a *known-plaintext attack*.

Let’s see how a known-plaintext attack would work against Turing’s code. Suppose that the Nazis know both m and m^* where:

$$m^* \equiv mk \pmod{p}$$

Now they can compute:

$$\begin{aligned}
m^{p-2} \cdot m^* &= m^{p-2} \cdot \text{rem}(mk, p) && \text{(def. (4.7) of } m^*) \\
&\equiv m^{p-2} \cdot mk \pmod{p} && \text{(by Cor 4.5.2)} \\
&\equiv m^{p-1} \cdot k \pmod{p} \\
&\equiv k \pmod{p} && \text{(Fermat's Theorem)}
\end{aligned}$$

Now the Nazis have the secret key k and can decrypt any message!

This is a huge vulnerability, so Turing's code has no practical value. Fortunately, Turing got better at cryptography after devising this code; his subsequent deciphering of Enigma messages surely saved thousands of lives, if not the whole of Britain.

4.6.5 Turing Postscript

A few years after the war, Turing's home was robbed. Detectives soon determined that a former homosexual lover of Turing's had conspired in the robbery. So they arrested him—that is, they arrested Alan Turing—because homosexuality was a British crime punishable by up to two years in prison at that time. Turing was sentenced to a hormonal "treatment" for his homosexuality: he was given estrogen injections. He began to develop breasts.

Three years later, Alan Turing, the founder of computer science, was dead. His mother explained what happened in a biography of her own son. Despite her repeated warnings, Turing carried out chemistry experiments in his own home. Apparently, her worst fear was realized: by working with potassium cyanide while eating an apple, he poisoned himself.

However, Turing remained a puzzle to the very end. His mother was a devoutly religious woman who considered suicide a sin. And, other biographers have pointed out, Turing had previously discussed committing suicide by eating a poisoned apple. Evidently, Alan Turing, who founded computer science and saved his country, took his own life in the end, and in just such a way that his mother could believe it was an accident.

Turing's last project before he disappeared from public view in 1939 involved the construction of an elaborate mechanical device to test a mathematical conjecture called the Riemann Hypothesis. This conjecture first appeared in a sketchy paper by Bernhard Riemann in 1859 and is now one of the most famous unsolved problem in mathematics.

4.7 Arithmetic with an Arbitrary Modulus

Turing's code did not work as he hoped. However, his essential idea—using number theory as the basis for cryptography—succeeded spectacularly in the decades after his death.

In 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman at MIT proposed a highly secure cryptosystem (called **RSA**) based on number theory. Despite decades of attack, no significant weakness has been found. Moreover, RSA has a major advantage over traditional codes: the sender and receiver of an encrypted mes-

The Riemann Hypothesis

The formula for the sum of an infinite geometric series says:

$$1 + x + x^2 + x^3 + \cdots = \frac{1}{1-x}$$

Substituting $x = \frac{1}{2^s}$, $x = \frac{1}{3^s}$, $x = \frac{1}{5^s}$, and so on for each prime number gives a sequence of equations:

$$\begin{aligned}
1 + \frac{1}{2^s} + \frac{1}{2^{2s}} + \frac{1}{2^{3s}} + \cdots &= \leftarrow \frac{1}{1 - 1/2^s} \\
1 + \frac{1}{3^s} + \frac{1}{3^{2s}} + \frac{1}{3^{3s}} + \cdots &= \leftarrow \frac{1}{1 - 1/3^s} \\
1 + \frac{1}{5^s} + \frac{1}{5^{2s}} + \frac{1}{5^{3s}} + \cdots &= \leftarrow \frac{1}{1 - 1/5^s} \\
&\text{etc.}
\end{aligned}$$

Multiplying together all the left sides and all the right sides gives:

$$\sum_{n=1}^{\infty} \frac{1}{n^s} = \leftarrow \prod_{p \in \text{primes}} \left(\frac{1}{1 - 1/p^s} \right)$$

The sum on the left is obtained by multiplying out all the infinite series and applying the Fundamental Theorem of Arithmetic. For example, the term $1/300^s$ in the sum is obtained by multiplying $1/2^{2s}$ from the first equation by $1/3^s$ in the second and $1/5^{2s}$ in the third. Riemann noted that every prime appears in the expression on the right. So he proposed to learn about the primes by studying the equivalent, but simpler expression on the left. In particular, he regarded s as a complex number and the left side as a function, $\zeta(s)$. Riemann found that the distribution of primes is related to values of s for which $\zeta(s) = 0$, which led to his famous conjecture:

Definition 4.6.5. *The Riemann Hypothesis*: Every nontrivial zero of the zeta function $\zeta(s)$ lies on the line $s = 1/2 + ci$ in the complex plane.

A proof would immediately imply, among other things, a strong form of the Prime Number Theorem.

Researchers continue to work intensely to settle this conjecture, as they have for over a century. It is another of the Millennium Problems whose solver will earn \$1,000,000 from the Clay Institute.

sage need not meet beforehand to agree on a secret key. Rather, the receiver has both a *secret key*, which she guards closely, and a *public key*, which she distributes as widely as possible. The sender then encrypts his message using her widely-distributed public key. Then she decrypts the received message using her closely-held private key. The use of such a *public key cryptography* system allows you and Amazon, for example, to engage in a secure transaction without meeting up beforehand in a dark alley to exchange a key.

Interestingly, RSA does not operate modulo a prime, as Turing's scheme may have, but rather modulo the product of *two* large primes. Thus, we'll need to know a bit about how arithmetic works modulo a composite number in order to understand RSA. Arithmetic modulo an arbitrary positive integer is really only a little more painful than working modulo a prime—though you may think this is like the doctor saying, "This is only going to hurt a little," before he jams a big needle in your arm.

4.7.1 Relative Primality

First, we need a new definition. Integers a and b are *relatively prime* iff $\gcd(a, b) =\leftarrow$ 1. For example, 8 and 15 are relatively prime, since $\gcd(8, 15) = 1$. Note that, except for multiples of p, every integer is relatively prime to a prime number p.

Next we'll need to generalize what we know about arithmetic modulo a prime to work modulo an arbitrary positive integer n. The basic theme is that arithmetic modulo n may be complicated, but the integers *relatively prime* to n remain fairly well-behaved. For example, the proof of Lemma 4.6.1 of an inverse for k modulo p extends to an inverse for k relatively prime to n:

Lemma 4.7.1. *Let n be a positive integer. If k is* relatively prime *to n, then there exists an integer k^{-1} such that:*

$$k \cdot k^{-1} \equiv 1 \pmod{n}$$

As a consequence of this lemma, we can cancel a multiplicative term from both sides of a congruence if that term is relatively prime to the modulus:

Corollary 4.7.2. *Suppose n is a positive integer and k is relatively prime to n. If*

$$ak \equiv bk \pmod{n}$$

then

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

This holds because we can multiply both sides of the first congruence by k^{-1} and simplify to obtain the second.

The following lemma is the natural generalization of Corollary 4.6.3.

Lemma 4.7.3. *Suppose n is a positive integer and k is relatively prime to n. Let $k_1, \ldots, k_r$ denote all the integers relatively prime to n in the range 1 to $n-1$. Then the sequence:*

$$\text{rem}(k_1 \cdot k, n), \quad \text{rem}(k_2 \cdot k, n), \quad \text{rem}(k_3 \cdot k, n), \quad \ldots \quad , \text{rem}(k_r \cdot k, n)$$

is a permutation of the sequence:

$$k_1, \quad k_2, \quad \ldots \quad , k_r.$$

Proof. We will show that the remainders in the first sequence are all distinct and are equal to some member of the sequence of k_j's. Since the two sequences have the same length, the first must be a permutation of the second.

First, we show that the remainders in the first sequence are all distinct. Suppose that $\text{rem}(k_i k, n) = \text{rem}(k_j k, n)$. This is equivalent to $k_i k \equiv k_j k \pmod{n}$, which implies $k_i \equiv k_j \pmod{n}$ by Corollary 4.7.2. This, in turn, means that $k_i = k_j$ since both are between 1 and $n-1$. Thus, none of the remainder terms in the first sequence is equal to any other remainder term.

Next, we show that each remainder in the first sequence equals one of the k_i. By assumption, $\gcd(k_i, n) = 1$ and $\gcd(k, n) = 1$, which means that

$$\begin{aligned} \gcd(n, \text{rem}(k_i k, n)) &= \gcd(k_i k, n) && \text{(by part (5) of Lemma 4.2.4)} \\ &= 1 && \text{(by part (3) of Lemma 4.2.4).} \end{aligned}$$

Since $\text{rem}(k_i k, n)$ is in the range from 0 to $n-1$ by the definition of remainder, and since it is relatively prime to n, it must (by definition of the k_j's) be equal to some k_j. ■

4.7.2 Euler's Theorem

RSA relies heavily on a generalization of Fermat's Theorem known as Euler's Theorem. For both theorems, the exponent of k needed to produce an inverse of k modulo n depends on the number of integers in the set $\{1, 2, \ldots, n\}$ (denoted $[1, n]$) that are relatively prime to n. This value is known as *Euler's ϕ function* (a.k.a. *Euler's totient function*) and it is denoted as $\phi(n)$. For example, $\phi(7) = 6$ since 1, 2, 3, 4, 5, and 6 are all relatively prime to 7. Similarly, $\phi(12) = 4$ since 1, 5, 7, and 11 are the only numbers in $[1, 12]$ that are relatively prime to 12.[5]

If n is prime, then $\phi(n) = n - 1$ since every number less than a prime number is relatively prime to that prime. When n is composite, however, the ϕ function gets a little complicated. The following theorem characterizes the ϕ function for

[5]Recall that $\gcd(n, n) = n$ and so n is never relatively prime to itself.

composite n. We won't prove the theorem in its full generality, although we will give a proof for the special case when n is the product of two primes since that is the case that matters for RSA.

Theorem 4.7.4. *For any number n, if $p_1, p_2, \ldots, p_j$ are the (distinct) prime factors of n, then*

$$\phi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_j}\right)$$

For example,

$$\begin{aligned}\phi(300) &= \phi(2^2 \cdot 3 \cdot 5^2)\\ &= 300\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right)\\ &= 300\left(\frac{1}{2}\right)\left(\frac{2}{3}\right)\left(\frac{4}{5}\right)\\ &= 80.\end{aligned}$$

Corollary 4.7.5. *Let $n = pq$ where p and q are different primes. Then $\phi(n) = (p-1)(q-1)$.*

Corollary 4.7.5 follows easily from Theorem 4.7.4, but since Corollary 4.7.5 is important to RSA and we have not provided a proof of Theorem 4.7.4, we will give a direct proof of Corollary 4.7.5 in what follows.

Proof of Corollary 4.7.5. Since p and q are prime, any number that is not relatively prime to $n = pq$ must be a multiple of p or a multiple of q. Among the numbers 1, 2, ..., pq, there are precisely q multiples of p and p multiples of q. Since p and q are relatively prime, the only number in $[1, pq]$ that is a multiple of both p and q is pq. Hence, there are $p + q - 1$ numbers in $[1, pq]$ that are *not* relatively prime to n. This means that

$$\begin{aligned}\phi(n) &= pq - p - q + 1\\ &= (p-1)(q-1),\end{aligned}$$

as claimed.[6] ■

We can now prove Euler's Theorem:

[6] This proof provides a brief preview of the kinds of counting arguments that we will explore more fully in Part III.

Theorem 4.7.6 (Euler’s Theorem)**.** *Suppose n is a positive integer and k is relatively prime to n. Then*

$$k^{\phi(n)} \equiv 1 \pmod{n}$$

Proof. Let $k_1, \ldots, k_r$ denote all integers relatively prime to n such that $0 \leq k_i < n$. Then $r = \phi(n)$, by the definition of the function ϕ. The remainder of the proof mirrors the proof of Fermat’s Theorem. In particular,

$$\begin{aligned}
&k_1 \cdot k_2 \cdots k_r \\
&\quad = \text{rem}(k_1 \cdot k, n) \cdot \text{rem}(k_2 \cdot k, n) \cdots \text{rem}(k_r \cdot k, n) && \text{(by Lemma 4.7.3)} \\
&\quad \equiv (k_1 \cdot k) \cdot (k_2 \cdot k) \cdots\cdot (k_r \cdot k) \pmod{n} && \text{(by Cor 4.5.2)} \\
&\quad \equiv (k_1 \cdot k_2 \cdots k_r) \cdot k^r \pmod{n} && \text{(rearranging terms)}
\end{aligned}$$

Part (3) of Lemma 4.2.4. implies that $k_1 \cdot k_2 \cdots k_r$ is relatively prime to n. So by Corollary 4.7.2, we can cancel this product from the first and last expressions. This proves the claim. ■

We can find multiplicative inverses using Euler’s theorem as we did with Fermat’s theorem: if k is relatively prime to n, then $k^{\phi(n)-1}$ is a multiplicative inverse of k modulo n. However, this approach requires computing $\phi(n)$. Computing $\phi(n)$ is easy (using Theorem 4.7.4) if we know the prime factorization of n. Unfortunately, finding the factors of n can be hard to do when n is large and so the Pulverizer is often the best approach to computing inverses modulo n.

4.8 The RSA Algorithm

Finally, we are ready to see how the *RSA public key encryption scheme* works. The details are in the box on the next page.

It is not immediately clear from the description of the RSA cryptosystem that the decoding of the encrypted message is, in fact, the original unencrypted message. In order to check that this is the case, we need to show that the decryption $\text{rem}((m')^d, n)$ is indeed equal to the sender’s message m. Since $m' = \text{rem}(m^e, n)$, m' is congruent to m^e modulo n by Corollary 4.5.2. That is,

$$m' \equiv m^e \pmod{n}.$$

By raising both sides to the power d, we obtain the congruence

$$(m')^d \equiv m^{ed} \pmod{n}. \tag{4.8}$$

The RSA Cryptosystem

Beforehand The receiver creates a public key and a secret key as follows.

1. Generate two distinct primes, p and q. Since they can be used to generate the secret key, they must be kept hidden.
2. Let $n = pq$.
3. Select an integer e such that $\gcd(e, (p-1)(q-1)) = 1$.
   The *public key* is the pair (e, n). This should be distributed widely.
4. Compute d such that $de \equiv 1 \pmod{(p-1)(q-1)}$. This can be done using the Pulverizer.
   The *secret key* is the pair (d, n). This should be kept hidden!

Encoding Given a message m, the sender first checks that $\gcd(m, n) = 1$.[a] The sender then encrypts message m to produce m' using the public key:

$$m' = \text{rem}(m^e, n).$$

Decoding The receiver decrypts message m' back to message m using the secret key:

$$m = \text{rem}((m')^d, n).$$

[a]It would be very bad if $\gcd(m, n)$ equals p or q since then it would be easy for someone to use the encoded message to compute the secret key If $\gcd(m, n) = n$, then the encoded message would be 0, which is fairly useless. For very large values of n, it is extremely unlikely that $\gcd(m, n) \neq 1$. If this does happen, you should get a new set of keys or, at the very least, add some bits to m so that the resulting message is relatively prime to n.

The encryption exponent e and the decryption exponent d are chosen such that $de \equiv 1 \pmod{(p-1)(q-1)}$. So, there exists an integer r such that $ed = 1 + r(p-1)(q-1)$. By substituting $1 + r(p-1)(q-1)$ for ed in Equation 4.8, we obtain

$$(m')^d \equiv m \cdot m^{r(p-1)(q-1)} \pmod{n}. \tag{4.9}$$

By Euler's Theorem and the assumption that $\gcd(m, n) = 1$, we know that

$$m^{\phi(n)} \equiv 1 \pmod{n}.$$

From Corollary 4.7.5, we know that $\phi(n) = (p-1)(q-1)$. Hence,

$$\begin{aligned}(m')^d &= m \cdot m^{r(p-1)(q-1)} \pmod{n} \\ &= m \cdot 1^r \pmod{n} \\ &= m \pmod{n}.\end{aligned}$$

Hence, the decryption process indeed reproduces the original message m.

Is it hard for someone without the secret key to decrypt the message? No one knows for sure but it is generally believed that if n is a very large number (say, with a thousand digits), then it is difficult to reverse engineer d from e and n. Of course, it is easy to compute d if you know p and q (by using the Pulverizer) but it is not known how to quickly factor n into p and q when n is very large. Maybe with a little more studying of number theory, you will be the first to figure out how to do it. Although, we should warn you that Gauss worked on it for years without a lot to show for his efforts. And if you do figure it out, you might wind up meeting some serious-looking fellows in black suits....