15 Number Theoretic Algorithms (November 12 and 14)

And it's one, two, three,
What are we fighting for?
Don't tell me, I don't give a damn,
Next stop is Vietnam; [or: This time we'll kill Saddam]
And it's five, six, seven,
Open up the pearly gates,
Well there ain't no time to wonder why,
Whoopee! We're all going to die.

— Country Joe and the Fish
"I-Feel-Like-I'm-Fixin'-to-Die Rag" (1967)

There are 0 kinds of mathematicians:
Those who can count modulo 2 and those who can't.

— anonymous

15.1 Greatest Common Divisors

Before we get to any actual algorithms, we need some definitions and preliminary results. **Unless specifically indicated otherwise, *all* variables in this lecture are integers.**

The symbol $\mathbb{Z}$ (from the German word "Zahlen", meaning 'numbers' or 'to count') to denote the set of integers. We say that one integer d *divides* another integer n, or that d is a *divisor* of n, if the quotient n/d is also an integer. Symbolically, we can write this definition as follows:

$$d \mid n \iff \left\lfloor \frac{n}{d} \right\rfloor = \frac{n}{d}$$

In particular, zero is not a divisor of any integer—∞ is *not* an integer—but every other integer is a divisor of zero. If d and n are positive, then $d \mid n$ immediately implies that $d \le n$.

Any integer n can be written in the form $n = qd+r$ for some non-negative integer $0 \le r \le |d-1|$. Moreover, the choices for the quotient q and remainder r are unique:

$$q = \left\lfloor \frac{n}{d} \right\rfloor \qquad \text{and} \qquad r = n \bmod d = n - d \left\lfloor \frac{n}{d} \right\rfloor.$$

Note that the remainder $n \bmod d$ is *always* non-negative, even if $n < 0$ or $d < 0$ or both.[1]

If d divides two integers m and n, we say that d is a *common divisor* of m and n. It's trivial to prove (by definition crunching) that any common divisor of m and n also divides any integer linear combination of m and n:

$$(d \mid m) \text{ and } (d \mid n) \implies d \mid (am + bn)$$

The *greatest common divisor* of m and n, written $\gcd(m,n)$,[2] is the largest integer that divides both m and n. Sometimes this is also called the greater common *denominator*. The greatest common divisor has another useful characterization as the *smallest* element of another set.

Lemma 1. *$gcd(m,n)$ is the smallest positive integer of the form $am + bn$.*

[1] The sign rules for the C/C++/Java % operator are just plain stupid. I can't count the number of times I've had to write `x = (x+n)%n;` instead of `x %= n;`. Idiots.

[2] Do *not* use the notation (m,n) for greatest common divisor. *Ever.*

Proof: Let s be the smallest positive integer of the form $am + bn$. Any common divisor of m and n is also a divisor of $s = am + bn$. In particular, $\gcd(m, n)$ is a divisor of s, which implies that $\boxed{\gcd(m, n) \le s}$.

To prove the other inequality, let's show that $s|m$ by calculating $m \bmod s$.

$$m \bmod s = m - s\left\lfloor \frac{m}{s} \right\rfloor = m - (am + bn)\left\lfloor \frac{m}{s} \right\rfloor = m\left(1 - a\left\lfloor \frac{m}{s} \right\rfloor\right) + n\left(-b\left\lfloor \frac{m}{s} \right\rfloor\right)$$

We observe that $m \bmod s$ is an integer linear combination of m and n. Since $m \bmod s < s$, and s is the smallest *positive* integer linear combination, $m \bmod s$ cannot be positive. So it must be zero, which implies that $s \mid m$, as we claimed. By a symmetric argument, $s \mid n$. Thus, s is a common divisor of m and n. A common divisor can't be greater than the *greatest* common divisor, so $\boxed{s \le \gcd(m, n)}$.

These two inequalities imply that $s = \gcd(m, n)$, completing the proof. □

15.2 Euclid's GCD Algorithm

The first part of this lecture is about computing the greatest common divisor of two integers. Our first algorithm for computing greatest common divisors follows immediately from two simple observations:

$$\gcd(m, n) = \gcd(m, n - m) \qquad \text{and} \qquad \gcd(n, 0) = n$$

The algorithm uses the first observation as a way to reduce the input and recurse; the second observation provides the base case.

```
SlowGCD(m, n):
    m ← |m|; n ← |n|
    if m < n
        swap m ↔ n
    while n > 0
        m ← m − n
        if m < n
            swap m ↔ n
    return m
```

The first few lines just ensure that $m \ge n \ge 0$. Each iteration of the main loop decreases one of the numbers by at least 1, so the running time is $O(m + n)$. This bound is tight in the worst case; consider the case $n = 1$. Unfortunately, this is terrible. The input consists of just $\log m + \log n$ bits; as a function of the input size, this algorithm runs in *exponential* time.

Let's think for a moment about what the main loop computes between swaps. We start with two numbers m and n and repeatedly subtract n from m until we can't any more. This is just a (slow) recipe for computing $m \bmod n$! That means we can speed up the algorithm by using mod instead of subtraction.

```
EuclidGCD(m, n):
    m ← |m|; n ← |n|
    if m < n
        swap m ↔ n
    while n > 0
        m ← m mod n      (⋆)
        swap m ↔ n
    return m
```

This algorithm swaps m and n at *every* iteration, because $m \bmod n$ is always less than n. This is usually called Euclid's algorithm, because the main idea is included in Euclid's *Elements*.[3]

The easiest way to analyze this algorithm is to work backward. First, let's consider the number of iterations of the main loop, or equivalently, the number times line ($\star$) is executed. To keep things simple, let's assume that $m > n > 0$, so the first three lines are redundant, and the algorithm performs at least one iteration. Recall that the Fibonacci numbers(!) are defined as $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$ for all $k > 1$.

Lemma 2. *If the algorithm performs k iterations, then $m \geq F_{k+2}$ and $n \geq F_{k+1}$.*

Proof (by induction on k): If $k = 1$, we have the trivial bounds $n \geq 1 = F_2$ and $m \geq 2 = F_3$.

Suppose $k > 1$. The first iteration of the loop replaces (m, n) with $(n, m \bmod n)$. Since the algorithm performs $k - 1$ more iterations, the inductive hypothesis implies that $n \geq F_{k+1}$ and $m \bmod n \geq F_k$. We've assumed that $m > n$, so $m \geq m + n(1 - \lfloor m/n \rfloor) = n + (m \bmod n)$. We conclude that $m \geq F_{k+1} + F_k = F_{k+1}$. $\square$

Theorem 1. EuclidGCD(m, n) *runs in $O(\log m)$ iterations.*

Proof: Let k be the number of iterations. Lemma 2 implies that $m \geq F_{k+2} \geq \phi^{k+2}/\sqrt{5} - 1$, where $\phi = (1 + \sqrt{5})/2$ (by the annihilator method). Thus, $k \leq \log_\phi(\sqrt{5}(m + 1)) - 2 = O(\log m)$. $\square$

What about the actual running time? Every number used by the algorithm has $O(\log m)$ bits. Computing the remainder of one b-bit integer by another using the grade-school long division algorithm requires $O(b^2)$ time. So crudely, the running time is $O(b^2 \log m) = O(\log^3 m)$. More careful analysis reduces the time bound to $\boxed{O(\log^2 m)}$. We can make the algorithm even faster by using a fast integer division algorithm (based on FFTs, for example).

15.3 Modular Arithmetic and Algebraic Groups

Modular arithmetic is familiar to anyone who's ever wondered how many minutes are left in an exam that ends at 9:15 when the clock says 8:54.

When we do arithmetic 'modulo n', what we're really doing is a funny kind of arithmetic on the elements of following set:

$$\boxed{\mathbb{Z}_n = \{0, 1, 2, \ldots, n - 1\}}$$

Modular addition and subtraction satisfies all the axioms that we expect implicitly:

- $\mathbb{Z}_n$ is *closed* under addition mod n: For any $a, b \in \mathbb{Z}_n$, their sum $a + b \bmod n$ is also in $\mathbb{Z}_n$

[3]However, Euclid's exposition was a little, erm, informal by current standards, primarily because the Greeks didn't know about induction. He basically said "Try one iteration. If that doesn't work, try three iterations." In modern language, Euclid's algorithm would be written as follows, assuming $m \geq n > 0$.

```
ActualEuclidGCD(m, n):
  if n | m
      return n
  else
      return n mod (m mod n)
```

This algorithm is *obviously* incorrect; consider the input $m = 3$, $n = 2$. Nevertheless, mathematics and algorithms students have applied 'Euclidean induction' to a vast number of problems, only to scratch their heads in dismay when they don't get any credit.

- Addition is *associative*: $(a + b \bmod n) + c \bmod n = a + (b + c \bmod n) \bmod n$.
- Zero is an additive *identity* element: $0 + a \bmod n = a + 0 \bmod n = a \bmod n$.
- Every element $a \in \mathbb{Z}_n$ has an *inverse* $b \in \mathbb{Z}_n$ such that $a+b \bmod n = 0$. Specifically, if $a = 0$, then $b = 0$; otherwise, $b = n - a$.

Any set with a binary operator that satisfies the closure, associativity, identity, and inverse axioms is called a *group*. Since Z_n is a group under an 'addition' operator, we call it an *additive* group. Moreover, since addition is commutative ($a+b \bmod n = b+a \bmod n$), we can call $(\mathbb{Z}_n, + \bmod n)$ is an *abelian* additive group.

What about multiplication? $\mathbb{Z}_n$ is closed under multiplication mod n, multiplication mod n is associative (and commutative), and 1 is a multiplicative identity, but some elements do not have multiplicative inverses. Formally, we say that $\mathbb{Z}_n$ is a *ring* under addition and multiplication modulo n.

If n is composite, then the following theorem shows that we can factor the ring $\mathbb{Z}_n$ into two smaller rings. The Chinese Remainder Theorem is named for Sun Tsu (or Sun Zi), the author of *the Art of War*, who proved a special case. (See the quotation for Lecture 1!)

The Chinese Remainder Theorem. *If $p \perp q$, then $\mathbb{Z}_{pq} \cong \mathbb{Z}_p \times \mathbb{Z}_q$.*

Okay, okay, before we prove this, let's define all the notation. The product $\mathbb{Z}_p \times \mathbb{Z}_q$ is the set of ordered pairs $\{(a, b) \mid a \in \mathbb{Z}_p, b \in \mathbb{Z}_q\}$, where addition, subtraction, and multiplication are defined as follows:

$$\begin{aligned}(a, b) + (c, d) &= (a + c \bmod p, b + d \bmod q)\\ (a, b) - (c, d) &= (a - c \bmod p, b - d \bmod q)\\ (a, b) \cdot (c, d) &= (ac \bmod p, bd \bmod q)\end{aligned}$$

It's not hard to check that $\mathbb{Z}_p \times \mathbb{Z}_q$ is a ring under these operations, where $(0, 0)$ is the additive identity and $(1, 1)$ is the multiplicative identity. The funky equal sign $\cong$ means that these two rings are *isomorphic*: there is a bijection between the two sets that is consistent with the arithmetic operations.

As an example, the following table describes the bijection between $\mathbb{Z}_{15}$ and $\mathbb{Z}_3 \times \mathbb{Z}_5$:

	0	1	2	3	4
0	0	6	12	3	9
1	10	1	7	13	4
2	5	11	2	8	14

For instance, we have $8 = (2, 3)$ and $13 = (1, 3)$, and

$$\begin{aligned}(2, 3) + (1, 3) &= (2 + 1 \bmod 3, 3 + 3 \bmod 5) = (0, 1) = 6 = 21 \bmod 15 = (8 + 13) \bmod 15.\\ (2, 3) \cdot (1, 3) &= (2 \cdot 1 \bmod 3, 3 \cdot 3 \bmod 5) = (2, 4) = 14 = 104 \bmod 15 = (8 \cdot 13) \bmod 15.\end{aligned}$$

Proof: The functions $n \mapsto (n \bmod p, n \bmod q)$ and $(a, b) \mapsto aq(q \bmod p) + bp(p \bmod q)$ are inverses of each other, and each map preserves the ring structure. □

We can extend the Chinese remainder theorem inductively as follows:

The Real Chinese Remainder Theorem. *Suppose $n = \prod_{i=1}^{r} p_i$, where $p_i \perp p_j$ for all i and j. Then $\mathbb{Z}_n \cong \prod_{i=1}^{r} \mathbb{Z}_{p_i} = \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_r}$.*

If we want to perform modular arithmetic where the modulus n is very large, we can improve the performance of our algorithms by breaking n into several relatively prime factors, and performing modular arithmetic separately modulo each factor.

So we can do modular addition, subtraction, and multiplication; what about division? As I said earlier, not every element of $\mathbb{Z}_n$ has a multiplicative inverse. The most obvious example is 0, but there can be others. For example, 3 has no multiplicative inverse in $\mathbb{Z}_{15}$; there is no integer x such that $3x \bmod 15 = 1$. On the other hand, 0 is the only element of $\mathbb{Z}_7$ without a multiplicative inverse:

$$1 \cdot 1 \equiv 2 \cdot 4 \equiv 3 \cdot 5 \equiv 6 \cdot 6 \equiv 1 \pmod 7$$

These examples suggest (I hope) that x has a multiplicative inverse in $\mathbb{Z}_n$ if and only if a and x are relatively prime. This is easy to prove as follows. If $xy \bmod n = 1$, then $xy + kn = 1$ for some integer k. Thus, 1 is an integer linear combination of x and n, so Lemma 1 implies that $\gcd(x, n) = 1$. On the other hand, if $x \perp n$, then $ax + bn = 1$ for some integers a and b, which implies that $ax \bmod n = 1$.

Let's define the set $\mathbb{Z}_n^*$ to be the set of elements if $\mathbb{Z}_n$ that have multiplicative inverses.

$$\boxed{\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid a \perp n\}}$$

It is a tedious exercise to show that $\mathbb{Z}_n^*$ is an abelian group under multiplication modulo n. As long as we stick to elements of this group, we can reasonably talk about 'division mod n'.

We denote the number of elements in $\mathbb{Z}_n^*$ by $\phi(n)$; this is called Euler's *totient* function. This function is remarkably badly-behaved, but there is a relatively simple formula for $\phi(n)$ (not surprisingly) involving prime numbers and division:

$$\boxed{\phi(n) = n \prod_{p \mid n} \frac{p-1}{p}}$$

I won't prove this formula, but the following intuition is helpful. If we start with $\mathbb{Z}_n$ and throw out all $n/2$ multiples of 2, all $n/3$ multiples of 3, all $n/5$ multiples of 5, and so on. Whenever we throw out multiples of p, we multiply the size of the set by $(p-1)/p$. At the end of this process, we're left with precisely the elements of $\mathbb{Z}_n^*$. *This is not a proof!* On the one hand, this argument throws out some numbers (like 6) more than once, so our estimate seems too low. On the other hand, there are actually $\lceil n/p \rceil$ multiples of p in $\mathbb{Z}_n$, so our estimate seems too high. Surprisingly, these two errors exactly cancel each other out.

15.4 Toward Primality Testing

In this last section, we discuss algorithms for detecting whether a number is prime. Large prime numbers are used primarily (but not exclusively) in cryptography algorithms.

A positive integer is *prime* if it has exactly two positive divisors, and *composite* if it has more than two positive divisors. The integer 1 is neither prime nor composite. Equivalently, an integer $n \geq 2$ is prime if n is relatively prime with every positive integer smaller than n. We can rephrase this definition yet again: n is prime if and only if $\phi(n) = n - 1$.

The obvious algorithm for testing whether a number is prime is *trial division*: simply try every possible nontrivial divisor between 2 and $\sqrt{n}$.

```
TrialDivPrime(n) :
  for d ← 1 to ⌊√n⌋
      if n mod d = 0
          return Composite
  return Prime
```

Unfortunately, this algorithm is horribly slow. Even if we could do the remainder computation in constant time, the overall running time of this algorithm would be $\Omega(\sqrt{n})$, which is exponential in the number of input bits.

This might seem completely hopeless, but fortunately most composite numbers are quite easy to detect as composite. Consider, for example, the related problem of deciding whether a given integer n, whether $n = m^e$ for any integers $m > 1$ and $e > 1$. We can solve this problem in polynomial time with the following straightforward algorithm. The subroutine $\text{Root}(n, i)$ computes $\lfloor n^{1/i} \rfloor$ essentially by binary search. (I'll leave the analysis as a simple exercise.)

```
ExactPower?(n):
  for i ← 2 to lg n
      if (Root(n, i))^i = n
          return True
  return False
```

```
Root(n, i):
  r ← 0
  for ℓ ← ⌈(lg n)/i⌉ down to 1
      if (r + 2^ℓ)^i ≤ n
          r ← r + 2^ℓ
  return r
```

To distinguish between arbitrary prime and composite numbers, we need to exploit some results about $\mathbb{Z}_n^*$ from group theory and number theory. First, we define the *order* of an element $x \in \mathbb{Z}_n^*$ as the smallest positive integer k such that $x^k \equiv 1 \pmod{n}$. For example, in the group

$$\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\},$$

the number 2 has order 4, and the number 11 has order 2. For any $x \in \mathbb{Z}_n^*$, we can partition the elements of $\mathbb{Z}_n^*$ into equivalence classes, by declaring $a \sim_x b$ if $a \equiv b \cdot x^k$ for some integer k. The size of every equivalence class is exactly the order of x. Since the equivalence classes must be disjoint, we can conclude that $\boxed{\text{the order of any element divides the size of the group}}$. We can express this observation more succinctly as follows:

Euler's Theorem. $a^{\phi(n)} \equiv 1 \pmod{n}$.[4]

The most interesting special case of this theorem is when n is prime.

Fermat's Little Theorem. *If p is prime, then* $a^p \equiv a \pmod{p}$.[5]

This theorem leads to the following efficient *pseudo*-primality test.

[4]This is not Euler's only theorem; he had thousands. It's not even his most famous theorem. His *second* most famous theorem is the formula $v + e - f = 2$ relating the vertices, edges and faces of any planar map. His most famous theorem is the magic formula $e^{\pi i} + 1 = 0$. Naming something after a mathematician or physicist (as in 'Euler tour' or 'Gaussian elimination' or 'Avogadro's number') is considered a high compliment. Using a lower case letter ('abelian group') is even better; abbreviating ('volt', 'amp') is better still. The number e was named after Euler.

[5]This is not Fermat's only theorem; he had hundreds, most of them stated without proof. Fermat's Last Theorem wasn't the last one he published, but the last one proved. Amazingly, despite his dislike of writing proofs, Fermat was almost always right. In that respect, he was *very* different from you and me.

```
FermatPseudoPrime(n):
    choose an integer a between 1 and n − 1
    if a^n mod n ≠ a
        return Composite!
    else
        return Prime?
```

In practice, this algorithm is both fast and effective. The (empirical) probability that a random 100-digit composite number will return PRIME? is roughly 10^{-30}, even if we always choose $a = 2$. Unfortunately, there are composite numbers that always pass this test, no matter which value of a we use. A *Carmichael number* is a composite integer n such that $a^n \equiv a \pmod n$ for every integer a. Thus, Fermat's Little Theorem can be used to distinguish between two types of numbers: (primes and Carmichael numbers) and everything else. Carmichael numbers are extremely rare; in fact, it was proved only a decade ago that there are an infinite number of them.

To deal with Carmichael numbers effectively, we need to look more closely at the structure of the group $\mathbb{Z}_n^*$. We say that $\mathbb{Z}_n^*$ is *cyclic* if it contains an element of order $\phi(n)$; such an element is called a *generator*. Successive powers of any generator *cycle* through every element of the group in some order. For example, the group $\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$ is cyclic, with two generators: 2 and 5, but $\mathbb{Z}_{15}^*$ is not cyclic. The following theorem completely characterizes which groups $\mathbb{Z}_n^*$ are cyclic.

The Cycle Theorem. *$\mathbb{Z}_n^*$ is cyclic if and only if $n = 2$, 4, p^e, or $2p^e$ for some odd prime p and positive integer e.*

This theorem has two relatively simple corollaries.

The Discrete Log Theorem. *Suppose $\mathbb{Z}_n^*$ is cyclic and g is a generator. Then $g^x \equiv g^y \pmod n$ if and only if $x \equiv y \pmod{\phi(n)}$.*

Proof: Suppose $g^x \equiv g^y \pmod n$. By definition of 'generator', the sequence $\langle 1, g, g^2, \ldots\rangle$ has period $\phi(n)$. Thus, $x \equiv y \pmod{\phi(n)}$. On the other hand, if $x \equiv y \pmod{\phi(n)}$, then $x = y + k\phi(n)$ for some integer k, so $g^x = g^{y+k\phi(n)} = g^y \cdot (g^{\phi(n)})^k$. Euler's Theorem now implies that $(g^{\phi(n)})^k \equiv 1^k \equiv 1 \pmod n$, so $g^x \equiv g^y \pmod n$. □

The $\sqrt{1}$ Theorem. *Suppose $n = p^e$ for some odd prime p and positive integer e. The only elements $x \in \mathbb{Z}_n^*$ that satisfy the equation $x^2 \equiv 1 \pmod n$ are $x = 1$ and $x = n - 1$.*

Proof: Obviously $1^2 \equiv 1 \pmod n$ and $(n-1)^2 = n^2 - 2n + 1 \equiv 1 \pmod n$.

Suppose $x^2 \equiv 1 \pmod n$ where $n = p^e$. By the Cycle Theorem, $\mathbb{Z}_n^*$ is cyclic. Let g be a generator of $\mathbb{Z}_n^*$, and suppose $x = g^k$. Then we immediately have $x^2 = g^{2k} \equiv g^0 = 1 \pmod{p^e}$. The Discrete Log Theorem implies that $2k \equiv 0 \pmod{\phi(p^e)}$. Since p is and odd prime, we have $\phi(p^e) = (p-1)p^{e-1}$, which is even. Thus, the equation $2k \equiv 0 \pmod{\phi(p^e)}$ has just two solutions: $k = 0$ and $k = \phi(p^e)/2$. By the Cycle Theorem, either $x = 1$ or $x = g^{\phi(n)/2}$. Because $x = n - 1$ is also a solution to the original equation, we must have $g^{\phi(n)/2} \equiv n - 1 \pmod n$. □

This theorem leads to a different *pseudo*-primality algorithm:

```
Sqrt1PseudoPrime(n):
    choose a number a between 2 and n − 2
    if a^2 mod n = 1
        return Composite!
    else
        return Prime?
```

As with the previous pseudo-primality test, there are composite numbers that this algorithm cannot identify as composite: powers of primes, for instance. Fortunately, however, the set of composites that always pass the $\sqrt{1}$ test is disjoint from the set of numbers that always pass the Fermat test. In particular, Carmichael numbers *never* have the form p^e.

15.5 The Miller-Rabin Primality Test

The following randomized algorithm, adapted by Michael Rabin from an earlier deterministic algorithm of Gary Miller*, combines the Fermat test and the $\sqrt{1}$ test. The algorithm repeats the same two tests s times, where s is some user-chosen parameter, each time with a random value of a.

```
MillerRabin(n):
  write n − 1 = 2^t u where u is odd
  for i ← 1 to s
      a ← Random(2, n − 2)
      if EuclidGCD(a, n) ≠ 1
          return Composite!          ⟨⟨obviously!⟩⟩
      x_0 ← a^u mod n
      for j ← 1 to t
          x_j ← x_{j−1}^2 mod n
          if x_j = 1 and x_{j−1} ≠ 1 and x_{j−1} ≠ n − 1
              return Composite!      ⟨⟨by the √1 Theorem⟩⟩
      if x_t ≠ 1                     ⟨⟨x_t = a^{n−1} mod n⟩⟩
          return Composite!          ⟨⟨by Fermat's Little Theorem⟩⟩
  return Prime?
```

First let's consider the running time; for simplicity, we assume that all integer arithmetic is done using the quadratic-time grade school algorithms. We can compute u and t in $O(\log n)$ time by scanning the bits in the binary representation of n. Euclid's algorithm takes $O(\log^2 n)$ time. Computing $a^u \mod n$ requires $O(\log u) = O(\log n)$ multiplications, each of which takes $O(\log^2 n)$ time. Squaring x_j takes $O(\log^2 n)$ time. Overall, the running time for one iteration of the outer loop is $O(\log^3 n + t \log^2 n) = O(\log^3 n)$, since $t \leq \lg n$. Thus, the total running time of this algorithm is $\boxed{O(s \log^3 n)}$. If we set $s = O(\log n)$, this running time is polynomial in the size of the input.

Fine, so it's fast, but is it correct? Like the earlier pseudoprime testing algorithms, a prime input will always cause MillerRabin to return Prime?. Composite numbers, however, may not always return Composite!; because we choose the number a at random, there is a small probability of error.[6] Fortunately, the error probability can be made ridiculously small—in practice, less than the probability that random quantum fluctuations will instantly transform your computer into a kitten—by setting $s \approx 1000$.

Theorem 2. *If n is composite,* MillerRabin(n) *returns* Composite! *with probability at least* $1 - 2^{-s}$.

[6]If instead, we try all possible values of a, we obtain an exact primality testing algorithm, but it runs in exponential time. Miller's original deterministic algorithm examined every value of a in a carefully-chosen subset of $\mathbb{Z}_n^*$. If the Extended Riemann Hypothesis holds, this subset has logarithmic size, and Miller's algorithm runs in polynomial time. The Riemann Hypothesis is a century-old open problem about the distribution of prime numbers. A solution would be at least as significant as proving Fermat's Last Theorem or P≠NP.

Proof: First, suppose n is not a Carmichael number. Let F be the set of elements of $\mathbb{Z}_n^*$ that pass the Fermat test:

$$F = \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\}.$$

Since n is not a Carmichael number, F is a *proper* subset of $\mathbb{Z}_n^*$. Given any two elements $a, b \in F$, their product $a \cdot b \bmod n$ in $\mathbb{Z}_n^*$ is also an element of F:

$$(a \cdot b)^{n-1} \equiv a^{n-1} b^{n-1} \equiv 1 \cdot 1 \equiv 1 \pmod{n}$$

We also easily observe that 1 is an element of F, and the multiplicative inverse (mod n) of any element of F is also in F. Thus, F is a proper *subgroup* of $\mathbb{Z}_n^*$, that is, a proper subset that is also a group under the same binary operation. A standard result in group theory states that if F is a subgroup of a finite group G, the number of elements of F divides the number of elements of G. (We used a special case of this result in our proof of Euler's Theorem.) In our setting, this means that $|F|$ divides $\phi(n)$. Since we already know that $|F| < \phi(n)$, we must have $|F| \leq \phi(n)/2$. Thus, at most half the elements of $\mathbb{Z}_n^*$ pass the Fermat test.

The case of Carmichael numbers is more complicated, but the main idea is the same: at most half the possible values of a pass the $\sqrt{1}$ test. See CLRS for further details. □