One of the most important domains for randomized algorithms is **number theory**, particularly with its applications to **cryptography**. Today we'll look at one of the several fast randomized algorithms for **primality testing**, but let's first set some context.

When we operate on very large integers, our normal measure of the input size is the number of bits required to store the number: $\log n$ for an integer n. In practice we might store the integer in a list of machine words, as in the Java `Bignum` class, but since only a constant number of bits will fit in a machine word, $O(\log n)$ is still a good measure of the size of n.

We can add, subtract, multiply, or divide large numbers in time linear or nearly linear in their size. We can run the **Euclidean algorithm** on two numbers and thus find whether they have a common factor. But other operations that we take for granted on small integers appear to become much harder on large ones.

How do we test small numbers to see whether they are prime? Using the definition, we get the **trial division** method, where we try every number from 2 through $\sqrt{n}$ to see whether it divides n exactly. This takes $O(\sqrt{n})$ time, which looks promising until you realize that this time is *exponential* in $\log n$. Until recently, no *deterministic* algorithm was known that ran in $(\log n)^{O(1)}$ time. This theoretical challenge was met in 2004 by Agrawal, Kayal, and Saxena, who have an algorithm that does it in about $O((\log n)^{15/2})$ and probably somewhat faster, about $O((\log n)^6)$. (A very readable August 2005 version of the paper is posted on Agrawal's web site at IIT Kanpur.) But their algorithm is not used in practice, because the randomized ones are much faster and give arbitrarily high confidence in their answers.

One of the most important uses for primality testing is in **generating** primes, particularly for use in public-key cryptography. The **prime number theorem** says that the number of primes less than x is about $\frac{x}{\ln x}$, so if you can test $O(k)$ random k-bit numbers you will probably find a prime.

The problem of **factoring** a large composite number appears to be much harder than primality testing. The security of the RSA cryptosystem depends absolutely on this difficulty, because the public key is the product of two large prime numbers and messages can be decoded with knowledge of the two individual primes.

In 1994 Peter Shor published an algorithm that would factor numbers in polynomial time on a **quantum computer**, a generalization of randomized computation that may or may not ever be practical. Researchers at IBM in 2001 were able to implement the algorithm to factor the number 15 (turns out it's 3 times 5), but scaling it up poses technical problems.

Understanding the randomized algorithm requires a bit of algebra. If n is any number, $\mathbf{Z}_n$ is the ring of integers modulo n. We define $\mathbf{Z}_n^*$ to be the integers modulo n that are relatively prime to n, viewed as a group under multiplication. If n is prime, $\mathbf{Z}_n^*$ is a group of $n-1$ elements. (It is a **cyclic group**, meaning that there is an element g such that every other element is a power of g. This is the meaning of "discrete logarithm" – if $g^a = b$ we say that b is the log of a (with respect to g).)

If n is not prime, some of the numbers modulo n are not relatively prime to n, and $\mathbf{Z}_n^*$ has fewer than $n-1$ elements. For example, if $n = pq$ and p and q are prime, $\mathbf{Z}_n^*$ has $(p-1)(q-1)$ elements. (By the Chinese Remainder Theorem, $\mathbf{Z}_n$ is isomorphic to the product of $\mathbf{Z}_{p^e}$ for each maximal prime-power divisor p^e of n, and $\mathbf{Z}_n^*$ is similarly the product of the groups $\mathbf{Z}_{p^e}^*$.)

The basic idea of the randomized algorithm is simple. Choose a random number a in $\mathbf{Z}_n$, and compute a^{n-1} modulo n. (How? Repeated squaring, reducing each product modulo n so we only deal with numbers in $\mathbf{Z}_n$. This takes $O(\log n)$ multiplications.)

If n is prime, then as long as a is in $\mathbf{Z}_n^*$, a^{n-1} will be 1. This is because any element of a group, raised to the order of the group, gives the identity. Our random choice *could* be 0, of course, but that is vanishingly unlikely.

If n is composite, however, we have a reasonable hope that the random number a, raised to a power that is *not* the order of its group, should be random. If a^{n-1} is not 1, we have a **proof** that n is not prime, though this still gives us no clue how to factor it. The technical difficulty of the randomized tests is in determining when we could get a "false positive" for primality – a number a where $a^{n-1} = 1$ even though n is composite.

Unfortunately, perhaps, there are composite numbers n for which any a in $\mathbf{Z}_n^*$ satisfies $a^{n-1} =$ modulo n. These are called **Carmichael numbers** and there are infinitely many of them, each (as it happens) a product of exactly three primes. If we choose a random a in $\mathbf{Z}_n$, we have a small chance of finding one that has a common factor with n, but for the product of three *large* primes this is vanishingly unlikely.

The smallest Carmichael number is 561, which factors as $3 \cdot 11 \cdot 17$. (Another interesting one is "Ramanujan's taxicab number", $1729 = 7 \cdot 13 \cdot 19$, which is also the smallest number that is the sum of two cubes in two different ways. But I digress.)

We can think of a number in $\mathbf{Z}_{561}$ as a triple of numbers in $\mathbf{Z}_3 \times \mathbf{Z}_{11} \times \mathbf{Z}_{17}$. For example, 43 can be viewed as $(1, 10, 9)$ because $43 = 1$ mod 3, $43 = 10$ mod 11, and $43 = 9$ mod 17. The numbers that are not in $\mathbf{Z}_{561}^*$ are those rare numbers that have a zero in one of the three components. When we raise a number to a power, we raise each component to that power with respect to its particular modulus.

What is $(1, 10, 9)^{560}$? Modulo a prime p, any number to the $p-1$ power is 1. So $(1, 10, 9)^{560} = (1^{560}, (10^{10})^{56}, (9^{16})^{35}) = (1, 1, 1)$. As you can see, because $3 - 1$, $11 - 1$, and $17 - 1$ all happen to divide 560, *any* $a \in \mathbf{Z}_{561}^*$ satisfies $a^{560} = 1$. So our simple randomized test does not work for this prime, or in fact for any Carmichael number.

The **Miller-Rabin** test is a randomized variant of this simple test that actually does work. We take our random a and check whether $a^{n-1} = 1$ modulo n. If this is false, of course we know that n is not prime. If it is true, we write $n - 1$ as $2^r s$, where s is odd. (We know that r is at least 1, because if $n - 1$ were odd we would know that n is even and thus not prime.)

Our test is to calculate the numbers a^s, a^{2s}, a^{4s}, ..., a^{n-1}, each modulo n, and see whether any of them *minus one* has a nontrivial common factor with n (by the Euclidean Algorithm). If any of them does, we know that n is composite. The key to the algorithm is that *if* n is composite and $a^{n-1} = 1$, it is reasonably likely that there will be a common factor with one of these numbers $a^{2^i s} - 1$.

Here we sketch the argument that if n is composite and $a^{n-1} = 1$ modulo n, one of the numbers $a^{2^i s} - 1$ is likely to have a nontrivial common factor with n. (The full argument is in [CLRS].) We'll assume that $n = p_1^{e_1} \ldots p_k^{e_k}$ where each p_i is prime. We view a as a sequence $(a_1 \ldots, a_k)$. If any a_i is divisible by p_i, a^{n-1} will not be 1 and we will find that n is composite. So each a_i is in $\mathbf{Z}^*_{p_i^{e_i}}$.

It is unlikely that $a^s = 1$ modulo n, because for each i, a_i^s is equally likely to be any of the s'th powers in $\mathbf{Z}^*_{p_i{}^{e_i}}$, and there are at least two of these because they include -1 and 1. So the sequence $a^s, a^{2s}, \ldots, a^{n-1}$ probably starts as something other than $(1, 1, \ldots, 1)$ but definitely finishes as $(1, 1, \ldots, 1)$. What we need to happen is that one of these numbers is a sequence that contains *some* but *not all* entries of 1. The argument we'll skip (which you may use on HW#3 without further proof) says that the chance of this happening is at least $1/2$ – the only way it could fail to happen is if the sequence starts as all ones or if all the non-ones become ones *at the same time*.

If some $a^{2^i s}$ is a sequence with an entry of 1 and an entry that is not 1, then $a^{2^i s} - 1$ has an entry of 0 and an entry that is not 0. That means that one of the primes p_i divides it, but n itself does not. So it shares a factor with n.

As with the Schwartz algorithm, we have a test with one-sided error – we might identify a composite number as prime, but not a prime number as composite. We can increase our confidence in our answer by running several independent trials – running the algorithm with several different a's and the same n. If n is prime, all the trials will say so. If n is composite, the chance that t independent trials will say that it is prime is at most 2^{-t}.

Before the deterministic poly-time primality test, it was still known that there is a **Las Vegas** poly-time algorithm for primality – one that always gets the right answer but only has *expected* polynomial time. This was because Adelman and Huang developed poly-time Monte Carlo algorithm with one-sided error *in the other direction*. Their algorithm always identifies composite numbers as composite but has a small chance of identifying a prime number as composite. The Las Vegas algorithm simply consists of running both Miller-Rabin and Adelman-Huang tests on n until one of them gives an answer than can be absolutely relied upon. There is no limit in principle on how many trials it might take for this to happen, but the *expected* number is a constant (no more than 2 if the individual success probabilities are at least $1/2$).

There is a problem with "expected value" results for running time – by themselves they give us no guarantee that any individual run of the algorithm might not be arbitrarily bad. If we are worried about the "worst plausible case", rather than the absolute worst case, we would like to have results of the form "the time is less than $T(n)$ with probability $1 - \epsilon$". In the remainder of this lecture we'll look at some general techniques to get results of this kind.

First a general example that we'll analyze in an *ad hoc* way. Suppose we have n bins and m balls, and we throw each ball independently into a uniformly chosen random bin. (This is the same situation as the **coupon collector's problem**, where we ask how large m should be before it is likely that all the bins have at least one ball.) Here our problem is to estimate the number of balls in the *largest* bin (the one with the most balls) in the case $m = n$.

The *average* number of balls in each bin is exactly 1, of course, but it's very unlikely that they wind up distributed exactly evenly. What we'll do here is find a number k such that the probability that *any* of the bins gets k or more balls is small. Another argument, that we'll omit here, shows that it's likely that some bin gets at least k' balls, where $k' = \Omega(k)$, giving us a tight asymptotic bound on the largest number.

Let's fix k and compute the probability that a particular bin, number i, gets k or more balls. There are $\binom{n}{k}$ different sets of exactly k balls, and each set gets sent entirely to bin i with probability n^{-k}. These various events overlap, but we can *overestimate* the total probability that bin i gets k balls as $\binom{n}{k}n^{-k}$.

We can estimate $\binom{n}{k}$ as between $(n/k)^k$ and $(ne/k)^k$. The lower bound is true because $\binom{n}{k}$ is $\frac{n}{k} \cdot \frac{n-1}{k-1} \cdot \ldots \cdot \frac{n-k+1}{k-k+1}$ and each of these k factors is at least n/k. The upper bound is true because $\binom{n}{k} \leq n/k!$ and by Stirling's formula, $k!$ is at most $(k/e)^k$.

So our upper bound on the probability that bin i gets k or more balls is $(ne/k)^k n^{-k} = (e/k)^k$. This gets smaller as k increases. We need to pick k large enough that it is at most n^{-2}. Then by the **union bound**, the probability that *any* bin gets this many items is at most $1/n$.

The proper value of k turns out to be about $\frac{2e \ln n}{\ln \ln n}$. The function $\frac{\log n}{\log \log n}$ tends to show up a lot in this sort of argument, in part because (up to asymptotics) it is the inverse function of n^n and $n!$.

If we know that a random variable never takes on a negative value, then there is an obvious limit to how often it can be much greater than its mean, given by **Markov's Inequality**.

Suppose you are designing a random variable X with $E[X] = \mu$, and you want to maximize the probability that X is at least as great as some bound α. It doesn't help you to ever make it more than α, or between 0 and α, so you are best off making it α with some probability p and 0 with probability $1 - p$. This means that the expected value $E[X]$ is $p\alpha$, so the value of p that makes $E[X] = \mu$ is just μ/α.

More formally, the probability that $X \geq \alpha$ cannot be greater than μ/α because then the expected value calculation would include a component of probability greater than μ/α of an event with $X \geq \mu$, which would make the expected value greater than μ (since no other component could subtract from this one as X is non-negative).

We can apply Markov's Inequality to get a better bound on the tail of a random variable if we know its **variance**. If X is a random variable with mean μ, its variance is the expected value of $(X - \mu)^2$. We write the variance as σ^2, because it is convenient to work with its square root, the **standard deviation** of x.

Suppose you are designing a random variable with mean μ and variance σ^2, and you want to maximize the probability that X is at least a certain distance t away from its mean. The natural thing is to make X either $\mu + t$, μ, or $\mu - t$, because it adds unnecesarily to the variance to make X further than t from the mean, or to have it deviate at all from the mean if it won't help the desired probability.

To make the mean μ, the probability of $\mu + t$ and $\mu - t$ must be the same – call it $p/2$. Thus $(X - \mu)^2$ is equal to t^2 with probability p and equal to 0 otherwise, so the variance is pt^2 and we make the variance equal to σ^2 by setting p to $(\sigma/t)^2$.

Formally, if the probability that $|X - \mu| \geq t$ is p, this is also the probability that the non-negative random variable $(X - \mu)^2$, which we know to have mean σ^2, is at least t^2. By Markov's Inequality, p cannot be greater than $(\sigma/t)^2$. We call this result **Chebyshev's Inequality**.

Another way to say this is that the probability of a random variable being k or more standard deviations away from its mean is at most $1/k^2$. You may recall that the probability of a **normal** random variable being k or more standard deviations away from its mean decreases exponentially with k. In the next lecture we'll look at **Chernoff Bounds**, which show that a similar exponential bound holds for the sum of independent copies of **binomial** random variables.

In the next lecture we'll also look at three ways to compute the **median** of a set of numbers, and use Chebyshev's Inequality to analyze the one that is (on average) the fastest.