CS265/CME309: Randomized Algorithms and Probabilistic Analysis
Lecture #3: Primality Testing

Gregory Valiant (Updated by Mary Wootters, 2020) *

August 2, 2020

1 Introduction

Prime numbers are extremely useful, and are an essential input to many algorithms in large part due to the algebraic structure of arithmetic modulo a prime. In everyday life, perhaps the most frequent use for prime numbers is in RSA encryption, which requires quite large primes (typically ≥ 128-bits long). Fortunately, there are lots of primes—for large n, the probability that a random integer less than n is prime is roughly $1/\log n$. (All logarithms are to the base e unless otherwise noted.) Rather tight bounds on the density of primes are known, and have been improved over the past century:

Theorem 1 ([7]). *For all $n \geq 55$, the number of prime numbers less than n, denoted $\pi(n)$ lies in the range*

$$\frac{n}{2+\log n} < \pi(n) < \frac{n}{-4+\log n}.$$

Thus a random number chosen between 1 and 2^{128} is prime with probability at least $1/100$ (and is larger if one makes sure not to pick a number that is even, or has small prime divisors). The question is how to efficiently *check* whether a given number is prime.

2 Brief Chronology of Primality Testing

- 200 BC: Eratosthenes of Cyrene (Greek mathematician, poet, astronomer, etc.) described the *prime number sieve* for finding all the prime numbers up to a certain value. (He also calculated the circumference of the earth...)
- 1976: At roughly the same time, Miller [5] and Rabin [6] came up with the randomized algorithm that we will describe below. Rabin explicitly formulated it as a randomized algorithm (which is frequently used in practice today). Rabin's algorithm is always correct when the input is a prime, and has a (small) probability of error when the input is a composite. Miller

described essentially the same algorithm in a deterministic fashion, and proved that if the Extended Riemann Hypothesis (ERH) holds, then the algorithm will have runtime bounded by polylog n, where n is the number one is trying to test for primality.

- 1977: Solovay and Strassen [8] described an alternate randomized algorithm, with similar properties to the Rabin/Miller algorithm, which is also employed today.
- 1987: Adleman and Huang [1] described an efficient randomized algorithm that is always correct on *composite* inputs, and has a small probability of error when the input is prime. When combined with either of the above algorithms, this yields a *Las Vegas* style algorithm for primality testing, which always outputs the correct answer, and has expected runtime polynomial in the length of the input.
- 1999: Agrawal and Biswas [2] gave a new type of randomized primality test, based on the fact that, as polynomials, for all integers a, $(x-a)^n \equiv x^n - a \mod n$, if, and only if n is prime.
- 2002: Agrawal, Kayal, and Saxena [3] described a polynomial time *deterministic* algorithm for primality testing, which proceeds by essentially de-randomizing the algorithm proposed in 1999.

3 Algebra Refresher

We provide a brief review of the key definitions and facts from algebra upon which the primality testing algorithms are based.

Definition 1. *A* group *is a set S together with a binary operation "*" that maps an (ordered) pair of elements of S to a third element of S, and satisfies four conditions:*

- *closure: for all $a, b \in S$, $a * b \in S$.*
- *associativity: for all $a, b, c \in S$, $a * (b * c) = (a * b) * c$.*
- *identity: there exists an element e such that for all $a \in S$, $a * e = e * a = a$.*
- *inverse: for every $a \in S$, there exists an element b s.t. $a * b = b * a = e$, where e is the identity element.*

Definition 2. *For an integer n, the group $\mathbb{Z}_n^+$ is defined to be the group consisting of the integers $0, 1, \ldots, n-1$, with the group operation being addition, modulo n.*

Definition 3. *For an integer n, the group $\mathbb{Z}_n^*$ is defined to be the group consisting of all positive integers $a < n$ satisfying $gcd(a, n) = 1$, with the group operation being multiplication, modulo n.*

In the above definition, it should be clear why we need the restriction that $gcd(a, n) = 1$: for example, if $n = 6$, the numbers 2 and 3 do not have any (multiplicative) inverse element modulo 6.

Fact 4 (Lagrange's Theorem)**.** *For any group G, and any* subgroup *$H \subset G$, $|H|$ divides $|G|$.*

A relevant and trivial corollary of the above fact (which we will rely on) is that if $H \neq G$, then $|H| \leq |G|/2$.

Definition 5. *A group is called* cyclic *if there exists an element x such that the set $\{x, x^2, x^3, x^4, ...\}$ is equal to the entire group, and such an x is called a* generator *of the group.*

$\mathbb{Z}_n^+$ is cyclic, as the number 1 generates the group. A bit of group theory lets one show the following fact:

Fact 6. *For n prime, $\mathbb{Z}_n^*$ is cyclic. And, more generally, every subgroup of the multiplicative group of any field will be cyclic.*

The above fact, together with the observation that for n prime, $|\mathbb{Z}_n^*| = n - 1$ yields the following useful theorem, due to Fermat:

Fact 7 (Fermat's Little Theorem)**.** *If n is prime, then for all $x \in \mathbb{Z}_n^*$,*

$$x^{n-1} \equiv 1 \mod n.$$

Fermat's Little Theorem is *not* an if, and only if statement. There exist composite numbers—known as *Carmichael numbers*—such that for all $x \in \mathbb{Z}_n^*$, $x^{n-1} \equiv 1 \mod n$. The smallest Carmichael number is $561 = 3 * 11 * 17$. Although Carmichael numbers are relatively rare–it was (relatively) recently shown that there are an infinite number of such numbers; in fact, there are at least $n^{2/7}$ Carmichael numbers less than any given number n. [4]. [Although Erdos conjectured that there are an infinite number of Carmichael numbers, this was only rigorously proved in 1994, and their asymptotic density is still not well understood.]

In contrast to the above, the following polynomial version of Fermat's Little Theorem *is* a characterization of primes (which we will see how to turn into a randomized primality test in the next homework):

Fact 8 (Fermat's Little Theorem for Polynomials)**.** *n is prime if, and only if, for all $a \in \mathbb{Z}_n^*$, the following polynomial equation holds: $(x - a)^n \equiv x^n - a \mod n$.*

4 Algorithms for Checking Primality

The simplest algorithm for checking the primality of an integer n is to simply check each integer $1, 2, \ldots, \sqrt{n}$ to see if it divides n. While there are slightly more clever variants of this type of search (for example, only checking the prime factors), they all have the property that their runtime is super-polynomial in the length of the representation of n, which is prohibitive in practice even when searching for modest-sized primes.

4.1 Fermat's Test

Fermat's Little Theorem (Fact 7) motivates the following very simple primality test, known as *Fermat's Test*

Algorithm 9. FERMAT'S TEST
Given n:

- *Choose $x \in \{1, \ldots, n-1\}$ uniformly at random.*
- *If $x^{n-1} = 1 \mod n$ output* ***prime*** *else output* ***composite.***

The runtime of the algorithm is dominated by the computation of $x^{n-1} \mod n$, which can be done by repeated squaring, involving at most $O(\log n)$ multiplications of $O(\log n)$-bit numbers.

If n is prime, then Fermat's Little Theorem guarantees that the above algorithm will correctly output that n is prime. If n is a Carmichael number, then the above algorithm will obviously fail, and will falsely say that n is a prime if we choose an x that is relatively prime to n. Nevertheless, the following proposition shows that Carmichael numbers are the *only* bad inputs for this algorithm, in that for any other composite input, the algorithm will correctly output "composite" with probability at least $1/2$ (which can be boosted to $1 - 1/2^d$ be repeating the algorithm d times).

Proposition 10. *For any n that is a composite, but is not a Carmichael number, the probability that the above algorithm outputs "composite" is at least $1/2$.*

Proof. First note that if the x chosen by the algorithm satisfies $gcd(n, x) \neq 1$, then $x^{n-1} \neq 1 \mod n$ (why?) and the algorithm is successful. We now argue that even if $gcd(n, x) = 1$, the probability that the algorithm correctly deduces that n is composite is at least $1/2$.

We begin by defining the subgroup $H := \{x : x \in \mathbb{Z}_n^* \text{ and } x^{n-1} = 1 \mod n\}$. [It is easy to check that H is, in fact a group.] Assuming that n is composite, but not a Carmichael number, it follows that there exists some $a \in \mathbb{Z}_n^*$ such that $a^{n-1} \neq 1 \mod n$, hence $a \notin H$, and thus $H \neq \mathbb{Z}_n^*$ is a proper subgroup. By Lagrange's Theorem (Fact 4), it follows that $|H| \leq \frac{1}{2}|\mathbb{Z}_n^*|$, and hence with probability at least $1/2$ the randomly chosen x will be a witness to the compositeness of n. □

Because Carmichael numbers are far less common than primes, the above algorithm is a reasonable approach to quickly finding primes, and is commonly used. The main drawback is that it fails for Carmichael numbers, and hence is not suitable for applications in which there is a severe penalty for returning a composite number (e.g. when cryptographic properties are lost because of the compositeness of one's secret keys).

Note: At this point we are done with the material required for *before* Class 3. The following notes (Section 4.2 and afterwards) are for the stuff we will talk about *during* Class 3. It's okay to come to class without having read it, and in fact we will try to derive it ourselves during class. The notes are here for reference after class.

4.2 The Rabin/Miller Algorithm

The Rabin/Miller Algorithm relies on the following Corollary to the fact that $\mathbb{Z}_n^*$ is cyclic for prime n:

Corollary 11. *For prime n, if $a \in \mathbb{Z}_n^*$ with $a^2 = 1$, then either $a = 1$ or $a = -1$.*

Proof. For $n = 2$, the claim trivially holds. Otherwise, note that $n - 1$ must be even. Let g be a generator of $\mathbb{Z}_n^*$, hence $g^1, g^2, \ldots, g^{n-1}$ are all distinct, with $g^{n-1} = 1$, and for any $j > n - 1$, $g^j = g^{j \mod n-1}$. Hence if $a^2 = (g^j)^2 = 1$, it must hold that $2j \equiv 0 \mod n - 1$, and hence the only two roots of 1 are g^{n-1} and $g^{(n-1)/2}$. (And hence these two roots must be ± 1.) □

As we will show, if n is an odd composite number that is not a power of a prime, then 1 has more than 2 square roots in $\mathbb{Z}_n^*$. The Rabin/Miller algorithm is a clever way of searching for such nontrivial square roots of 1.

Algorithm 12. RABIN/MILLER PRIMALITY TEST
Given n:

- *If n is even, or is a power of a prime, then output* ***composite.*** *[Exercise: How does one check this efficiently?]*
- *Otherwise, find k, m s.t. $n-1 = 2^k m$ for some odd number m.*
- *Choose $x \in \{1, \ldots, n-1\}$ uniformly at random, and compute the list of k numbers $(x^m, x^{2m}, x^{2^2 m}, \ldots, x^{2^k m} \mod n)$.*
- *If $x^{2^k m} \not\equiv 1 \mod n$, then n fails Fermat's Test, so output* ***composite.***
- *Check whether any of the k numbers in the list is a non-trivial square-root of unity. Specifically, for $i = 0, 1, \ldots, k-1$, check whether $\left(x^{2^i m}\right)^2 = 1$, but $x^{2^i m} \neq \pm 1$. If any such non-trivial root of 1 was found, output* ***composite,*** *else output* ***prime.***

Theorem 2. *If n is prime, the Rabin/Miller test will always output "prime", and if n is composite, it will output "composite" with probability at least $1/2$. Additionally, the runtime is polynomial in $\log n$ [the representation size of the number n].*

We leave the analysis of the runtime as an exercise, and focus on proving that the algorithm succeeds with the claimed probability. In the case that n is prime, Corollary 11 guarantees that the algorithm will always output "prime". We now show that if n is composite, and is not of the form $n = p^r$ for some prime p, then with probability at least $1/2$ over the choice of random $x \in \{1, \ldots, n-1\}$, the algorithm will find a nontrivial square root of 1 (a witness to the compositeness of n) and hence will output "composite".

At a high level, the proof will follow by carefully constructing a proper subgroup $S \subset \mathbb{Z}_n^*$, and then arguing that if the algorithm fails, it must have been the case that the randomly selected element x was actually in the set S. Since S is a proper subgroup, however, $|S| \leq |\mathbb{Z}_n^*|$, and hence $\Pr[x \in S] \leq 1/2$, implying the theorem.

We now begin to define the construction of the special proper subgroup S.

Definition 13. *For an integer r, define $S_r = \{x \in \mathbb{Z}_n^* : x^r = \pm 1 \mod n\}$.*

It is not hard to show that for any r, S_r is closed under multiplication and if $x \in S_r$ then the inverse of x will also be in S_r, and hence it is a subgroup of $\mathbb{Z}_n^*$. The following lemma gives a condition for S_r being a *proper* subgroup.

Lemma 14. *Assume n is composite and not a power of a prime. Given an integer r, if there is some $x \in \mathbb{Z}_n^*$ for which $x^r = -1 \mod n$, then S_r is a proper subgroup of $\mathbb{Z}_n^*$, and by Lagrange's Theorem has size $|S_r| \leq \frac{1}{2}|\mathbb{Z}_n^*|$.*

Proof. Since n is composite and not a power of a prime, we can express it as $n = s \cdot t$ for integers s, t with $gcd(s, t) = 1$. Fix an x s.t. $x^r = -1 \mod n$. By the Chinese Remainder theorem, there exists a y s.t.

$$y = x \mod s, \text{ and } y = 1 \mod t.$$

We now argue that $y \in \mathbb{Z}_n^*$ but $y \notin S_r$. To show the first inclusion, since $x \in \mathbb{Z}_n^*$, it must be that $gcd(x, n) = 1$. This implies that $gcd(x, s) = 1$, since $s|n$. Then this implies that $gcd(y, s) = 1$,

since $y = x \mod s$. By definition, $gcd(y, t) = 1$, so then $gcd(y, s) = gcd(y, t) = 1$. This implies that $gcd(y, s \cdot t) = 1$. Since $n = s \cdot t$, this implies that $y \in \mathbb{Z}_n^*$.

To argue that $y \notin S_r$, consider $y^r \mod n$. We have that $y^r = 1^r = 1 \mod t$, and $y^r = x^r = -1 \mod s$. There are two cases: either $y^r = 1 \mod n$ or else $y^r = -1 \mod n$. If $y^r = -1 \mod n$, then y^r would be equivalent to -1 modulo both s and t, since $n = s \cdot t$. But this contradicts the fact that $y^r = 1 \mod t$. Similarly, if $y^r = 1 \mod n$, then $y^r = 1 \mod s$ which contradicts the fact that $y^r = -1 \mod s$. Either way we have a contradiction and we conclude that $y^r \neq \pm 1 \mod n$, and hence $y \notin S_r$, as desired. □

We now complete our proof of Theorem 2. The high level idea is that we will cleverly choose an r for which the conditions of the above lemma are satisfied (namely, there is some $c \in \mathbb{Z}_n^*$ for which $c^r = -1 \mod n$), and hence by the above lemma the corresponding set S_r satisfies $|S_r| \leq \frac{|\mathbb{Z}_n^*|}{2}$. We will then argue that in order for the algorithm to falsely output "prime", it must be the case that the x that is randomly chosen in the algorithm is actually an element of S_r. Since the above lemma guarantees that $|S_r| < \frac{n-1}{2}$, the probability that a "bad" x is chosen is at most $1/2$.

Proof of Theorem 2. We now prove that if n is composite, and not a power of a prime, then with probability at least $1/2$ the algorithm will correctly output "composite". Define b to be the largest integer less than k (where, as above $n - 1 = 2^k m$) such that there exists a $y \in \mathbb{Z}_n^*$ with the property that $y^{2^b m} = -1 \mod n$. (Hence if n is composite and not a power of a prime, then by Lemma 14 we know $|S_{2^b m}| \leq \frac{1}{2}|\mathbb{Z}_n^*|$.) We know that such a b exists, because $(-1)^{2^0 m} = -1$.

We now argue that if the algorithm wrongly outputs "prime", then $x \in S_{2^b m}$. In order for the algorithm to wrongly output "prime", it must be the case that the randomly chosen x of the algorithm satisfies either:

$$x^m = x^{2m} = \ldots = x^{2^k m} = 1 \mod n \quad \text{or} \quad \exists i \text{ s.t. } x^{2^i m} = -1 \mod n.$$

In the first case, trivially, $x \in S_{2^b m}$. In the second case, from our definition of b, it follows that $b \geq i$, and hence $x^{2^b m} = \pm 1$ so $x \in S_{2^b m}$. Thus in either of these cases, $x \in S_{2^b m}$. By Lemma 14, $S_{2^b m}$ is a proper subgroup of $\mathbb{Z}_n^*$. Hence the probability of choosing an x that fails to find a nontrivial root of 1 is at least $1/2$, as claimed. □

5 Fingerprinting

Note: We won't talk about this during class, but this is a cool application of random primes! It's similar to the fingerprinting scheme using polynomials that we saw in Class 1.

There are a number of important applications of primality testing beyond cryptography. One set of use-cases is related to "fingerprinting" and pattern matching. One of the canonical examples of such a use case is in designing a succinct hash of a given string/file, which can be used for making sure that the file was successfully copied or transferred.

Formally, consider the setting where there is an n-bit file $A = (a_1, \ldots, a_n)$, which we have attempted to copy, resulting in file $B = (b_1, \ldots, b_n)$. The following protocol, defined in terms of a parameter N which we will pin down later, is one very simple randomized scheme:

Algorithm 15. FINGERPRINTING
Given n-bit strings $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, and integer N:

- *Let p be chosen uniformly at random from the set of prime numbers less than N.*
- *Regard A and B as n-bit numbers, and check if $A \equiv B \mod p$. If so, declare "$A = B$," otherwise we know that $A \neq B$.*

Proposition 16. *If $A = B$, then $A \equiv B \mod p$ and the algorithm will be correct. We claim that if $A \neq B$, then for any $c < n$, if we set $N = cn \log n$, the probability that the algorithm is correct is at least $1 - 3/c$.*

Proof. If $A \neq B$, the algorithm is correct unless p divides $A - B$. Since $A, B \leq 2^n$, $|A - B| \leq 2^n$, and hence it has at most n distinct prime factors. (This is true because for any number, m, the product of the distinct prime factors is at most m, and each prime factor must be at least 2.) Together with the fact that the number of primes less than N is at most $N/(2 + \log N)$, we have:

$$\begin{aligned} Pr[failure] &\leq \frac{n}{N/(2+\log N)} = \frac{n(2+\log(cn\log n)}{cn\log n} \\ &= \frac{n(2+\log c+\log n+\log\log n)}{cn\log n} \leq \frac{1}{c} + \frac{\log c}{c\log n} + \frac{2+\log\log n}{c\log n} \leq \frac{3}{c}. \end{aligned}$$

□

Should we be happy with the above theorem? Consider a practical example. Suppose we want to send a gigabyte video file, and check that the correct file was received, with a probability of falsely thinking we received the correct file to be one in a million. Namely, $n \approx 2^{30}$, and we want the probability that the algorithm fails to be bounded by $\approx 2^{-20}$, so $c \approx 3 \cdot 2^{20}$. The above proposition says that we should pick $N = cn \log n \approx 2^{30} \cdot 3 \cdot 2^{20} \cdot \log 2^{30} < 2^{57}$. This means, after sending our gigabyte file, all we need to do is send a 57-bit prime, and the 57-bit number that is the file modulo our prime. Each of these is less than a single 64-bit word!!!! [And if we want the probability of failure to be less than 2^{-50}, this just means we need an extra 50 bits!.]

5.1 Did we need to use a prime?

What goes wrong if we didn't use a prime p in this fingerprinting scheme? Namely, if we picked q uniformly at random between 1 and N, and then just checked whether $A = B \mod N$? The core of the proof of the success guarantee was the fact that an n-bit number can have at most n distinct prime factors, which was the numerator in our probability of failure. If we are not restricting ourselves to primes, then this numerator would become the number of factors of our n-bit number. How much worse could this be? Quite a lot. Consider the following back-of-the-envelope calculation: imagine that $A - B$ is the product of the first k prime numbers. The first k prime numbers are all at most $O(k \log k)$, and hence their product is at most $(k \log k)^k \approx 2^{k \log k}$, so we can think that $n \approx k \log k$. How many factors does $A - B$ have? Well, any subset of these k prime factors can be multiplied to produce a distinct factor of $A - B$, hence there are 2^k such factors (since each of the k primes has 2 choices, either it is in the factor, or not). Since $n \approx k \log k$, as opposed to the bound of n distinct prime factors, we would now have $2^k \approx 2^{n/\log n}$ distinct factors—nearly exponentially worse than before!!!

References

[1] Leonard Adleman and M Huang. Recognizing primes in random polynomial time. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 462–469, 1987.

[2] Manindra Agrawal and Somenath Biswas. Primality and identity testing via chinese remaindering. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 202, 1999.

[3] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[4] William R Alford, Andrew Granville, and Carl Pomerance. There are infinitely many carmichael numbers. *Annals of Mathematics*, 139(3):703–722, 1994.

[5] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976.

[6] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

[7] Barkley Rosser. Explicit bounds for some functions of prime numbers. *American Journal of Mathematics*, 63(1):211–232, 1941.

[8] Robert Solovay and Volker Strassen. A fast monte-carlo test for primality. *SIAM journal on Computing*, 6(1):84–85, 1977.