Exploring Different Primality Tests

Dallas Anderson

December 10 2023

Abstract

In this paper we'll explore different primality testing algorithms, examining some of their advantages and disadvantages. We'll explore the ideas behind the algorithms, observing how the algorithms develop from these ideas. From Fermat's Little Theorem to elliptic curves to modular arithmetic, there are a lot of different techniques we can apply. We'll explore the tradeoff between efficiency and 100% results, seeing how different tests compare based on these factors.

1 Introduction

There are many different primality tests: Algorithims that test when a number is prime. One example is just, for a given number $n > 1$, checking every number between 2 and $n - 1$ and seeing if it divides n. If none do, it's prime. Otherwise it's composite. But you can see how, for bigger and bigger n, there are more and more numbers to check, so it very quickly becomes inefficient. There are ways of getting around this, like checking everything up to $\sqrt{n}$ instead, but you'll still run into the same problem as before, even if a bit later. And you could also just check primes up to $\sqrt{n}$, but that would mean you'd have to do this test on the numbers between 2 and $\sqrt{n}$ and this algorithim would have to be recursive, yikes!

And a lot of primality tests are like this, where it takes more and more steps for bigger and bigger numbers. The (non-recursive) one we've set up takes $\sqrt{n} - 1$ steps (plus the steps for computing $\sqrt{n}$ in the first place), which is pretty inefficient.

We want a more efficient test. There are some tests people have developed that work like this: If the test spits out composite, then the input is composite. If it spits out prime, then the input is prime with high probability. So these aren't 100% secure, but some of them don't have the same problem we observed above: They take the same amount of time for all inputs n, or at least the amount of time is bounded and doesn't grow forever. One example is the Fermat Primality Test.

2 The Fermat Primality Test

What is the Fermat primality test? Well, it's a test based on Fermat's Little Theorem:

Theorem 2.1 If p is a prime and a is an integer not divisible by p, then

$$a^{p-1} \equiv 1 \pmod{p}.$$

You can think of the a's as being integers mod p. We'll call this congruence the *Fermat congruence*, or FC, and we'll say it *holds on* a, p (note: This terminology isn't standard). So if p is prime, then the FC holds on a, p for all $1 \leq a \leq p - 1$. Could we use this as a test: That is, could we check each a between 1 and $p - 1$ and see if the FC holds? I mean, the converse is true too because if n is composite, it has a nontrivial divisor m, and even though $n \not| m$ we still have

$$m^{p-1} \not\equiv 1 \pmod{n}.$$

The problem is again efficiency, because we have to check all those a. Before we continue, we'll need to introduce some more new terminology (which is standard):

Definition 2.2 We say that a number m is a *Fermat witness* for n if $n \nmid m$ but

$$m^{p-1} \not\equiv 1 \pmod{p},$$

thus making n composite. We say m is a *Fermat lier* if n is composite but $n \nmid m$ and

$$m^{p-1} \equiv 1 \pmod{p}.$$

Instead of checking all $1 \leq a \leq n-1$, we can check only a fixed amount of (not necessarily distinct) values of a, and when n is composite we'll reach a Fermat witness for n with high probability (which we'll check). Except, there's a minor inconvenience with that: The FC trivially holds on $1, p$ for all p, and whenever p is odd the FC holds on $-1, p$. To fix this problem, we only test $1 < a < p-1$. There are none if $n \leq 3$, but those cases we already know.

So here's our algorithim:

Set a parameter k, and input a value $n > 1$.

Step 1: If $n = 2, 3$, output "probably prime".

Otherwise, move on to step 2: Generate k random numbers $a_1, \ldots, a_k$ strictly between 1 and $n-1$.

Step 3: Test the FC on a_i, n for each a_i.

Step 4: If not all hold, output "composite" but if all hold, output "probably prime".

If the test outputs "composite", then it's not too hard to see that n is definitely composite. However, we still need to check that if it outputs "probably prime", it's prime with high probability.

Firstly, if $n = 2, 3$, then it's definitely prime. Otherwise, the FC has held for all random $1 < a_1, \ldots, a_k < n-1$. Suppose n actually turns out to be composite. First of all, the a_i would have to be coprime to n; it's certainly possible for one of them to hit something strictly between 1 and $n-1$ that isn't coprime to n. Those exist since n is composite. Next, suppose they do all happen to hit values coprime to n. It could be that n is a *Carmichael number*:

Definition 2.3 We say that n is a *Carmichael number* if n is composite but every a coprime to n is a Fermat liar.

However, Carmichael numbers get increasingly rare as you get bigger and bigger, so it's unlikely that n is a Carmichael number. There's a nice theorem we can apply when n isn't:

Theorem 2.4 If n is composite but not a Carmichael number, then at least half of all $a \pmod{n}$ coprime to n are Fermat witnesses.

Proof. Let A be the set of all $a \pmod{n}$ coprime to n, and let $a_1, \cdots, a_s$ be Fermat Liars. We can take any Fermat witness a and find that aa_k are Fermat witnesses because

$$(aa_k)^{n-1} = a^{n-1}a_k^{n-1} \equiv a^{n-1} \not\equiv 1 \pmod{p}.$$

So if there were less than half Fermat witnesses, more than half would be Fermat liars (the complement of Fermat witnesses), meaning we could multiply those by a and get more than half Fermat witnesses, a contradiction. Thus there are indeed at least half Fermat witnesses in A.

So at most half of everything in A, and thus half of everything nonzero mod n, is a Fermat liar if n isn't Carmichael, and this provides a bound on the probability that it outputs "probably prime" (if n isn't Carmichael).

All this that we've done is just estimating probabilities since, after all, n could be Carmichael. However, there's a test, called the Goldwasser-Killian Test, that is 100% which we'll talk about. It involves *elliptic curves*.

3 Elliptic Curves and the Goldwasser-Killian Test

Before we get into this test, let's first define elliptic curves:

An elliptic curve E is a set of points along a curve of the form $y^2 = x^3 + ax + b$, along with a special point called the point at ∞ (and denoted ∞). There's a certain group operation on E, denoted $+$, defined by the following:

Take any two points $P, Q \in E$. If one of P or Q is ∞ then $P + Q$ is the other one (∞ is the identity element). Otherwise, let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. If $x_1 \neq x_2$, you use the regular addition formula: Letting $m = \frac{y_2 - y_1}{x_2 - x_1}$, we get

$$P + Q = (x_3, y_3) = \left(m^2 - x_1 - x_2, m(x_1 - x_3) - y_1\right).$$

If $x_1 = x_2$, then by the equation of E, $y_1^2 = y_2^2$, so $y_1 = \pm y_2$. If $y_1 = -y_2$, we get $P + Q = \infty$. If $y_1 = y_2$ and they're not equal to 0, then $P = Q$ (with $y_1 = y_2 \neq 0$) and we use the duplication formula: This time setting $m = \frac{3x_1^2 + a}{2y_1}$, we get

$$2P = P + P = (x_3, y_3) = \left(m^2 - 2x_1, m(x_1 - x_3) - y_1\right).$$

So that's the definition of addition on an elliptic curve E. This may seem arbitrary, with the weird formulae and such, but it has motivation. We won't get into motivation here though, and we'll just take it for granted that this does form a group, abelian in fact. For motivation and other details about elliptic curves, check out [4]. In this paper, we're just concered about the algorithm, and here's the Goldwasser-Killian test:

You input an integer $n > 1$.

Step 1: Choose three integers at random, a, x, y with the property that, setting

$$b \equiv y^2 - x^3 - ax \pmod{n},$$

we have $\gcd(4a^3 + 27b^2, n) = 1$. Now $P = (x, y)$ is a point on E, where E is the elliptic curve defined by $y^2 = x^3 + ax + b$.

Step 2: Apply Schoof's algorithm (say) to E, producing a number m which is the number of points on E over $\mathbb{F}_n$, provided n is prime.

Step 3: If this algorithm stops at an undefined expression, output "composite".

Otherwise, move on to step 4: If we can write m in the form $m = kq$ where $k > 1$ is a small integer and q a large probable prime (a number that passes a probabilistic primality test, for example), then we do not discard E. Otherwise, we discard our curve and randomly select another triple (a, x, y) to start over.

Step 5: Assuming we find a curve which passes this criterion, you then calculate mP and kP.

Step 6: If any of the two calculations produce an undefined expression, output "composite". If $mP \neq \infty$, output "composite".

Otherwise, move on to step 7: If $kP = \infty$, discard E and start over with a different a, x, y triple.

Step 8: If $mP = \infty$ and $kP \neq \infty$, output "prime".

Note that the only-probabilistic primality of q is verified using the same algorithm, and all the way down until we reach some threshold where the number is small enough so that we can use a simple 100% algorithm

(this is a recursive algorithm basically).

What is the idea behind this algorithim? Well, we'll need the following proposition:

Proposition 3.1 Let n be a positive integer and E be the elliptic curve $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2$ is coprime to n. Consider E over $\mathbb{Z}/n\mathbb{Z}$ (using the elliptic curve addition laws we mentioned before). Let m be an integer. If there is a prime q which divides m, and is greater than $(\sqrt[4]{n} + 1)^2$, and there exists a point P on E such that

$$mP = \infty \tag{1}$$

$$(m/q)P \text{ is defined and not equal to } \infty \tag{2}$$

Then n is prime.

Proof. If n is composite, then there exists a prime $p \leq \sqrt{n}$ dividing n. Let E_p be the elliptic curve defined by the same equation as E but evaluated modulo p instead of modulo n (this is still an elliptic curve since $4a^3 + 27b^2$ is coprime to n). Define m_p as the order of the group E_p. *Hasse's theorem on elliptic curves* shows that

$$m_p \leq p + 1 + 2\sqrt{p} = (\sqrt{p} + 1)^2 \leq \left(\sqrt[4]{n} + 1\right)^2 < q,$$

thus $\gcd(q, m_p) = 1$ (since q is prime) and there exists an integer u such that

$$uq \equiv 1 \bmod m_p.$$

Let P_p be the point P evaluated modulo p. Thus, on E_p we have

$$(m/q)P_p = uq(m/q)P_p = umP_p = \infty,$$

by (1). This is true because mP_p is calculated using the same method as mP, except modulo p rather than modulo n (and $p|n$). But this contradicts (2), because if $(m/q)P$ is defined and not equal to ∞ mod n, then the same method calculated modulo p instead of modulo n will give

$$(m/q)P_p \neq \infty. \square$$

The basic idea in the first section of the algorithm (steps 1-4) is to find an m to use to implement this proposition. First of all, if Schoof's algorithm for computing m gives an error, then we know n is composite. Assuming it doesn't, we want an m that is divisible by a large prime q which we can use in the proposition. If $mP \neq \infty$, it is clear that n is not prime, because if n were prime then E would be a group with order m, and any element of E would become ∞ on multiplication by m. If $kP = \infty$, we discard E and start over because we need it not to be ∞ in order to use the proposition. And in the other case we use the proposition and get that n is prime.

There are some problems with this algorithm, though. For one, we're relying on Schoof's algorithm to compute m, an algorithm that turns out to be very slow and cumbersome. Another problem is the difficulty of finding the curve E whose number of points is of the form kq as above. There is no known theorem which guarantees we can find a suitable E in polynomially many attempts (in $\log(n)$, which I'll explain about later). So efficiency is again a problem, as with our $\sqrt{n}$ method earlier.

It might start to seem like every primality test is either super inefficient (for big numbers) or doesn't give for-sure results, and it's been this way for a long time. But mathematicians have more recently developed a test, called the AKS test, that is 100% and is much more efficient than the Goldwasser-Killian Test (and the $\sqrt{n}$ method). It still requires more and more steps for bigger and bigger numbers, but the number of steps doesn't grow quite as much as for the other methods.

4 The AKS Test and its Advantages

First, before we get into the AKS test, we'll need some new notation:

Definition 4.1 Take functions f, g, from positive reals to reals. We say that $f(x) \in O(g(x))$ if there exist positive real numbers M and x_0 such that $|f(x)| \leq Mg(x)$ for all $x \geq x_0$.

For example, if $f(x) = \sqrt{x} - 1$, then $f(x) \in O(\sqrt{x})$ since we can take M to be 2 and x_0 to be 1.

Many authors write $f(x) = O(g(x))$ but here we won't use that, since it gets confusing when you say things like $f(x) = O(g(x))$ and $h(x) = O(g(x))$ but $f(x) \neq h(x)$, for example.

In this paper, we will use the notation $f(x) \equiv g(x) \pmod{h(x), n}$ to represent the equation $f(x) \equiv g(x)$ in the ring $\mathbb{Z}/n\mathbb{Z}[x]/(h(x))$. We'll use $\ln(x)$ to mean $\log_e(x)$, and we use $\log(x)$ to mean $\log_2(x)$. We use the symbol $O^{\sim}(t(n))$ for $O(t(n) \cdot \text{poly}(\log[t(n)]))$, where $t(n)$ is some function of n and poly just means "some polynomial of". For example,

$$O^{\sim}([\log(n)]^k) = O^{\sim}([\log(n)]^k \cdot \text{poly}(\log\log n)) = O([\log n]^{k+\epsilon})$$

for any $\epsilon > 0$. Given $r \in \mathbb{N}$ and $a \in \mathbb{Z}$ with $\gcd(a, r) = 1$, the order of a mod r is the smallest number k such that $a^k \equiv 1 \pmod{r}$. It is denoted as $o_r(a)$. Here, $\mathbb{N}$ denotes the positive integers. For $r \in \mathbb{N}$, $\phi(r)$ is *Euler's totient function* giving the number of positive integers less than r that are relatively prime to r. With basic group theory, it is easy to see that $o_r(a) \big| \phi(r)$ for any a such that $\gcd(a, r) = 1$.

It turns out that the number of steps still grows, so what makes the AKS test more efficient than the others? Well, this algorithm takes polynomial time in $\log(n)$, here meaning it takes a polynomial in $\log(n)$ amount of steps to carry out. The others take polynomial time in n, a polynomial in n amount of steps to carry out. So, what is the algorithm?

As with the Fermat primality test, we'll first explain the idea behind the algorithim. We'll need a lemma:

Lemma 4.2 Let $a \in \mathbb{Z}$, $n \in \mathbb{N}$ so that $n > 1$ and $\gcd(a, n) = 1$. Then n is prime iff (if and only if)

$$(x + a)^n \equiv x^n + a \pmod{n}. \tag{3}$$

This suggests a simple test for primality: Given an input n, just take any a coprime to n and check if this congruence holds. However, we need to evaluate n coefficients in the worst case so this is still pretty inefficient. One way to reduce the number of coefficients is to evaluate both sides of (3) modulo a polynomial of the form $x^r - 1$ for a chosen small r. In other words, to test if the following congruence is satisfied:

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, n}. \tag{4}$$

From Lemma 4.2 it is immediate that all primes n satisfy the equation (4) for all values of a and r. The problem now is that some composites n may also satisfy the equation for a few values of a and r (and indeed they do). However, we can almost restore the characterization: We show, for appropriately chosen r, that if the equation (4) is satisfied for several a's then n must be a prime power. The number of a's and the appropriate r are both bounded by a polynomial in $\log(n)$ and therefore, we get a 100% polynomial time in $\log(n)$ algorithim for testing primality.

With that in mind, here's the algorithim:

You input an integer $n > 1$.

Step 1: If $n = a^b$ for some $a \in \mathbb{N}$ and $b > 1$, output "composite".

Otherwise, move on to step 2: Find the smallest r such that $o_r(n) > [\log n]^2$.

Step 3: If $1 < \gcd(a, n) < n$ for some $a \leq r$, output "composite".

Otherwise, move on to step 4: If $n \leq r$, output "prime".

Otherwise, step 5: For each a from 1 to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ check whether equation (4) holds.

Step 6: If it doesn't for at least one of them, output "composite" but if it holds for all a, output "prime".

And it's provable that this algorithm is 100%: It outputs "prime" iff n is prime. The details of this proof are complicated, but one direction is easy: If n is prime, the algorithm outputs "prime". See [1] for more details.

Lemma 4.3 If n is prime, the AKS test outputs "prime".

Proof. By primality of n the algorithm can never return "composite" in steps 1 or 3. The only other time when it could is if equation (4) doesn't hold for one of the a's, which can't happen if n is prime by Lemma 4.2.□

What is the efficiency of our algorithm?

Theorem 4.4 The efficiency (*asymptotic time complexity*) of the algorithm is in

$$O^{\sim}[\log n]^{21/2}.$$

Proof. Step 1 takes asymptotic time in $O^{\sim}[\log n]^3$ (see [3]). In step 2, we find an r with $o_r(n) > [\log n]^2$. This can be done by trying out successive values of r and testing if $n^k \neq 1 \pmod{r}$ for every $k \leq [\log n]^2$. For a particular r, this will involve at most $O[\log n]^2$ multiplications modulo r and so will take time $O([\log n]^2 \log r)$. Next, by a certain lemma which we won't get into here, only $O[\log n]^5$ different r's need to be tried, so step 2 has total time complexity $O^{\sim}[\log n]^7$. In step 3, we compute a total of r gcd's. Each gcd computation takes time in $O(\log n)$ (again, see [3]), so the time complexity of this step is $O(r \log n) = O[\log n]^6$. The time complexity of step 4 is just $O(\log n)$.

In step 5, we need to verify $\lfloor \sqrt{\phi(r)} \log n \rfloor$ equations. Each equation requires $O(\log n)$ multiplications of degree r polynomials with coefficients of size $O(\log n)$. So each equation can be verified in $O(r[\log n]^2)$ steps. This means the total time complexity of step 5 is

$$O^{\sim}\left(r\sqrt{\phi(r)}[\log n]^3\right) = O^{\sim}\left(r^{\frac{3}{2}}[\log n]^3\right) = O^{\sim}[\log n]^{21/2}.$$

(And step 6 isn't really much, just outputs). Step 5 dominates all others, so that's the final time complexity.□

Thus the algorithm is polynomial time in $\log(n)$.

There's still a lot to learn about primes and primality testing. Maybe in the future, we'll come up with better algorithims, perhaps even 100% ones that don't have more and more steps for bigger and bigger inputs. More and more mathematics is coming out every day, and more and more breakthroughs will keep coming. For now, all we can do is keep studying and grow our knowledge.

For more details on the Goldwasser-Killian test, see pages 323-324 of [2], or see [5]. You could also check out [6] for more on the Fermat primality test.

Acknowledgements. I would like to thank my teacher Simon Rubinstein-Salzedo for making the writing of this paper possible, and I would like to thank my T.A. Albert Zhang for providing feedback along the way.

References

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

[2] David A Cox. *Primes of the Form x2+ ny2: Fermat, Class Field Theory, and Complex Multiplication. with Solutions*, volume 387. American Mathematical Soc., 2022.

[3] Jürgen Gerhard and Joachim Von zur Gathen. *Modern computer algebra*. Cambridge University Press, 2013.

[4] Simon Rubinstein-Salzedo. *Number Theory*. Rubinstein-Salzedo, 2023.

[5] Wikipedia. Elliptic curve primality — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Elliptic\%20curve\%20primality&oldid=1184114564`, 2023. [Online; accessed 09-December-2023].

[6] Wikipedia. Fermat primality test — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Fermat\%20primality\%20test&oldid=1181252338`, 2023. [Online; accessed 09-December-2023].