Primality Testing in Polynomial Time

Pitchayut Saengrungkongka

November 25, 2025

This is the notes written in preparation for a talk in HMMT November 2025 Education. The goal is to describe the AKS primality testing and cover most of the proof.

The key reference is the original paper itself: `https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf`.

§1 Background on Primality Testing

Our problem is pretty simple.

📋 **Problem Statement.**

Given a positive integer n, determine whether n is prime.

We will not discuss the application of having a fast primality testing other than mentioning that it is useful in cryptography (which typically have to handle primes with hundreds digits). Before looking at the fast algorithm, let us look at some of the earlier algorithm.

▶ **Trial Division.** The most naive algorithm to solve this problem is as follows.

- For each $2 \le d \le \lfloor \sqrt{n} \rfloor$.
  - If d divides n, return `composite`.
- If none of these d's divide n, return `prime`.

This algorithm is not very fast. It requires $\sqrt{n}$ divisions. Concretely, your computer can handle about 10^8 divisions in a second, so a 30-digit number would take a CPU year to run. Let alone 100-200 digit numbers.

▶ **Fermat's Little Theorem.** The following theorem could be used to test for prime.

📝 **Proposition 1.1.**

For any prime p and a positive integer a, p always divides $a^p - a$.

In particular, this leads to the following algorithm:

- Pick a random positive integer a.
- If $a^n \not\equiv a \pmod{n}$, return `composite`.

- Otherwise, return `maybe prime`.

How do we compute $a^n \pmod{n}$ quickly? It turns out that the trick is to use squaring to quickly reduce the exponents. This is easier to say through an example:

$$\begin{aligned} a^{19} &= a \cdot (a^9)^2 \\ a^9 &= a \cdot (a^4)^2 \\ a^4 &= (a^2)^2. \end{aligned}$$

Thus, we can compute a^{19} in 6 multiplications. In general, computing a^n takes around $\log_2 n$ multplications because the exponent increase to roughly a factor of two eahc step. This is very fast compared to $\sqrt{n}$ divisions above.

How good is this algorithm though? Unfortunately, this test fail to recognize some kinds of composite numbers.

Proposition 1.2.

$a^{561} \equiv a \pmod{561}$ for all positive integers a, and the same holds when 561 is replaced by 1105, 1729, $\ldots$

This means that the algorithm will always return "`maybe prime`" for $561 = 3 \cdot 11 \cdot 17$, and several other numbers. These numbers are called **Carmichael numbers**, and it is known (proven by Alford, Granville, and Pomerance) that there are infinitely many of them.

Remark 1.3 (Other primality tests, not mentioned during talk)**.**

In the process of computing $a^n \pmod{n}$, there are a lot of squaring. In each computation of squares, we can perform and additional check that "if there exists x such that $x^2 \equiv 1 \pmod{n}$ but $x \not\equiv \pm 1 \pmod{n}$, then return `composite`." This leads to **Miller-Rabin Test**, which is correct for approximately $\frac{3}{4}$ of the values of a. In particular, if we run this 100 times, the probability that this is correct is 4^{-100}, which makes it fairly accurate (although not undisputably correct).

Another primality test worth mentioning is **Elliptic Curve Primality Proving (ECPP)**, which is based on the Hasse-Weil bound of the elliptic curves. This algorithm runs very fast in practice (and it is preferred over AKS) and gives undisputably correct results. However, the runtime is only heuristic and not proven.

§2 The AKS Algorithm

The AKS algorithm, found by Agrawal, Kayal, and Saxena (hence the name AKS) in 2002 is the only primality testing algorithm to date that is

1. **deterministic**, i.e., the output is undisputably correct.
2. **runs in polynomial time**, i.e., runs fast (we will be more precise in a moment).
3. **unconditional**, i.e., the correctness and runtime does not depend on any unproven conjecture.

Unfortunately, AKS is **not practical**; the runtime is very slow in practice, even with the optimized variants that were found later. Still, it is of immense theoretical significance, and unlike many recent breakthroughs in number theory, the proof of this algorithm is **completely elementary** (this does not mean simple, but it requires a lot less background compared to e.g., the proof of Fermat's Last Theorem).

The idea of AKS is based on the following proposition, which is a stronger form of Fermat's little theorem above.

Proposition 2.1.

For any prime p, we have

$$(X+a)^p \equiv X^p + a \pmod p$$

as polynomials in variables X.

▶ *Proof.* The left hand side is

$$X^p + \binom{p}{1} aX^{p-1} + \binom{p}{2} a^2 X^{p-2} + \cdots + \binom{p}{p-1} a^{p-1} X + a^p.$$

Since p divides $\binom{p}{1}, \binom{p}{2}, \ldots, \binom{p}{p-1}$ and $a^p \equiv a \pmod p$, the result follows. □

Thus, if $(X+a)^n \not\equiv X^n + a \pmod n$, we can say that n is composite. It turns out that the converse is true as well (but we will not prove it here).

However, $(X+a)^n$ is a large polynomial. To make it smaller, we take modulo $X^r - 1$ where r is small enough that we can actually check all coefficients. In particular, we compare two polynomials $(X+a)^n$ and $X^n + a$ by

- reducing all coefficients modulo n; and
- reducing the entire polynomial modulo $X^r - 1$. (For example, $X^{2r+3} \equiv X^3 \pmod{X^r - 1}$.)

If they are not the same, i.e., $(X+a)^n \not\equiv X^n + a \pmod{n, X^r - 1}$, for some a or r, we can say immediately that n is composite.

This leads to the following algorithm.

The AKS Algorithm.

1. If n is a perfect power, return `composite`.
2. Find (by brute force) the smallest positive integer $r \geq 2$ such that $\operatorname{ord}_r n > \log_2^2 n$.
   (Recall that $\operatorname{ord}_r n$ is the smallest positive integer k such that $r \mid n^k - 1$.)
   If we ever encounter any r such that $\gcd(r, n) > 1$, return `composite`.
3. For each $1 \leq a \leq \lfloor 2\sqrt{r} \log n \rfloor$,
   - If $(X+a)^n \not\equiv X^n + a \pmod{n, X^r - 1}$, return `composite`.
4. Return `prime`.

It's clear from the discussion above that if the algortihm returns `composite`, n is actually composite. The major miracle is that if the algorithm returns `prime` in Step 4., then n is actually prime. We will prove this in the next Section.

▶ **Runtime Analysis** First of all, the glaring issue of this algorithm is that if r is large, then we are comparing too many values of a and comparing polynomials that are too large. Fortunately, r is small.

Lemma 2.2.

There exists $r < \log_2^5 n$ such that $\operatorname{ord}_r n > \log_2^2 n$.

▶ *Proof (Skipped during talk).* Assume not. Let $x = \lceil \log_2^5 n \rceil$. Then, $\operatorname{lcm}(1, 2, 3, \dots, x)$ would divide the product

$$(n-1)(n^2-1)\dots(n^{\lfloor \log_2^2 n \rfloor} - 1).$$

Thus, we deduce that

$$2^{x-1} \le \operatorname{lcm}(1, 2, \dots, x) \le n^{1+2+3+\dots+\lfloor \log_2^2 n \rfloor} < n^{\log_2^4 n/2} = 2^{\log_2^5 n/2},$$

which is a contradiction. Here we have used the fact that $\operatorname{lcm}(1, 2, \dots, x) \ge 2^{x-1}$. □

What does it mean for this algorithm to run fast? To analyze algorithm of this complexity, we often write the runtime in **big O-notation**, which indicates that we don't care about the constant factor. For example, $O(n^2)$ means that the runtime is proportional to n^2 (but it could be $10n^2$ or $100n^2$).

In our case, Step 3. dominates the run time.

- We have to compare $\lfloor \sqrt{r} \log n \rfloor = O(\log^{3.5} n)$ values of a.
  - For each such a, we have to compute $(X+a)^n$ and $X^n + a$, while keeping its remainder when divided by n and $X^r - 1$. The remainder is a polynomial of degree $< r$, and each coefficient has size at most n.
  - To compute this, we use the idea of repeated squaring as in the Fermat's primality test. In particular, this takes $O(\log n)$ multiplications.
  - Each multiplication is multiplying two degree r polynomials with coefficients having $\log n$ digits. With a naive multiplication algorithm (requiring multiplying every *pair* of digits and sum them), this takes $O(r^2 \log^2 n) = O(\log^{12} n)$.
  - Thus, each such a takes $O(\log^{12+1} n) = O(\log^{13} n)$ times to process.
- Hence, the whole algorithm takes $\boxed{O(\log^{16.5} n)}$ to run.

We note that if faster multiplication algorithms (i.e., the Fast Fourier Transform based ones) is used, the runtime can be reduced to $O(\log^{10.5+\varepsilon} n)$.

Remark 2.3 (About runtime, not mentioned during the talk)**.**

The bound $r = O(\log^5 n)$ above is not tight. We conjecture that $r = O(\log^2 n)$ is true, which will give the **conjectured** runtime of $O(\log^{6+\varepsilon} n)$. The best known result to date, utilizing Fouvry's result in analytic number theory, is $r = O(\log^3 n)$, giving a runtime of $O(\log^{7.5+\varepsilon} n)$.

In a sequel paper "Primality Testing via Gaussian Periods" by Lenstra and Pomerance, the algorithm above was generalized by replacing $X^r - 1$ by a more carefully chosen polynomial of smaller degree. Their version of AKS primality test has a **proven** runtime of $O(\log^{6+\varepsilon} n)$.

§3 Proof of Correctness

To prove that Step 4. works, we prove the following theorem.

Theorem 3.1.

Let n and r be positive integers such that n is not a perfect power, $\gcd(n, r) = 1$, and $\operatorname{ord}_n r > \log_2^2 n$. Let $\ell = \lfloor 2\sqrt{r} \log n \rfloor$. Suppose that for each $1 \le a \le \ell$, we have

$$(X+a)^n \equiv X^n + a \pmod{n, X^r - 1},$$

then n is prime.

In the proof below, we assume that n is sufficiently large ($n \geq 1000$ is more than enough).

Assume for a contradiction that p is a prime divisor of n. We start by using the condition $(X+a)^n \equiv X^n + a \pmod{n, X^r - 1}$ to generate many congruences of a similar form.

Definition 3.2.

A pair (m, f), where m is a positive integer and f is a polynomial with integer coefficients, is called a **critical** pair if

$$f(X^m) \equiv f(X)^m \pmod{p, X^r - 1}.$$

We have the following observations. Part (a) and (b) give us initial critical pair, while part (c) and (d) explain how to generate a lot of critical pairs.

Proposition 3.3.

(a) $(n, X + a)$ is a critical pair for all $0 \leq a \leq \ell$.

(b) $(p, X + a)$ is a critical pair for all a.

(c) If (m, f) and (m, g) are critical pairs, then so is (m, fg).

(d) If (m, f) and (m', f) are critical pairs, then so is (mm', f).

► *Proof (skipped during the talk).*

(a) This is obviously true when $a = 0$, and for $1 \leq a \leq \ell$, this is clear from the assumption.

(b) Follows from Proposition 2.1: $(X + a)^p = X^p + a \pmod{p}$.

(c) Follows from

$$f(X^m)g(X^m) \equiv f(X)^m g(X)^m \equiv (f(X)g(X))^m \pmod{p, X^r - 1}.$$

(d) Follows from

$$f(X^{mm'}) \equiv f(X^m)^{m'} \equiv f(X)^{mm'} \pmod{p, X^r - 1}.$$ □

Let

$$I = \left\{n^i p^j : i, j \in \mathbb{Z}_{\geq 0}\right\},$$
$$t = \text{number of distinct remainders of } I \text{ modulo } r,$$
$$G = \left\{X^{e_0}(X+1)^{e_1}(X+2)^{e_2}\cdots(X+\ell)^{e_\ell} : \begin{array}{c} e_0, e_1, \ldots, e_\ell \geq 0 \\ e_0 + e_1 + \cdots + e_\ell = t - 1. \end{array}\right\} \subseteq \mathbb{F}_p[X].$$

(In particular, each element of G is degree $t-1$.) Then from the above proposition, (m, f) is critical pair for all $m \in I$ and $f \in G$. We also note that by Stars and Bars,

$$|G| = \binom{t+\ell}{t-1} \tag{1}$$

The intuition now is that this generates a lot of critical pairs. To get a contradiction, we need to better understand the structure of G and of critical pairs. In order to do this, we now restrict from modulo

$X^r - 1$ (which is a product of many polynomials in $\mathbb{F}_p[X]$) to modulo an $\phi(X)$, which is an irreducible factor of $X^r - 1$.

More precisely, we recall that the ***r*-th cyclotomic polynomial** $\Phi_r(X)$ is a polynomial in $\mathbb{Z}[x]$ that whose roots are primitive r-th root of unity (i.e., $e^{2\pi i a/r}$ where $\gcd(a, r) = 1$). Let $\phi(X)$ be an irreducible factor of $\Phi_r(X)$ in modulo p, and define

$$K = \frac{\mathbb{Z}[X]}{(p, \phi(X))} = \{\text{polynomial with integer coefficients in modulo } p \text{ and } \phi(X)\}.$$

Since ϕ is irreducible, it is straightforward to show that every nonzero element in K is invertible (i.e., for every polynomial $f(X) \not\equiv 0 \pmod{p, \phi(X)}$, there exists $g(X)$ such that $f(X)g(X) \equiv 1 \pmod{p, \phi(X)}$.) Then we have the following fact, whose proof follows its analogue over $\mathbb{Z}$ or $\mathbb{Q}$.

Fact 3.4.

Any polynomial in $K[Y]$ with degree d has at most d roots in K.

(An example of an element in $K[Y]$ would be $Y^2 - (X+6)Y + 7$, which has degree 2, so at most two roots, each of which can be in form of X, but it should not be the same modulo $\phi(X)$.)

We now start analyzing the structure of critical pairs more closely.

Lemma 3.5.

Any two polynomials $f, g \in G$ leave different residues modulo $\phi(X)$.

▶ *Proof.* Assume not, i.e., $f(X) \equiv g(X) \pmod{\phi(X)}$. Then for any $m \in I$, we have that

$$f(X^m) \equiv f(X)^m \equiv g(X)^m \equiv g(X^m) \pmod{p, \phi(X)},$$

so $\{X^m : m \in I\}$ are roots of polynomial $f(Y) - g(Y) \in K[Y]$. We claim that $\{X^m : m \in I\}$ are distinct modulo $\phi(X)$.

(Proof of this claim, which is technical and not important: Since $\phi(X)$ is a divisor of $\Phi_r(X)$, we get that X is a primitive r-th root of unity. In particular, the polynomial $Y^r - 1$ splits completely into r linear factors in $K[Y]$. Since the derivative of $Y^r - 1$ is rY^{r-1}, which does not vanish because $p \nmid r$, we get that these r linear factors of $Y^r - 1$ are distinct. These r linear factors are $(Y - X^j)$ for $j = 0, 1, \ldots, r-1$, so $1, X, X^2, \ldots, X^{r-1}$ are distinct modulo $\phi(X)$.)

Therefore, we have t distinct roots of a polynomial of degree $t - 1$, which is a contradiction. ☐

Thus, from Lemma 3.5 and (1), we get that G leaves at least $\binom{t+\ell}{t-1}$ distinct remainders when divided by $\phi(X)$. On the other hand, we upper bound this quantity.

Lemma 3.6.

The number of distinct remainders of G in modulo $\phi(X)$ is at most $n^{2\sqrt{t}}$.

▶ *Proof.* By pigeonhole, two elements in the set

$$\left\{n^i p^j : 0 \le i, j \le \sqrt{t}\right\} \subseteq I$$

are the same modulo r. (The condition that n is a perfect power ensures that $n^i p^j$ are distinct.) In particular, there exists distinct $m_1, m_2 \in I$ such that $m_1 \equiv m_2 \pmod{r}$ but $m_1, m_2 \le n^{2\sqrt{t}}$. However, we note that for any $f(X) \in G$, we have

$$f(X)^{m_1} \equiv f(X^{m_1}) \equiv f(X^{m_2}) \equiv f(X)^{m_2} \pmod{p, X^r - 1},$$

so any $f(X)$ is a root of the polynomial $Y^{m_1} - Y^{m_2} \in K[Y]$. There can be at most $n^{2\sqrt{t}}$ roots, so there can be at most $n^{2\sqrt{t}}$ distinct residues from G in modulo $\phi(X)$. □

We now get the final contradiction. By comparing the number of distinct residues of G in modulo $\phi(X)$ from Lemma 3.5, (1), and Lemma 3.6, we get

$$n^{2\sqrt{t}} \geq \binom{t+\ell}{t-1}. \tag{2}$$

We now need a lower bound on t so that it beats the n in the base. Fortunately, this is easy: we have $t > \log_2^2 n$ since $1, n, n^2, \dots, n^{\lfloor \log_2^2 n \rfloor}$ are all in I and distinct modulo r. Therefore,

$$\begin{aligned}
\binom{t+\ell}{t-1} &\geq \binom{\lfloor \sqrt{t}\log_2 n\rfloor + \ell + 1}{\lfloor \sqrt{t}\log_2 n\rfloor} && \text{(since } t > \sqrt{t}\log n\text{)} \\
&\geq \binom{3\lfloor \sqrt{t}\log_2 n\rfloor + 1}{\lfloor \sqrt{t}\log_2 n\rfloor} && \text{(since } \ell > 2\sqrt{t}\log_2 n\text{)} \\
&\geq 4^{\lfloor \sqrt{t}\log_2 n\rfloor + 1} && \text{(since } \tbinom{3k+1}{k} \geq 4^{k+1} \text{ for all } k \geq 5\text{)} \\
&> 4^{\sqrt{t}\log_2 n} = n^{2\sqrt{t}},
\end{aligned}$$

which contradicts (2). This completes the proof of Theorem 3.1.