PRIMALITY PROVING ALGORITHMS

GREGORY MINTON

Prime numbers are central to modern discrete mathematics. It is thus natural to ask the question, given a natural number, of whether it is prime or not. In fact, since there is no formula for the n^{th} prime, we use the answer to this question to generate large primes: to find a prime, repeatedly generate a random number until one is prime. By an appeal to the Prime Number Theorem, on average it will take about $\log n$ tries to find a prime. Thus if we had a polynomial-time algorithm[1] for testing a number for primality, we would have a polynomial-time algorithm for generating a prime.

The importance of generating large primes has grown recently, with the rise of public-key cryptography systems (in particular, RSA and Diffie-Hellman methods). The security of these methods depends directly on the availability of a sizable stock of large primes. In this context, "large" primes typically have on the order of a thousand bits; using trial division to check primality (an exponential-time approach) is very much out of the question.

There are very fast probabilistic methods available for checking primality, which could perhaps more accurately be called compositeness tests: given a number, they return either "definitely composite" or "probably prime." If a number passes many iterations of such a test, it is extremely likely to be prime (with some tests, it can be proven that the probability of incorrectly identifying a prime vanishes exponentially with the number of iterations). Cryptographic applications typically use such methods, and in particular a test due to Miller and Rabin [23] for generating primes.

In this paper, we focus instead on primality *proving* algorithms, techniques which generate a proof of primality. Further, we focus on general purpose algorithms, excluding any approaches which require a special form for the input number. (For one example of a special case algorithm, the distributed internet project searching for Mersenne primes [25] uses Pollard's $p-1$ method.) In particular, we survey three major primality proving algorithms, which rely on disparate concepts in mathematics: Jacobi sums, elliptic curves, and exponentiation in a polynomial ring. Our goal is to present a summary of the techniques which explains the broad concepts while being suitable for implementation. While we will discuss asymptotic runtime and proof of correctness to some extent, we typically just cite such results from the original papers. To complement our description of the algorithms, we also wrote implementations (source code available upon request). We close the paper with a brief comparison of the runtimes of these implementations.

1. Jacobi Sum APR Method

The Jacobi sums test of L. Adleman, C. Pomerance, and R. Rumley, when introduced in [1], was a breakthrough in primality proving. Previously there had existed probabilistic methods and methods which assumed the Extended Riemann Hypothesis, but the fastest unconditional primality tests took exponential time.

[1]The running times of algorithms related to large numbers are typically quoted in terms of the number of digits, so polynomial-time refers to $O(\log^c n)$ for some constant c.

The APR method, as published, provably has a runtime of

$$(\log n)^{c \log\log\log n},$$

where the constant c is "a positive constant for which an upper bound could in principle be computed" [1]. Further, this runtime is sharp; the method does not run in polynomial time, but it is subexponential. The method given in [1] was revised and simplified by H. Cohen and H.W. Lenstra in [12], and details of an implementation were given in [11]. The simplification relies on replacing Jacobi sums with Gauss sums, but the latter are not as well suited for computation. Thus they in the end replace the Gauss sums again with Jacobi sums. The method has been revisited in more recent years, by the dissertations of W. Bosma and M.P.M. van der Hulst [8] and P. Mihailescu [18]. Their improvements have made the method competitive with the more recent elliptic curve method; in particular, Mihailescu's CYCLOPROV software [21] broke records when it was released for the largest primality proof by a general algorithm [20].

We first describe the motivating background for this test, as given in the two original papers [1] and [12]. This discussion gives an intuitive feel for the mathematics behind the APR test. We then state the actual algorithm; we follow the revised approach of [12], but we shall use the somewhat simplified language from Cohen's text [10].

The APR test relies on residue symbols and reciprocity laws, so we begin with a discussion of the simplest case, quadratic residues. For an integer a and an odd prime p, we define the Legendre symbol

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \bmod p, \\ +1 & \text{if } a \not\equiv 0 \bmod p \text{ and } a \equiv x^2 \text{ for some } x, \\ -1 & \text{otherwise.} \end{cases}$$

To compute Legendre symbols efficiently, we generalize our quadratic residue symbol to situations where the base is not prime, defining the Jacobi symbol

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1}\left(\frac{a}{p_2}\right)^{e_2}\cdots\left(\frac{a}{p_k}\right)^{e_k}, \qquad \text{where } n = p_1^{e_1}p_2^{e_2}\cdots p_k^{e_k},$$

for odd integers n. This clearly agrees with the Legendre symbol when the base is prime. Computation with the Jacobi symbol is done by observing that it is multiplicative in both top and bottom and applying the following facts:

- $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2} = \begin{cases} 1 & \text{if } n \equiv 1 \bmod 4 \\ -1 & \text{if } n \equiv 3 \bmod 4 \end{cases}$,
- $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8} = \begin{cases} 1 & \text{if } n \equiv 1, 7 \bmod 8 \\ -1 & \text{if } n \equiv 3, 5 \bmod 8 \end{cases}$, and
- (Quadratic Reciprocity) If m, n are odd positive integers, then
$$\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)(-1)^{(m-1)(n-1)/4}.$$

Thus by flipping the top and bottom (and then reducing the top modulo the bottom), we can quickly compute Jacobi/Legendre symbols. The running time of such an operation is comparable to computing the greatest common divisor of two integers.

Now it can be easily verified that, for prime n,

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \bmod n.$$

Further, it is known that if n is not prime, then this fails for at least half of the possible values for a. This is the basis for a relatively simple probabilistic test, which existed prior to the APR test.

The major idea of the APR test is to use the above condition not just as a "is n prime" test, but as a "what can it say about the prime factors of n" test. Using the language of [12], the idea is to formulate a series of tests such that, if n passes all of them, then $\psi(r)$ is a power of $\psi(n)$ for every divisor r of n (defining ψ as the multiplicative map from the set of divisors to $\mathbb{Z}/s\mathbb{Z}$ with $\psi(r) = (r \bmod s)$, for a suitably chosen $s > n^{1/2}$). This allows us to conclude the primality of n, since all of its divisors are powers of it. This formulation of the test is one of the reinterpretations of [12]; thus our final algorithm will clearly reflect it, but our discussion of the connection between primality and residue symbols (which follows the original paper [1]) takes a somewhat different tone.

In more detail, suppose we have a set of primes (called Euclidean primes in [1] and q-primes in [11]; we use the latter terminology), the product of which exceeds $\sqrt{n}$. Suppose r is a divisor of n at most $\sqrt{n}$; if we knew the value of r modulo each of these q-primes, then we would know the value of r by the Chinese Remainder Theorem.

Now define the collection of q-primes with respect to a set $\mathcal{P}$ of primes to be all primes q such that $q-1$ is square-free and every prime factor thereof lies in $\mathcal{P}$. (Note: in the simplification of [12], which we follow for our implementation, the square-free requirement is not needed.) The reason we create this level of abstraction is that the collection of q-primes with respect to $\mathcal{P}$ can have large product (in particular, at least $\sqrt{n}$) while $\mathcal{P}$ itself can have small product. In particular, [1] proves that the set $\mathcal{P}$ can be chosen to have

$$\prod_{p\in\mathcal{P}} p < (\log n)^{c\log\log\log n}$$

for sufficiently large n, which is the source of the APR test's runtime.

The connection between q-primes and $\mathcal{P}$ is as follows. For each q-prime q, $\mathbb{Z}/q\mathbb{Z}$ is cyclic; fix a generator g_q. Now for x not divisible by q, define the index $\mathrm{Ind}_q(x) \in \mathbb{Z}/(q-1)\mathbb{Z}$ to be the exponent i such that $g_q^i \equiv x \bmod q$. As above, we use r to denote some divisor of n less than $\sqrt{n}$; r is specified by the values $r \bmod q$ over all q-primes q, and any $r \bmod q$ is completely specified by $\mathrm{Ind}_q(r)$. (Note: if any q divides n, then we are immediately done with the primality proof — either $n = q$, in which case it is prime, or $n > q$, in which case it is composite. Otherwise $q \nmid r$, so $\mathrm{Ind}_q(r)$ is defined.) Since the index is an integer mod $q-1$ (the modulus being square-free), again by the Chinese Remainder Theorem we know that $\mathrm{Ind}_q(r)$ is specified by its value modulo every prime divisor of $q-1$. Each of these prime divisors is in the set $\mathcal{P}$. We summarize this discussion in the following proposition.

Proposition 1. *Suppose $\mathcal{P}$ is a collection of primes such that the product of all the q-primes w.r.t. $\mathcal{P}$ is greater than $\sqrt{n}$. Then a divisor $r \leq \sqrt{n}$ of n is completely specified by the value of $\mathrm{Ind}_q(r)$ modulo p, for each q-prime q and $p \in \mathcal{P}$ dividing $q-1$.*

The APR test uses p^{th} residue symbols to give information on the indicators $\mathrm{Ind}_q(r) \bmod p$. The case $p = 2$ can be clearly explained in terms of the already-defined quadratic residue symbol, the Jacobi symbol.

For some q-prime q, if the index $\mathrm{Ind}_q(r)$ is even, then clearly r is a square modulo q. The converse holds as well, so knowing $(\frac{r}{q})$ is equivalent to knowing $\mathrm{Ind}_q(r) \bmod 2$. We cannot compute $(\frac{r}{q})$ since we do not know r; however, this is solved by the "Extraction Lemma" of [1].

Definition 2. *For any integers b, n, the* <u>*mock residue symbol*</u> $\langle b/n\rangle_2$ *is defined as*

$$\left\langle \frac{b}{n} \right\rangle_2 = \begin{cases} \pm 1 \equiv b^{(n-1)/2} \text{ mod } n & \text{if such a congruence holds,} \\ 0 & \text{otherwise.} \end{cases}$$

Mock residue symbols may be easily computed by exponentiation; further, it is clear that if we ever encounter $\langle b/n\rangle_2 = 0$ for $0 < b < n$, then n is composite. There is a corresponding definition of the mock residue symbol for $p > 2$, although (as will be discussed in a moment) the setting is more complicated in that case. The following is a special case of the APR Extraction Lemma, for the case $p = 2$.

Theorem 3 (Extraction Lemma ($p = 2$))**.** *Suppose r is a prime divisor of n. If b is such that $\langle b/n\rangle_2 = -1$, then for any m, c,*

$$\left\langle \frac{b}{n} \right\rangle_2^m = \left\langle \frac{c}{n} \right\rangle_2 \quad \textit{implies} \quad \left(\frac{b}{r} \right)^m = \left(\frac{c}{r} \right).$$

We apply the Extraction Lemma as follows. Identify a q-prime q_0 such that $\langle q_0/n\rangle_2 = -1$, and suppose for simplicity that all of the q-primes are congruent to 1 mod 4. Now for each q-prime q, we can find an integer m_q such that

$$\left\langle \frac{q_0}{n} \right\rangle_2^{m_q} = \left\langle \frac{q}{n} \right\rangle_2.$$

(This statement is obvious; the right-hand side is either 1 or -1, and we simply pick $m_q = 0$ or $m_q = 1$ accordingly. The exponents m_q become more interesting for larger values of p.) We can compute all of these "transition data" m_q by direct exponentiation. They become useful, because by the reciprocity law and Extraction Lemma (recall the congruences mod 4, which cause the correction factor in the reciprocity law to be 1), any prime factor r of n must satisfy

$$\left(\frac{r}{q} \right) = \left(\frac{q}{r} \right) = \left(\frac{q_0}{r} \right)^{m_q}.$$

Thus all the quadratic symbols (r/q) (correspondingly, the indices $\mathrm{Ind}_q(r)$ mod 2) are determined by the single value (q_0/r). There are thus only two choices for the entire set of indices $\{\mathrm{Ind}_q(r) \text{ mod } 2\}$ over all q. Analogously, we expect there to be p choices for the set $\{\mathrm{Ind}_q(r) \text{ mod } p\}$, and thus $\prod_{p\in\mathcal{P}} p$ choices for r. This provides some justification for the claim that the runtime is controlled by the product of the p's.

This theory, at some level, holds for all primes p. Combining these ideas, in general terms the following is the original APR primality test.

1. Find a collection $\mathcal{P}$ of primes p such that the product of the q-primes they generate exceeds $\sqrt{n}$. Find generators for the q-primes. Compute certain Jacobi sums (sums of residue symbols, discussed below).
2. For each prime p in $\mathcal{P}$, compute the "transition data" exponents $m_{i,q}$ relating a fixed (not equal to 1) mock residue symbol to each other mock residue symbol. (This is where the Jacobi sums are used, for $p > 2$.)
3. Using the Extension Lemma, find all possibilities for prime factors r of n. Test each; if one divides n, deduce that n is composite. If none do, deduce that n is prime.

The proof of correctness and runtime analysis for the APR Jacobi sums test (which we, of course, have not yet fully described) are given by the following, Theorem 1 from [1].

(Note: they define $f(n)$ to be the product of the primes in $\mathcal{P}$; recall that it is asymptotically $O((\log n)^{c \log\log\log n})$.)

Theorem 4. *The above algorithm [the Jacobi sums test] correctly determines whether n is prime or composite, if it terminates. There is an absolute, calculable constant $c_1 > 0$ such that for every $k \geq 1$, if n is prime, the algorithm terminates within $T_k(n)$ steps with probability greater than $1 - 2^{-k}$, where*

$$f(n) \leq T_k(n) \leq kf(n)^{c_1}.$$

The above discussion, while it captures the flavor of the test, is unfortunately limited to $p = 2$. In particular, the reciprocity law is not as straightforward for $p > 2$; the natural field in which to state it is $\mathbb{Q}[\zeta_p]$, where $\zeta_p = e^{2\pi i/p}$ is a primitive p^{th} root of unity. (In the case $p = 2$, $\zeta_2 = -1$ so this is actually not a proper extension.) This is natural, as one may expect that generalized residue symbols $(a/b)_p$ would have values which are p^{th} roots of unity. There is a reciprocity law and Extraction Lemma in these settings, but the correct base values to consider are primes in the extension ring $\mathbb{Z}[\zeta_p]$ (not rational primes).

This is where Jacobi sums make their appearance. For a prime $\mathfrak{p}$ of $\mathbb{Q}[\zeta_p]$ not dividing p and integers a, b, [1] defines the Jacobi sum

$$J_{a,b}(\mathfrak{p}) = \sum \left(\frac{x}{\mathfrak{p}}\right)_p^{-a} \left(\frac{1-x}{\mathfrak{p}}\right)^{-b}$$

with the sum taken over coset representatives of $\mathbb{Z}[\zeta_p]/\mathfrak{p}$. (Note: Jacobi sums are generally defined in terms of characters; the (powers of) generalized residue symbols are examples of such objects.) The usefulness of Jacobi sums is that they have a known factorization into primes in the extension ring, and also yield a trivial coefficient in the power reciprocity law (the analogue of an integer congruent to 1 mod 4 in the quadratic reciprocity law).

A full explanation of this is well beyond the scope of this paper, so we content ourselves with the above motivation and background. To achieve our goal of an implementable description, we resort to repeating a summary from [10, pp. 455–457]. With the exception of the intermediate discussion, a few unsubstantial comments and rewordings, and one correction, the following procedure is copied verbatim. The outline refers to Jacobi sums of characters (which have not been defined); it also gives formulas for these sums, though, so the actual definition is not necessary.

Preparatory steps

1. Let B be an upper bound on the numbers N which we will test. The following steps will depend only on B.

2. For even integers t, define

$$e(t) = 2 \prod_{q \text{ prime}, (q-1)|t} q^{v_q(t)+1},$$

where $v_q(t)$ is the number of powers of q dividing t. Using a table (such as Table 1, [12]), find a t such that $e^2(t) > B$.

3. For every prime q dividing $e(t)$ with $q \geq 3$, do the following:

(a) Find a primitive root g_q modulo q, and make a table of the function $f(x)$ mapping $\{1, 2, \ldots, q-2\}$ to itself, given by $g_q^{f(x)} = 1 - g_q^x$.

(b) For every prime p dividing $q-1$, let $k = v_p(q-1)$ and let $\chi_{p,q}$ be the character defined by $\chi_{p,q}(g_q^x) = \zeta_{p^k}^x$.

(c) For $p \geq 3$ compute

$$J(p,q) = \sum_{1 \leq x \leq q-2} \zeta_{p^k}^{x+f(x)}.$$

If $p=2$ and $k \geq 2$, compute $J(2,q)$ as above. If $p=2$ and $k \geq 3$, compute in addition

$$j(\chi_{2,q}^2, \chi_{2,q}) = \sum_{1 \leq x \leq q-2} \zeta_{2^k}^{2x+f(x)},$$

$$J_3(q) = j_3(\chi_{2,q}, \chi_{2,q}, \chi_{2,q}) = J(2,q) j(\chi_{2,q}^2, \chi_{2,q}),$$

and

$$J_2(q) = j^2\left(\chi_{2,q}^{2^{k-3}}, \chi_{2,q}^{2^{k-3}}\right) = \left(\sum_{1 \leq x \leq q-2} \zeta_8^{3x+f(x)}\right)^2.$$

This concludes the preparatory stages. It should be noted that the powers of ζ_8 in computing $J_2(q)$ are actually viewed as the corresponding elements of $\mathbb{Z}[\zeta_{2^k}]$, by the correspondence $\zeta_8 = \zeta_{2^k}^{2^{k-3}}$.

In our implementation, as suggested in [12], we use $t = 5040 = 2^4 \cdot 3^2 \cdot 5 \cdot 7$, which has $e(5040) \sim 10^{52}$. We can thus handle test primes up to 100 decimal digits (so upwards of 300 binary digits). The prime factorization of $e(5040)$ (which gives the values of q) and of $q-1$ (which gives the primes p) are given below, from Table 2 of [12].

q^{v_q}	$q-1$	q^{v_q}	$q-1$	q^{v_q}	$q-1$
2^6	1	31	$2\cdot 3\cdot 5$	181	$2^2\cdot 3^2\cdot 5$
3^3	2	37	$2^2\cdot 3^2$	211	$2\cdot 3\cdot 5\cdot 7$
5^2	2^2	41	$2^3\cdot 5$	241	$2^4\cdot 3\cdot 5$
7^2	$2\cdot 3$	43	$2\cdot 3\cdot 7$	281	$2^3\cdot 5\cdot 7$
11	$2\cdot 5$	61	$2^2\cdot 3\cdot 5$	337	$2^4\cdot 3\cdot 7$
13	$2^2\cdot 3$	71	$2\cdot 5\cdot 7$	421	$2^2\cdot 3\cdot 5\cdot 7$
17	2^4	73	$2^3\cdot 3^2$	631	$2\cdot 3^2\cdot 5\cdot 7$
19	$2\cdot 3^2$	113	$2^4\cdot 7$	1009	$2^4\cdot 3^2\cdot 7$
29	$2^2\cdot 7$	127	$2\cdot 3^2\cdot 7$	2521	$2^3\cdot 3^2\cdot 5\cdot 7$

We see from this table that, while there are many q-primes (which is necessary, since their product must be large), there are only four primes p, and the largest power present of any of them is $p^k = 2^4$. The computations of Jacobi sums take place in extension rings $\mathbb{Z}[\zeta_{p^k}]$. We handle elements of this ring as vectors of length p^k, where multiplication takes place by treating a given vector as a polynomial in ζ_{p^k} and using $\zeta_{p^k}^x = \zeta_{p^k}^{x \bmod p^k}$. (In fact, looking ahead to the AKS method, we treat $\mathbb{Z}[\zeta_{p^k}]$ like the polynomial quotient ring $\mathbb{Z}[X]/(X^{p^k}-1)$.) In [11], it is suggested that the extension ring be stored with dimension $(p-1)p^{k-1}$; this would save storage space but (in the author's opinion) complicates the implementation overall.

In our implementation, we use the simple "grammar school" method for multiplying polynomials, which multiplies every pair of coefficients and adds up the corresponding terms.

Since multiplications are slow, while additions are fast, we should use a recursive, asymptotically improved method based on the observation that the product

$$(a_1X + a_0)(b_1X + b_0) = c_2X^2 + c_1X + c_0$$

can be computed, instead of $c_2 = a_1b_1$, $c_1 = a_0b_1 + a_1b_0$, $c_0 = a_0b_0$ (which requires 4 multiplications), as $c_2 = a_1b_1$, $c_0 = a_0b_0$, and $c_1 = c_0 + c_2 - (a_1 - a_0)(b_1 - b_0)$ (which requires only 3 multiplications). Such an algorithm is described in [10, pg. 109].

We now give the actual Jacobi sums primality test, which proves that a suspected prime N (which is $\leq B$, the bound from the preparatory stages) is actually prime. Again, most of our description is simply copied from [10, pp. 455–457], with minor explanations added. The one change worth noting is a correction: step 4b below had a typo in [10], as we discovered by comparison with [11]. (The δ_N exponent was originally doubled; it should not be, as J_2 is already defined to be squared.)

We need some notation concerning group rings. The Galois group G of the extension $\mathbb{Q}[\zeta_{p^k}]/\mathbb{Q}$ is the set of automorphisms σ_a for integers a relatively prime to p^k, mapping $\sigma_a(\zeta_{p^k}) = \zeta_{p^k}^a$. We view the group ring $\mathbb{Z}[G]$ as the set of formal sums $\sum_a c_a\sigma_a$ with coefficients c_a in $\mathbb{Z}$. We extend the natural action of G on $\mathbb{Q}[\zeta_{p^k}]$ to a *multiplicative* action of $\mathbb{Z}[G]$ on this field; i.e. the action of $\sum_a c_a\sigma_a$ on x is $\prod_a \sigma_a(x)^{c_a}$. We denote the action of $f \in \mathbb{Z}[G]$ on $x \in \mathbb{Q}[\zeta_{p^k}]$ by x^f.

Primality test

1. With the t from the preparatory stage, check that $(te(t), N) = 1$; otherwise N is composite.

2. For every prime $p \mid t$, define the constant l_p to be $l_p = 1$ if $p \geq 3$ and $N^{p-1} \not\equiv 1 \pmod{p^2}$, and $l_p = 0$ otherwise.

3. For each pair (p, q) of primes such that p^k exactly divides $q-1$ ($p^k \mid (q-1)$ but $p^{k+1} \nmid (q-1)$) and $q - 1$ divides t, go to step 4a if $p \geq 3$, step 4b if $p = 2$ and $k \geq 3$, step 4c if $p = 2$ and $k = 2$, or step 4d if $p = 2$ and $k = 1$. After this, proceed to step 5.

4a. Let E be the set of integers between 0 and p^k which are not divisible by p. Define the element Θ of the group algebra $\mathbb{Z}[G]$ by $\Theta = \sum_{x\in E} x\sigma_x^{-1}$, the constant $r = N \bmod p^k$, and the element $\alpha = \sum_{x\in E}\lfloor \frac{rx}{p^k}\rfloor\sigma_x^{-1}$ in $\mathbb{Z}[G]$. Compute $s_1 = J(p, q)^\Theta \bmod N$, $s_2 = s_1^{\lfloor N/p^k\rfloor} \bmod N$, and $S(p, q) = s_2J(p, q)^\alpha \bmod N$.

If there does not exist a p^k-th root of unity η such that $S(p, q) \equiv \eta \pmod N$, then N is composite. Otherwise, if η exists and is a primitive root of unity, set $l_p = 1$.

4b. Let E be the set of integers between 0 and 2^k which are congruent to 1 or 3 modulo 8. Define $\Theta = \sum_{x\in E} x\sigma_x^{-1}$ in $\mathbb{Z}[G]$, set $r = N \bmod 2^k$, and define $\alpha = \sum_{x\in E}\lfloor \frac{rx}{2^k}\rfloor\sigma_x^{-1}$ in $\mathbb{Z}[G]$. Compute $s_1 = J_3(q)^\Theta \bmod N$, $s_2 = s_1^{\lfloor N/p^k\rfloor} \bmod N$, and finally $S(2, q) = s_2J_3(q)^\alpha J_2(q)^{\delta_N}$, where $\delta_N = 0$ if $r \in E$ and equals 1 otherwise.

If there does not exist a 2^k-th root of unity η such that $S(2, q) \equiv \eta \bmod N$, then N is composite. Otherwise, if η exists and is a primitive root of unity, and in addition $q^{(N-1)/2} \equiv -1 \bmod N$, then set $l_2 = 1$.

4c. Set $s_1 = J(2, q)^2 \cdot q \bmod N$, $s_2 = s_1^{\lfloor N/4\rfloor} \bmod N$, and finally $S(2, q) = s_2$ if $N \equiv 1 \bmod N$ and $S(2, q) = s_2J(2, q)^2$ if $N \equiv 3 \bmod 4$.

If there does not exist a fourth root of unity η such that $S(2,q) \equiv \eta \bmod N$, then N is composite. Otherwise, if such an η exists and is a primitive root, and in addition $q^{(N-1)/2} \equiv -1 \bmod N$, set $l_2 = 1$.

4d. Compute $S(2,q) = (-q)^{(N-1)/2} \bmod N$. If $S(2,q) \not\equiv \pm 1 \bmod N$, then N is composite. If $S(2,q) \equiv -1 \bmod N$ and $N \equiv 1 \bmod 4$, set $l_2 = 1$.

5. For every $p \mid t$ such that $l_p = 0$, do as follows. Choose random primes q such that $q \nmid e(t)$, $q \equiv 1 \bmod p$, and $(q, N) = 1$. Execute the appropriate step 4(a–d) according to the pair (p, q). (This requires computing new Jacobi symbols; we use the same algorithm as in the preparatory stages.)

If after many attempts, some l_p is still equal to 0, then the test fails.

6. For $i = 1, 2, \ldots, t-1$, compute (by induction) $r_i = N^i \bmod e(t)$. If for some i, r_i is a non-trivial divisor of N, then N is composite. Otherwise, N is prime.

While this algorithm is somewhat opaque, it is nonetheless possible to implement without delving into the details of power reciprocity laws over cyclotomic extensions. While lengthy, the final algorithm really only requires basic arithmetic over cyclotomic extensions (although the proof of correctness requires arguments well beyond our present discussion).

There is an issue in the cyclotomic arithmetic which requires clarification. The roots of unity in $\mathbb{Z}[\zeta_{p^k}]$ have representations which are all zero, except for a single one. However, when testing whether an element x is a root of unity (modulo some number N), it is not enough to check if the representation of x has this form. One complication is that it is possible for an element other than $\zeta_{p^k}^j$ to be a root of unity, when the ring is taken modulo N. (If N is prime, then this is possible iff $p \mid (N-1)$. In this case, the problem is more aptly stated as the cyclotomic extension $(\mathbb{Z}/N\mathbb{Z})[\zeta_{p^k}]$ not having the full degree expected. However, as is stated in [11], we are actually working not in $(\mathbb{Z}/N\mathbb{Z})[\zeta_{p^k}]$, but in $\mathbb{Z}[\zeta_{p^k}]/(N)$.) To sidestep this issue, when checking whether x is a root of unity, we simply check whether powers of x equal unity.

There is, however, a further complication. Recall that we are representing elements of $\mathbb{Z}[\zeta_{p^k}]$ as length p^k vectors, but the extension actually has dimension $(p-1)p^{k-1}$. (The dimension follows directly from the order of the Galois group.) This implies that there is a vector space of dimension p^{k-1} of length p^k vectors which are actually zero; these are spanned by the combinations

$$\left\{ \sum_{j=0}^{p-1} \zeta_{p^k}^{i+p^{k-1}j} : i \in \{0, 1, \ldots, p^{k-1}-1\} \right\}.$$

Thus when checking whether x is unity, we check if (after removing one), the representation of x repeats with period p^{k-1}.

2. Elliptic Curve Method

In contrast with APR methods, which provably have a superpolynomial runtime, the elliptic curve methods developed by A.O.L. Atkin and F. Morain have a conjectured polynomial runtime. Their method is similar to an approach previously proposed by S. Goldwasser and J. Kilian [15], and the Goldwasser-Kilian (henceforth G-K) test was in turn based on the DOWNRUN process of [26]. After describing the DOWNRUN concept and the basis for the G-K test, we will describe the Atkin-Morain method.

The DOWNRUN process can be easily explained in broad terms: given some number p which we want to prove prime, we construct a decreasing sequence $p = p_0 > p_1 > p_2 > \dots$ of likely primes such that the primality of p_j implies the primality of p_{j-1}. An example helps develop the concept. (The algorithm developed in [26] is, of course, more general than this example suggests.) Suppose we want to prove p is prime, and have already checked it is not a perfect power. Then we have primality if and only if $\mathbb{Z}/p\mathbb{Z}$ is cyclic, so we just need to demonstrate a generator; to prove an element g is a generator, it suffices to show $g^{(p-1)/r}$ is not the identity for all primes r dividing $p-1$. Suppose we can factor $p-1$ into a fully factored piece a, and a likely prime b. Then we search for an element g which has $g^{(p-1)/r} \not\equiv 1$ for all primes r dividing a, as well as $r = b$. From this we can deduce that g is a generator (so p is prime) *as long as b is in fact prime.*

If we could always make this reduction, then we would have an effective prime proving algorithm. Unfortunately, sometimes $p-1$ is not easy to factor. The core idea of the elliptic curve method is thus to move out of the group $(\mathbb{Z}/p\mathbb{Z})^*$ (which has order $p-1$) to groups of elliptic curves over the field $\mathbb{Z}/p\mathbb{Z}$. Different curves have different orders, and if we can successfully factor one of these orders into a fully factored part and a probable prime, then we can apply the same sort of DOWNRUN reduction.

The study of elliptic curves is itself an interesting subject, but here we just give an operating definition and some basic theory which can be found in any text on the subject. For a fixed field $\mathbb{F}$ (with characteristic 0 or relatively prime to 6) and parameters a, b (satisfying $4a^3 + 27b^2 \neq 0$), consider the set of solutions (x, y, z) to the equation

$$y^2 z = x^3 + axz^2 + bz^3.$$

This equation is scaling invariant, so we can instead consider solutions as lying in the projective plane over $\mathbb{F}$. (We denote the equivalence class containing (x, y, z) by $(x : y : z)$.) This set of solutions is an elliptic curve, denoted $E(\mathbb{F})$.

$E(\mathbb{F})$ contains a single point with $z = 0$, $O_E = (0 : 1 : 0)$. For all other points, we report the representative with $z = 1$. We can place an operation on $E(\mathbb{F})$ which makes it an abelian group with identity O_E. For two non-identity points $(x_1 : y_1 : 1)$ and $(x_2 : y_2 : 1)$ on $E(\mathbb{F})$, we define their product $(x_3 : y_3 : 1)$ by

$$\begin{array}{rcl} x_3 & = & \lambda^2 - x_1 - x_2 \\ y_3 & = & \lambda(x_1 - x_3) - y_1 \end{array} \qquad \text{with} \qquad \lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{if } x_2 \neq x_1 \\ (3x_1^2 + a)(2y_1)^{-1} & \text{otherwise.} \end{cases}$$

When attempting to prove an integer p is prime, we will consider elliptic curves over $\mathbb{Z}/p\mathbb{Z}$. This is, of course, only a field if p is actually prime; if an inversion in the group operation fails, then we naturally conclude p is composite.

As a brief aside, whenever exponentiating a point on an elliptic curve, we use fast log-time operations. The simplest such (and the one actually implemented) is the "square first" technique which computes a^b by recursively finding $a^{\lfloor b/2 \rfloor}$, squaring, and then multiplying by another a if b was odd. More efficient algorithms, such as Brauer's algorithm, improve on this. If multiple powers of the same base are desired, there are improved algorithms, such as Straus's, Yao's, and Pippenger's. A summary of these is provided in [4]. We denote the kth power of a point P additively by the notation kP.

Another elliptic curve algorithm we will need is point finding; this can be done quickly by (setting $z = 1$ and) randomly generating an x until $x^3 + ax + b$ is a quadratic residue modulo N (checked using the fast algorithm for computing Legendre symbols), and then extracting the square root. There exist efficient algorithms for finding a root; see [13, pg. 100].

Using the wording of [3], Goldwasser and Kilian give the following theorem and corollary.

Theorem 5. *Let N be an integer prime to 6, E an elliptic curve over $\mathbb{Z}/N\mathbb{Z}$, together with a point P on E and m and s two integers with $s \mid m$. For each prime divisor q of s, we put $(m/q)P = (x_q : y_q : z_q)$. We assume that $mP = O_E$ and $\gcd(z_q, N) = 1$ for all q. Then, if p is a prime divisor of N, one has $\#E(\mathbb{Z}/p\mathbb{Z}) \equiv 0$ mod s.*

Corollary 6. *With the same conditions, if $s > (\sqrt[4]{N} + 1)^2$, then N is prime.*

For cleanliness, we restate this in the form we will use.

Theorem 7 (Goldwasser-Kilian Prime Condition)**.** *Suppose N is an integer not divisible by two or three, E is an elliptic curve over $\mathbb{Z}/N\mathbb{Z}$, and we have integers q, m with q prime, $q > (\sqrt[4]{N} + 1)^2$, and $q \mid m$. If there exists a point P on E such that $mP = O_E$ but $(m/q)P \neq O_E$, then N is prime.*

Using the polynomial-time algorithm for checking the number of points on an elliptic curve over a finite field (Schoof's algorithm, see [16]), the Goldwasser-Kilian elliptic curve test proves primality of a number N by (1) generating elliptic curves until one has number of points $2q$, q a likely prime, (2) using the G-K Prime Condition to prove N is prime (assuming q is prime), and then (3) using the same algorithm to prove primality of q, as in the DOWNRUN process.

Unfortunately, though, Schoof's algorithm for counting the number of points seems not to be feasible for an actual implementation. This leads us to the closely related elliptic curve prime proving (ECPP) algorithm of Atkin and Morain. It is described in [3], with additional improvements in [20]. We work with the original algorithm, ignoring the special case speedups of later papers (although our description follows most closely the summary in [20]).

There is an asymptotic improvement of the elliptic curve technique, called fastECPP and originally due to J.O. Shallit. The interested reader is referred to [19] for a description thereof.

There are two main implementations available of the ECPP algorithm, although both are binary-only releases. One is a package called ECPP, written by Morain [24]. This package's record of largest proven prime appears to be $907^{694} + 694^{907}$, with 2578 decimal digits. (Morain, using a private distributed version of his program, proved the primality of the Mills' prime, with 20562 decimal digits.) The other implementation is Primo (formerly Titanix), due to Marcel Martin [22]. This package holds the single processor record, having proven the primality of the number $18517\# + 39317$ with 7993 decimal digits. The software package MAGMA also has an internal implementation of ECPP (due to Morain), as does (according to [3]) the Mathematica package.

The basic idea for moving from the G-K algorithm to the ECPP is that, instead of generating random elliptic curves and testing their order, we generate elliptic curves in such a way that we know their orders in advance. The outline of ECPP, as given in [20], is as follows.

```
function ECPP($N$): boolean;
  1. if $N < 1000$ then check the primality of $N$ directly and return the answer.
  2. Find an imaginary quadratic field $\mathbb{K} = \mathbb{Q}(\sqrt{-D})$ ($D > 0$) for which the equation
     $4N = x^2 + Dy^2$ has solutions in rational integers $x$ and $y$.
```

3. For each pair (U, V) of solutions, try to factor $m = N + 1 - U$. If one of these can be written as $F \times \bar{\omega}$ where F is completely factored and $\bar{\omega}$ is a probable prime, then go to step 4 else go to step 2.

4. Find the equation of the curve E having m points modulo N and a point P on it. If the primality condition is satisfied for (N, E, P), then return ECPP($\bar{\omega}$). Otherwise, return composite.

5. end.

There is a fast algorithm due to Cornacchia for finding solutions to the equation in step (2) (dependent, as one may expect, on the algorithm for extracting square roots modulo a prime) which is described in [3]. We will discuss techniques for factoring m later. The most opaque step is (4), finding a curve E with m points. This is accomplished by looking at elliptic curves with complex multiplication, for which the order is given by an appeal to class field theory. While the details are well outside the scope of this paper, the summary is this:

(a) For a given discriminant D with an integral solution to $4N = U^2 + DV^2$:
(b) Compute the Hilbert class polynomial for this discriminant,
(c) Find a root j to this polynomial mod N, and use this to compute elliptic curve parameters.

The curves thus generated have a particular order (actually, depending on D there are either 2, 4, or 6 possibilities for the order). For step (b), the class polynomial may be computed by using modular forms over the complex numbers. There exists an efficient algorithm for finding roots to a polynomial modulo a prime, and then the elliptic curve parameters depend on this root via a simple formula. This process is explained in a very clear and implementable fashion in [13, pg. 360].

Since the class polynomial is an integer coefficient polynomial dependent only on the discriminant, we can (and should) precompute it. Further, for certain discriminants (specifically, those with class number $h(D) = 1$ or 2) the solutions to the elliptic curve parameters may be expressed using only square roots; thus, if we are willing to limit ourselves to those discriminants, we can precompute and store the result in a manner which allows us to neatly sidestep any foray into class field theory.

The discriminants of this type are

$$D = 3, 4, 7, 8, 11, 19, 43, 67, 163, 15, 20, 24, 35, 40, 51, 52, 88, 91, 115, 123, 148, 187, 232, 235, 267, 403, 427.$$

(Reminder: the actual discriminant of the quadratic field is negative, but since we only ever deal with negative discriminants we make the convenient cheat of always referring to it as positive.) In what follows, suppose (U, V) is a solution to $4N = U^2 + DV^2$, and g is a non-quadratic residue mod N (if $D = 3$, we also require that it not be a cubic residue, but this is a simple additional check). We will give the parameters (a, b) defining our particular elliptic curves of interest. It deserves to be emphasized that this theory only works if N is indeed prime.

The cases $D = 3$ and $D = 4$ are special. For $D = 3$, we have

Curve Orders	Curve Parameters
$\{N + 1 \pm U, N + 1 \pm (U \pm 3V)/2\}$	$(a, b) = \{(0, -g^k \bmod N) : k = 0, 1, 2, 3, 4, 5\}$

and for $D = 4$,

Curve Orders	Curve Parameters
$\{N+1\pm U, N+1\pm 2V\}$	$(a,b)=\{(-g^k \bmod N, 0) : k=0,1,2,3\}$.

For all the other discriminants, the curve orders are $N+1\pm U$ and the curve parameters are $(a,b)=\{(r,s),(rg^2 \bmod N, sg^3 \bmod N)\}$ where r and s are the parameters given in Table 7.1 of [13, pg. 365]. For example, the $D=15$ parameters are

$$r = 1225 - 2080\sqrt{5} \qquad \text{and} \qquad s = 5929.$$

It should be noted that there is no obvious correspondence telling which parameters correspond to which curve orders; in our implementation, we just search for points which distinguish between the possible orders. (For example, for the cases $D>4$ we look for a point P which has $(2U)P \neq O_E$; then checking $(N+1+U)P$ will determine which curve order we have.) Some recent results, in particular of Stark, tell which curve has order $N+1+U$ by checking quadratic residues; this is mentioned in [20] but not implemented here.

Once we generate a curve, checking the conditions of the G-K Primality Test just amounts to finding a point with $FP \neq O_E$ and verifying $mP = O_E$. Since our method actually generated curves of a few different orders, we could actually relax step (3) to try any of these orders m. This is the advantage of the elliptic curve method; there are many different orders to try to factor.

The final step to discuss is actually performing the factorization in step (3). Following [3], we first do trial division on m with several small primes, removing any multiples found. This suffices for small numbers, but when m is large it is likely that such trial division will not result in a prime quotient. Thus, we also use rounds of Pollard's ρ test, a factorization method whose elegance justifies a brief aside to explain it.

The following description is from [13, pg. 230]. Suppose, as is the case, that we want to find a prime factor p of a number m. Suppose we knew p, and let f be a random function from $\mathbb{Z}/p\mathbb{Z}$ to itself. Since this set is finite, for any starting value s, the sequence $\left\{f^{(k)}(s)\right\}_k$ must repeat itself, likely (with an appeal to probability theory) after $\sqrt{p}$ steps. Now we do not know p, so we instead must use a function F on $\mathbb{Z}/m\mathbb{Z}$. The iterates of this function will repeat modulo p, so p will divide $F^{(j)}(s) - F^{(k)}(s)$, for some j,k, as well as m; in particular, it divides their gcd. Since we expect $p \ll m$, we should be able to find iterates which have repeated modulo p but not modulo m; thus $\gcd(F^{(j)}(s) - F^{(k)}(s), m)$ should be a nontrivial factor. This is the idea of Pollard's ρ test; we use a random starting value $s \in \mathbb{Z}/m\mathbb{Z}$, the pseudorandom function $F(x) = x^2 + a$ (for a randomly selected a), and check $\gcd(F^{(i)}(s) - F^{(2i)}(s), m)$ for some specified number of iterations i. If it is ever larger than 1 but smaller than m, then we have found a nontrivial factor.

In [3], the authors also suggest an elliptic curve factorization method. We do not implement that, and thus our algorithm is limited to situations in which the ρ test can successfully factor m.

We have completed an algorithmic description of the elliptic curve primality proving technique. A proof of correctness for the algorithm is far outside the scope of this work; the reader is referred to [3] for a more formal approach to this theory. As for runtime, we only have conjectures; there appears to be no reasonable hope for a theoretical derivation of the runtime for ECPP. A conjectured runtime of $O((\log N)^{6+\epsilon})$ is given in [3]; in [19], it is suggested that the runtime is actually $O((\log N)^{5+\epsilon})$ and that the fastECPP improvement has runtime $O((\log N)^{4+\epsilon})$.

Before closing this section, we should note the issue of verification. The elliptic curve method is unique (of the three studied here) in that, once generated, a proof of primality can be quickly verified. Indeed, scanning over the work of the ECPP algorithm, much of the runtime is in finding the appropriate values to use (picking a discriminant, finding a solution $4N = U^2 + DV^2$, and especially finding a large prime factor of m). The values chosen can be saved once generated, and then an outside program could check this "certificate" rather quickly. In fact, the ECPP package [24] comes with a verification program, and a description of how to write one. The ability to quickly verify a proof is a clear advantage for the elliptic curve primality proving method.

3. Polynomial AKS Method

By contrast with the first two methods, the polynomial exponentiation method for primality proving, due to M. Agrawal, N. Kayal, and N. Saxena, is quite simple. It offers a completely deterministic, provably polynomial-time algorithm for proving primality, thus proving that the PRIMES problem is in fact in the complexity class P. This explains the simple title of their paper [2], a relatively short paper offering their algorithm along with proof and analysis of runtime. Indeed, it is quite surprising that such a key result could come from relatively simple arguments "suited for undergraduates" [7].

The AKS paper has undergone a few revisions; the most recent version offers a simplified proof due to H. Lenstra. An asymptotic improvement of the AKS algorithm, due to H. Lenstra and C. Pomerance, is given in [17]. We, however, describe and implement the original algorithm.

This method might be described as "proving primality with combinatorics" [5], since it relies on combinatorial-inspired identities of the form

$$(x + a)^p \equiv x^p + a \bmod p.$$

More specifically, consider the following well-known lemma.

Lemma 8. *Suppose $n \geq 2$ and $(a, n) = 1$. Then n is prime if and only if $(X + a)^n \equiv X^n + a$ (mod n).*

Proof. By the binomial theorem, the leading coefficient of $(X + a)^n$ is one, the constant term is a^n (which is congruent to a if n is prime), and the intermediate coefficients are $\binom{n}{j}a^j$. These coefficients all vanish mod n if n is prime, proving ($\Rightarrow$). If n is not prime, then consider q a prime divisor such that q^k exactly divides n (i.e. $q^k \mid n$ and $q^{k+1} \nmid n$). Then $\binom{n}{q}$ is not divisible by q^k (write the binomial coefficient as $n(n-1)\cdots(n-q+1)/q!$; then the only term divisible by q in the numerator is n, and one factor of q is canceled by $q!$). Since $(a, n) = 1 \implies q \nmid a^q$, the q^{th} coefficient does not vanish mod q^k, so does not vanish mod n. $\square$

Evaluating $(X + a)^n$ can be done with (approximately) $\log n$ multiplications; however, the resulting polynomial has degree n. In particular, one of the polynomials in the last multiplication will have degree at least $n/2$. This multiplication alone takes time polynomial in n (so exponential in $\log n$). The idea of the AKS algorithm is to consider the term $(X + a)^n$ not in the infinite ring $\mathbb{Z}[X]/(n)$, but in the finite ring $\mathbb{Z}[X]/(n, X^r - 1)$ for some appropriately chosen r.

The algorithm, as copied from [2], is as follows. For two relatively prime numbers a and r, we denote the order of a modulo r (the order of a in the group $(\mathbb{Z}/r\mathbb{Z})^*$) by $o_r(a)$. Recall that

$\phi(r)$, the Euler totient function, is the number of elements of $(\mathbb{Z}/r\mathbb{Z})^*$. Now the algorithm is as follows.

```
Input: integer $n > 1$.
  1. If ($n = a^b$ for $a \in \mathbb{N}$ and $b > 1$), output COMPOSITE.
  2. Find the smallest $r$ such that $o_r(n) > \log^2 n$.
  3. If $1 < (a, n) < n$ for some $a \leq r$, output COMPOSITE.
  4. If $n \leq r$, output PRIME.
  5. For $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ do
           if ($(X + a)^n \neq X^n + a \ (mod\ X^r - 1, n)$), output COMPOSITE;
  6. Output PRIME;
```

Some explanation of how to implement these steps is in order. First, as we shall mention below, an r as in step (2) always exists, and is polynomial in the number of bits $\log n$. Thus it is reasonable to compute the order of n in $(\mathbb{Z}/r\mathbb{Z})^*$ by either factoring $\phi(r)$ and raising n to the factors, or just exponentiating n to consecutive powers until 1 is encountered. We take the latter approach. Since r will be small, we compute $\phi(r)$ by simply testing against a precomputed table of primes to find the prime divisors of r.

In step (3), it suffices to verify that n has no divisors less than r.

The tests in step (5) may be performed by fast exponentiation techniques, storing polynomials as vectors of size r. As in the Jacobi sums test, we will use the simple "grammar school" algorithm for polynomial multiplication; a recursive approach would offer significant improvements.

The remaining (potentially) unclear step is (1). Actually, according to Exercise 4.28 of [13, pg. 222], this step may be skipped, but we will take this opportunity anyway to discuss an interesting algorithm for testing if a number is a perfect power. First, it clearly suffices to check that n is not a power a^b for $b \leq \log_2 n$. Thus, we get an efficient test by simply trying every such power b (alternatively, every prime b up to the bound) and checking if $(\lfloor n^{1/b} \rfloor)^b = n$. The floor of this root can be computed using the following integer Newton's method. This pseudocode is copied directly from [14] (with minor typos corrected), which was the author's source for this algorithm.

```
int_root($p$, $b$) { /* Returns $\lfloor p^{1/b} \rfloor$. */
  $x = 2^{\lceil B(p)/b \rceil}$; /* $B(n)$ = number of bits in $n$. */
  while(1) {
      $y = \lfloor ((b-1)x + \lfloor p/x^{b-1} \rfloor)/b \rfloor$;
      if ($y \geq x$) return $x$;
      $x = y$;
  }
}
```

The deterministic nature of the AKS primality proving algorithm should be emphasized; unlike the elliptic curve algorithm, which used random algorithms to (for example) find points on a given elliptic curve, every step in the AKS method can be implemented completely deterministically.

Since this algorithm is more accessible than the other methods we have considered, we will discuss the proof of correctness. However, a full proof is not appropriate for this context, so we shall instead give a sketch, following [2] closely; the proof is quite accessible, and the interested reader is referred to the original paper.

First, we can indeed find an r in step (2); according to a lemma, the smallest such r is bounded by $\max\{3, \lceil \log^5 n \rceil\}$. The proof of this lemma is a counting argument relying on the fact that, for $m \geq 7$, the least common multiple of $\{1, 2, \ldots, m\}$ is at least 2^m.

Now if n is prime then the algorithm does return PRIME; the equations $(X+a)^n \equiv X^n + a$ all hold in $\mathbb{Z}[x]/(n)$, so they hold in any quotient ring of this. Suppose that the algorithm returns PRIME for some n. Now since $o_r(n) > 1$, we can find some prime divisor p of n (possibly $p = n$) with $o_r(p) > 1$. Consider the following definition (dependent on the fixed values of p and r).

Definition 9. *A number $m \geq 0$ is <u>introspective</u> for a polynomial $f(X)$ if $[f(X)]^m \equiv f(X^m)$ $(mod\ X^r - 1, p)$.*

Define $\ell = \lfloor \sqrt{\phi(r)} \log n \rfloor$. Now since the algorithm returned prime, we have that n is introspective for $f(X) = X + a$, for each a with $0 \leq a \leq \ell$. Since p is also introspective, it follows that n/p is introspective. Now one shows that both (1) the set of introspective numbers for a given polynomial and (2) the set of polynomials for which a given number is introspective are closed under multiplication. Define the group G of (residues modulo r of) elements $\{(\frac{n}{p})^i \cdot p^j : i, j \geq 0\}$ (all of which are introspective for $X + a$, for each $a \leq \ell$) and the group $\mathcal{G}$ of (residues modulo p and a certain polynomial of) polynomials $\{\prod_{a=0}^{\ell} (X+a)^{e_a} : e_a \geq 0\}$ (for which all of G is introspective). Let t be the cardinality of G.

A counting argument now gives a lower bound on the cardinality of $\mathcal{G}$; namely $\#\mathcal{G} \geq \binom{t+\ell}{t-1}$. The crux of this argument is showing that two distinct polynomials of the form $\prod_{a=0}^{\ell}(X+a)^{e_a}$, each with degree less than t, must map to distinct polynomials in the quotient ring.

Now if n is not a power of p, we get the upper bound $\#\mathcal{G} \leq n^{\sqrt{t}}$. This comes from noticing that, if n is not a power of p, then $\{(\frac{n}{p})^i \cdot p^j : 0 \leq i, j \leq \sqrt{t}\}$ has $(\lfloor \sqrt{t} \rfloor + 1)^2 > t$ elements (recall that t was the size of the group of such elements, reduced modulo r). Thus two are equivalent mod r, say m_1 and m_2. Now $X^{m_1} \equiv X^{m_2}$ modulo $X^r - 1$; it follows that each function $f \in \mathcal{G}$ satisfies $f^{m_1} = f^{m_2}$, so is a root of the polynomial $Q(f) = f^{m_1} - f^{m_2}$, in a certain field. Thus $\deg Q = \max\{m_1, m_2\} \geq \#\mathcal{G}$, which gives the desired bound.

Some elementary inequalities show that these bounds conflict, so we deduce that n must indeed be a power of the prime p; we have already removed perfect powers, so n must be prime.

The asymptotic complexity of this algorithm is $O((\log n)^{7.5+\epsilon})$, as analyzed in [2]. The runtime is dominated, not surprisingly, by checking the congruence $(X+a)^n \equiv X + a$ for each value of a up to $\sqrt{\phi(r)} \log n$. An improvement on the bound for r would thus improve the runtime. Such an improvement is possible if either Artin's Conjecture or the Sophie-Germain Prime Density Conjecture holds; this would reduce the runtime to $O((\log n)^{6+\epsilon})$.

While this algorithm certainly represents a theoretical step forward, it is not (yet) an improvement in practice. However, the Lenstra-Pomerance improvement brings the runtime provably to $O((\log n)^{6+\epsilon})$, which is more competitive with (for example) elliptic curve techniques. In a certain special case (namely, when $n - 1$ is divisible by a power of two on the order of $(\log n)^2$), P. Berrizbeitia showed that primality can be proven by checking a single polynomial equality [6]. A description of this algorithm, which has vastly improved runtime $O((\log n)^{4+\epsilon})$, can be found in [13, pp. 213–217].

When this special case holds, the AKS-variant method is extremely fast. There are many primes for which it fails though. It is proposed in [9] that when the prime is not of this special form, we apply a round of ECPP. One round is relatively quick, and gives us a new prime whose primality implies the primality of the original number. Assuming randomness of this reduction, the new prime is likely to fall under the special case heading, at which point we use a round of the AKS-variant to quickly finish the proof. It remains to be seen if this is a computationally worthwhile scheme.

4. Runtime Comparisons

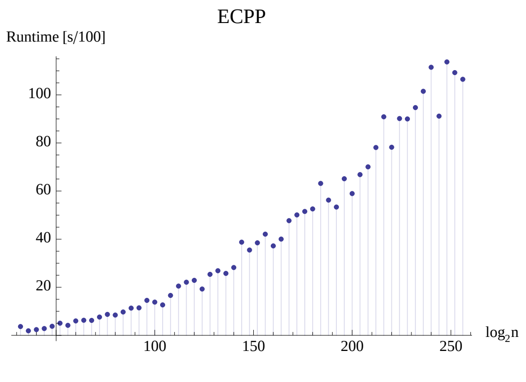
It should be stressed at the beginning that the implementations discussed here are not, by any stretch of the imagination, optimal. In all three cases we used algorithms from early papers, ignoring more recent improvements. Further, no care was taken for optimizing the code.

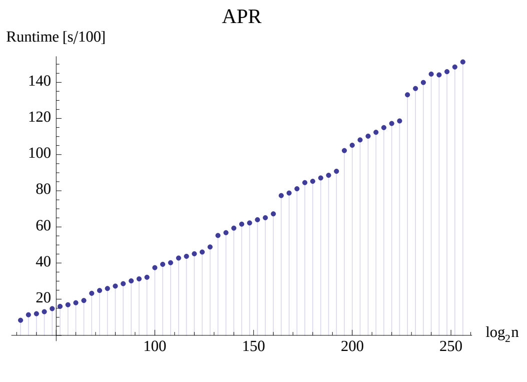
Nonetheless, we offer our results for two reasons. Firstly, with the notable exception of the AKS method, there is limited public availability of these primality proving methods. In particular, the distribution of ECPP seems generally limited to binaries. While our code is not fit for general consumption, the task of preparing a working implementation helps provide a deeper understanding than sketchy descriptions of the methods. Secondly, a comparison of asymptotic runtime does not tell the whole story; constants can vary, so that sometimes the asymptotically inferior algorithm performs better on the input sizes of interest. Thus, we offer a comparison of actual runtimes on a single machine, with a single author's (uniformly unoptimized) code.

In all cases, quoted runtimes are from an Apple Powerbook with a Motorola G4 1.33 GHz processor and 512 MB of RAM. The algorithms were written in `C++`, using the `GMP` library for arbitrary precision integer arithmetic. We use the built-in `clock()` function for runtimes.

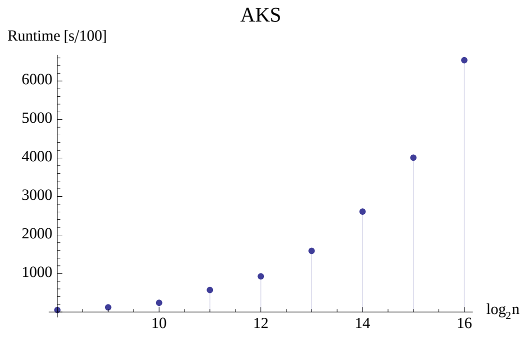
For the ECPP method and the APR Jacobi sums method, we proved primality of 20 different (randomly generated) primes of sizes ranging from 32 bits to 256 bits and found the average runtime for each size. Recall from the discussion above that our implementation of the elliptic curve method is limited by its factorization methods; in particular, some primes fail simply because the elliptic curve orders are not successfully factored by Pollard's ρ test. When generating this plot, we simply threw out such numbers and selected new random primes instead. (This is negligible for smaller sizes, but for some of the larger sizes approximately half of the generated primes failed.) This creates a bias likely to reduce the average runtime of ECPP, as discussed in more detail below.

The runtimes for these methods, with the above caveat, are plotted below.





These runtimes are quite comparable. By contrast, the AKS method ran much slower. For each integer size from 8 to 16 bits, we tested 10 different randomly generated primes; the average runtimes are plotted below.



Notice the difference in scale; while the longest time for the first two methods is approximately one second, proving primality of a number with 16 bits (half the size of the smallest primes considered for the other methods) takes approximately a minute with the AKS method. For numbers of this size, naive trial division would execute much quicker. We close this paper with an individual discussion of each algorithm's performance.

The APR Jacobi sums method, despite being superpolynomial in complexity, performed quite well in practice. In particular, its runtime seems to increase slowly with the size of the primes tested. This is to be expected, though; recall that the APR test first fixes a bound for all numbers to be tested, and then finds a corresponding set of small primes. The runtime arises from the growth rate of that set of primes. We use a fixed bound, though, so that aspect is removed at the cost of having an upper limit on the size of primes we can support.

In fact, the main effect of increasing the size of the prime is just to increase the modulus used in the algorithm's arithmetic operations. A brief discussion of large integer arithmetic can now explain why the runtime appears to be discontinuous, with a jump every 32 bits. The `GMP` library's internal structure stores large integers in chunks, called "limbs." Because of this limb structure, the runtime of basic arithmetic operations will generally increase when the number of limbs increases. The size of a limb, as we could hypothesize from the runtime plot, is indeed 32 bits. (In general, `GMP` defaults to using the machine's native `long` structure, which is 32 bits in this case.)

Given the speedy performance of the APR algorithm in the range considered, it is natural to probe the upper limit. Our implementation is restricted to only handling primes up to about 100 decimal digits; for primes of this size (specifically, we use 332 bits), proving primality with our implementation takes approximately 2.8 seconds. This brings to life the claim of [1] that their algorithm "make[s] routine the testing of primality for numbers some hundreds of digits long."

The ECPP method of Atkin and Morain performs similarly well, supporting the conjectured polynomial run time. In fact, over the range considered, ECPP is consistently faster than the Jacobi sums method. These results even outperform the conjectured runtime of $O(\log^{6+\epsilon}(n))$; our runtime data is well fit by a quadratic polynomial. There are two notable factors, though, which depress our measured runtimes.

First, as already mentioned, our implementation is deficient in its factorization techniques. This forced us to drop some primes from the sample space. To have a general algorithm, we would need to support these "bad" primes via a more powerful factorization technique; in particular, the suggestion of [3] was to use an elliptic curve factoring method. This is asymptotically much slower than the factorization techniques that we implemented, so implementation of it (and handling of the primes that require it) would inflate average runtime.

A second and somewhat related factor is that we used a hardcoded and small set of discriminants. Namely, our implementation only tests discriminants with class number 1 or 2, which are especially "good" according to [3]. For all the primes tested, this set suffices; however, larger primes require more options.

The third method, the AKS polynomial exponentiation technique, is by far the slowest (for the sizes of primes considered here). The advantage of having a provably polynomial runtime is thus primarily theoretical; it is not computationally feasible to run this algorithm for any reasonably large prime.

Of course, there are several improvements possible in the implementation. For one, there is much room for optimization in the representation of polynomials. In particular, polynomial multiplication would be much quicker if we used the recursive algorithm rather than the grammar school method. Since the vast majority of the test's time is spent computing powers of polynomials, this single change could have a drastic effect on the overall runtime. (This issue also affects the APR method, although not to such an exaggerated extent.) Another, more technical improvement would come from better memory management; computing powers of polynomials requires a lot of big integer objects, the allocation and destruction of which is relatively slow. This issue is especially exaggerated for the AKS test, somewhat present in the APR test (which also computes exponentials of vector objects), but is not a terribly important factor in ECPP.

Our results seem to confirm the modern state of affairs that elliptic curves and Jacobi sums are the only two practical options for the general-purpose primality proving of large numbers. However, the AKS combinatorial approach is relatively new; perhaps future developments (see [5], for example) will make this technique more competitive in practice. Also, in theoretical value (namely, *proving* that primality can be deterministically tested in polynomial time), the AKS approach remains unmatched.

References

[1] L. Adleman, C. Pomerance, and R. Rumely, "On distinguishing prime numbers from composite numbers," *Annals of Math.* **117** (1983), pp. 173–206.

[2] M. Agrawal, N. Kayal, and N. Saxena, "PRIMES is in P," *Annals of Math.* **160** (2004), pp. 781–793.

[3] A.O.L. Atkin and F. Morain, "Elliptic Curves and Primality Proving," *Math. of Comput.* **61** (1993), pp. 29–68.

[4] D.J. Bernstein, "Pippenger's Exponentiation Algorithm," preprint (2002), `http://cr.yp.to/papers.html#pippenger`.

[5] D.J. Bernstein, "Proving Primality in Essentially Quartic Random Time," *Math. of Comput.* **76** (2007), pp. 389–403.

[6] P. Berrizbeitia, "Sharpening 'Primes is in P' for a large family of numbers" (2002), `http://arxiv.org/abs/math.NT/0211334`.

[7] F. Bornemann, "Primes is in P: A Breakthrough for Everyman," *Notices of the AMS* **50** (2003), pp. 545–552.

[8] W. Bosma and M.P.M. van der Hulst, *Primality Proving with Cyclotomy*, Ph.D. thesis, Universiteit van Amsterdam, 1990.

[9] Q. Cheng, "Primality proving via one round in ECPP and one iteration in AKS," *Journal of Crypt.* **20** (2007), pp. 375–387.

[10] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer-Verlag, 1993.

[11] H. Cohen and A.K. Lenstra, "Implementation of a New Primality Test," *Math. of Comput.* **48** (1987), pp. 103–121.

[12] H. Cohen and H.W. Lenstra, Jr., "Primality Testing and Jacobi Sums," *Math. of Comput.* **42** (1984), pp. 297–330.

[13] R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*, Springer, 2005.

[14] R. Crandall and J. Papadopoulos, "On the implementation of AKS-class primality tests," Tech. Report (2003), `http://developer.apple.com/hardware/ve/pdf/aks3.pdf`.

[15] S. Goldwater and J. Kilian, "Almost all primes can be quickly certified," *Proc. 18th STOC* (1986), pp. 316–329.

[16] H.W. Lenstra, Jr., "Elliptic curves and number theoretic algorithms," Tech. Rep. Report 86-19, Math. Inst., Univ. Amsterdam, 1986.

[17] H.W. Lenstra, Jr. and C. Pomerance, "Primality Testing with Gaussian Periods," preprint (2005), `http://www.math.dartmouth.edu/~carlp/PDF/complexity12.pdf`.

[18] P. Mihailescu, *Cyclotomy of Rings and Primality Testing*, Ph.D. thesis, ETH Zurich, 1997.

[19] F. Morain, "Implementing the Asymptotically Fast Version of the Elliptic Curve Primality Proving Algorithm," *Math. of Comput.* **76** (2007), pp. 493–505.

[20] F. Morain, "Primality proving using elliptic curves: an update." In *Algorithmic Number Theory*, vol. 1423 of Lecture Notes in Comput. Sci. (1998), pp. 111–127.

[21] "Preda Mihailescu," `http://math-www.uni-paderborn.de/~preda/`.

[22] "Primo overview," `http://www.ellipsa.net/public/primo/primo.html`.

[23] M.O. Rabin, "Probabilistic algorithm for testing primality," *Journal of Number Theory* **12** (1980), pp. 128–138.

[24] "The ECPP home page," `http://www.lix.polytechnique.fr/~morain/Prgms/ecpp.english.html`.

[25] "The Great Internet Mersenne Prime Search," `http://www.mersenne.org/`.

[26] M.C. Wunderlich, "A performance analysis of a simple prime-testing algorithm," *Math. of Comput.* **40** (1983), pp. 709-714.