PRIMES is in P

Manindra Agrawal, Neeraj Kayal and Nitin Saxena*

Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur-208016, INDIA

August 6, 2002

Abstract

We present a deterministic polynomial-time algorithm that determines whether an input number n is prime or composite.

"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length... Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated."

- Karl Friedrich Gauss, *Disquisitiones Arithmeticae*, 1801 (translation from [Knu98])

1 Introduction

Since ancient times, mathematicians have been fascinated by problems concerning prime numbers. One of the fundamental problems concerning prime numbers is to determine if a given number is prime. In modern times, primality testing has also become important from a practical perspective because of its applications in cryptography.

Starting from ancient Chinese and Greek, many have worked on the problem of finding an efficient algorithm for testing primality. The Sieve of Eratosthenes (ca. 240 BC) is the most ancient algorithm that works correctly for all primes, however, its time complexity ($= \Omega(n)$ where n is input number) is exponential in the size of input. In 17th Century, Fermat proved what is referred as *Fermat's Little Theorem* stating that for any prime number p, and any number a not divisible by p, $a^{p-1} = 1 \pmod{p}$. Although the converse of this theorem does not hold (and in fact fails spectacularly for *Carmichael numbers*), this result has been the starting point for several efficient primality testing algorithms. In 1976, Miller [Mil76] used this property to obtain a deterministic polynomial-time algorithm for primality testing assuming *Extended Riemann Hypothesis (ERH)*. His test was modified by Rabin [Rab80] to yield an unconditional but randomized polynomial-time algorithm. Solovay and Strassen [SS77] obtained another randomized polynomial-time algorithm using quadratic residues. (Their algorithm can also be derandomized under ERH). Since then, a number of randomized polynomial-time algorithms have been proposed for primality testing.

In 1983, Adleman, Pomerance, and Rumely achieved a major breakthrough by giving a deterministic algorithm for primality that runs in $(\log n)^{O(\log\log\log n)}$ time (all the previous deterministic algorithms required exponential time). In 1986, Goldwasser and Kilian [GK86] proposed a randomized algorithm based on Elliptic curves running in expected polynomial-time on almost all inputs (*all* inputs under a widely believed hypothesis) that produces a certificate for primality (until then, all randomzied algorithms produced certificates for compositeness only). A similar algorithm was developed by Atkin [Atk86]. Adleman and Huang [AH92] modified Goldwasser-Kilian algorithm to obtain a randomized polynomial-time algorithm that always produced a certificate for primality.

*Email addresses: manindra@cse.iitk.ac.in, kayaln@iitk.ac.in, nitinsa@cse.iitk.ac.in

The ultimate goal of this line of research is, of course, to obtain an unconditional deterministic polynomial-time algorithm for primality testing. Despite the impressive progress made in primality testing so far, this goal has remained elusive. In this paper, we achieve this. We give a deterministic, $\tilde{O}((\log n)^{12})$ time algorithm for testing if a number is prime. Heuristically, our algorithm does much better: under a widely believed conjecture on the density of Sophie Germain primes (primes p such that $2p+1$ is also prime), the algorithm takes only $\tilde{O}((\log n)^6)$ steps. The correctness proof of our algorithm requires only simple tools of algebra (except for appealing to a sieve theory result on the density of primes p with $p-1$ having a large prime factor). In contrast, the correctenss proofs of deterministic algorithms of [APR83, GK86, Atk86] are much more complex.

In section 2, we summarize the basic idea behind our algorithm. In section 3, we state some preliminary theorems and fix the notation used here. Thereafter, we state the algorithm in full detail and present the proof of correctness.

2 Basic Idea and Approach

Our test is based on the following identity for prime numbers. This same identity was basis for a randomized polynomial-time algorithm in [AB99]:

Identity *Suppose that a is coprime to p. Then p is prime if and only if*

$$(x-a)^p \equiv (x^p - a) (\text{mod } p) \tag{1}$$

Proof. For $0 < i < p$, the coefficient of x^i in $((x-a)^p - (x^p - a))$ is $(-1)^i \binom{p}{i} a^{p-i}$. Now if p is prime, $\binom{p}{i} \equiv 0 (\text{mod } p)$ and hence all the coefficients are zero.
If p is composite: consider a prime q that is a factor of p and let $q^k || p$. Then q^k does not divide $\binom{p}{q}$ and is coprime to a^{p-q} and hence the coefficient of x^q is not zero (mod p). Thus $((x-a)^p - (x^p - a))$ is not identically zero over F_p. □

Thus given a p as input, one could pick a polynomial $P(x) = x - a$ and compute whether the congruence (1) is satisfied or not. However, this takes time $\Omega(p)$ because we need to evaluate p coefficients in the LHS in the worst case. Therefore, to make it feasible we will evaluate both sides of (1) modulo a polynomial of the form $x^r - 1$. One iteration of our algorithm will consist of evaluating whether the following holds:

$$(x-a)^p \equiv (x^p - a) (\text{mod } x^r - 1, p) \tag{2}$$

From the Identity it is immediate that all primes p satisfy the above congruence for all values of a and r; however some composites p may also satisfy (2) for a few values of (a, r). The above congruence takes $O(r^2 \log^3 p)$ time for verification (lhs is evaluated by repeated squaring), or even better $O(r \log^2 p)$ if Fast Fourier Multiplication [Knu98] is used. Our algorithm first chooses a "suitable" r. (An r is "suitable" for us if it is a prime$= O(\log^6 p)$ and $r-1$ contains a prime factor of size at least $r^{\frac{1}{2}+\delta}$, for some constant $\delta > 0$. [Fou85, BH96] assures us that such a "suitable" r exists.) Thereafter, the algorithm verifies the congruence (2) for a "small" ($O(\sqrt{r} \log p)$) number of a's. We prove that this idea works: i.e. the algorithm correctly determines whether p is prime or not.

3 Notation and Preliminaries

This section states some algebraic and number theoretic results which we will be using in the later proofs.

In the rest of the paper F_{p^d} denotes the finite field, where p is a prime. Recall that if p is a prime and $h(x)$ is a polynomial of degree d and irreducible in F_p, then $F_p[x]/(h(x))$ is a finite field of order p^d. In the rest of the paper $h(x)$ will be a factor of $\frac{x^r-1}{x-1}$ unless stated otherwise.

We will use the symbol $\tilde{O}(t(n))$ for $O(t(n) poly(\log t(n)))$, where $t(n)$ is some function of n. Unless stated otherwise, log will be to base 2 in this paper.

We now collect some simple facts from algebra that can be found in any standard text, e.g. [LN86, Fra90]. We also prove some of these for the sake of completeness.

Lemma 3.1. *Let p and r be prime numbers, $p \neq r$.*

1. *The multiplicative group of any field F_{p^t} for $t > 0$, denoted by $F^*_{p^t}$ is cyclic.*

2. *Let $f(x)$ be a polynomial with integral coefficients. Then*

$$f(x)^p \equiv f(x^p) \ (mod\ p).$$

3. *Let $h(x)$ be any factor of $x^r - 1$. Let $m \equiv m_r \ (mod\ r)$. Then*

$$x^m \equiv x^{m_r} \ (mod\ h(x)).$$

4. *Let $o_r(p)$ be the order of p modulo r. Then in F_p, $\frac{x^r-1}{x-1}$ factorises into irreducible polynomials each of degree $o_r(p)$.*

Proof. 1. See, e.g., [LN86].

2. Let $f(x) = a_0 + a_1 x + \cdots + a_d x^d$. The coefficient of x^i in $f(x)^p$ is

$$\sum_{\substack{i_0+\cdots+i_d=p \\ i_1+2i_2+\cdots+di_d=i}} a_0^{i_0} \cdots a_d^{i_d} \frac{p!}{i_0! \cdots i_d!}.$$

Note that this sum is divisible by p unless one of the i_j's is p. In the latter case $i = pj$ and the coefficient of x^i is $a_j^p = a_j$. This gives us the required congruence.

3. Let $m = kr + m_r$. Now

$$\begin{array}{rrcll} & x^r & \equiv & 1 & (\text{mod } x^r - 1) \\ \Rightarrow & x^{kr} & \equiv & 1 & (\text{mod } x^r - 1) \\ \Rightarrow & x^{kr+m_r} & \equiv & x^{m_r} & (\text{mod } x^r - 1) \\ \Rightarrow & x^m & \equiv & x^{m_r} & (\text{mod } h(x)). \end{array}$$

4. Let $d = o_r(p)$ and $Q_r(x) = \frac{x^r-1}{x-1}$. Suppose that $Q_r(x)$ has an irreducible factor, $h(x)$ in F_p of degree k. Now $F_p[x]/h(x)$ forms a field of size p^k and the multiplicative subgroup of $F_p[x]/h(x)$ is cyclic with a generator, say $g(x)$. Also, in this galois field, by fact (2) above, we have

$$\begin{array}{rrcl} & g(x)^p & \equiv & g(x^p) \\ \Rightarrow & g(x)^{p^d} & \equiv & g(x^{p^d}) \\ \Rightarrow & g(x)^{p^d} & \equiv & g(x) \ \text{[By fact (3) above]} \\ \Rightarrow & g(x)^{p^d-1} & \equiv & 1. \end{array}$$

Since $(p^k - 1)$ is the order of $g(x)$, we get $(p^k - 1)|(p^d - 1)$ which implies that $k|d$.
We also have that $h(x)|(x^r - 1)$ in F_p and therefore in the field $F_p[x]/h(x)$ we have

$$x^r \equiv 1.$$

Thus the order of x in this field must be r (since r is prime and $x \not\equiv 1$). Therefore $r|(p^k - 1)$, i.e. $p^k \equiv 1 \ (\text{mod } r)$. Hence, $d|k$. Therefore, $k = d$, and the lemma follows.

□

In addition to the above algebraic facts, we will need the following two number theoretic facts.

Lemma 3.2. *[Fou85, BH96] Let $P(n)$ denote the greatest prime divisor of n. There exist constants $c > 0$ and n_0 such that, for all $x \geq n_0$*

$$|\{p | p \text{ is prime, } p \leq x \text{ and } P(p-1) > x^{\frac{2}{3}} \ \}| \geq c\frac{x}{\log x}.$$

The above lemma is, in fact, known to hold for exponents upto 0.6683 (see [BH96] for a summary of results of this kind.

Lemma 3.3. *[Apo97] Let $\pi(n)$ be the number of primes $\leq n$. Then for $n \geq 1$:*

$$\frac{n}{6 \log n} \leq \pi(n) \leq \frac{8n}{\log n}.$$

4 The Algorithm

Input: integer $n > 1$

1. if (n is of the form a^b, $b > 1$) output COMPOSITE;
2. $r = 2$;
3. while($r < n$) {
4. if (gcd(n,r) $\neq 1$) output COMPOSITE;
5. if (r is prime)
6. let q be the largest prime factor of $r - 1$;
7. if ($q \geq 4\sqrt{r}\log n$) and ($n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$)
8. break;
9. $r \leftarrow r + 1$;
10. }
11. for $a = 1$ to $2\sqrt{r}\log n$
12. if ($(x-a)^n \not\equiv (x^n - a) \pmod{x^r - 1, n}$) output COMPOSITE;
13. output PRIME;

Theorem 4.1. *The algorithm above returns PRIME if and only if n is prime.*

In the remainder of the section, we establish this theorem through a sequence of lemmas. First note that the algorithm has two loops. The first loop tries to find a prime r such that $r-1$ has a large prime factor $q \geq 4\sqrt{r}\log n$, and that $q | o_r(n)$, where $o_r(n)$ is the order of n modulo r. Let us first bound the number of iterations of the `while` loop after which such an r is found.

Lemma 4.2. *There exist positive constants c_1 and c_2 for which there is a prime r in the interval $[c_1(\log n)^6, c_2(\log n)^6]$ such that $r-1$ has a prime factor $q \geq 4\sqrt{r}\log n$ and $q | o_r(n)$.*

Proof. Let c and $P(n)$ be as given in Lemma 3.2. Thus, the number of prime r's (lets call them *special* primes) between $c_1(\log n)^6$ and $c_2(\log n)^6$ such that $P(r-1) > (c_2(\log n)^6)^{\frac{2}{3}} > r^{\frac{2}{3}}$ is (for large enough n)

$$\begin{aligned} &\geq \text{No of special primes in } [1 \cdots c_2(\log n)^6] - \text{No of primes in } [1 \cdots c_1(\log n)^6] \\ &\geq \frac{cc_2(\log n)^6}{7\log\log n} - \frac{8c_1(\log n)^6}{6\log\log n} \text{ (using Lemma 3.3)} \\ &= \frac{(\log n)^6}{\log\log n}\left(\frac{cc_2}{7} - \frac{8c_1}{6}\right). \end{aligned}$$

Choose constants $c_1 \geq 4^6$ and c_2 so that the quantity in braces is a positive constant, say c_3.

Let $x = c_2(\log n)^6$. Consider the product

$$\Pi = (n-1)(n^2-1)\cdots(n^{x^{\frac{1}{3}}} - 1).$$

This product has atmost $x^{\frac{2}{3}}\log n$ prime factors. Note that:

$$x^{\frac{2}{3}}\log n < \frac{c_3(\log n)^6}{\log\log n}.$$

Therefore, there is at least one special prime, say r, that does not divide the product Π.

This is the required prime: $r-1$ has a large prime factor $q \geq r^{\frac{2}{3}} \geq 4\sqrt{r}\log n$ (since $c_1 \geq 4^6$), and $q | o_r(n)$. □

Once we know that the `while` loop halts, we are ready to show:

Lemma 4.3. *If n is prime, the algorithm returns PRIME.*

Proof. The `while` loop cannot return COMPOSITE since $\gcd(n, r) = 1$ for all $r \leq c_2(\log n)^6$, where c_2 is as in Lemma 4.2. By Lemma 3.1 (fact 2), the `for` loop also cannot return COMPOSITE. Thus, algorithm will identify n as PRIME. □

Now let us turn our attention to the case where a composite n is input to our algorithm. The significance of the r found by the `while` loop arises when n is composite with say $p_i, 1 \leq i \leq k$, as its prime factors. In this case $o_r(n) \mid \text{lcm}_i\{o_r(p_i)\}$ and hence there exists a prime factor p of n such that $q \mid o_r(p)$, where q is the largest prime factor of $r-1$. For the remainder of the argument, let p be such a prime factor of n.

The second loop of the algorithm uses the value of r obtained to do polynomial computations on $\ell = 2\sqrt{r}\log n$ binomials: $(x-a)$ for $1 \leq a \leq \ell$. By Lemma 3.1 (fact 4), we have a polynomial $h(x)$ (factor of $x^r - 1$) of degree $d = o_r(p)$ irreducible in F_p. Note that

$$(x-a)^n \equiv (x^n - a)(\text{mod } x^r - 1, n)$$

implies that

$$(x-a)^n \equiv (x^n - a)(\text{mod } h(x), p).$$

So the identities on each binomial hold in the field $F_p[x]/(h(x))$. The set of ℓ binomials form a large cyclic group in this field:

Lemma 4.4. *In the field $F_p[x]/(h(x))$, the group generated by the ℓ binomials: $(x-a), 1 \leq a \leq \ell$ i.e,*

$$G = \{ \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} | \alpha_a \geq 0, \forall\ 1 \leq a \leq \ell \},$$

is cyclic and of size $> \left(\frac{d}{\ell}\right)^\ell$.

Proof. It is clear that G is a group and since it is a subgroup of the cyclic group $(F_p[x]/(h(x)))^*$, it is also cyclic.

Now consider the set

$$S = \{ \prod_{1 \leq a \leq \ell} (x-a)^{\alpha_a} \mid \sum_{1 \leq a \leq \ell} \alpha_a \leq d-1,\ \alpha_a \geq 0,\ \forall 1 \leq a \leq \ell \}.$$

The following argument shows that all the elements of S are distinct in $F_p[x]/(h(x))$. The `while` loop ensures that once it halts the final r is such that $r > q > 4\sqrt{r}\log n > \ell$. Also step 4 of the algorithm checks `gcd` of r and n. If any of the a's are congruent modulo p, then $p < \ell < r$ and thus step 4 of the algorithm identifies n as composite. Thus, none of the a's are congruent modulo p. So any two elements of S are distinct modulo p. This implies that all elements of S are distinct in the field $F_p[x]/(h(x))$ since degree of any element of S is less than d—the degree of $h(x)$.

The cardinality of the set S is:

$$\begin{aligned} \binom{\ell + d - 1}{\ell} &= \frac{(\ell + d - 1)(\ell + d - 2)\cdots(d)}{\ell!} \\ &> \left(\frac{d}{\ell}\right)^\ell. \end{aligned}$$

Since S is just a subset of G we get the result. □

Since $d \geq 2\ell$, size of G is $> 2^\ell = n^{2\sqrt{r}}$. Let $g(x)$ be a generator of G. Clearly, order of $g(x)$ in $F_p[x]/(h(x))$ is $> n^{2\sqrt{r}}$. We now define a set related to $g(x)$ which will play an important role in the remaining arguments. Let

$$I_{g(x)} = \{m \mid g(x)^m \equiv g(x^m)(\text{mod } x^r - 1, p)\}.$$

Here is a nice property of $I_{g(x)}$:

Lemma 4.5. *The set $I_{g(x)}$ is closed under multiplication.*

Proof. Let $m_1, m_2 \in I_{g(x)}$. So,

$$g(x)^{m_1} \equiv g(x^{m_1}) \pmod{x^r - 1, p},$$

and

$$g(x)^{m_2} \equiv g(x^{m_2}) \pmod{x^r - 1, p}.$$

Also we have by substituting x^{m_1} in place of x in the second congruence:

$$\begin{array}{rrcl} & g(x^{m_1})^{m_2} & \equiv & g(x^{m_1 m_2}) \pmod{x^{m_1 r} - 1, p} \\ \Rightarrow & g(x^{m_1})^{m_2} & \equiv & g(x^{m_1 m_2}) \pmod{x^r - 1, p}. \end{array}$$

From these, we get

$$\begin{aligned} g(x)^{m_1 m_2} &\equiv (g(x)^{m_1})^{m_2} \pmod{x^r - 1, p} \\ &\equiv g(x^{m_1})^{m_2} \pmod{x^r - 1, p} \\ &\equiv g(x^{m_1 m_2}) \pmod{x^r - 1, p}. \end{aligned}$$

Hence $m_1 m_2 \in I_{g(x)}$. □

Now we prove a property of $I_{g(x)}$ that plays a crucial role in our proof.

Lemma 4.6. *Let the order of $g(x)$ in $F_p[x]/(h(x))$ be o_g. Let $m_1, m_2 \in I_{g(x)}$. Then $m_1 \equiv m_2 (mod\ r)$ implies that $m_1 \equiv m_2 (mod\ o_g)$.*

Proof. Since $m_1 \equiv m_2 (\text{mod } r)$, $m_2 = m_1 + kr$ for some $k \geq 0$. Since $m_2 \in I_{g(x)}$: (in what follows, the congruences are in the field $F_p[x]/(h(x))$ unless indicated otherwise.)

$$\begin{array}{rrcl} & g(x)^{m_2} & \equiv & g(x^{m_2}) \pmod{x^r - 1, p} \\ \Rightarrow & g(x)^{m_2} & \equiv & g(x^{m_2}) \\ \Rightarrow & g(x)^{m_1 + kr} & \equiv & g(x^{m_1 + kr}) \\ \Rightarrow & g(x)^{m_1} g(x)^{kr} & \equiv & g(x^{m_1}) \text{ [By Lemma 3.1, fact 3]} \\ \Rightarrow & g(x)^{m_1} g(x)^{kr} & \equiv & g(x)^{m_1}. \end{array}$$

Now $g(x) \not\equiv 0$ implies $g(x)^{m_1} \not\equiv 0$ and hence we can cancel $g(x)^{m_1}$ from both sides leaving us with

$$g(x)^{kr} \equiv 1.$$

Therefore,

$$\begin{array}{rrcl} & kr & \equiv & 0 \pmod{o_g} \\ \Rightarrow & m_2 & \equiv & m_1 \pmod{o_g}. \end{array}$$

□

The above property implies that there are "very few" ($\leq r$) numbers in $I_{g(x)}$ that are less than o_g. Now we are ready to prove the most important property of our algorithm.

Lemma 4.7. *If n is composite, the algorithm returns COMPOSITE.*

Proof. Suppose that the algorithm returns PRIME instead. Thus, the `for` loop ensures that for all $1 \leq a \leq 2\sqrt{r} \log n$,

$$(x - a)^n \equiv (x^n - a) \pmod{x^r - 1, p}. \tag{3}$$

Notice that $g(x)$ is just a product of powers of ℓ binomials $(x - a)$, $(1 \leq a \leq \ell)$ all of which satisfy equation (3). Thus,

$$g(x)^n \equiv g(x^n) \pmod{x^r - 1, p}.$$

Therefore, $n \in I_{g(x)}$. Also, $p \in I_{g(x)}$ by Lemma 3.1, fact 2 and, trivially, $1 \in I_{g(x)}$. We will now show that the set $I_{g(x)}$ has "many" numbers less than o_g contradicting Lemma 4.6.

Consider the set

$$E = \{n^i p^j \mid 0 \leq i, j \leq \lfloor \sqrt{r} \rfloor\}.$$

By Lemma 4.5, $E \subseteq I_{g(x)}$. Since $|E| = (1 + \lfloor\sqrt{r}\rfloor)^2 > r$, there are two elements $n^{i_1}p^{j_1}$ and $n^{i_2}p^{j_2}$ in E with $i_1 \neq i_2$ or $j_1 \neq j_2$ such that $n^{i_1}p^{j_1} \equiv n^{i_2}p^{j_2} \pmod{r}$ by pigeon-hole principle. But then by Lemma 4.6 we have $n^{i_1}p^{j_1} \equiv n^{i_2}p^{j_2} \pmod{o_g}$. This implies

$$n^{i_1 - i_2} \equiv p^{j_2 - j_1} \pmod{o_g}.$$

Since $o_g \geq n^{2\sqrt{r}}$ and $n^{|i_1 - i_2|}$, $p^{|j_1 - j_2|} < n^{\sqrt{r}}$, the above congruence turns into an equality. Since p is prime, this equality implies $n = p^k$ for some $k \geq 1$. However, in step 1 of the algorithm, composite numbers of the form p^k for $k \geq 2$ are already detected. Therefore, $n = p$: a contradiction. □

This completes the proof of theorem.

5 Time Complexity Analysis

It is straightforward to calculate the time complexity of the algorithm.

Theorem 5.1. *The asymptotic time complexity of the algorithm is* $\tilde{O}(\log^{12} n)$.

Proof. The first step of the algorithm takes asymptotic time: $O(\log^3 n)$. As noted during the analysis of the algorithm in the previous section, the `while` loop makes $O(\log^6 n)$ iterations.

Let us now measure the work done in one iteration of the `while` loop. The first step (gcd computation) takes $\text{poly}(\log \log r)$ asymptotic time. The next two steps would take atmost $r^{\frac{1}{2}}\text{poly}(\log \log n)$ time in the brute-force implementation. The next three steps take atmost $\text{poly}(\log \log n)$ steps. Thus, total asymptotic time taken by the `while` loop is $\tilde{O}(\log^6 n \cdot r^{\frac{1}{2}}) = \tilde{O}(\log^9 n)$.

The `for` loop does modular computation over polynomials. If repeated-squaring and Fast-Fourier Multiplication is used then one iteration of this `for` loop takes $\tilde{O}(\log n \cdot r \log n)$ steps. Thus, the `for` loop takes asymptotic time $\tilde{O}(r^{\frac{3}{2}} \log^3 n) = \tilde{O}(\log^{12} n)$. □

In practice, however, our algorithm is likely to work much faster. The reason is that even though we only know that there are "many" primes r such that $P(r-1) > r^{\frac{2}{3}}$, a stronger property is believed to be true. In fact it is believed that for many primes r, $P(r-1) = \frac{r-1}{2}$. (Such primes are called *Sophie Germain primes.*)

Definition 5.2. If both r and $\frac{r-1}{2}$ are primes, then $\frac{r-1}{2}$ is a Sophie Germain prime. We will call such r's as co-Sophie Germain primes.

The following conjecture gives the density of Sophie Germain primes. This conjecture has been verified for $r \leq 10^{10}$:

Conjecture. [HL22] *The number of co-Sophie Germain primes is asymptotic to* $\frac{Dx}{\log^2 x}$, *where D is the twin prime constant (estimated by Wrench and others to be approximately 0.6601618...).*

If this conjecture is true, then the `while` loop exits with a "suitable" r of size $O(\log^2 n)$:

Lemma 5.3. *Assuming the conjecture 5, there exists "suitable"* r *in the range* $64 \log^2 n$ *to* $c_2 \log^2 n$ *for all* $n > n_0$, *where* n_0 *and* c_2 *are positive constants.*

Proof. First of all note that if r is prime and $q = \frac{r-1}{2}$ is a prime, then the only possible orders of n modulo r are $\{1, 2, q, 2q = r-1\}$. But the order of n modulo r can be 1 or 2 for at most $2 \log n$ primes r. (This is because $(n^2 - 1)$ can have at most $\log(n^2 - 1)$ prime factors.) Let us leave aside these prime factors of $(n^2 - 1)$ and consider the other co-Sophie Germain primes r for which the order of n modulo r is at least $\frac{r-1}{2}$. We would now like that

$$\begin{array}{rrcl} & \frac{r-1}{2} & \geq & 4\sqrt{r} \log n \\ \Rightarrow & \sqrt{r} & \geq & 8 \log n \\ \Rightarrow & r & \geq & 64 \log^2 n. \end{array}$$

Hence we consider the range $64 \log^2 n$ to $c_2 \log^2 n$ and we show that choosing c_2 large enough, we find at least one desired r in this range. By the conjecture 5, there are $\frac{Dc_2 \log^2 n}{\log^2(c_2 \log^2 n)}$ co-Sophie-Germain primes

less than $c_2 \log^2 n$. Out of these, at most $\frac{D64 \log^2 n}{\log^2(64 \log^2 n)}$ are less than $64 \log^2 n$ (again by the conjecture). From the remaining ones, there are at most $2 \log n$ primes for which order of n modulo r is 1 or 2. Thus we will choose c_2 such that

$$\begin{aligned} & \frac{Dc_2 \log^2 n}{\log^2(c_2 \log^2 n)} > \frac{D64 \log^2 n}{\log^2(64 \log^2 n)} + 2 \log n \\ \text{or,} \quad & \frac{c_2 \log^2 n}{(\log \log n)^2} > \frac{100 \log^2 n}{(\log \log n)^2} \quad \text{[for large enough n]} \\ \text{or,} \quad & c_2 > 100 \quad \text{[for large enough n].} \end{aligned}$$

□

This immediately leads us to a heuristic time complexity of $\tilde{O}(r^{\frac{1}{2}}(\log n)^2)$ (for `while` loop) + $\tilde{O}(r^{\frac{3}{2}}(\log n)^3)$ (for `for` loop) = $\tilde{O}(\log^6 n)$ for our algorithm.

6 Future Work and Improvements

In our algorithm, the `for` loop needs to run for $1 \leq a \leq 2\sqrt{r} \log n$ in order to ensure that the size of the group G referred to in Lemma 4.4 is large enough ($> n^{2\sqrt{r}}$). The upper limit for a could be improved if we can show that a still smaller set of $(x - a)$'s generates a group of the required size. This seems very likely.

We can further improve the complexity to $\tilde{O}(\log^3 n)$ if we can prove the following conjecture given in [BP01] and verified for $r \leq 100$ and $n \leq 10^{10}$ in [KS02]:

Conjecture. *If r does not divide n and if*

$$(x - 1)^n \equiv (x^n - 1)(mod\ x^r - 1, n), \tag{4}$$

then either n is prime or $n^2 \equiv 1(mod\ r)$.

If this conjecture is true, we can modify the algorithm slightly to first search for an r which does not divide $n^2 - 1$. Such an r can assuredly be found in the range $[2, 4 \log n]$. This is because the product of prime numbers less than x is at least e^x (see [Apo97]). Thereafter we can test whether the congruence (4) holds or not. Verifying congruence (4) takes time $\tilde{O}(r(\log^2 n))$ using FFT for multiplication. This would give us a time complexity of $\tilde{O}(\log^3 n)$.

Acknowledgements

We thank Rajat Bhattacharjee for useful discussions. We thank Erich Bach, Abhijit Das, G. Harman, Roger Heath-Brown, H. W. Lenstra, Pieter Moree, Richard Pinch, and Carl Pomerance for providing us with useful references. We thank Shafi Goldwasser and Erich Bach for pointing out incorrect citations in an earlier draft.

References

[AB99] M. Agrawal and S. Biswas. Primality and identity testing via chinese remaindering. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 202–209, 1999.

[AH92] L. M. Adleman and M.-D. Huang. Primality testing and two dimensional Abelian varieties over finite fields. *Lecture Notes in Mathematics*, 1512, 1992.

[Apo97] T. M. Apostol. *Introduction to Analytic Number Theory.* Springer-Verlag, 1997.

[APR83] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Ann. Math.*, 117:173–206, 1983.

[Atk86] A. O. L. Atkin. Lecture notes of a conference, boulder (colorado). Manuscript, August 1986.

[BH96] R. C. Baker and G. Harman. The Brun-Titchmarsh Theorem on average. In *Proceedings of a conference in Honor of Heini Halberstam, Volume 1*, pages 39–103, 1996.

[BP01] Rajat Bhattacharjee and Prashant Pandey. Primality testing. Technical report, IIT Kanpur, 2001. Available at http://www.cse.iitk.ac.in/research/btp2001/primality.html.

[Fou85] E. Fouvry. Theoreme de Brun-Titchmarsh; application au theoreme de Fermat. *Invent. Math.*, 79:383–407, 1985.

[Fra90] J. B. Fraleigh. *A First Course in Abstract Algebra.* Narosa, 1990.

[GK86] S. Goldwasser and J Kilian. Almost all primes can be quickly certified. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 316–329, 1986.

[HL22] G. H. Hardy and J. E. Littlewood. Some problems of 'Partitio Numerorum' III: On the expression of a number as a sum of primes. *Acta Mathematica*, 44:1–70, 1922.

[Knu98] D. E. Knuth. *The Art of Computer Programming, Vol II, Seminumerical Algorithms.* Addison Wesley, 1998.

[KS02] Neeraj Kayal and Nitin Saxena. Towards a deterministic polynomial-time test. Technical report, IIT Kanpur, 2002. Available at http://www.cse.iitk.ac.in/research/btp2002/primality.html.

[LN86] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1986.

[Mil76] G. L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Sys. Sci.*, 13:300–317, 1976.

[Rab80] M. O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12:128–138, 1980.

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6:84–86, 1977.