RIEMANN'S HYPOTHESIS AND TESTS FOR PRIMALITY

by

Gary L. Miller † * ‡

Research Report CS-75-27

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
October 1975

†Research sponsored by NSF GJ-35604X1

*A preliminary version of this paper was presented at the 7th ACM symposium on the theory of computing.

‡Ph.D. Dissertation submitted to University of California, Berkeley.

(abbreviated title)

TESTS FOR PRIMALITY

Department of Computer Science

University of Waterloo

ABSTRACT

In this paper we present two algorithms for testing primality of an integer. The first algorithm runs in O(n1/7) steps; while, the second runs in $O(\log^4 n)$ step but assumes the Extended Reimann Hypothesis. We also show that a class of functions which includes the Euler phi function are computationally equivalent to factoring integers.

INTRODUCTION

Two classic computational problems are finding efficient algorithms for: 1) testing primality (deciding whether an integer is prime or composite), 2) factoring integers. The best upper bounds on the number of steps needed by algorithms for 1) or 2) are due to Pollard [14]. Pollard proves an upper bound of $O(n^{(1/8)+\varepsilon})$ steps for testing primality and an upper bound of $9(n^{(1/4+\varepsilon)})$ steps for factoring, where ε is any constant > 0. We give an algorithm which tests primality and runs in $O(n^{1/7})$ steps. By slightly modifying this algorithm and assuming the Extended Riemann Hypothesis (ERH) we produce an algorithm which tests primality and runs in $O(\log^4 n)$ steps. Thus we show primality is testable in time polynomial in the length of the binary representation of n. Using the terminology of Cook [6] and Karp [9], we say primality is testable in polynomial time on the ERH.

One of the values of having a fast algorithm for factoring integers is that then many other computational problems could be done quickly. For example, the Euler phi function can obviously be computed quickly given the prime factorization of n.

As a by-product of the work on tests for primality we show that in fact the converse is true, assuming the ERH. Thus, computing the Euler phi function is computationally equivalent to factoring, assuming the ERH.

In the last section we discuss the relationship between recognition problems and computational problems. We show that a class of functions which includes prime factorization and the Euler phi function has the property that the graph of each function in this class is recognizable in polynomial time on the ERH.

Tests for Primality

Our main goal in this section is Theorem 2 but first we make precise the notion of a test for primality in $O(f(n))$ steps.

Definition. We say an algorithm tests primality in $O(f(n))$ steps if there exists a deterministic Turing machine which implements this algorithm, and this machine correctly indicates whether n is prime or composite in less than $K \cdot f(n)$ steps, for some constant K.

Using this definition we can state Theorem 1:

Theorem 1. There exists an algorithm which tests primality in $O(n^{.134})$ steps.

If we then assume the Extended Riemann Hypothesis (see Appendix), Theorem 1 can be vastly improved. Since the running time is small it seems more convenient to state the running time in terms of the length of the binary representation. Thus, let $|n|$ denote the length of the binary representation of n. Using this notation the main theorem is:

Theorem 2 (ERH). There exists an algorithm which tests primality in $O(|n|^4 \log \log |n|)$ steps.

The difficult step in the proof of the above two theorems is in demonstrating that there is a "small" quadratic nonresidue. In Theorem 1, we appeal to the work of Burgess who uses Weil's proof of Riemann Hypothesis over finite fields, while in Theorem 2 we use Ankeny's reduction of the size of the first quadratic nonresidue to the Extended Riemann Hypothesis.

Throughout the paper we will use the following conventions or notations:

Notation. We will assume that n, the number to be factored or tested for primality, is odd, for the even case easily can be reduced to the odd case. We let p, q vary over odd primes, and (a,b) denote the greatest common divisor of a and b. The number of 2's in n will be denoted by $\#_2(n)$, i.e., $\#_2(n) = \max\{K: 2^K | n\}$.

We will also need the following functions:

Definition. Let $n = p_1^{v_1} \cdots p_m^{v_m}$ be the prime factorization of the odd number n. We let "prime factorization" denote the function from the natural numbers to some fixed appropriate coding of the prime factors and their exponents. We also consider the following three functions:

i) $\phi(n) = p_1^{v_1-1}(p_1-1)\cdots p_m^{v_m-1}(p_m-1)$ (Euler's ϕ-function),

ii) $\lambda(n) = \operatorname{lcm}\{p_1^{v_1-1}(p_1-1),\ldots,p_m^{v_m-1}(p_m-1)\}$ (The Carmichael λ-function),

iii) $\lambda'(n) = \operatorname{lcm}\{p_1-1,\ldots,p_m-1\}$.

Motivation of Proofs

Fermat proved that for p prime

$$a^{p-1} \equiv 1 \bmod p \quad \text{if} \quad (a,p) = 1 .$$

Therefore, if for some a, $1 < a < n$,

$$a^{n-1} \not\equiv 1 \bmod n , \tag{1}$$

then n must be composite. Now, $a^m \bmod n$ can be computed in $O(|m|M(|n|))$ steps (where $M(|n|)$ denotes the cost of multiplying two

numbers of length $|n|$) using standard techniques described in [7]. A possible technique for recognizing composite numbers might be to systematically search for an a satisfying (1). This technique could fail for composite n for two reasons:

a) There could be composite n which satisfies Fermat's Congruence. That is,

$$a^{n-1} \equiv 1 \bmod n \quad \text{for all} \quad (a,n) = 1 .$$

b) The first a satisfying (1) could be very large which would give us an inefficient method.

The rest of this section will be devoted to handling these two problems. We start by showing that in fact some composite numbers satisfy Fermat's Congruence.

Theorem (Carmichael [5]). n satisfies Fermat's Congruence if and only if $\lambda(n)|n-1$.

For example, the composite number $561 = 3 \cdot 11 \cdot 17$ is such that $\lambda(n) = \operatorname{lcm}(2,10,16) = 80$, and 80 divides 560. It follows that $(a,561) = 1$ implies $a^{560} \equiv 1 \bmod 561$ for all natural numbers a. Thus there are composite numbers which satisfy Fermat's Congruence. At first these numbers seem more difficult to recognize as composite. Not only will we recognize them as composite, but we will quickly find a divisor. By what we have done it would seem that the obvious approach would be to use Fermat's test to recognize composite n such that $\lambda(n) \nmid n-1$ and some other test for n such that $\lambda(n)|n-1$. Instead we shall separate the composite numbers into sets according to whether $\lambda'(n) \nmid n-1$ or $\lambda'(n)|n-1$.

Since the algorithms used in Theorems 1 and 2 are essentially the same we shall define the following class of algorithms:

Definition of A_f. Let f be a computable function on the natural numbers. We define A_f on input n as follows:

1) Check if n is a perfect power, i.e. $n = m^s$ where $s \geq 2$. If n is a perfect power then output "composite" and halt.

2) Carry out steps i) - iii) for each $a \leq f(n)$. If at any stage i), ii) or iii) holds output "composite" and halt:

i) $a \mid n$

ii) $a^{n-1} \not\equiv 1 \bmod n$

iii) $((a^{(n-1)/2^k} \bmod n) - 1, n) \neq 1$ for some k, $1 \leq k \leq \#_2(n-1)$.

3) Output "prime" and halt.

Note. A_f as defined above is a simplified version of the algorithm needed to get Theorem 2. A_f will give an algorithm for testing primality in $O(|n|^5 \log^2 |n|)$ steps.

Before we prove Theorems 1 and 2 we must develop the technical hardware to define f and to show that there is an $a \leq f(n)$ which "works".

We start by considering those composite numbers n which satisfy $\lambda'(n) \not| n-1$. In the following lemma we give a characterization of some of the a's which satisfy $a^{n-1} \not\equiv 1 \bmod n$.

Lemma 1. If $\lambda'(n) \not| n-1$ then there exist primes p and q so that:

1) $p \mid n$, $p-1 \not| n-1$, $q^m \mid p-1$, and $q^m \not| n-1$ for some integer $m \geq 1$.

2) If a is any q^{th} nonresidue mod p then $a^{n-1} \not\equiv 1 \bmod n$.

See the Appendix for the definition of q^{th} nonresidue mod p and the definition of index of a mod p which we will denote by $\text{ind}_p a$.

Proof of Lemma 1. Let $q_1,\ldots,q_n$ be the distinct prime divisors of n. Thus $\lambda'(n) = \text{lcm}\{q_1-1,\ldots,q_n-1\} \not| n-1$ which implies $q_i-1 \not| n-1$ for some i. By setting $p = q_i$ we have $p|n$ and $p-1 \not| n-1$. Since $p-1 \not| n-1$, there must exist a prime q and an integer $m \geq 1$ so that $q^m | p-1$ and $q^m \not| n-1$. Thus p and q satisfy condition 1). We next show that p, q satisfy condition 2).

Suppose the lemma is false, i.e. $a^{n-1} \equiv 1 \bmod n$. Since $p|n$ we have

$$a^{n-1} \equiv 1 \bmod p \ . \tag{2}$$

Let b be a generator mod p; then by (2) we have $b^{(\text{ind}_p a)(n-1)} \equiv 1 \bmod p$. Since $b^m \equiv 1 \bmod p$ implies $p-1|m$ we have

$$p-1 | (\text{ind}_p a)(n-1) \ . \tag{3}$$

Now a is a q^{th} nonresidue implies $q \not| \text{ind}_p a$. Thus

$$q \not| \text{ind}_p a \quad \text{and} \quad q^m | p-1 \ . \tag{4}$$

Applying (4) to (3) gives $q^m | n-1$ which is a contradiction. □

Lemma 1 motivates the definition of the first q^{th} nonresidue mod p.

Definition. Let N(p,q) be the least a so that a is a q^{th} nonresidue mod p defined only when $q|p-1$. Using index arguments it is not hard to show that N(p,q) is prime.

Theorem (Ankeny [1]) (ERH). $N(p,q) = O(|p|^2)$

Using Ankeny's Theorem and Lemma 1 we have that if $\lambda'(n) \nmid n-1$ then there exists an $a \leq O(|n|^2)$ such that $a^{n-1} \not\equiv 1 \bmod n$.

We now return to a discussion of composite numbers n which have the property that $\lambda'(n) | n-1$. Let $q_1,\ldots,q_m$ be the distinct prime divisors of n; then by the definition of λ' we know that $\#_2(\lambda'(n)) = \max(\#_2(q_1-1),\ldots,\#_2(q_m-1))$. Thus for some $1 \leq i \leq m$, $\#_2(\lambda'(n)) = \#_2(q_i-1)$. We next make a distinction between two types of numbers as follows:

Definition. Let $q_1,\ldots,q_m$ be the distinct prime divisors of n. We say n is of type A if for some $1 \leq j \leq m$, $\#_2(\lambda'(n)) > \#_2(q_j-1)$. On the other hand, we say n is of type B if $\#_2(\lambda'(n)) = \#_2(q_1-1) = \cdots = \#_2(q_m-1)$.

Digressing for a moment to motivate the next three lemmas, suppose we have a composite number $n = pq$. Suppose further that we have a number m so that

$$m \equiv 1 \bmod q \quad \text{and} \quad m \equiv -1 \bmod p \ . \tag{5}$$

The first of the restrictions in (5) implies $q | m-1$ and the second implies $m \not\equiv 1 \bmod n$. Thus $q = (m-1,n)$. If we could quickly compute some m satisfying (5), we would quickly know a divisor of n. In the following lemmas we develop a method for finding m satisfying (5). We say b has a non-trivial GCD with n if $(b,n) \neq 1$ or n.

Lemma 2A. Let n be a composite number of type A where, say, p and $q|n$, and $\#_2(\lambda'(n)) = \#_2(p-1) > \#_2(q-1)$. Assume further that $0 < a < n$ is so that $(\frac{a}{p}) = -1$ where $(\frac{a}{p})$ is the Jacobi symbol (cf. Appendix), then either a or $(a^{\lambda'(n)/2} \bmod n) - 1$ has a nontrivial GCD with n.

Proof. Suppose a has a trivial GCD with n. Since $1 < a < n$ it must be that $(a,n) = 1$. Since $q-1|\lambda'(n)$ and $\#_2(q-1) < \#_2(\lambda'(n))$, we have $q-1|\frac{\lambda'(n)}{2}$, thus

$$a^{\lambda'(n)/2} \equiv 1 \bmod q \ . \tag{1}$$

Since $\left(a^{\lambda'(n)/2}\right)^2 \equiv 1 \bmod p$ then $a^{\lambda'(n)/2} \equiv \pm 1 \bmod p$. Suppose $a^{\lambda'(n)/2} \equiv 1 \bmod p$ then $p-1|(\mathrm{ind}_p a)(\frac{\lambda'(n)}{2})$ which implies that $\mathrm{ind}_p a$ is even. On the other hand, $(\frac{a}{p}) = -1$ implies $\mathrm{ind}_p a$ is odd (see Appendix). So

$$a^{\lambda'(n)/2} \equiv -1 \bmod p \ . \tag{2}$$

By (1), $q|(a^{\lambda'(n)/2} \bmod n) - 1$. By (2), $p\nmid(a^{\lambda'(n)/2} \bmod n) - 1$ since p is an odd prime. Thus $((a^{\lambda'(n)/2} \bmod n) - 1, n) \neq 1,n$. □

Lemma 2B. Let n be a composite number with at least two distinct prime divisors, say p and q. Further suppose n is of type B and $1 < a < n$ is so that $(\frac{a}{pq}) = -1$. Then, either a or $(a^{\lambda'(n)/2} \bmod n) - 1$ has a nontrivial divisor with n.

Proof. As in the proof of Lemma 2A we assume that a has a trivial GCD with n, thus $(a,n) = 1$. Without loss of generality we assume that $(\frac{a}{p}) = -1$ and $(\frac{a}{q}) = 1$. Using techniques similar to above we

show $a^{\lambda'(n)/2} \equiv -1 \bmod p$ and $a^{\lambda'(n)/2} \equiv 1 \bmod q$. The rest of the argument follows from the above proof. □

Lemma 3. If $p|n$, $\lambda'(n)|m$, and $k = \#_2[\frac{m}{\lambda'(n)}] + 1$ then $a^{\frac{\lambda'(n)}{2}} \equiv a^{\frac{m}{2^k}} \bmod p$.

Proof. Since $a^{\lambda'(n)} \equiv 1 \bmod p$ it follows that $a^{\lambda'(n)/2} \equiv \pm 1 \bmod p$. We consider the two possible values of $a^{\lambda'(n)/2}$ separately:

1) If $a^{\lambda'(n)/2} \equiv 1 \bmod p$ then $a^{m/2^k} \equiv 1 \bmod p$, since by our choice of k and the fact that $\lambda'(n)|m$ we have $\frac{\lambda'(n)}{2}|\frac{m}{2^k}$.

2) If, on the other hand, $a^{\lambda'(n)/2} \equiv -1 \bmod p$ we note that:

$$a^{\frac{m}{2^k}} \equiv \left(a^{\frac{\lambda'(n)}{2}}\right)^{\frac{m}{\lambda'(n)2^{k-1}}} \equiv (-1)^{\frac{m}{\lambda'(n)2^{k-1}}} \bmod p .$$

Since $m/\lambda'(n)2^{k-1}$ is odd, $a^{m/2^k} \equiv -1 \bmod p$.

Using Lemmas 2A and 3 we see that: if n is a type A composite number, $\lambda'(n)|n-1$, and $a = N(p,2)$ then either $a|n$ or $((a^{(n-1)/2} \bmod n) - 1, n) \neq 1,n$. For type B numbers we will need the following definition.

Definition. Let $N(pq)$ be the minimum a so that $(\frac{a}{pq}) \neq 1$ where $(\frac{a}{pq})$ is the Jacobi symbol and $N(pq)$ is defined only when $p \neq q$. Note again that $N(pq)$ is prime.

Theorem (Ankeny [1]) (ERH). $N(pq) = O(|pq|^2)$

Ankeny doesn't actually state the case $N(pq)$ but it follows without any change in his argument. We only need to use the stronger

form of Selberg's Theorem 6 [16] referred to as Lemma 2(c) in [1]. Also see [12] for the statement and proof of Ankeny's theorem.

Proof of Theorem 2 (weak form). By Theorems of Ankeny we can pick an integer $c \geq 1$ so that

$$N(p,q) \leq c|p|^2 \quad \text{and} \quad N(pq) \leq c|pq|^2 .$$

Consider A_f where $f(n) = c|n|^2$.

Analysis of Running Time

1) A_f must first check to see if n is a perfect power which will take $O(|n|^4)$ steps. We leave it to the reader to verify this bound.

2) A_f must check i), ii) and iii) for $f(n)$ different a's.

Check i) takes say $O(|n|^2)$ steps.

Check ii) takes $O(|n|M(|n|))$ steps.

Check iii) takes $O((|n|M(|n|) + |n|^2)|n|)$ steps since GCD can be computed in $O(|n|^2)$ steps, see [7], and $1 \leq k \leq |n|$. Now multiplication takes at least $|n|$ steps thus check iii) takes at most $O(|n|^2M(|n|))$ steps.

So A_f runs in $O(|n|^4M(|n|))$ steps. If we use the Schonhage-Strassen algorithm ([18]) for multiplying binary numbers, $M(|n|) = O(|n|\log|n|\log\log|n|)$ and we have $O(|n|^5\log|n|\log\log|n|)$ steps.

Proof of Correctness of A_f. If n is prime A_f will indicate correctly that n is prime so we need only show that A_f recognizes composite n. If n is composite n it will fall into one of the following three cases.

1) n is a prime power.

2) $\lambda'(n) \nmid n-1$

3) $\lambda'(n) | n-1$ and n is not a prime power.

Case 1. If n is a prime power then n is a perfect power and in this case A_f will indicate that A_f is composite.

Case 2. If $\lambda'(n) \nmid n-1$ then by Lemma 1 we have a p and q such that if $a = N(p,q)$ then $a^{n-1} \not\equiv 1 \bmod n$. Thus we need only note that $N(p,q) \leq f(n)$, which follows by our choice of f.

Case 3. If $\lambda'(n) | n-1$ and n is not a prime power:

A) Suppose n is of type A then by Lemmas 2A and 3 we can choose p and k ($k \leq \#_2(n-1)$ such that if $a = N(p,2)$ then either $a | n$ or $((a^{(n-1)/2^k} \bmod n)-1, n) \neq 1,n$. Since $N(p,2) \leq f(n)$, n will be recognized as composite by either step i) or ii).

B) Suppose n is of type B. Then by Lemmas 2B and 3 and the assumption that n is not a perfect power, we can choose p, q and $k \leq \#_2(n-1)$ so that if $a = N(pq)$ then either $a | n$ or $((a^{(n-1)/2^k} \bmod n)-1, n) \neq 1,n$. Since $N(pq) \leq f(n)$, A_f will indicate that n is composite. □

To prove Theorem 1 we need the following results of Burgess.

Theorem (Burgess) [2,3,5].

$$N(p,q) = O(p^{(1/4\sqrt{e})+\varepsilon}) \quad \text{any } \varepsilon >$$
$$N(pq) = O((pq)^{1/4\sqrt{e})+\varepsilon}) \quad \text{any } \varepsilon > 0$$

Proof of Theorem 1. By the Theorem of Burgess we can pick an integer $c \geq 1$ so that

$$N(p,q) \leq cp^{1/4\sqrt{2.71}} \quad \text{and} \quad N(pq) \leq c(pq)^{1/4\sqrt{2.71}}.$$

Set $\ell = 4\sqrt{2.71}$. Consider A_f where $f(n) = \lceil cn^{\frac{1}{\ell+1}} \rceil \leq \lceil cn^{.133} \rceil$. Since A_f runs in $O(n^{.134})$ steps we need only show that A_f tests primality. If n is prime then A_f will indicate that n is prime.

Suppose that n is composite. Then n must lie in at least one of the following four cases.

Case 1. n is a prime power.

Case 2. n has a divisor $\leq f(n)$.

Case 3. $\lambda'(n)/n-1$, n has no divisor $\leq f(n)$.

By Lemma 1 there exist primes p, q such that if $a = N(p,q)$ then $a^{n-1} \not\equiv 1 \bmod n$. So we need only show that $a = N(p,q) \leq f(n)$. We have

$$a \leq \lceil cp^{1/\ell} \rceil \tag{5}$$

from above. Since n is composite and for all $a \leq f(n)$ implies $a \nmid n$, we have

$$p \leq \frac{n}{f(n)}, \quad \text{i.e.,} \quad p \leq \lceil \tfrac{1}{c} n^{\ell/(\ell+1)} \rceil . \tag{6}$$

Substituting (6) into (5) we have

$$a \leq \lceil n^{1/(\ell+1)} \rceil \leq f(n) \quad \text{since} \quad c \geq 1 .$$

Case 4. $\lambda'(n) \mid n-1$ and n has no divisor $\leq f(n)$ and n is not a prime power.

A) Suppose n is of type A. Then as in Case 3A of Theorem 1 we need only show $a = N(p,2) \leq f(n)$ where $p \mid n$. Since in this case (5) and (6) hold we get $a \leq f(n)$.

B) Suppose n is of type B. Since n is not a prime power n has at least two distinct prime divisors, say p, q. We need to show that $N(pq) \leq f(n)$ which will follow if we show $pq \leq \frac{n}{f(n)}$.

Claim. $n \neq pq$ (see [5]).

Suppose $n = pq$ where $p < q$. Now $q-1|pq-1$, since $\lambda'(n)|n-1$. But this implies $q-1|p-1$. Hence $q \leq p$ which contradicts the assumption that $p < q$.

By claim $n = pqr$ where $r \neq 1$. Since $r|n$ we have $r \geq f(n)$. Thus $pq \leq \frac{n}{f(n)}$. □

Modification to Algorithm A_f

First note that a in step 2) of A_f need not vary over all numbers $\leq f(n)$ but only prime numbers $\leq f(n)$. Since the number of prime $\leq f(n)$ is $O(\frac{f(n)}{\log f(n)})$, by the prime number theorem, we have the upper bound for Theorem 2 of $O(|n|^5 \log \log |n|)$ steps.

We amend A_f as follows:

1) If n is perfect power output composite.

2) Compute $p_1,\ldots,p_m$ where p_i is the i-th prime number and m is so that $p_m \leq f(n) < p_{m+1}$. Compute Q, S so that $n-1 = Q2^S$ and Q is odd. Let $i = 1$ and proceed to ii) (let a denote p_i throughout).

i) If $i < m$ set i to $i+1$. If $i = m$ then output "prime" and halt.

ii) If $a|n$ then output "composite" and halt.

Compute $a^Q \bmod n$, $a^{Q2} \bmod n$, ..., $a^{Q2^S} \bmod n$.

iii) If $a^{Q2^S} \bmod n \neq 1$ then output "composite" and halt.

iv) If $a^Q \bmod n = 1$ go to i).

Set $J = \max(J: a^{Q2^J} \bmod n \neq 1)$.

v) If $a^{Q2^J} \bmod n = n-1$ go to i).

vi) Output "composite" and halt.

The running time A_f is $O(|n|^4 \log\log|n|)$. To show that A_f tests primality we need only reconsider Case 3:

Case 3. $\lambda'(n)|n-1$ and n is not a prime power.

A) Suppose n is of type A with $\#_2(\lambda'(n)) = \#_2(p-1) > \#_2(q-1)$ and $p,q|n$. Let $a = N(p,2)$ (thus a is prime). Thus we need only show that either step ii), iii) or vi) outputs "composite" for this a. So suppose $a \nmid n$ and $a^{n-1} \equiv 1 \bmod n$. We show that A_f reaches step vi). If $a^S \equiv 1 \bmod p$ then $2|S$, since $(\frac{a}{p}) = -1$ and p is odd. Since $p|n$ we have $a^Q \not\equiv 1 \bmod n$. Thus A_f will reach step v). By Lemmas 2A and 3, we know there exists a k so that $a^{Q2^k} \equiv 1 \bmod q$ and $a^{Q2^k} \equiv -1 \bmod p$. Suppose $a^{Q2^J} \equiv -1 \bmod n$ then $a^{Q2^J} \equiv -1 \bmod p$ and q. Now $a^{Q2^k} \equiv a^{Q2^J} \equiv -1 \bmod p$ implies $k = J$. On the other hand, $a^{Q2^k} \equiv 1 \bmod q$ and $a^{Q2^J} \equiv -1 \bmod q$ implies $k > J$. Thus by contradiction $a^{Q2^J} \not\equiv -1 \bmod n$. Hence A_f reaches step vi).

B) Suppose n is of type B. The proof in this case follows the argument in Case A.

Relative Computational Complexity

In this section we discuss the relative computational complexity of certain functions from number theory.

To begin, consider the following example: The Euler phi function, $\phi(n)$, is defined to equal the number of integers between 1 and n which are relatively prime to n. Computing $\phi(n)$ via this definition, checking each number less than n and seeing if it is relatively prime to n, requires at least n steps. Thus this method requires an exponential number of steps in terms of $|n|$. Now given the prime factorization of n say $p_1^{v_1} \cdots p_m^{v_m}$ we can evaluate $\phi(n)$ via the product

$$\phi(n) = p_1^{v_1-1}(p_1-1) \cdots p_m^{v_m-1}(p_m-1)$$

in at most $\log_2 n$ multiplications, thus, in time at most a polynomial in terms of $|n|$. We can restate the product formula from a complexity point of view as: If the prime factorization of n could be computed "quickly" then $\phi(n)$ could be computed "quickly". We now proceed to formalize the above statement and prove its converse assuming the ERH.

Definitions for reducibility amongst recognition problems (sets) have been introduced by many authors, see in particular Cook [6] and Karp [9]. Since we are primarily concerned with functions, we introduce the notation of functional reducibility.

Definition. Given functions f and g we say that f is polynomial time reducible to g denoted $f \leq_p g$, if there exists a Turing machine which on inputs n and $g(n)$ computes $f(n)$ in $O(|n|^k)$ steps for some constant k. We say f is polynomial time equivalent

to g if $f \leq_p g$ and $g \leq_p f$ and denote this relation by $f \approx_p g$.

The above definition of polynomial time reducible is very strong. It says that if two functions are polynomial time equivalent then upper (lower) bounds on their running time differ by at most an additive polynomial uniformly. In a later example we shall make a definition of polynomial time reducibility which need only preserve the asymptotic running times.

We now formalize our statement about Euler's function.

Lemma 4. The functions ϕ, λ, λ' are all polynomial time reducible to "prime factorization", i.e., $\phi, \lambda, \lambda' \leq_p$ "prime factorization".

Proof. $\phi \leq_p$ "prime factorization" follows by our discussion in the introduction of this section. To show that λ, λ' are reducible to prime factorization we note the following two facts about the LCM function:

1) $\text{lcm}(a,b) = a \cdot b/(a,b)$

2) $\text{lcm}(a,b,c) = \text{lcm}(\text{lcm}(a,b),c)$ □

By Lemma 4 we have that if the Euler ϕ function cannot be computed in polynomial time then neither can we factor integers in polynomial time. But it may be the case that computing $\phi(n)$ can be done quickly while factoring is difficult. The next lemma shows that all functions from a certain class which includes ϕ are no easier to compute than prime factorization, assuming the ERH.

Lemma 5 (ERH). Let g be any function such that

1) $\lambda'(n) | g(n)$

2) $|g(n)| = O(|n|^k)$ for some constant k.

Then "prime factorization" $\underset{p}{\leq}$ g.

Proof. Consider the following procedure on n and m.

1) Check if n is a perfect power.

2) Carry out steps i) and ii) for each $a \leq f(n)$ (where f is as in the proof of Theorem 1):

i) $a|n$

ii) $((a^{m/2^k} \bmod n)-1, n) \neq 1$ for some $a \leq k \leq \#_2(m)$.

If $\lambda'(n)|m$ then we know by arguments similar to Case 3 of the proof of Theorem 2 that this procedure will produce a divisor of n if n is composite. If we set $m = g(n)$ then in $O(|g(n)||n|^3M(|n|))$ steps we will either know that n is prime or that n' is a divisor of n, for some n'. If in the above procedure we replace n by n' then $\lambda'(n')|g(n)$ since $n'|n$ implies $\lambda'(n')|\lambda'(n)$. Thus in $O(|g(n)||n'|^3M(n'))$ steps we will either know n' is prime or n'' is a factor of n'. Iterating this procedure at most $|n|$ times we will have all prime factors of n. Thus, we get a prime factorization of n in $O(|g(n)||n|^4M(|n|))$ steps. Since $|g(n)| = O(|n|^k)$ it runs in $O(|n|^{k+4}M(|n|))$ steps. □

Thus we have the following theorem.

Theorem 3 (ERH). The functions ϕ, λ, λ' and "prime factorization" are all polynomial time equivalent, i.e. "prime factorization" $\underset{p}{\approx}$ ϕ $\underset{p}{\approx}$ λ $\underset{p}{\approx}$ λ'.

Another problem related to factoring numbers is finding the period of a rational number. We know that every rational number is periodic in any base. Thus the function "Period"(a,b) = the minimum period of

l/a base b is well defined. It does not seem possible to prove equivalence between "period" and "prime factorization" using the previous definition of reducibility. Thus we introduce a weaker definition of reducibility similar to Turing reducibility from recursion theory.

Definition. Given functions f and g we say that f is polynomial time Turing reducible to g denoted $f \leq_p^T g$ if there exists a Turing machine with the following properties:

1) The machine has a distinguished tape on which it can call for values of g, where the cost of calling for g(m) is $|m| + |g(n)|$ steps.

2) The machine computes f(n) in $O(|n|^K)$ steps for some constant K.

We say f and g are polynomial time Turing equivalent if $f \leq_p^T g$ and $g \leq_p^T f$ denoted by $f \approx_p^T g$.

In Lemma 5 we made certain restrictions on the growth of the function g. We required that the length of g grow by at most a polynomial in terms of the length of its argument. We shall say that such a function has syntactic polynomial growth.

Lemma 6. Over the class of functions with syntactic polynomial growth the relations $\leq_p$, $\approx_p$, $\leq_p^T$ and $\approx_p^T$ have the following properties:

1) $\leq_p$ and $\leq_p^T$ are transitive relations. Thus $\approx_p$ and $\approx_p^T$ are equivalence relations.

2) $f \leq_p g$ implies $f \leq_p^T g$.

3) The class of functions computable in polynomial time forms an equivalence class mod $\approx_p$ and mod $\approx_p^T$.

Using our second definition of reducibility we can now prove

equivalence between period and prime factorization.

Theorem 4 (ERH). "period" is polynomial time Turing equivalent to "prime factorization", i.e. "period" $\approx_p^T$ "prime factorization".

Proof. By a standard theorem in number theory, see Hardy and Wright [8], we have if $a = uv$ where $(v,b) = 1$ and u consists only of primes which divide b then $\text{Period}(a,b) = \min\{m: b^m \equiv 1 \bmod v\}$. That is, the $\text{Period}(a,b)$ equals the order of $b \bmod v$.

We start by showing that "period" $\leq_p^T$ "prime factorization". Assume the input is $[a,b]$. The machine first computes u,v as above by successive applications of GCD. For completeness we give a possible method. Consider the sequences $u_0,u_1,\ldots$ and $v_0,v_1,\ldots$ defined by $u_0 = 1$, $v_0 = a$, $u_{i+1} = (v_i,b)u_i$ and $v_{i+1} = a/u_{i+1}$. Now $a = u_i v_i$ and u_i consists only of primes dividing b. If $u_i \neq u$ then $2u_i \leq u_{i+1}$. Therefore when $i = |a|$ then $u_i = u$ and $v_i = v$. The machine now calls for the prime factorization of v from which it computes $\lambda(v)$. It now calls for the prime factorization of $\lambda(v)$, say $p_1^{\tau_1}\ldots p_m^{\tau_m}$. We know by Carmichael's Theorem, see [5], that the order of $b \bmod v$ divides $\lambda(v)$. Thus we need only determine which of the p_i's to discard. This can be done by computing the minimum h_i satisfying $b^{p_1^{\tau_1}\ldots p_{i-1}^{\tau_{i-1}} p_i^{h_i} p_{i+1}^{\tau_{i+1}}\ldots p_m^{\tau_m}} \equiv 1 \bmod v$ for each i between 1 and m. It follows that the order $b \bmod v$ equals $p_1^{h_1}\ldots p_m^{h_m}$.

To prove "prime factorization" $\leq_p^T$ "period", let f be as in the proof of Theorem 2 and let the number to be factored by n. Assume that n has no factors between 2 and $f(n)$, otherwise just factor them out.

Claim (ERH). Let $h(n) = \text{lim}\{\text{Period}(n,2),,..,\text{Period}(n,f(n))\}$. Then $h(n) \leq \lambda(n)$ and $\lambda'(n)|h(n)$.

Since $\text{Period}(n,i)|\lambda(n)$ for $2 \leq i \leq f(n)$ we have $h(n)|\lambda(n)$ hence $h(n) \leq \lambda(n)$. Suppose $q^m|\lambda'(n)$. Then $q^m|p-1$ for some $p|n$. Let a be the minimum q-th nonresidue mod p, i.e., $a = N(p,q)$. Then we have the following:

1) By the Extended Riemann Hypothesis $a \leq f(p)$. Thus $a \leq f(n)$.

2) $q^m|$order of a mod p since a is a q-th nonresidue mod p and $q^m|p-1$. Since $(a,n) = 1$ by 1) $q^m|$order of a mod n.

From these two facts $q^m|h(n)$. Thus the claim follows.

Since h satisfies the hypothesis of Lemma 5, "prime factorization" $\leq_p h$. $h \leq_p^T$ "period" since we can define a machine which simply calls for $\text{Period}(n,2),...,\text{Period}(n,f(n))$ and computes their LCM. This machine runs in polynomial time since $f(n) = O(\log^2 n)$. Finally by Lemma 6 we have "prime factorization" $\leq_p^T$ "period". □

Factoring and P-NP

Probably the most interesting open question in computational complexity theory is the P-NP question. Cook [6] and Karp [9] showed that a surprising number of recognition problems were NP-complete. One recognition problem which was not shown to be NP-complete was the set of composite numbers, {composites}. Pratt [15] showed that the set of prime numbers, {primes}, is in NP. Thus it seems unlikely that {composites} are NP-complete, for this would imply that $NP = \overline{NP}$, where $\overline{NP}$ consists of those sets whose complements are in NP. Further, by Theorem 2 we have that {composites} $\in P$ on the ERH. These two facts to a certain extent settle the relation of {composites} to the P-NP question. The complexity of factoring seems more elusive.

In light of Pratt's work it seems natural to view factoring in terms of nondeterminism, not as a recognition problem, but rather as a function which is nondeterministically computed. In the next definition we introduce the notion of deterministic (nondeterministic) polynomial time computable functions.

Definition. Let P^* denote those total functions over the natural numbers computable in polynomial time. We say a nondeterministic machine computes f in T steps if the machine on input n has some path which halts and any path which halts must output $f(n)$ in $O(T(n))$ steps. Using this definition, we let NP^* denote those total functions over the natural numbers computable in nondeterministic polynomial time.

As in the introduction of this section, we let P (NP) denote those subsets of the natural numbers recognizable in deterministic (nondeterministic) polynomial time. It is clear that the set of

composite numbers is contained in NP, i.e. {composites} $\in$ NP. Pratt proved the following surprising result:

Theorem (Pratt) [15]. {primes} $\in$ NP

Using Pratt's result we get the following corollaries:

Corollary 1. "prime factorization" $\in NP^*$

Proof. The machine simply guesses a prime factorization, recognizes each of the factors as prime and then outputs the "prime factorization". Since there are at most log n factors the machine runs in polynomial time. □

Corollary 2. "period", ϕ, λ, λ' $\in NP^*$

Proof. Since all four functions are polynomial time Turing reducible to the function "prime factorization" and "prime factorization" is in NP^* we need only show the following lemma:

Lemma 7. $f \leq_p^T g$ and $g \in NP^*$ then $f \in NP^*$.

Proof. Let M be a machine which computes f via the method given by $f \leq_p^T g$. Let M' be a machine which computes g nondeterministically in polynomial time. To construct a machine which computes f we simply replace the calls for values of g in M by computations using M'. Now the new machine runs in polynomial time since M can call for at most a polynomial number of values of g each of which can be computed in polynomial time. □

At this time we introduce two different constructions for producing

recognition problems from functions and examine their properties in light of the results of the preceding sections. Let $\langle a,b \rangle$ be some encoding of the ordered pair a,b as a natural number which is "efficient", that is, we can encode and decode in polynomial time.

Definition. If f is a total function over the natural numbers, we let the graph of f be

$$G_f = \{\langle n,f(n)\rangle \mid n \text{ is a natural number}\}$$

and the projection of f be

$$P_f = \{\langle n,m\rangle \mid f(n) \leq m\} .$$

Using these two definitions we get the following lemmas. (We let SPG denote those total functions with syntactic polynomial growth, defined in the previous section.)

Lemma 8. If $f \in SPG$ then the following statements are equivalent:

1) $G_f \in NP \cap \overline{NP}$

2) $G_f \in NP$

3) $f \in NP^*$

4) $P_f \in NP \cap \overline{NP}$

Proof. The cases 1) $\Rightarrow$ 2) $\Rightarrow$ 3) $\Rightarrow$ 4) and 3) $\Rightarrow$ 1) are straightforward. We prove the case 4) $\Rightarrow$ 3). Since $P_f \in NP \cap \overline{NP}$ there exists a nondeterministic machine which computes the characteristic function of P_f in polynomial time. Since f is in SPG there exist constants k and c such that $|f(n)| \leq c|n|^k$. The value of $f(n)$ lies between 0 and $2^{c|n|^k}$. Thus, using a binary search we need only compute $c|n|^k$ values of the characteristic function of P_f. □

Lemma 9. If $f, g \in SPG$ then the following hold:

1) $P_f \in P$ if and only if $f \in P^*$

2) If $f \approx_p g$ and $G_g \in P$ then $G_f \in P$.

Proof. 1) $f \in P^* \Rightarrow P_f \in P$ is clear whereas $P_f \in P \Rightarrow f \in P^*$ follows by the same argument used to show that $P_f \in NP \cap \overline{NP} \Rightarrow f \in NP^*$.

2) Consider the following machine, say M, on input $\langle n,m \rangle$:

i) M decodes $\langle n,m \rangle$ into n and m and attempts to compute $g(n)$ by the algorithm given by $g \leq_p f$ on inputs n and m. If it halts with a possible value for $g(n)$, say h, M continues to step ii). If the algorithm uses more than some $c|n|^k$ steps it rejects $\langle n,m \rangle$, where c, k are given by the reduction.

ii) M computes $\langle n,h \rangle$ and checks if $\langle n,h \rangle \in G_g$ by the algorithm given by $G_g \in P$. If $\langle n,h \rangle \notin G_g$ then M rejects $\langle n,m \rangle$ otherwise it continues to iii) and we know $h = g(n)$.

iii) M computes $f(n)$ using $n, g(n)$ by the algorithm given by $f \leq_p g$. If $f(n) = m$ then M accepts $\langle n,m \rangle$ otherwise it rejects $\langle n,m \rangle$.

It should be clear that M runs in polynomial time and that it accepts precisely G_f. □

Using the last two lemmas and the reudctions of the last section we get:

Theorem 5 (ERH). The graphs of ϕ, λ, λ' and "prime factorization" are recognizable in polynomial time.

Proof. By Lemma 9 we need only show that the graph of "prime factorization" is in P since by Theorem 3 all four functions are

polynomial time equivalent. But, the graph of prime factorization is in P by Theorem 2. □

Theorem 6. The projections of "period", ϕ, λ, λ' and "prime factorization" are members of $NP \cap \overline{NP}$ and if any of these projections are members of P then all the functions are in P^*.

Proof. The first part of Theorem 6 follows from Corollary 2 and Lemma 8 while the second part follows from Lemma 9 part 1). □

These results permit us to make a distinction between our two methods of constructing recognition problems from functions. Theorem 5 suggests that the graph of a function may be easy to recognize while the function may be difficult to compute. Lemmas 8 and 9 show that projection is a natural complexity preserving map from functions to relations. Theorem 6 exhibits possible candidates for recognition problems in $(NP \cap \overline{NP}) - P$.

Appendix

Let Z_n denote the ring of integers mod n. Let Z_n^* denote the integers relatively prime to n under multiplication mod n. Z_n^* is a group and if p is a prime then Z_p^* is a cyclic group of order p-1. Thus, the only solutions to the equation $x^2 \equiv 1 \bmod p$ are ± 1. We may pick a generator of the cyclic group Z_p^*, say b; then we define $\text{ind}_p a = \min\{m: b^m \equiv a \bmod p\}$. We note that $\text{ind}_p a$ is dependent on our choice of a generator. We say a is a q-th residue mod p if there exists b $(b^q \equiv a \bmod p)$.

Note. If p, q are primes and $q|p-1$ then a is a q-th residue mod p if and only if $q|\text{ind}_p a$.

Definition. The Legendre symbol $(\frac{a}{p})$ is defined by:

$$(\frac{a}{p}) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue mod } p \text{ and } (a,p) = 1; \\ -1 & \text{if } a \text{ is a quadratic nonresidue mod } p \text{ and } (a,p) = 1; \\ 0 & \text{if } (a,p) \neq 1. \end{cases}$$

The Jacobi symbol $(\frac{a}{pq})$ is defined by:

$$(\frac{a}{pq}) = (\frac{a}{p}) \cdot (\frac{a}{q})$$

where $(\frac{a}{p})$ and $(\frac{a}{q})$ are the Legendre symbols.

The above two symbols for fixed denominators define functions which fall into a general class of functions called characters. We define one more character as follows:

$$\chi(a) = \begin{cases} e(2\pi i(\text{ind}_p a)/q) & \text{if } (a,p) = 1 \\ 0 & \text{if } (a,p) \neq 1 \end{cases}$$

where $q|p-1$ and $e(\ )$ is the exponential function.

Dirichlet's L functions are defined by:

$$L(S,\chi) = \sum_{n-1}^{\infty} \chi(n)/n^S$$

where χ is a character.

Extended Riemann's Hypothesis (ERH). The zeros of $L(S,\chi)$ in the critical strip, $0 \leq$ (real part of S) ≤ 1 all lie on the line (real part of S) $= \frac{1}{2}$, where χ is any of the three characters above.

References

[1] N.C. Ankeny, "The Least Quadratic Non-Residue," Annals of Mathematics 55 (1952) 65-72.

[2] D.A. Burgess, "The Distribution of Quadratic Residues and Non-Residues," Mathematika 4 (1957) 106-112.

[3] D.A. Burgess, "On Character Sums and Primitive Roots," Proc. London Mathematical Society 12 (3) (1962) 179-192.

[4] D.A. Burgess, "On Character Sums and L-series," Proc. London Mathematical Society 12 (3) (1962) 193-206.

[5] R.D. Carmichael, "On Composite Numbers p Which Satisfy the Fermat Congruence $a^{p-1} \equiv p$," American Mathematical Monthly 19 (1912) 22-27.

[6] S.A. Cook, "The Complexity of Thoerem-proving Procedures," Conference Record of Third ACM Symposium on Theory of Computing (1970) 151-158.

[7] H. Davenport and P. Erdös, "The Distribution of Quadratic and Higher Residues," Publ. Math. Debreien 2 (1952) 252-265.

[8] G. Hardy and E. Wright, An Introduction to the Theory of Numbers, Oxford Press (1968) 111.

[9] R.M. Karp, "Reducibility Among Combinatorial Problems," Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York (1972) 85-103.

[10] D. Knuth, The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Addison-Wesley, Reading, Mass. (1969).

[11] G.L. Miller, "Riemann's Hypothesis and Tests for Primality," Proceedings of Seventh Annual ACM Symposium on Theory of Computing (1975) 234-239.

[12] H. Montgomery, Topics in Multiplicative Number Theory, Springer-Verlag Lecture Notes #227, 120.

[13] J. Pollard, "An Algorithm for Testing the Primality of Any Integer," Bulletin of the London Mathematical Society 3 (1971) 337-340.

[14] J. Pollard, "Theorems on Factorization and Primality Testing," Proceedings of Camp. Philosophical Society 76 (1974) 521-528.

[15] V. Pratt, "Every Prime Has a Succinct Certificate," to appear.

[16] A. Selberg, "Contributions to the Theory of Dirichlet L Functions," Avhandlinger utgett av Det Norske Videnskops, Akademi i Oslo (1934).

[17] D. Shanks, "Class Number, A Theory of Factorization and Genera," Proceedings of Symposia in Pure Mathematics 20 (1969) Number Theory Institute, American Mathematical Society (1971) 415-440.

[18] A. Schönhage and V. Strassen, "Schnelle Multiplikation Grosser Zahlen," Computing 7 (1971) 281-292.