PRIME GENERATION DRAFT

JASON MEDCOFF

1. Introduction

Recall the following definition.

Definition 1.1. A natural number $p > 1$ is said to be prime if its only positive divisors are 1 and p. If p is not prime, it is said to be composite.

One of the most important theorems with regards to the primes and integers follows.

Theorem 1 (Fundamental Theorem of Arithmetic)**.** *If $n > 1$ is a natural number, it can be written as a product of primes unique up to the order of the factors.*

Proof. Suppose there exists an integer $n > 1$ such that n cannot be written as a product of primes. By the well-ordering principle, we know that there must be a smallest n satisfying this. Write n as a product, $n = xy$, with $x > 1$ and $y < n$. Since n is the smallest positive integer that cannot be written as a product of primes, and x and y must be less than n, we can write x and y as products of prime factors. Then n can be written as a product of primes, which contradicts the above assumption. So, every integer $n > 1$ can be written as a product of primes.

Next, suppose an integer $n > 1$ has two prime factorizations, such that

$$n = j_1 \cdots j_s = k_1 \cdots k_t$$

where all j_i and k_i are prime. By the well-ordering principle, we know that there must be a smallest n satisfying this as well. Then $j_1 | n = k_1 \cdots k_t$ and since j_1 is prime, it must divide k_i for some i. But we know that k_i is prime too, so it must be that $j_1 = k_i$. Then we relabel k_i as k_1 for simplicity, and we now cancel j_1 to obtain

$$j_2 \cdots j_s = k_2 \cdots k_t.$$

Since n is the smallest positive integer that has a non-unique prime factorization, and $j_2 \cdots j_s = k_2 \cdots k_t < n$ is a product of primes, we arrive at a contradiction. Then there cannot exist an integer $n > 1$ with two different factorizations. □

An immediate question resulting from this theorem is how we can best obtain the prime factors of arbitrary numbers. This question is still open in number theory and theoretical computer science; no classical algorithm is known that can factor a number in polynomial time on the number of digits. The difficulty of factoring very large numbers (on the order of thousands of digits) led to the creation of the RSA cryptosystem.

Given some number n, we may want to find out whether n is prime or composite. Naively, we can simply enumerate the natural numbers starting at 2, and check for divisibility, stopping when we reach n. However, in the interest of saving time, we can apply a result that will cut down the time spent checking divisors.

Lemma 1. *If n is composite, at least one of its factors lies in $[2, \sqrt{n}]$.*

Proof. Let $m = \sqrt{n}$. Then we can write n as

$$n = m \cdot m = a \cdot b.$$

Consider a. One of the following must be true: $a > m$, $a < m$, or $a = m$. In the first case, b must be less than m; in the second case, b will be greater than m. If $a = m$, clearly $b = m$. Therefore, if we search the integers up to m, we will find at least one factor of n. □

We can combine this result with the fundamental theorem of arithmetic, and easily show the following.

Corollary 1. *Let $n > 1$ be an integer. If n does not have a prime factor in the range $[2, \sqrt{n}]$, it is prime.*

This is a combination of the contrapositive of lemma 1 with theorem 1. So now, we have a way to check for primality of small numbers. For some n, we need only check divisibility by primes in $[2, \sqrt{n}]$. This algorithm, called trial division, is shown to be very slow on larger numbers.

2. Generating Primes with the Sieve of Eratosthenes

The Sieve of Eratosthenes is a process for generating primes up to some positive integer n. Roughly speaking, the process is to cross off the multiples of the numbers up to the square root of n. Those numbers that remain are the primes up to n.

Consider the following algorithm.

Figure 1. Sieve of Eratosthenes

```
def eratosthenes(n):
    composites = set()
    for i in range(2, n+1):
        if i not in composites:
            yield i
            composites.update(range(i**2, n+1, i))
```

Theorem 2. *The algorithm `eratosthenes` produces the prime numbers in the range $2, n$.*

Proof. We know the algorithm must terminate because its loop bounds are finite. The algorithm's correctness is easy to see based on the use of `composites`. The loop invariant here is that the set `composites` contains all composite numbers between 2 and i; this holds for every iteration, where we yield i if it is not composite, then add multiples of i from i^2 to n to the `composites` set. At the end of the last loop, every prime has been yielded in the interval $[2, n]$. □

Informally speaking, we are using the `composites` set to "mark" composite numbers in the range, then returning along the way those numbers which are not marked.