Number Theory and Primality Tests

UCLA Math Circle Advanced 1

by Jack Fasching

Week 1: January 12, 2025

Introduction

We start with some simple definitions:

Definition 0.1 *A natural number greater than 1 is* ***prime*** *if it cannot be written as a product of two natural numbers that are both greater than 1.*

Definition 0.2 *A natural number greater than 1 is* ***composite*** *if it is not prime.*

For small numbers, we can easily determine whether they are prime or not:

Problem 0.1 *Of the natural numbers 1-20, which of them are primes?*

However, determining whether large numbers are prime or not is difficult for humans. However, computers are much more useful for determining the a number's primality (whether or not it is prime).

This is where **primarily tests** come in: they are computational algorithms (allowing them to be coded into computer software) which determine whether a number is prime or not. They are usually coded into software instead of being used by humans, since computers can perform billions of calculations in seconds. Despite this, mathematicians and computer scientists still try to make these algorithms as efficient as possible, determining primarily in the fewest computations. These seemingly small changes in computation time make a huge difference, as we will see later in the packet.

Primality tests are especially useful in cryptography as they quickly provide large prime numbers for modern encryption algorithms.

Additionally, primality tests rely on theorems from number theory to justify their approach. When introducing primarily tests, we will prove the theorems they use.

1 Trial Division

1.1 Introduction

This is the simplest primality test. Given natural number $n > 1$, we can write out the algorithm in pseudocode:

```
Let integer i = 2;
While i is not greater than n-1, do the following:

    If the remainder of n divided by i is 0:
        Return "n is composite" and exit code.
    Otherwise, increase i by 1.

If code goes through all values of i:
    Return "n is prime" and exit code.
```

Problem 1.1 *Follow the algorithm to determine whether n is prime for $n = 5$ and $n = 12$.*

One way to measure the speed of this program is to count how many times we check if i is a divisor of n.

Problem 1.2 *How many iterations of checking divisors do we go through for $n = 7$, $n = 15$, $n = 53$ (prime), and $n = 91$ (not prime)?*

Problem 1.3 *If we do not know the primality of n, what is the maximum number of iterations of checking divisors do we go through? For what n is this the case? (this is the worst-case scenario for our algorithm, where we make the least computations).*

Problem 1.4 *If we already know n is large and composite, under what conditions do we have the best-case scenario (the most computations)? Under what conditions do we have the worst-case scenario (the least computations)? Give both the conditions for n and the min/max number of iterations.*

(Fill in the blanks) From problems 1.3 and 1.4, we see that it takes a maximum of _____ divisor checks to determine n is prime, but a maximum of _____ divisor checks to determine n is composite.

If we still have not proved n is (Circle one) COMPOSITE / PRIME after _____ divisor checks, then it must be COMPOSITE / PRIME.

So, we can make our algorithm more efficient by just doing _____ divisor checks.

Problem 1.5 *What line(s) of pseudocode from our algorithm do we modify to limit our number of divisor checks? Explain why our new algorithm still determines the primality of n.*
Have an instructor check your work before moving on.

Problem 1.6 *How many iterations of checking divisors do we go through for $n = 7$, $n = 15$, $n = 53$, and $n = 91$?*

Problem 1.7 *In general, as we increase the value of n, how will the run time of our new algorithm compare to that of our old algorithm?*

We can further make this program more efficient by proving the Fundamental Theorem of Arithmetic:

1.2 Proving the Fundamental Theorem of Arithmetic

This next section will focus more on number theory. Let's start with a few helpful definitions:

Definition 1.1 *The **greatest common divisor** of nonzero integers a and b is the greatest integer that divides both a and b. It is denoted by* $\gcd(a, b)$.

Problem 1.8 *Compute the following:*

- $\gcd(4, 6) =$
- $\gcd(20, 28) =$
- $\gcd(9, 27) =$
- $\gcd(11, 1) =$
- $\gcd(11, 0) =$

Definition 1.2 (The Division Algorithm) *Given two integers a, b, we can find two integers q, r, where $0 \leq r < b$ and $a = qb + r$. In other words, we can divide a by b to get q remainder r.*

Problem 1.9 *Find q, r making $14 = 3q + r$ as in the division algorithm.*

Problem 1.10 *Find q, r making $87 = 12q + r$ as in the division algorithm.*

Problem 1.11 *Prove that q, r are unique to a, b.*

Using the two definitions above, we can show the following:

Theorem 1.1 (Bezout's Identity) *Let a, b be integers with greatest common divisor d. Then there exist integers x, y such that $ax + by = d$.*

Problem 1.12 *Prove Bezout's Identity.*

Hint: By the well-ordering principle, a nonempty subset of the positive integers has a minimum element. So, if you can show that the set

$\{ax + by \mid x, y \in \mathbf{Z} \text{ and } ax + by > 0\} \subseteq \mathbf{Z}^+$

is nonempty, then you can say it has a minimum element $d = as + bt$ for some s, t. Now you only need to prove that $d = \gcd(a, b)$.

*** Challenge Problem 1.1** *Prove that integers of the form $az + bt$ are exactly the multiples of d.*

This theorem is especially helpful in justifying our next step:

Lemma 1.1 (Euclid's Lemma) *If a prime p divides the product ab of two integers a and b, then p must divide at least one of those integers a or b.*

Problem 1.13 *Prove Euclid's Lemma.*

*** Challenge Problem 1.2** *Generalize Euclid's Lemma to the following:*
If an integer n divides the product ab of two integers, and is coprime with a (in other words $\gcd(n, a) = 1$), then n divides b.

Finally, we can use Euclid's Lemma to prove the Fundamental Theorem of Arithmetic:

Theorem 1.2 (Fundamental Theorem of Arithmetic) *Every integer greater than 1 can be represented uniquely as a product of prime numbers, up to the order of their factors (in other words, this unique representation $p_1^{e_1}...p_k^{e_k}$ has $p_1 < ... < p_k$).*

Problem 1.14 *Prove the Fundamental Theorem of Arithmetic.*

The proof involves two parts:

EXISTENCE: Show that every integer greater than 1 is either a prime or a product of primes.

UNIQUENESS: Show that every integer greater than 1 has a unique prime factorization.
In other words, if $p_1^{e_1}...p_k^{e_k} = q_1^{f_1}...q_m^{f_m}$ for primes p_i, q_j with $p_1 < ... < p_k$ and $q_1 < ... < q_m$, then $k = m$, $p_1 = q_1,\ ...,\ p_k = q_k$, and $e_1 = f_1,\ ...,\ e_k = f_k$. (Hint: Apply Euclid's Lemma to create a concise proof.)

1.3 Further Improving Trial Division

After looking at the Fundamental Theorem of Arithmetic, we can see that every natural number n greater than 1 that we input into the algorithm is divisible by at least one prime number.

So instead of checking all possible divisors of n from 2 to $\sqrt{n}$, we can check all possible prime divisors of n from 2 to $\sqrt{n}$. However, we still need to check if these divisors are prime.

One way we can do this is to use a pre-computed list of primes no greater than $\sqrt{n}$:

```
Let L be an empty list of primes not greater than sqrt(n).
For each element l in L, do the following:
    If the remainder of n divided by l is 0:
        Return "n is composite" and exit code.

If code goes through all values of L:
    Return "n is prime" and exit code.
```

Problem 1.15 *Detail an algorithm, similar in spirit to the one we wrote above, that given a natural number $n > 1$ generates a list of all natural primes no greater than $\sqrt{n}$.*

However, we cannot always use this approach:

Problem 1.16 *Give a possible scenario where it is not feasible to compute such a list of primes.*

An alternative method is only checking divisors that are relatively prime to a few primes.

For example, we can check divisors 2, 3, and 5, and then check all divisors up to floor($\sqrt{n}$) which aren't divisible by 2, 3, and 5.

Problem 1.17 *Detail an algorithm in pseudocode for the primality test described above. Check with an instructor before continuing.*

Problem 1.18 *How many iterations of checking divisors do we go through for $n = 97$ and $n = 391 = 17 \cdot 23$?*

1.4 Back to Number Theory

However, we can once again use number theory to make our code more efficient:

Problem 1.19 *Prove that all primes $p > 3$ are in the form of $6k + i$ for non-negative integer k and $i \in \{1, 5\}$.*

Problem 1.20 *For any prime $p > 5$ in the form of $30k + i$ for non-negative integers k and $i < 30$, what are all possible values of i? Justify your answer.*

Problem 1.21 *Let p_j be the jth prime number for any natural number j.*
In general, for any prime $p > p_j$ in the form of $(p_1p_2p_3 \cdots p_j)k + i$ for non-negative integers k and $i < p_1p_2p_3 \cdots p_j$, what can be say about i?

So, instead of checking divisibility for each divisor greater than 5 in our algorithm, we can check divisors in the form $30k + i$ for certain i.

Problem 1.22 *Rewrite the algorithm from Problem 1.17 so that it checks divisors as described above. Check with an instructor before continuing.*

Problem 1.23 *How many iterations of checking divisors do we go through for* $n = 97$ *and* $n = 391 = 17 \cdot 23$*?*

Problem 1.24 *As* n *gets larger and larger, what is approximately the number of divisor checks our new algorithm makes (in terms of* n*)?*

We will continue to focus on primality tests in the second part of this packet.

2 Challenge Problems

Problem 2.1 *(2020 AMC 10B Problem 22) What is the remainder when* $2^{202} + 202$ *is divided by* $2^{201} + 2^{51} + 1$*?*

Problem 2.2 *(2020 AMC 10B Problem 24) Let* n *be the least positive integer greater than 1000 for which* $\gcd(63, n + 120) = 21$ *and* $\gcd(n + 63, 120) = 60$*. What is the sum of the digits of* n*?*

Problem 2.3 *(USAMO 1997) Let* p_n *be the* n*th prime and let* $a \in (0, 1)$ *be a real number. Define the sequence* x_n *by*

$$x_0 = a \qquad x_n = \begin{cases} \textit{the fractional part of } p_n/x_{n-1} & x_{n-1} \neq 0 \\ 0 & \textit{else} \end{cases}$$

Find all a *for which the sequence is eventually* 0*.*