# 15 Number Theory

## 15.1 Primality Tests (9 units)

*This project is related to material in the Part II course Number Theory.*

A primality test is an algorithm used to determine whether or not a given integer is prime. In this project we consider several different primality tests, and apply them to numbers in the range from 3 up to $10^{10}$.

## 1 Trial division

The simplest primality test is *trial division*: $N$ is prime if and only if it is not divisible by any integer $t$ with $1 < t \leqslant \sqrt{N}$.

> **Question 1** Write a program to test for primality using trial division. Use your program to list the primes in the intervals $[188000, 188200]$ and $[10^9, 10^9 + 200]$.

## 2 The Fermat test

Fermat's Little Theorem states that if $p$ is prime then $a^{p-1} \equiv 1 \bmod p$ for any $a$ coprime to $p$. The *Fermat test base* $a$ for $N$, if $1 < a < N$, is to compute $a^{N-1} \bmod N$: if this is not $\equiv 1 \bmod N$ then $N$ is certainly composite. A *Fermat pseudoprime base* $a$ is a composite $N$ which passes the Fermat test base $a$.

> **Question 2** Write a program to carry out the Fermat test base $a$, capable of working for $N$ up to $10^{10}$. Run it on the intervals in Question 1, say for $a$ up to 13. Check that the output is consistent with your earlier answer, and make a note of any Fermat pseudoprimes that you find.
>
> You should take care to devise a good algorithm for computing $a^b \bmod N$, ensuring that there is no possibility of integer overflow during the calculation, in view of the possibility that the integers in your chosen language may be limited to $10^{15}$ or similar. (See the note on programming at the end of the project.) If the integers in your language are large enough for there to be no chance of overflows, then you should still comment briefly on how you might manage if you were limited to $10^{15}$. (*Hint*: A multiplication modulo $N$ can be done in two pieces.)
>
> Briefly discuss the complexity of your algorithm, *i.e.* the time theoretically taken. To do this, you should consider (without coding it) how your program might extend to arbitrarily large $N$, and work out roughly how many basic operations are needed as $N \to \infty$, where a basic operation could be addition or multiplication of two numbers of similar size to $N$. (A more sophisticated analysis might also take into account the time required to add and multiply large numbers on a finite machine, but it isn't necessary to go into such details here.)

An *absolute Fermat pseudoprime*, also called a *Carmichael number*, is a composite number which passes the Fermat test for any base $a$ with $(a, N) = 1$.

 

**Question 3** Find the Fermat pseudoprimes base 2 and the absolute Fermat pseudoprimes up to $10^6$. Can you think of any very simple methods (not involving any elaborate theory) to speed up the checking of the absolute pseudoprimes?

How many values of $a$, starting from $a = 2$, are necessary to determine the primality or compositeness of those $N$ in this range which are not absolute Fermat pseudoprimes? Note that if $a$ has a non-trivial common factor with $N$ then we regard $a$ as having determined that $N$ is composite.

The existence of infinitely many absolute Fermat pseudoprimes (proven in 1994) means that the Fermat test cannot be relied on to prove the primality of $N$ any faster than trial division, although it can usually detect compositeness quickly.

## 3 The Euler test

We can improve the Fermat test by using Euler's theorem, that if $p$ is an odd prime then $a^{(p-1)/2} \equiv (a/p) \bmod p$ where $(a/p)$ is the Legendre symbol. The *Euler test base* $a$, for an odd integer $N$, is to compute $a^{(N-1)/2} \bmod N$ and check if this is $\pm 1$ and equal to the *Jacobi symbol* $(a/N)$: for $N$ odd and positive this satisfies the properties

$$\left(\frac{a}{N}\right) = \left(\frac{a \bmod N}{N}\right),$$

$$\left(\frac{ab}{N}\right) = \left(\frac{a}{N}\right)\left(\frac{b}{N}\right),$$

$$\left(\frac{-1}{N}\right) = (-1)^{(N-1)/2},$$

$$\left(\frac{2}{N}\right) = (-1)^{(N^2-1)/8},$$

and, if $M$ and $N$ are odd positive and coprime,

$$\left(\frac{M}{N}\right)\left(\frac{N}{M}\right) = (-1)^{(M-1)(N-1)/4}.$$

If $N$ is prime then $(a/N)$ is just the Legendre symbol. As before, an *Euler pseudoprime base* $a$ is a composite $N$ which passes the Euler test base $a$, and an *absolute Euler pseudoprime* is a composite number which passes the Euler test for any base $a$ with $(a, N) = 1$.

**Question 4** Write a procedure to evaluate the Jacobi symbol and modify your previous program to carry out the Euler test. Find the Euler pseudoprimes base 2 and the absolute Euler pseudoprimes up to $10^6$. How many values of $a$ are necessary to determine the primality or otherwise of those $N$ in this range which are not absolute Euler pseudoprimes?

## 4 The strong test

If $p$ is prime then the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ is cyclic and so $-1$ is the only element of order 2. If we put $p - 1 = 2^r s$ where $s$ is odd then either $a^s \equiv 1 \bmod p$ or there is $j$, $0 \leqslant j < r$ such that $a^{2^j s} \equiv -1 \bmod p$. The *strong*, or *Miller–Rabin test base* $a$ is to put $N - 1 = 2^r s$ with $s$ odd and to compute the sequence $a^s$, $a^{2s}$, …, $a^{2^{r-1}s}$: if the sequence begins with 1 or contains $-1$ then $N$ passes the strong test.

 

**Question 5** Modify your program to carry out the strong test. As before, find the strong pseudoprimes base 2 and the absolute strong pseudoprimes up to $10^6$. How many values of $a$ are required to determine the character of the integers $N$ in this range?

**Question 6** Use your programs to tabulate the number of Fermat/Euler/strong pseudoprimes base 2 in the intervals $[10^k, 10^k + 10^5]$ for $5 \leqslant k \leqslant 9$. For comparison you should also list the numbers of primes in these intervals. Repeat for base $a = 3$, and also for numbers passing the corresponding tests for both $a = 2$ and $a = 3$.

Briefly comment on the relation between the Fermat, Euler and strong tests.

**Question 7** Combining all your previous results, devise the most efficient algorithm you can for determining the primality or otherwise of a number $N$ in the range from 3 up to $10^{10}$. (This may be a mixture of the above methods.)

Experiment on a suitable number of random numbers, say 10000, in the range to compare the time taken by your algorithm and trial division. Comment also on the time required theoretically for these algorithms for large $N$, making clear any assumptions you are making.

An important question in applications is the reliability of these tests for determining the primality of randomly chosen numbers in some range.

**Question 8** Fix a value of $k$ and suppose that $N$ is chosen uniformly at random from all odd integers of exactly $k$ bits, that is, between $2^{k-1}$ and $2^k - 1$. (You should be able to take $k$ at least 15.) Investigate the probability that if $N$ passes $t$ rounds of the strong test with randomly chosen $a$, then $N$ is in fact composite.

## Programming note

The quantities of interest in this project are natural numbers. You might be aware that computer languages typically like to be told if a number is an integer rather than a general real, because it means there is no need to save a fractional part, and it avoids concerns over rounding errors.

MATLAB, though, is incorrigibly real-minded. Hence, although MATLAB allows you to tell it a number is an integer with the `int32` command (see the CATAM manual), it is better for this project if you don't do so. In other words, you need not worry about distinguishing between reals and integers.

You should, however, be aware that MATLAB can handle integers in this way only up to about 15 digits. (Try `eps(10^15)` and `eps(10^16)`. Try also `10^8 * 10^8 - (10^16-1)`.) So you must avoid calculations involving integers exceeding $10^{15}$.

You may use the MATLAB functions `conv` and `deconv` to multiply and divide polynomials.

Obviously no credit can be given in this project for using inbuilt functions such as `isprime` or `factor` — you are expected to write and analyse your own programs. You may however use the inbuilt MATLAB function `gcd`.

 

## References

[1] Riesel, H., *Prime numbers and computer methods for factorisation*, 2nd edition, Progress in Mathematics 126, Birkhauser, 1994

[2] Koblitz, N., *A Course in Number Theory and Cryptography*, Graduate Texts in Mathematics 114, Springer, 1987.

 