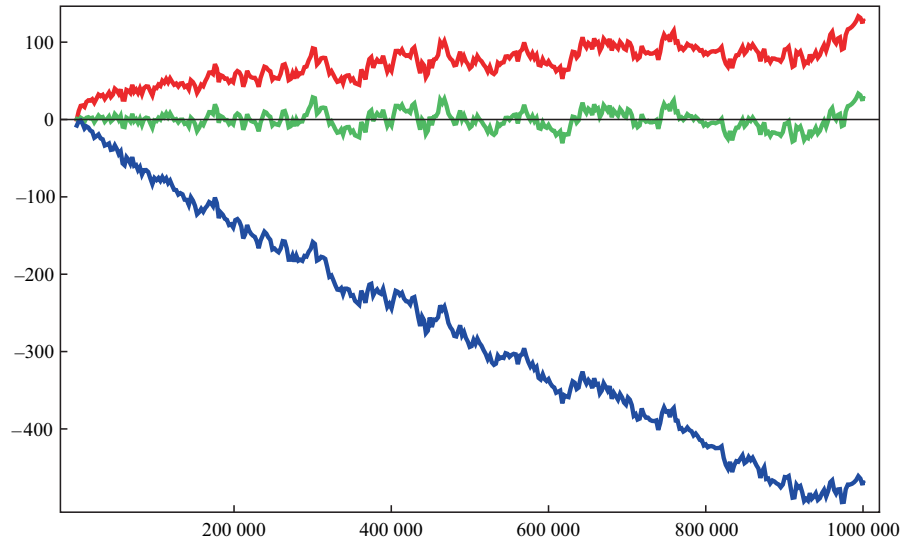


2 Prime Numbers



Many famous mathematicians have found relatively simple functions that model the behavior of $\pi(x)$, the number of primes below x . This graph shows the error, up to one million, of three such approximations: Legendre and Chebyshev used logarithms (blue graph; beyond 10^{12} , Chebyshev is better than Legendre), Gauss (red) used the integral of the reciprocal of the logarithm, and Riemann (green) enhanced Gauss's integral with an infinite series. As an example of the power of such formulas, note that Gauss's estimate for $\pi(10^{18})$,

$$\int_2^{10^{18}} \frac{1}{\log t} dt$$

is 24739954309690414 while the actual value is 24739954287740860; the relative error is about 1 part in a billion.

This chapter uses elementary number theory to introduce a variety of elementary and advanced features of *Mathematica*. We will see how to use *Mathematica* as a supercalculator and how to write short programs. Several important techniques are introduced, as well as different approaches to graphing data. Only a little bit of number theory is assumed — not much beyond modular arithmetic and the definition of a prime number.

2.1 Basic Number Theory Functions

There are several easy-to-use functions that can help us understand prime numbers. `Prime[n]` returns the n th prime number.

```
Prime[1000]
```

```
7919
```

`Prime` is a `Listable` function, which means that it works as expected when the argument is a list of integers. We can also feed it a range of integers. `Range` is the quickest way to generate an interval of integers, and we can raise 10 to the entries in a list to get a bunch of powers of 10.

```
10Range[5]
```

```
{10, 100, 1000, 10 000, 100 000}
```

So now we can see the 10th prime, 100th prime, and so on.

```
Prime[10Range[9]]
```

```
{29, 541, 7919, 104 729, 1 299 709,  
15 485 863, 179 424 673, 2 038 074 743, 22 801 763 489}
```

Another important function related to the primes goes by the name of $\pi(x)$, called `PrimePi` in *Mathematica*; $\pi(x)$ is the number of primes less than or equal to x .

```
PrimePi[106]
```

```
78 498
```

So there are 78498 primes less than 1 million. Because $\pi(x)$ is essentially the inverse of `Prime`, if we ask for the 78499th prime, we get the first prime beyond 1 million.

```
Prime[78499]
```

```
1 000 003
```

There are limits to how far one can go with `Prime` and `PrimePi`. The largest cur-

rently known value of these functions at a power of 10 is $\pi(10^{23}) = 1\,925\,320\,391\,606\,803\,968\,923$ (due to T. Oliveira e Silva; see [Wei1]) but that is the result of many hours of computation using specialized algorithms. *Mathematica's* PrimePi works below 10^{14} (and Prime works up to about 10^{12}).

For the complete factorization of an integer into primes, use FactorInteger, but again be aware that this is a very difficult problem for large numbers. The output consists of pairs: primes and the exponents to which they occur.

```
FactorInteger[11737654214175]
{{3, 2}, {5, 2}, {3607, 1}, {3803, 2}}
```

If you want to check such an output, it can be done as follows. First we gather the primes and the exponents by treating the output as a matrix and transposing (% refers to the preceding output).

```
Transpose[%]
{{3, 5, 3607, 3803}, {2, 2, 1, 2}}
```

Then we raise the primes to the exponents.

```
%[[1]]%[[2]]
{9, 25, 3607, 14462809}
```

And finally we Apply Times. Recall that applying a function to a set turns the elements of the set into an argument sequence for the function.

```
Apply[Times, %]
11737654214175
```

Some large numbers can be factored, of course, but most cannot in reasonable time. The initialization group for this chapter contains a function called FactorForm that puts factorizations in a familiar form.

```
FactorInteger[100!] // FactorForm
297 348 524 716 119 137 175 195 234 293
313 372 412 432 472 53 59 61 67 71 73 79 83 89 97
```

Other useful functions are Divisors, which returns the set of all divisors of an integer, GCD and LCM, which denote the greatest common divisor and least common multiple functions, respectively, and PrimeQ, which attempts to return True or False according as the input is prime (explanation of "attempts" is given shortly).

```
Divisors[123456789]
{1, 3, 9, 3607, 3803, 10821, 11409,
 32463, 34227, 13717421, 41152263, 123456789}
```

```
GCD[76 895, 16 302 982 080]
```

```
35
```

```
LCM[165, 150]
```

```
1650
```

The GCD and LCM functions are based on the Euclidean algorithm, which is extremely fast. Thus they can be used on 100-digit numbers with no excessive slowdown. Two other very fast and very important functions are Mod and PowerMod. Mod simply reduces its first argument modulo its second. There is also Divisible, which checks whether one number is divisible by another.

```
Mod[1 238 719 479 147 974, 100]
```

```
74
```

We can easily check Wilson's theorem that $(p - 1)! + 1$ is divisible by p if and only if p is prime.

```
Divisible[100! + 1, 101]
```

```
True
```

Here is how one would get the numbers for which the Wilson condition holds.

```
WilsonQ[n_] := Divisible[(n - 1)! + 1, n]
```

```
w = Select[Range[2, 200], WilsonQ]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199}
```

And we can check the result.

```
Table[Mod[w[[i]]!, i], {i, Length[w]}]
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Although the syntax of the preceding example is quite simple, it is worthwhile to learn how pure functions are used in such a situation. Here is a one-line approach to the same problem. `Divisible[(# - 1)! + 1, #] &` is a pure function that returns True or False. The `#` is viewed as the generic variable, and the `&` indicates the end of the pure function construction. The advantage of this approach is that it is faster, more elegant, and requires less code. The disadvantage is that it is harder to read.

```
Select[Range[2, 200], Divisible[(# - 1)! + 1, #] &]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199}
```

The reader might try to prove Wilson's theorem. For a clue to the proof, look at the result of

```
Table[Mod[(n - 1)! + 1, n], {n, 2, 200}]
```

Wilson's theorem is not a workable test for primality because of the excessive effort required to compute a large factorial.

For modular powers such as $9^{1000000} \pmod{7}$, one should not raise 9 to the millionth power and reduce modulo 7; that is doomed because of the excessive time and memory needed to compute $9^{1000000}$. There is a much better way based on looking at the base-2 representation of the exponent, working through these a digit at a time, and reducing modulo 7 at each stage (see [BW]). `PowerMod` uses this algorithm and is very fast. Moreover, any of the three arguments can be quite large.

```
PowerMod[9, 1 000 000, 7]
```

```
2
```

```
PowerMod[9 999 999 999, 10100, 700]
```

```
501
```

`PowerMod` also has the nifty property that it accepts negative exponents. And if the exponent is -1 , then `PowerMod` computes the modular inverse (if it exists).

```
PowerMod[57, -1, 100]
```

```
93
```

And indeed $57 \cdot 93$ leaves a remainder of 1 (mod 100).

```
57 x 93
```

```
5301
```

One can even use fractional exponents to get a modular root (though not the full set of modular roots).

```
PowerMod[444,  $\frac{1}{2}$ , 1000]
```

```
38
```

Here is how to get all the square roots of 444 modulo 1000.

```
x /. Solve[{x2 == 444, Modulus == 1000}]
```

```
{38, 462, 538, 962}
```

The `PrimeQ` function is unusual among *Mathematica* functions in that it is not guaranteed to tell the truth. Briefly, it uses some tests that are known to tell the truth if the input number is less than 10^{16} . Possibly a counterexample exists beyond that

point, but it is hard to say where it might be; it could be that the first counterexample has billions of digits. Still, the point is important to understand, so we will discuss the issue in the context of some other elementary tests for primality. Note that there is no issue of probability here; the tests are completely deterministic. There are other tests that, while slower, are guaranteed to give correct results; see §21.3.

The first point to make is that it is not a good idea to test n for primality by checking all the potential divisors. There are too many of them (one would have to check potential divisors up to $\sqrt{n}$). But there are some clever ideas that are much, much faster. For example, Fermat's little theorem states that if p is prime, then $a^{p-1} \equiv 1 \pmod{p}$, provided $\gcd(a, p) = 1$. Here is the data for the prime 541, with a taking on all possible values.

```
Union[PowerMod[Range[1, 540], 540, 541]]
{1}
```

Now, it is tempting to hope that if n is odd and $2^{n-1} \equiv 1 \pmod{n}$, then n is prime. Such a test is very fast to execute thanks to `PowerMod`. Unfortunately, the test is not universally valid. Here is how `Select` can be used to find all counterexamples under 1000; there are 22 of them. These numbers are called *pseudoprimes*.

```
PseudoprimeQ[n_] := ! PrimeQ[n] && PowerMod[2, n - 1, n] == 1
Select[Range[10 000], PseudoprimeQ]
{341, 561, 645, 1105, 1387, 1729, 1905, 2047, 2465, 2701, 2821,
 3277, 4033, 4369, 4371, 4681, 5461, 6601, 7957, 8321, 8481, 8911}
```

Now, while a 99.8% success ratio is not too bad, it is far from perfect. A number that passes the base-2 test but is not in fact a prime number is called a *2-pseudoprime*. There is a slightly more complicated notion called a *b-strong pseudoprime*, which we will discuss in §2.5. And there are other varieties of pseudoprimes, such as Lucas pseudoprimes, Euler pseudoprimes, and Perrin pseudoprimes (see [BW]). The current version of `PrimeQ` is based on three tests: 2-strong pseudoprime, 3-strong pseudoprime, and Lucas pseudoprime. While no counterexample has been discovered up to 10^{16} (in other words, `PrimeQ` is proved to be reliable up to 10^{16}), it is quite possible that counterexamples do exist.

Finally, we mention `EulerPhi`, which computes the ϕ function: $\phi(n)$ gives the number of positive integers less than n that are relatively prime to n .

An important theorem is Euler's theorem, which states that $a^{\phi(n)} \equiv 1 \pmod{n}$ if a and n are relatively prime.

```
PowerMod[3, EulerPhi[1016], 1016]
```

1

We mention in passing a famous unsolved problem concerning ϕ called *Carmichael's conjecture*. The conjecture asserts that if $\phi(n) = m$, then there is an integer $r \neq n$ such that $\phi(r) = m$. Here is a quick example.

```
EulerPhi[267]
```

```
176
```

But we can find many other numbers that take on the value 176 under ϕ . Here is how `Select` would be used to find all of them less than 1000.

```
good[n_] := EulerPhi[n] == 176
Select[Range[1000], good]
{267, 345, 356, 368, 460, 534, 552, 690}
```

EXERCISE 1. Use a pure function to get the same result.

A new capability is an algorithm that finds all values of n so that $\phi(n)$ is a given value. Here is how that works. We use 400000, which is $\phi(10^6)$. In fact, there are 56 numbers m so that $\phi(m)$ is 400000.

```
Reduce[n > 0 && EulerPhi[n] == 400 000, n, Integers]

n == 401 851 || n == 404 101 || n == 430 967 || n == 445 511 || n == 500 125 ||
n == 503 255 || n == 517 625 || n == 533 375 || n == 551 375 || n == 566 005 ||
n == 584 375 || n == 751 875 || n == 771 825 || n == 796 875 || n == 803 702 ||
n == 805 208 || n == 808 202 || n == 811 232 || n == 845 625 || n == 861 934 ||
n == 891 022 || n == 905 608 || n == 1 000 000 || n == 1 000 250 ||
n == 1 002 500 || n == 1 004 000 || n == 1 006 510 || n == 1 010 000 ||
n == 1 014 040 || n == 1 025 000 || n == 1 029 100 || n == 1 035 250 ||
n == 1 062 500 || n == 1 066 750 || n == 1 100 000 || n == 1 102 750 ||
n == 1 104 400 || n == 1 111 000 || n == 1 127 500 || n == 1 132 010 ||
n == 1 168 750 || n == 1 207 812 || n == 1 216 848 || n == 1 358 412 ||
n == 1 500 000 || n == 1 503 750 || n == 1 506 000 || n == 1 515 000 ||
n == 1 521 060 || n == 1 537 500 || n == 1 543 650 || n == 1 593 750 ||
n == 1 650 000 || n == 1 656 600 || n == 1 666 500 || n == 1 691 250
```

Here is how to transform this output to a list.

```
Short[
  n /. {ToRules[Reduce[EulerPhi[n] == 400 000 && n > 0, n, Integers]]}]
{401 851, 404 101, 430 967, 445 511, 500 125, 503 255, 517 625,
  533 375, 551 375, <<38>>, 1 515 000, 1 521 060, 1 537 500,
  1 543 650, 1 593 750, 1 650 000, 1 656 600, 1 666 500, 1 691 250}
```

We can now look at the multiplicities of the inverse of ϕ .

```
Union[
  Table[Length[Reduce[n > 0 && EulerPhi[n] == k, n, Integers]], {k, 500}]]
```

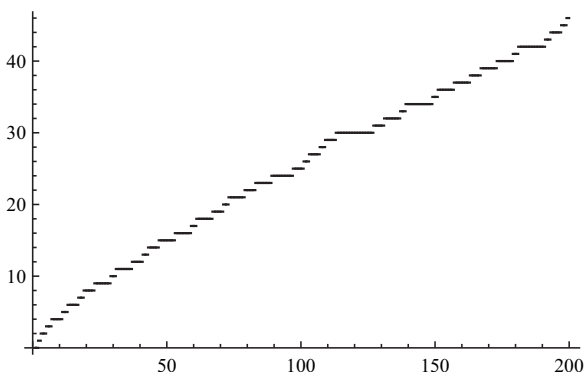
```
{0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
 13, 16, 17, 18, 19, 21, 25, 27, 28, 31, 34, 37}
```

We see that all multiplicities up to 13 show up, except for 1 (e.g., there are 13 numbers with ϕ -value equal to 396). This data relates to two old problems of number theory. The first is Carmichael's Conjecture, which asserts that 1 does not show up as a multiplicity; that is, if $\phi(n) = k$ then there is another integer with the same ϕ -value. This problem is unsolved. The second problem is due to Sierpiński, who asked whether every integer other than 1 does show up as a multiplicity. This problem was solved in 1998 by Kevin Ford [For].

2.2 Where the Primes Are

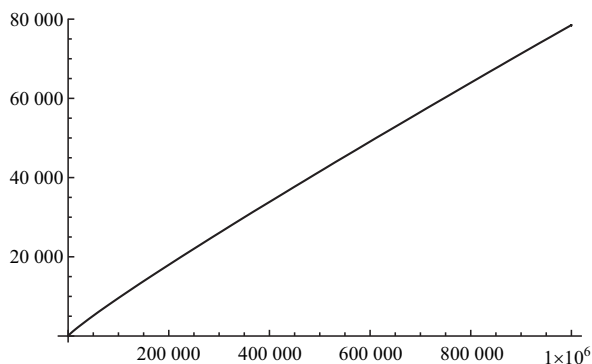
We can get a sense of where the primes are, the places they might cluster at or be absent from, by looking at a graph of $\pi(x)$, which gives the number of primes less than or equal to x .

```
Plot[PrimePi[x], {x, 1, 200},
  PlotStyle -> {Thickness[0.004], Black}, Exclusions -> Range[200]]
```



On a larger scale the piecewise nature of the graph disappears.

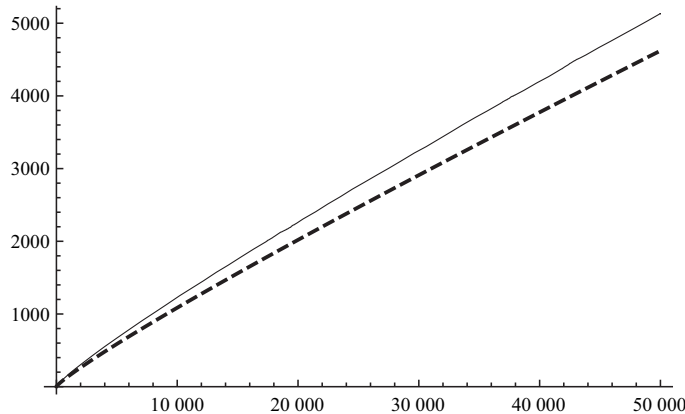
```
Plot[PrimePi[x], {x, 1, 10^6}, PlotStyle -> {Thickness[0.004], Black}]
```



Note how straight this curve is. Only a slight curvature visible at the lower end gives some indication of the nonlinearity of $\pi(x)$. The primes do become less dense as x increases (that is, $\pi(x)$ is concave down), but it is hard to see in a graph. The smoothness of this graph is noteworthy because the primes seem, at first glance, to show up randomly among the integers. For example, the interval $[114, 126]$ consists entirely of composites. But the smoothness of $\pi(x)$'s growth means that the prime distribution apparently obeys some guiding principles. As Don Zagier has observed [Zag], "The smoothness with which this curve climbs is one of the most astonishing facts in mathematics."

Of course, the next step is to find some functions that describe the prime growth rate. This is a well-studied problem, and we can compare the ideas of Legendre, Chebyshev, Gauss, and Riemann. The celebrated prime number theorem states that $\pi(x)$ is asymptotic to $x/\log x$. This means that the ratio of $\pi(x)$ to $x/\log x$ approaches 1 as x approaches infinity. (We use $\log$ to denote $\log_e$, to conform with *Mathematica*'s usage; that is, $\text{Log}[x]$ is $\log_e x$, while $\text{Log}[10, x]$ and $\text{Log}[2, x]$ are used for bases 10, 2, and so on.) Here is a view of the $x/\log x$ approximation (the dashed curve). Note how `PlotStyle` is set to be a list of two lists: the first is a list of instructions for the first function (the empty list in this case), and the second applies to the second function.

```
Plot[{PrimePi[x],  $\frac{x}{\text{Log}[x]}$ }, {x, 2, 50 000},
PlotStyle → {{Black}, {Black, Thickness[0.006], Dashed}}]
```



EXERCISE 2. Generate a table of $\pi(x)/(x/\log x)$ as x goes from 10 to 10^9 .

The fit of the preceding graph is clearly not ideal. In fact, there are much better approximations to $\pi(x)$. Legendre discovered empirically that $x/[(\log x) - 1.08366]$ is much better, although in fact this is true only in the short term; the best estimate

of this form in the long-term is $x / (\log(x) - 1)$ (this is sometimes called Chebyshev's estimate). Gauss, also working empirically (the prime number theorem was not proved until 1896), found that the following integral is an excellent approximation:

$$\int_2^t \frac{1}{\log t} dt$$

This integral (taken from 0 to t ; the singularity at $t = 1$ is easily dealt with) is called the *logarithmic integral* of x , usually denoted by $\text{li}(x)$ (*Mathematica* uses `LogIntegral[x]`). And Riemann's approximation is the most sophisticated:

$$R(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n} \text{li}(x^{1/n}),$$

where μ denotes the Möbius function ($\mu(n) = 0$ unless $n = p_1 p_2 \cdots p_r$, in which case $\mu(n) = (-1)^r$). Let's see how these three functions compare as approximations to $\pi(x)$.

It is simple enough to translate Legendre's and Gauss's functions into *Mathematica*. Moreover, because `Log`, `LogIntegral`, and the usual arithmetic functions are `Listable`, we can apply them to lists and get a list of values in return. For Riemann's function, however, we will use an alternative formulation. It is known that $R(x)$ equals the following infinite series

$$R(x) = 1 + \sum_{m=1}^{\infty} \frac{(\log x)^m}{m! m \zeta(m+1)},$$

where ζ is the Riemann ζ -function, discussed in much more detail in Chapter 20. For now, we need know only that it is built in as `Zeta`.

The ideas of the following implementation are due to Ilan Vardi. We predefine a list of 200 values, which will be large enough to handle inputs in the range of interest. Then `RiemannR` forms a dot product in order to get the 200th partial sum of the series above. Note that `RiemannR` will produce large symbolic output; we use adaptive precision when we want a certain number of decimal places. `RiemannR` is built into version 7, so only users of earlier versions should evaluate the following.

```
RiemannRData = 1. / (Range[200]! Range[200] Zeta[Range[200] + 1]);
RiemannR[x_?NumberQ] := 1 + Log[x]Range[200] . RiemannRData;
```

Before going into visual comparisons of the various approximations, note that computing exact values of $\pi(x)$ is very difficult and `PrimePi` works only up to about a trillion. However, a few values beyond that are known, and the initialization group for this chapter contains an extension of `PrimePi` that works for all powers of 10 up to 10^{23} . So we can compare our three approximations, as well as $x/(\log(x)-1)$, to $\pi(10^{23})$ as follows. Note how well both Gauss's and Riemann's approximations do. The `Grid` command generates a nicely formatted table, and generally replaces `GridBox` constructions of previous versions.

```
h[z_] := Style[z, FontFamily -> "Times"];
x = 1023; Grid[Map[h, {
  {"Legendre", Round[ $\frac{x}{\log[x] - 1.08366}$ ]},
  {"Chebyshev", Round[ $\frac{x}{\log[x] - 1}$ ]},
  {"Gauss", Round[LogIntegral[x]]},
  {"Riemann", Round[RiemannR[x]]},
  {" $\pi(10^{23})$ ", PrimePi[x]}}, {2}], Dividers -> All,
  Background -> RGBColor[1., 1., 0.8]]
```

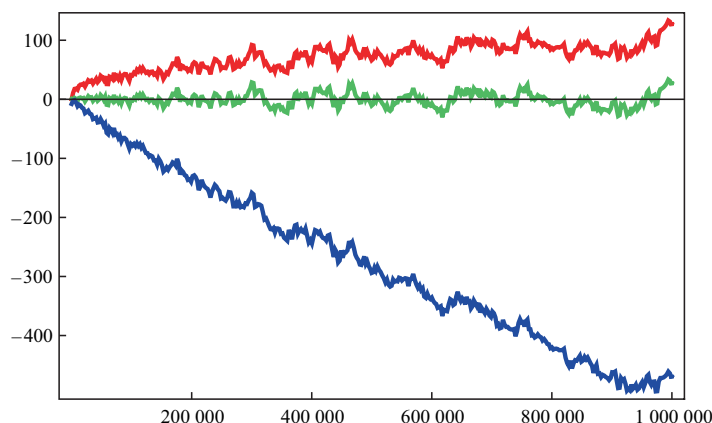
Legendre	1927 681 221 597 738 565 632
Chebyshev	1924 577 459 166 813 514 800
Gauss	1925 320 391 614 054 155 139
Riemann	1925 320 391 607 837 268 776
$\pi(10^{23})$	1925 320 391 606 803 968 923

Note that $\text{li}(x)$ agrees with $\pi(x)$ for about half its digits. A general statement of this form is equivalent to the Riemann hypothesis.

Now we are ready to generate some plots that compare all three approximations. Because Chebyshev's choice of constant, 1, is the correct one in the sense that it is the only c -value for which $x/(\log(x)-1)$ is asymptotic to $\pi(x)$ [Pin], we use Chebyshev and not Legendre in the visual comparisons to follow.

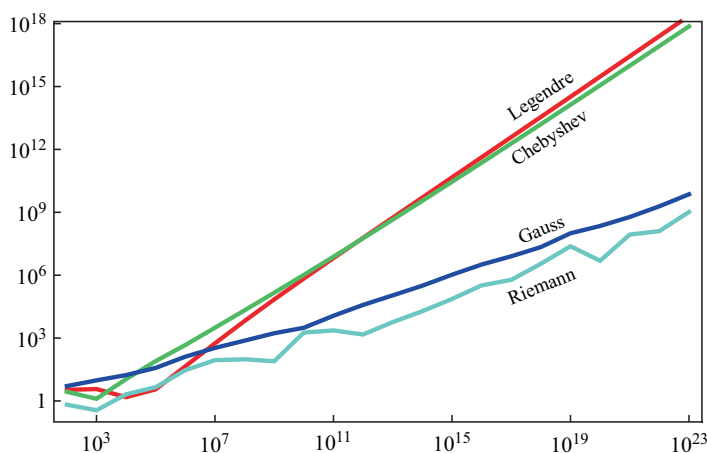
In other words, Legendre's choice is ad hoc and is incorrect when x is large. The `MaxRecursion -> 3` option is used to cut down the running time; remove it and it will take a little longer and yield more accurate graphs.

```
Plot[{LogIntegral[x] - PrimePi[x], RiemannR[x] - PrimePi[x],
   $\frac{x}{\log[x] - 1}$  - PrimePi[x]}, {x, 2, 106}, MaxRecursion -> 3, Frame -> True,
  PlotStyle -> {{Thick, Red}, {Thick, Green}, {Thick, Blue}}]
```



This image shows how good Riemann's approximation — its error curve straddles the x -axis — is. We have the data to carry this out to 10^{23} . The fourth argument to `Text` is used to tilt the labels. Note that Chebyshev becomes better than Legendre at around 10^{12} .

```
ListLinePlot[
  Table[{i, Log[10, Abs[#[10i] - PrimePi[10i]]]}, {i, 2, 23}] & /@
    {
      #
      Log[#] - 1.08366 &,
      #
      Log[#] - 1
    },
  LogIntegral, RiemannR}, Frame → True,
PlotStyle → {{Thick, Red}, {Thick, Green},
  {Thick, Blue}, {Thick, Cyan}},
FrameTicks → {{Table[{i, "10"i}, {i, 0, 18, 3}], None},
  {Table[{i, "10"i}, {i, 3, 23, 4}], None}},
Axes → None, PlotRange → {-1, 18.1},
Epilog → {Text["Gauss", {18, 8.2}, {0, 0}, {1, 0.4}],
  Text["Riemann", {18, 5.5}, {0, 0}, {1, 0.4}],
  Text["Legendre", {18, 14.5}, {0, 0}, {1, 0.65}],
  Text["Chebyshev", {18.3, 12.5}, {0, 0}, {1, 0.65}]}
```



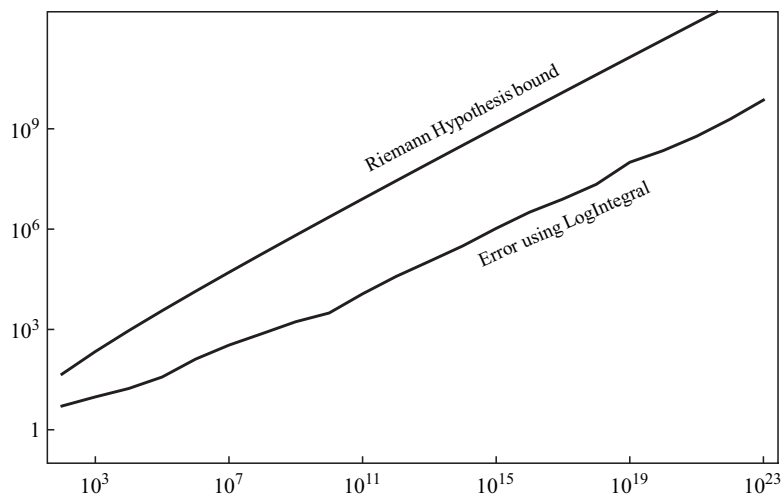
Gauss's approximation is worse than Chebyshev's for a while, but beyond 100 000 it is far superior. Note also the phenomenal accuracy of Riemann's $R(x)$: it differs from $\pi(x)$ by no more than 966 for integers up to 1 billion (this is an estimate; 966 occurs for 905 055 690; thanks to David Baugh for spotting an error here in the second edition). Look at how close it is for x equal to 1 billion.

```
{PrimePi[10^9], Round[RiemannR[10^9]]}
{50 847 534, 50 847 455}
```

Note that Riemann's function both overestimates and underestimates $\pi(x)$. It seems as if $\text{li}(x)$ only overestimates. But our computations are in sharp contrast to the spectacular result of Littlewood, which states that $\text{li}(x) - \pi(x)$ is not always positive; in fact, it crosses the x -axis infinitely often. The first crossing is called the *Skewes number*, after S. Skewes, who proved that it was less than $10^{(10^{34})}$. It is now known that the Skewes number is less than 10^{371} (due to H. te Riele [teR]; for a further discussion of the Skewes number see [Boa]). This remarkable phenomenon — that millions of hours of computation might lead to overwhelming evidence in favor of a conclusion that is, in fact, false — is a striking warning against basing conclusions solely on numerical evidence.

The Riemann hypothesis, a conjecture about the zeros of the complex function ζ that is considered by many to be the most important unsolved problem in mathematics, is equivalent to the assertion that for some constant c , $|\text{li}(x) - \pi(x)| \leq c \sqrt{x} \log x$. The ζ -function and its connection to the distribution of primes is discussed in more detail in Chapter 20. The image that follows generates a $\log_{10}$ plot that compares the two sides of this inequality when $c = 1$. The evidence looks good, but the Skewes phenomenon is always hovering in the background to remind us of the potential danger of deducing too much from computations involving primes.

```
ListLinePlot[{
  Table[{i, Log[10, LogIntegral[10^i] - PrimePi[10^i]]}, {i, 2, 23}],
  Table[{i, Log[10, Log[10^i] Sqrt[10^i]]}, {i, 2, 23}],
  Frame -> True, PlotStyle -> {{Thickness[0.004], Black}},
  FrameTicks -> {{Table[{i, "10^i"}, {i, 0, 9, 3}], None},
    {Table[{i, "10^i"}, {i, 3, 23, 4}], None}},
  Axes -> None, PlotRange -> {-1, 12.5}, Epilog ->
    {Text["Error using LogIntegral", {17, 6.1}, {0, 0}, {1, 0.44}],
     Text["Riemann Hypothesis bound", {14, 9.3}, {0, 0}, {1, 0.51}]}
```



2.3 The Prime Number Race

The prime number theorem has an extension that explains the growth of the sequence of primes in the congruence classes modulo some integer. Let $\pi_n(x, m)$ be the number of primes p less than or equal to x such that $p \equiv m \pmod{n}$. Then the famous theorem of Dirichlet on primes in arithmetic progressions guarantees that each congruence class contains infinitely many primes (provided $\gcd(m, n) = 1$); that is, each function $\pi_n(x, m)$ approaches infinity as x approaches infinity. Moreover, the aforementioned extension to the prime number theorem states that the $\phi(n)$ classes are uniformly distributed.

One of my favorite illustrations of the power of *Mathematica*'s high-level functions is the visualization of the prime number race in the mod-4 case. In this case every prime (except 2) falls into either the 1-class or the 3-class modulo 4. First we look at the first n primes, where n is 50.

Prime[Range[50]]

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107,
 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167,
 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229}
```

Reducing modulo 4, and using a third argument to give the start of the residue classes used, gives us ± 1 s (we eliminate the prime 2 here).

Mod[Prime[Range[2, 50]], 4, -1]

```
{-1, 1, -1, -1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1,
 -1, 1, -1, -1, 1, -1, -1, 1, 1, 1, -1, -1, 1, 1, -1, -1, 1,
 -1, 1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1, -1, -1, -1, 1}
```

Now, in order to see how the race is progressing, we need only look at the partial sums of this sequence. This is easily done with `Accumulate` (in earlier versions one would use `FoldList[Plus, 0, s]`).

```
Accumulate[{a, b, c}]
```

```
{a, a + b, a + b + c}
```

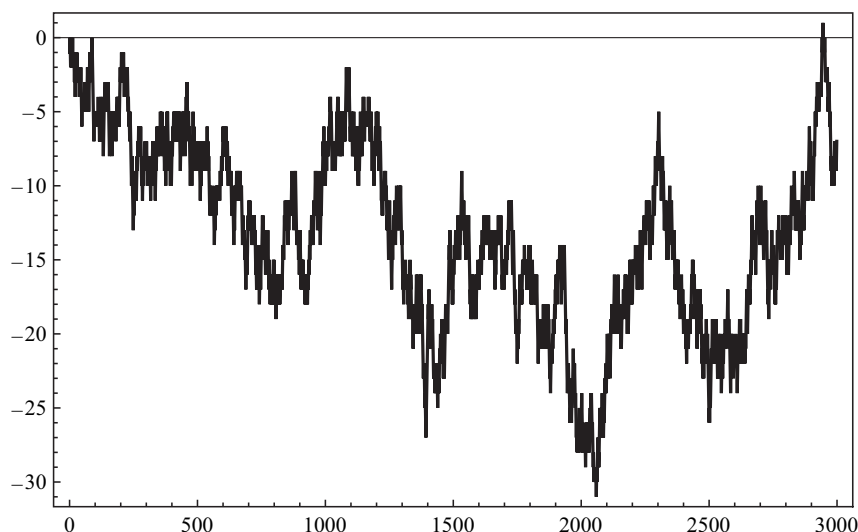
So we can watch the race unfold.

```
Accumulate[Mod[Prime[Range[2, 50]], 4, -1]]
```

```
{-1, 0, -1, -2, -1, 0, -1, -2, -1, -2, -1, 0, -1, -2, -1, -2, -1,  
-2, -3, -2, -3, -4, -3, -2, -1, -2, -3, -2, -1, -2, -3, -2, -3,  
-2, -3, -2, -3, -4, -3, -4, -3, -4, -3, -2, -3, -4, -5, -6, -5}
```

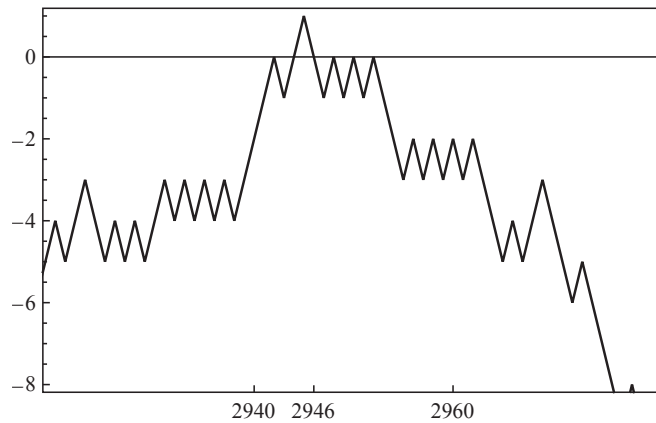
Of course it is nice to get graphic output, and that is easily done with `ListLinePlot`. At this point we increase 50 to 3000.

```
race = ListLinePlot[Accumulate[Mod[Prime[Range[2, 3000]], 4, -1]],  
Frame → True, Axes → False, GridLines → {{}, {0}}]
```



It is remarkable that so much information can be computed and displayed using hardly any code at all. And what have we learned about the prime number race? For one thing, the $4k-1$ primes, while they do sprint out to an early lead, are eventually caught by the $4k+1$ primes somewhere near the 2900th prime. We can find the exact place where the lead changes as follows.

```
Show[race, PlotRange → {{2920, 2980}, {-8, 1}},  
FrameTicks → {{Automatic, None}, {{2940, 2946, 2960}, None}}]
```



We see that the lead changes hands at the 2947th prime (because 2 was omitted in the preceding computations).

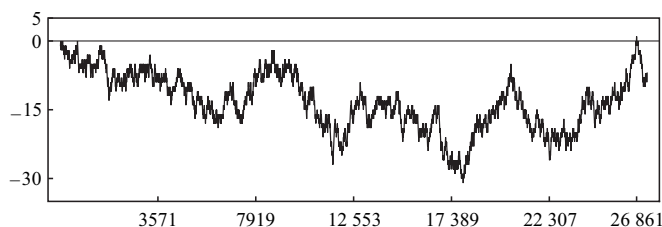
```
Prime[2947]
```

```
26 863
```

It is noteworthy that this first lead change was not even discovered until 1958. However, Hardy and Littlewood proved in 1914 that the lead necessarily changes hands infinitely often.

These plots look better if we tweak a few options, especially the ticks option, which we can use to put the actual primes, as opposed to their indices, at the tick marks.

```
ListLinePlot[Accumulate[Mod[Prime[Range[2, 3000]], 4, -1]],
  Frame → True, Axes → False, GridLines → {{}, {0}},
  FrameTicks → {{{-30, -15, 0, 5}, {}}, {Append[Table[{n - 1, Prime[n]},
    {n, 500, 2500, 500}], {2945, Prime[2946]}], {}},
  AspectRatio → 0.3, PlotRange → {-35, 5}]
```



One can now go further: the next lead change is at the prime 616 843.

One might wish to look at the prime number race with respect to other bases. Standard notation defines $\pi_n(x, a)$ to be the number of primes less than or equal to x that are congruent to $n \pmod{a}$. We can define this as follows, using a pattern query so that we can use Count, which counts the occurrence of a pattern. A more elementary approach would use `Length[Select[list, Mod[#, n] == a &]]`.

```

PrimePiMod[x_?NumberQ, n_Integer, a_Integer] :=
  Count[Prime[Range[PrimePi[x]]], _?(Mod[#, n] == a &)];
{PrimePi[100], PrimePiMod[100, 4, 1], PrimePiMod[100, 4, 3]}
{25, 11, 13}

```

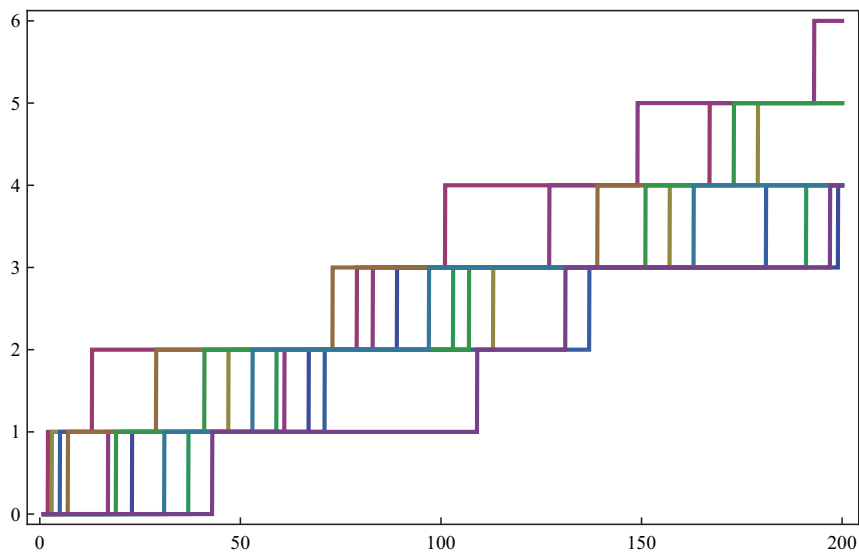
The discrepancy ($11 + 13 < 25$) is because of that odd prime, 2.

The approach we have taken here is not terribly efficient in terms of generating lots of data, since we have to repeatedly compute the same set of primes. We can use PrimePiMod to generate some small graphics, but the graph is difficult to decipher. Evaluate is used only to make the colors different, as explained in §1.4.

```

Plot[Evaluate[Table[PrimePiMod[x, 11, i], {i, 10}]],
  {x, 1, 200}, PlotStyle -> Thick]

```



So here is a function of the same name that takes arguments x and n and returns a data set that consists of a list of lists, one for each congruence class. The auxiliary function AttachPosns attaches the positions to the entries in the list.

```

AttachPosns[m_] := Transpose[{m, Range[Length[m]]}]
PrimePiMod[x_, n_] := AttachPosns /@
  Table[Select[Prime[Range[PrimePi[x]]], Mod[#, n] == i &], {i, n - 1}]

```

An example will clarify what this does.

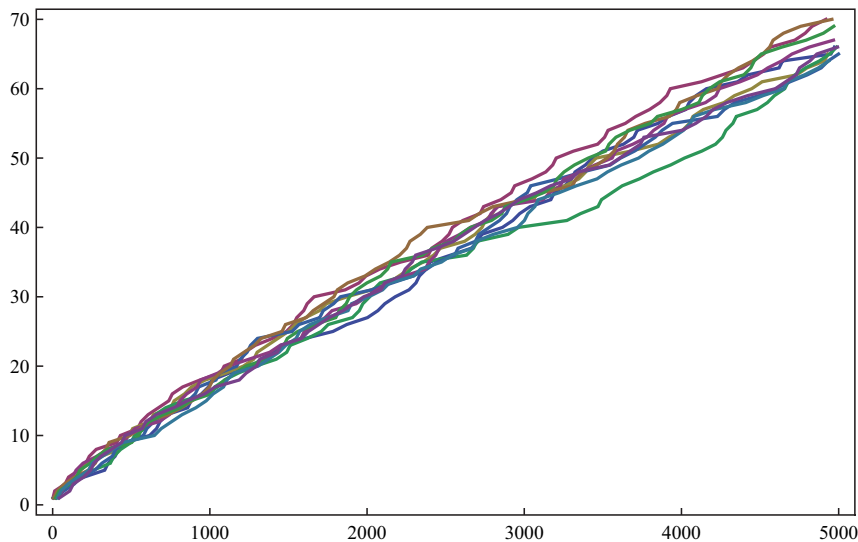
```

PrimePiMod[100, 5]
{{{11, 1}, {31, 2}, {41, 3}, {61, 4}, {71, 5}},
 {{2, 1}, {7, 2}, {17, 3}, {37, 4}, {47, 5}, {67, 6}, {97, 7}},
 {{3, 1}, {13, 2}, {23, 3}, {43, 4}, {53, 5}, {73, 6}, {83, 7}},
 {{19, 1}, {29, 2}, {59, 3}, {79, 4}, {89, 5}}}

```

The first list consists of the primes congruent to 1 (mod 5), together with their positions in the list, while the second list is the same for 2 (mod 5), and so on. So this is a discrete version of the true $\pi_n(x, a)$, which is a step function. We can now create a visual image of the mod-11 prime number race, using `ListLinePlot` to plot all the lists.

```
ListLinePlot[PrimePiMod[5000, 11],
  PlotStyle → Thickness[0.004], Frame → True]
```



2.4 Euclid and Fibonacci

As the reader no doubt knows, Euclid proved that there are infinitely many prime numbers by considering the sum of 1 and the product of the first n primes and concluding that this number either is prime or has a prime factor greater than the first n primes. To illustrate the use of recursion, let us examine some of the numbers that arise in this proof. We'll use `PrimeProduct[n]` for the product of the first n primes, calling the next integer a *Euclid number*. It is easy to program this recursively, but some care is necessary.

```
PrimeProduct[n_] := PrimeProduct[n - 1] Prime[n]
PrimeProduct[1] = 2;
Table[PrimeProduct[n] + 1, {n, 20}]
```

```
{3, 7, 31, 211, 2311, 30031, 510511, 9699691, 223092871,
 6469693231, 200560490131, 7420738134811, 304250263527211,
13082761331670031, 614889782588491411, 32589158477190044731,
1922760350154212639071, 117288381359406970983271,
7858321551080267055879091, 557940830126698960967415391}
```

This works, but each entry in the table requires the recomputation of all preceding values, since they have not been saved in any way. This is very inefficient. To see this happening, we modify `PrimeProduct` using `nCount` to record the values it sees.

```
PrimeProduct[n_] := (AppendTo[nCount, n]; PrimeProduct[n - 1] Prime[n])
PrimeProduct[1] = 2;

nCount = {};
Table[PrimeProduct[n] + 1, {n, 8}];
nCount
{2, 3, 2, 4, 3, 2, 5, 4, 3, 2, 6, 5,
 4, 3, 2, 7, 6, 5, 4, 3, 2, 8, 7, 6, 5, 4, 3, 2}
```

Note how much longer the list of *ns* is than is actually necessary for the computation. We can avoid this problem by caching the values as they are computed. The definition of `PrimeProduct` contains the explicit value of `PrimeProduct[1]` as a base for the recursion. We wish to add all new values to this list as they are computed, since *Mathematica* will scan this list before applying the general rule. There is an elegant way to do this. First, we clear out the old definition.

```
Clear[PrimeProduct]
PrimeProduct[n_] := PrimeProduct[n] = PrimeProduct[n - 1] Prime[n]
PrimeProduct[1] = 2;
```

The phrase `PrimeProduct[n] = PrimeProduct[n - 1] Prime[n]` causes *Mathematica* to store the values as they are computed, which is just what we want. We can see this by computing `PrimeProduct[5]` and then examining the internal representation of `PrimeProduct`.

```
PrimeProduct[5]
```

```
2310
```

```
? PrimeProduct
```

```
Global`PrimeProduct
```

```
PrimeProduct[1] = 2
```

```
PrimeProduct[2] = 6
```

```
PrimeProduct[3] = 30
```

```
PrimeProduct[4] = 210
```

```
PrimeProduct[5] = 2310
```

```
PrimeProduct[n_] := PrimeProduct[n] = PrimeProduct[n - 1] Prime[n]
```

In short, we had *Mathematica* teach itself the values as it computed them. Caching is essential with a recursive approach to the Fibonacci numbers, since otherwise there is exponential blowup that renders impossible computing something as conceptually simple as the 100th Fibonacci number. Here is the naive approach to the 22nd Fibonacci number (the bar | refers to alternatives among patterns; thus 0 | 1 can be read as "0 or 1").

```
Fib[0 | 1] = 1;
Fib[n_] := Fib[n - 1] + Fib[n - 2]
Timing[Fib[26]]
{0.519756, 196 418}
```

Now we use caching and see a huge speedup.

```
Clear[Fib];
Fib[0 | 1] = 1;
Fib[n_] := Fib[n] = Fib[n - 1] + Fib[n - 2]
Timing[Fib[26]]
{0.000278, 196 418}
```

Of course, the fastest way to get Fibonacci numbers is to use the built-in `Fibonacci` function, discussed in Chapter 1.

Returning to Euclid's numbers, let us examine whether they are more often prime or composite.

```
Attributes[PrimeProduct] = Listable;
PrimeQ[PrimeProduct[Range[20]] + 1]
{True, True, True, True, True, False, False, False, False, False, True,
 False, False, False, False, False, False, False, False}
```

We can generate the indices of the prime Euclid numbers as follows.

```
Select[Range[100], PrimeQ[PrimeProduct[#] + 1] &]
{1, 2, 3, 4, 5, 11, 75}
```

Is this last output definitely correct? Almost surely. Recall that if `PrimeQ` thinks a number is composite, then that number has failed a basic test and is definitely composite. But `PrimeQ` is definitive only up to 10^{16} . Since the 75th Euclid number has 154 digits, we do not have a proof of primality in this case. However, there are several certification procedures available in *Mathematica*, and they are capable of providing a proof in this instance (see §21.3).

The prime Euclid numbers quickly become rare, and it is not known whether infinitely many exist (see [Rib]).

EXERCISE 3. Find the next prime Euclid number. Carry out similar investigations on the Fermat numbers, $2^{(2^n)} + 1$. It was once thought that they were all prime. It is now thought that except for the first few, they are all composite! See [Guy, Rib].

2.5 Strong Pseudoprimes

To give the reader an idea of some of the stronger pseudoprime tests, and also because it is an interesting programming exercise, we discuss the notion of strong pseudoprimes. As pointed out in section 1, $b^{n-1} \equiv 1 \pmod{n}$ if n is prime and $1 < b < n$. Let us consider the path one might take to this 1. If the odd part of $n - 1$ is m (that is, $n - 1 = m \cdot 2^r$ where m is odd), then one can first compute $b^m \pmod{n}$ and then square it r times. Let us call the resulting sequence a *b-sequence*. Here are various forms the *b-sequence* might take, where $*$ denotes an integer that is not $\pm 1 \pmod{n}$.

b^m	b^{2m}	b^{4m}	b^{8m}	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	b^{n-1}	
+1	+1	+1	$\cdot$	$\cdot$	$\cdot$	+1	+1	+1	+1	Type 1: b^m is +1
*	*	*	$\cdot$	$\cdot$	$\cdot$	*	-1	+1	+1	Type 1: $b^m 2^k$ is -1 for some $k < r$
*	*	*	$\cdot$	$\cdot$	$\cdot$	*	+1	+1	+1	Type 2: composite; an entry different from ± 1 squares to 1
*	*	*	$\cdot$	$\cdot$	$\cdot$	*	*	*	*	Type 2: composite; $b^m 2^k$ is never ± 1
*	*	*	$\cdot$	$\cdot$	$\cdot$	*	*	*	-1	Type 2: composite; b^{n-1} is -1

Table 2.1 Different types of *b-sequences* that an integer n might yield, where an $*$ denotes an integer that is not $\pm 1 \pmod{n}$.

Now, it turns out that the bottom three possibilities (type 2) cannot occur if n is truly prime. The reason for this is that, if n is prime, there are only two square roots of +1: +1 and -1. Of course, we are speaking here about reduced residues, and -1 is really synonymous with $n - 1$. The proof of this is very nice. Suppose p is prime and x squares to -1 $\pmod{p}$. Then p divides $x^2 - 1$, which means, because p is prime, that p divides $x - 1$ or $x + 1$, so $x \equiv \pm 1 \pmod{p}$, as claimed. So we define a *b-strong pseudoprime* to be an odd composite integer n whose *b-sequence* is of type 1. Note that we assume throughout this discussion that n is odd.

This argument means that if the *b-sequence* for n has type 2, n is definitely not prime. How good a discriminator is this? Let's find out. First we make an auxiliary function to compute the odd part of an integer and the power of 2 in the even part. Aside: it is not hard to show that if n is odd then $b^{n-1} \pmod{n}$ cannot be -1 (see [BW, ex. 4.18]); thus the last line of the table is not really necessary.

```

OddPart[n_] := With[{p = IntegerExponent[n, 2]}, { $\frac{n}{2^p}$ , p}];
Map[OddPart, {100, 101, 102}]
{{25, 2}, {101, 0}, {51, 1}}

```

Now we look at some b -sequences. Recall that `NestList` returns the list of iterates of a function. In the code that follows, we iterate the mod- n squaring function the number of times specified by s . And for legibility, we replace $n - 1$ by -1 by using a third argument to `Mod`.

```

spspSequence[b_, n_] := Module[{m, s}, {m, s} = OddPart[n - 1];
  NestList[Mod[#^2, n, -1] &, Mod[PowerMod[b, m, n], n, -1], s];

```

When we look at the 1000th prime (7919) or 10000th prime, we see the proper type -1 behavior.

```

spspSequence[2, 7919]
{1, 1}

spspSequence[2, Prime[10 000]]
{36 639, -1, 1, 1}

```

The 2-pseudoprime 341 is quickly unmasked by this test, since the 2-sequence has a 1 preceded by a 32.

```

spspSequence[2, 341]
{32, 1, 1}

```

Here is complete code to recognize b -strong pseudoprimes. It is sufficient to check that the b -sequence either begins with $+1$ or has a -1 anywhere, for such a -1 guarantees that the form of the sequence is typical of primes. We treat $n = 1$ as a special case, using the case-restrictor `/;` to restrict the general case to odd $n > 1$. Note that n is assumed to be odd, and that is why we can ignore the last line of Table 2.1, which cannot occur.

```

StrongPseudoprimeQ[b_][n_] := Module[{bSeq = spspSequence[b, n]},
  (bSeq[[1]] == 1 || MemberQ[bSeq, -1]) && ! PrimeQ[n]] /; n > 1 && OddQ[n]
StrongPseudoprimeQ[_][1] = False;

```

We can now select all the 2-spmps below 10000, where the term *2-spmp* denotes 2-strong pseudoprimes.

```

Select[Range[10 000], StrongPseudoprimeQ[2]]
{2047, 3277, 4033, 4681, 8321}

```

Well, perfection remains tantalizingly out of reach, but the 2-spsp test does have an impressive success rate of 99.95% up to 10000. There are several 3-spsps in this interval, but if we run the test on 2 and 3 together, it is formidable indeed. The first bad integer for this double test is 1 373 653.

```
StrongPseudoprimeQ[3][1 373 653]
```

```
True
```

```
FactorInteger[1 373 653]
```

```
{{829, 1}, {1657, 1}}
```

Indeed, if we made a test that consisted of combining the base- b strong pseudo-prime tests for $b = 2, 3, 5, 7, 11$ then that test would be pretty efficient, as there are no counterexamples less than $25 \cdot 10^9$. The composite integer 3 215 031 751 is the first integer that is a b -spsp for $b = 2, 3, 5$, and 7. It is caught by $b = 11$.

```
Table[StrongPseudoprimeQ[Prime[i]][3 215 031 751], {i, 1, 10}]
```

```
{True, True, True, True, False, False, False, True, False, False}
```

```
FactorInteger[3 215 031 751]
```

```
{{151, 1}, {751, 1}, {28 351, 1}}
```

A record of sorts was set by the composite number

$$18215745452589259639 \cdot 4337082250616490391 \cdot 867416450123298079$$

found by D. Bleichenbacher in 1993 [Ble]. It is a b -spsp for all $b \leq 100$.

```
n = 18 215 745 452 589 259 639 ×
```

```
4 337 082 250 616 490 391 × 867 416 450 123 298 079;
```

```
Select[Range[101], ! StrongPseudoprimeQ[#][n] &]
```

```
{101}
```

If one is carrying out such a search and does not know where the target lives, or even if it exists, one would use a Do-loop, set to break when the example is found, as follows.

```
Do[If[! StrongPseudoprimeQ[i][n], Print[i]; Break[]], {i, 200}]
```

```
101
```

It is strongly suspected (indeed, it follows from the unproved extended Riemann hypothesis) that checking all bases up to $2(\log n)^2$ is a true test of primality, and such a test would run in polynomial time. A spectacular breakthrough occurred in

2002 when it was proved by Agrawal, Kayal, and Saxena that primality can indeed be determined in polynomial time [Wei2]; but their algorithm is nowhere near as fast as a combination of strong pseudoprime tests.

Now we have a better understanding of *Mathematica's* PrimeQ function. It combines a 2-spsp test, a 3-spsp test, and a Lucas pseudoprime test (a detailed discussion of Lucas pseudoprimes is in [BW]). There is no known counterexample to the assertion that these three tests pass the primes and only the primes, and it has been checked by D. Bleichenbacher that there is no counterexample under 10^{16} .

Mathematica® in Action

Problem Solving Through Visualization and Computation

Wagon, S.

2010, XI, 580 p. With online files/update., Softcover

ISBN: 978-0-387-75366-9