Prime Counting

How many prime numbers are there up to a given n? In the "Prime sum" example we wrote a function to compute the prime counting function, $\pi(n)$, which gives the exact answer. It works by checking the factorization of all the numbers up to n, so it would be too slow using Calc-50 for $n = 10^9$ or $n = 10^{20}$.

For really big values of n, we might be satisfied with a good approximation of $\pi(n)$. We will compare three formulas that approximate $\pi(n)$.

The obvious place to start (obvious if you are a number theorist, anyway), is the Prime Number Theorem:

$$\pi(x) \approx \frac{x}{\ln(x)}$$

The second is the logarithmic integral $\mathrm{li}(x)$ from screen 4:

$$\pi(x) \approx \mathrm{li}(x) = \int_0^x \frac{1}{\ln(t)}\, dt$$

The third is Riemann's prime counting function:

$$\pi(x) \approx R(x) = \sum_{k=1}^{\infty} \frac{\mu(k)}{k}\, \mathrm{li}(x^{1/k})$$

where $\mu(k)$ is the Möbius function from number theory.

The only one that is complicated for us is $R(x)$. While $\mathrm{li}(x)$ is built into the calculator, $\mu(k)$ is not. For $\mu(k)$, a positive integer k is called *square-free* if there is no prime number p for which k/p^2 is an integer.

$\mu(k) = \ \ 1$ if k is square-free and has an even number of prime factors.

$\mu(k) = -1$ if k is square-free and has an odd number of prime factors,

$\mu(k) = \ \ 0$ if k has a squared (or higher power) prime factor.

Here are the first few values of $\mu(k)$:

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mu(k)$	1	−1	−1	0	−1	1	−1	0	0	1	−1	0	−1	1	1

To create a function for $\mu(k)$, the fctr key on screen 6 can find the prime factors, but we won't know in advance how many factors a given k has. Here is one possible algorithm.

1. Store k in register 2, initialize register 0 to -1, apply fctr to k, and store the factor in register 1.
2. Repeat step 3 ten times using the sum key.
3. Recall registers 2 and 1, divide to remove the previous factor and store the result in register 2,
   if register 2 is 1, don't change register 0,
   else fctr register 2 to get the next prime factor of k and store it in register 3,
      if register 3 equals register 1, set register 0 to 0,
      else store register 3 in register 1 and multiply register 0 by -1.

4. After the loop, recall register 0 as $\mu(k)$.

The idea is to look at the prime factors one at a time. If the next factor ever equals the previous one (in register 1), set $\mu(k)$ to zero. Otherwise multiply register 0 by -1 for each factor found, so we have -1 after an odd number of factors and $+1$ after an even number of factors.

If k has fewer than 11 prime factors, we will divide them all out and reduce the number to 1. For the remaining trips through the loop, register 2 will be 1, so register 0 doesn't change.

Functions f1 through f6 perform the algorithm above. The select (sel) function is used as an if statement.

f1(k) returns $\mu(k)$. It initializes, uses the sum function to call f2 10 times, then recalls register 0 as $\mu(k)$.

f2 divides out the previous factor and saves the result in register 3. If that factor is 1 call f3, else call f4.

f3 puts 1 into registers 1 and 2, and doesn't change register 0.

f4 gets the next factor of register 2, puts it into register 3.
If it is the same as the previous factor in register 1 call f5, else call f6.

f5 sets register 0 to zero when a repeat factor is found

f6 moves register 3 to register 1 and multiplies register 0 by -1

f1: 2, sto, -1, enter, 0, sto, roll, fctr, 1, sto, 7, func, 1, enter, 10, enter, 1, enter, 2, sum 0, rcl

f2: 2, rcl, 1, rcl, /, 2, sto, 7, func, 1, enter, 1, enter, 3, sel

f3: 1, enter, 1, sto, 2, sto

f4: 2, rcl, fctr, 3, sto, 7, func, 1, rcl, 1, rcl, 5, sel

f5: 0, enter, 0, sto

f6: 3, rcl, 1, sto, 0, rcl, chs, 0, sto

A compromise we made here comes from the fact that Calc-50 isn't designed for complicated programming, so it doesn't have a "while" loop. We need to keep factoring k until all its prime factors have been found, but the program above will stop too soon if k has more than 11 prime factors.

If there are any repeated prime factors in the first 11, then we know that $\mu(k) = 0$, so stopping too soon still gets the right answer. The smallest k for which our function returns the wrong answer is the product of the first 12 primes, $k = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37$ =7,420,738,134,810. That is much larger than any k we will need for the $R(x)$ sums.

Now define f7(n, x) as the n^{th} partial sum of $R(x)$. f7 will store x in register 7 and sum the series from 2 to n, then add the $k = 1$ term by hand since $\mu(1) = 1$ is a special case that the f1 function doesn't handle. f8 will compute the k^{th} term of the $R(x)$ series.

f7: 7, func, 7, sto, x↔y, 2, x↔y, 1, enter, 8, sum, 4, func, 7, rcl, li, +

f8: 1, func, 8, sto, 1, f_n, 8, rcl, /, 7, rcl, 8, rcl, 1/x, y^x, 4, func, li, $*$

How many terms will be needed for $R(x)$? Since we are approximating $\pi(x)$, which is an integer, getting R to one place after the decimal should be fine. We can see that about half of the non-zero values of $\mu(k)$ are negative by summing just $\mu(k)$ to 500, 1000, and 2000 terms.

7, func, 2, enter, 500, enter, 1, enter, 1, sum	-7
2, enter, 1000, enter, 1, enter, 1, sum	1
2, enter, 2000, enter, 1, enter, 1, sum	4

That means the R sum should converge roughly like an alternating series, where the error in a partial sum is about the size of the last term added. Check the size of the $k = 1003$ term (which has $\mu(1003) = 1$) for $x = 10^9$ and $x = 10^{20}$ by storing x in register 7 and evaluating f8.

e9, enter, 7, sto, 1003, enter, 8, f_n	-0.003
e20, enter, 7, sto, 1003, enter, 8, f_n	-0.002

This makes it appear that 1000 terms of the R series will be enough. Knowing that, if it turns out that our function f7 for R runs slower than we like, we could do some "code-tuning" on f1 by replacing the 10 upper limit of the sum function there by 4, since the product of the first 5 primes is more than 1000. That would make the R sum run about twice as fast.

Compare the three approximations to $\pi(x)$ for several large x's. Round them to the nearest integer. f7 has two inputs, n and x, so to get the sum of the first 1000 terms of the series for $R(10^9)$,

1000, enter, e9, enter, 7, f_n, -2, fix

Define f9(x) to compute the top 3 values in the x column below. e9, enter, 9, f_n, roll, roll shows first the $x/\ln(x)$ value, then li(x), then $R(x)$.

f9: 9, sto, 1000, xch, 7, f_n, 2, func, nint, 9, rcl, 4, func, li, 2, func, nint, 1, func, 9, rcl, enter, ln, /, 2, func, nint, 0, fix

$x =$	10^9	10^{12}	10^{15}	10^{20}
$x/\ln(x)$	48,254,942	36,191,206,825	28,952,965,460,217	2,171,472,409,516,259,138
li(x)	50,849,235	37,607,950,281	29,844,571,475,288	2,220,819,602,783,663,484
$R(x)$	50,847,455	37,607,910,542	29,844,570,495,887	2,220,819,602,556,027,015
$\pi(x)$	50,847,534	37,607,912,018	29,844,570,422,669	2,220,819,602,560,918,840

All three of these functions are good approximations to $\pi(x)$ in the sense that as $x \to \infty$, the limit of the quotient of one of the three functions divided by $\pi(x)$ tends to 1.

But for finite values of x, even as large as 10^{20}, li(x) and $R(x)$ give much more accurate approximations than the prime number theorem. The number of primes up to 10^{20} has 19 digits, with the prime number theorem giving 2 significant digits correct, li(x) giving 10 digits, and $R(x)$ giving 12 digits.