APL and the Search for the Ultimate Prime Number Program (part 1)

D. McCormick

For some time now I have made a hobby of finding prime numbers. Over the years I've written and collected a number of different methods for finding primes. My most recent program is in C but most of the preceding ones have been in APL.

For anyone who may not know: a prime number is a positive integer with exactly 2 integral divisors: itself and 1. Historically, primes have had little practical value; they have been of interest mainly to mathematicians, especially those concerned with number theory. However, the importance of primes is reflected in the statement of the fundamental theorem of arithmetic: **every positive whole number other than 1 is either prime or a product of a unique collection of primes.** (Parts of this essay are based on an article by Dr. Keith Devlin, *Factoring Fermat numbers*, in the 25 September 1986 issue of *New Scientist* magazine.)

Since the time of Euclid (a covert APL user), primes have remained fundamentally important to mathematicians and no one else. The past 10 years or so have seen an awakening of a more general interest in primes. Large prime numbers form the basis of the RSA Public Key System, a method for encrypting data very securely. The U.S. Defense Department considered this method to be sensitive information when it was first discovered and they attempted to restrict it in the interest of national security.

This new interest in large primes has given rise to some sophisticated methods for finding them. However, we will get to these later. Right now, let's look at some decidedly unsophisticated "brute force" approaches to finding primes.

Anyone who has ever written a prime number program in APL probably first came up with something similar to the following:

```
     ∇ R←PRIME0 N;CO;⎕IO
[01]    ⎕IO←1 ⋄ R← 2 3 ⋄ CO←5 ⋄ →(N>3)⍴GO ⋄
                          ←[(¯1+(R≤N)⍳0)↑R ⋄ →0
[02]   GO:R←R,(~0∊(((R≤⌈CO*0.5)⍳0)↑R)|CO)⍴CO
                                ⋄ →(N≥CO←CO+2)⍴GO
     ∇
```

Like many algorithms, this one starts with a few "seed" primes (the initial value of *R*.) We then search for successive primes (up to number *N*.) by setting a counter *CO*) and checking for primality using the "residue" function. Notice that we increment the counter by 2 each time since we know that even numbers greater than 2 are all non-prime.

Note further that the left argument to the residue function is all the primes up the the test number's square root. This is a reflection of the fundamental theorem ("*either* prime *or a product* of · · · primes.") Furthermore, we only check primes up to the test number's square root because any factor greater than the square root will have a corresponding one less than the square root; we need only find one factor to disqualify the test number as a prime.

The main problem with prime number generators in APL is that they are so numerically intensive that they run a long time for any great number of primes. In studying the above function, I realized that selecting primes up to the square root is quite time-consuming. Up to a certain point, it's faster to use too many primes for the left argument rather than select only those needed.

The following function incorporates these observations:

```
     ∇ R←PRIME1 N;C
[01]    →(N>⍴R←2,C←3)⍴GO ⋄ R←N↑R ⋄ →0
[02]   GO(→(N>⍴R←R,(~0∊R|C)⍴C←C+2)⍴GO
     ∇
```

Note too that *PRIME*1 construes its argument *N* differently than does *PRIME*0. Whereas *PRIME*0 finds primes up to *N* (inclusive), *PRIME*1 finds the first *N* primes. There is a well known formula to approximate the number of primes up to the number *N*:

```
     ∇ NP←NUMPRIMESTO N
[01]    NP←N÷⍟N
     ∇
```

This is an approximation which is less accurate for smaller values of *N* and more accurate for larger values *N*. I don't know the inverse of this function though the following approximates it:

```
     ∇ N←PRIMESTONUM NP
[01]    N←NP×(⍟NP)+(⍟⍟NP)+.1×⍟⍟⍟NP
     ∇
```

Can you come up with a better inverse function? This one falls short somewhere between 1E13 and 1E14.

P.O. Box 6821
FDR Station
New York, NY 10150
U.S.A.