SOME COMPUTATIONAL ALGORITHMS IN NUMBER THEORY

The following algorithms are given in pseudocode, i.e. in the format of a computer program, but without regard to the precise syntax of any particular programming language. If you are used to programming (in MAPLE, C++, etc) then you should find it reasonably straightforward to translate these into working programs.

We write `div` and `mod` for integer division and remainder of integer division: thus `a = (a div b)*b + (a mod b)` for natural numbers `a` and `b`.

1 The Extended Euclidean Algorithm

```
ALGORITHM eea

Input: integers a,b>0.
Output: [g,x,y] where g is the greatest common divisor of a,b
        and where g=xa+by.

x0:=1; x1:=0; y0:=0; y1:=1;
WHILE b<>0 DO
  q:=a div b; r:=a mod b;
  a:=b; b:=r;
  x:=x0-x1*q; x0:=x1; x1:-x;
  y:=y0-y1*q; y0:=y1; y1:-x;
END-WHILE
RETURN([g,x0,y0]);
STOP;
```

In the sequel we will assume that we have a subroutine `gcd` such that `gcd(a,b)` returns the greatest common divisor of a and b (without giving the corresponding x and y).

2 The Binary Powering Algorithm

```
ALGORITHM powermod

Input: integers a, k>=0, n>=2.
Output: a^k mod n.

c:=a; b:=1;
WHILE k>0 DO
  IF (k mod 2)=1 THEN k:=k-1; b:=b*c mod n END-IF
  c:=c*c mod n;
  k:=k div 2;
END-WHILE
RETURN(b);
STOP;
```

In the sequel we will call this as a subroutine: `powermod(a,k,n)` should return $a^k \bmod n$.

3 The Miller-Rabin Primality Test

```
ALGORITHM miller-rabin

Input: integers  n>=3, a with 2<=a<n.
Output: ‘Pass’ if n is either prime or
         a strong pseudoprime to base a;
        ‘Fail’ otherwise.
        [In some cases, a proper factor of n is also output.]

IF (n mod 2) = 0 THEN OUTPUT(‘Fail’); OUTPUT(2); STOP; END-IF;
IF gcd(a,n)>1 THEN OUTPUT(‘Fail’); OUTPUT(gcd(a,n)); STOP; END-IF;
m:=n-1; s:=0;
WHILE (m mod 2)=0 DO m:=m div 2; s:=s+1; END-WHILE;
b:=powermod(a,m,n);
IF b=1 THEN OUTPUT(‘Pass’); STOP; END-IF
IF b=n-1 THEN OUTPUT(‘Pass’); STOP; END-IF
FOR i=1 TO s-1 DO
  b1:=b*b mod n;
  IF b1=n-1 THEN OUTPUT(‘Pass’); STOP; END-IF;
  IF b1=1 THEN OUTPUT(‘Fail’); OUTPUT(gcd(b-1,n)); STOP; END-IF;
  b:=b1;;
END-FOR;
OUTPUT(‘Fail’);
IF b*b mod n = 1 THEN OUTPUT(gcd(b-1,n)); STOP; END-IF;
STOP;
```

4 Pollard's "$p-1$" Algorithm

```
ALGORITHM p-minus-1

Input: integers  n>=2.
Output: Either a proper factor of n, or 'Fail'.

a:=2; kmax:=1000;   [These parameters may be varied.]
FOR k=2 TO kmax DO
  a:=powermod(a,k,n);
  g:=gcd(a-1,n);
  IF g=n THEN OUTPUT('Fail'); STOP; END-IF;
  IF g>1 THEN OUTPUT(g); STOP; END-IF;
END-FOR
OUTPUT('Fail');
STOP
```

5 Pollard's "Rho" Algorithm

```
ALGORITHM rho

Input: integers  n>=2.
Output: Either a proper factor of n, or 'Fail'.

c:=1; x0:=2; imax:=1000;   [These parameters may be varied.]
x:=x0; y:=x0;
FOR i=1 TO imax DO
  x:=x^2+c mod n;
  y:=y^2+c mod n;
  y:=y^2+c mod n;
  g:=gcd(x-y,n);
  IF g=n THEN OUTPUT('Fail'); STOP; END-IF
  IF g>1 THEN OUTPUT(g); STOP; END-IF
END-FOR
OUTPUT('Fail');
STOP
```

Nigel Byott, October 2002