Universität Wien
Prof. Dr. D. Burde

Computeralgebra
— PARI —

---

1. Lucas-Lehmer Primzahltest. Der Lucas-Lehmer Test ist ein deterministischer Primzahltest, der entscheidet, ob eine Mersenne-Zahl $M_n = 2^n - 1$ für ein vorgelegtes $n > 2$ prim ist oder nicht. Wie wir aus der Vorlesung wissen, kann M_n nur dann prim sein, wenn n bereits prim ist. Sei also $p > 2$ prim. Das folgende Kriterium geht auf Lucas (1878) und Lehmer (1935) zurück. Sei $(u_i)_{i\geq 1}$ eine Folge natürlicher Zahlen, die wie folgt rekursiv definiert ist:

$$u_1 = 4$$
$$u_i = u_{i-1}^2 - 2, \quad i \geq 1$$

Dann ist M_p genau dann prim, wenn $u_{p-1} \equiv 0 \mod M_p$ ist.
Hier ist ein PARI Programm, das einen Algorithmus für den Lucas-Lehmer Test beschreibt:

```
lucas(p) =
{
  local(u,q); u=4; q=1<<p - 1;
  for(k=3,p, u = (sqr(u)-2)%q);
  u == 0
}
```

Man überlege sich, wie dieser Algorithmus funktioniert. Sei p ein Input. Dann wird eine Funktion lucas(p) definiert. Man zeige folgendes: wenn der Algorithmus dafür den Wert 1 ausgibt, so ist M_p prim. Gibt er 0 aus, so ist M_p zusammengesetzt. Man beachte, daß $\mathrm{sqr}(x) = x^2$ die Quadratfunktion ist, und nicht die Quadratwurzel $\mathrm{sqrt}(x) = \sqrt{x}$. Weiterhin ist $x << n = x \cdot 2^n$ für $n \geq 1$. Man bestimme mit diesem Programm, welche der folgenden Mersenne-Zahlen prim sind, und vergleiche die Laufzeit mit dem Befehl factor, factorint oder anderen Primzahltests:

$$M_{127},\ M_{227},\ M_{521},\ M_{541},\ M_{1277},\ M_{1279},\ M_{9941},\ M_{9973},\ M_{21701}$$

Die letzte Zahl hat immerhin schon 6533 Stellen.

2. Der AKS Primzahltest. Hier ist der Primzahltest von M. Agrawal, N. Kayal und N. Saxena in einer verbesserten Version von H.W. Lenstra und J.F. Voloch. Man versuche diesen Test z.B. mit $n = 628363443011$ und $n = 2222222222222222222222222223$.

```
\\ ---------------------------------------------------------------------------
\\ Hilfsroutine: isprimitiveroot testet ob n eine Primitivwurzel mod r ist
\\ ---------------------------------------------------------------------------

isprimitiveroot(n,r) =
{
    local (out,q);
```

```
    q = component(factor(r-1),1)~;
    out=1;
    for(i=1,length(q),out = out && (Mod(n,r)^((r-1)/q[i])!=1));
    out
}

\\ ------------------------------------------------------------------------
\\ Das ist die Funktion, die wir haben wollen
\\ ------------------------------------------------------------------------

AKS(n) =
{
    local(t,r,q,s,X,a,k,d,p,out);

\\ Input
\\      n : eine natuerliche Zahl, die man testen moechte
\\ Output
\\      0 : n ist zusammengesetzt
\\      1 : n ist prim
\\ ------------------------------------------------------------------------
\\ Step 1: testing for perfect power using 2-adic Newton as provided by PARI's
\\         "polrootspadic" command
\\ ------------------------------------------------------------------------
    if( issquare(n),
        return(0),
\\  else
        m = floor(log(n)/log(3));
        forprime( k=3, m,
            d = ceil(log(n)/log(2)/k)+1;
            p = component(lift(
                Mod(polrootspadic(x^k-n,2,d+1),2^(d+1))),1);
            if( n==p^k, return(0) );
        );
    );

\\ ------------------------------------------------------------------------
\\ Step 2: Looking for admissible (r,s) as in Theorem 2 of Daniel Bernstein:
\\         An Exposition of the Argrawal-Kayal-Saxena Primality-Proving
\\         Theorem, http://cr.yp.to/papers/aks.ps .
\\         The number s is essential cut to half by an idea of Jos Felipe
\\         Voloch: ftp://ftp.ma.utexas.edu/pub/papers/voloch/aks.pdf.
\\         The admissible pair is chosen in a way as to optimize run-time for
\\         Step 3 in a simplified asymptotic model.
\\ ------------------------------------------------------------------------
\\ t = 17.4929;     \\ asymptotic optimal for FFT-multiplication
\\ t = 61.5789;     \\ asymptotic optimal for Karatsuba-multiplication
\\ t = 144.8858;    \\ asymptotic optimal for normal multiplication
\\ PARI seems to use Karatsuba-multiplication for polynomials
\\ ------------------------------------------------------------------------
```

```
    t = 61.5789;
    r = nextprime(((1/((t+1)*log(t+1)-t*log(t)))^2*log(n)^2));
    until(isprimitiveroot(n,r),r = nextprime(r+2)); print(r);
    s = ceil(solve(x=1,floor(2*t*(r-1)),
         (lngamma(r-1+x)-lngamma(r-1)-lngamma(x+1))/log(n)/floor(sqrt(r))-1));
    s = ceil((s+2)/2);
    print(s);
    if( component(factor(n,(s-1)^2),1) != [n]~, return(0) );

\\ --------------------------------------------------------------------------
\\ Step 3: AKS type of for-loop
\\ --------------------------------------------------------------------------

    X = Mod(Mod(1,n)*x,x^r-1);
    for( a=1, s-1,
        if( (X-a)^n  != X^n-a, return(0) );
    );
    return(1);
}
```

3. Pollards $p-1$ Methode. Hier ist ein PARI Programm, geschrieben von F. Voloch, das den Pollard-Algorithmus mit potenzglatten Zahlen implementiert. Man versuche diesen Algorithmus mit

$$n = 3076923076923076923076923077, \quad t = 10^5$$

$$n = 2222222222222222222222222223, \quad t = 10^5$$

Was ist das Problem bei der zweiten Zahl ? (Antwort in der Vorlesung !)

```
\\ Modified Pollard p-1 method. Usage pollard(n,t) where n is the number to be
\\ factored and t is the size of the factor base, i.e., the first t primes.
\\ Adjust to taste. Modified to handle case where p-1 is smooth for all p|n.

pollard(n,t)=
        {local(m,b,p,j,d,g,lgn);
        until(b,b=random()%n);
        if(n<=1,error("Invalid input"));
        g=gcd(b,n);
        if(g>1,print(g," is a factor of ",n),
        p=3;j=1;d=1;lgn=log(n);
        while(d==1&&j<=t,
                b=lift(Mod(b,n)^(p^floor(lgn/log(p))));
                d=gcd(b-1,n);
                        if(d>1&&d<n,print(d," is a factor of ",n));
                        j=j++;p=nextprime(p++));m=1;
                        while(d==1&&m<=floor(lgn/log(2)),b=lift(Mod(b,n)^2);
                d=gcd(b-1,n);if(d>1&&d<n,print(d," is a factor of ",n));m=m++);
                        if(d==1||d==n,print("Try increasing t")))
}
```

4. Pollards ρ-Methode. Man teste das Beispiel der Vorlesung, nämlich $n = 82123$, und die vorherige Zahl

$$n = 2222222222222222222222222222223$$

mit dem folgenden Programm:

```
\\ A simple implementation of Pollard's rho method. The function
\\ rho(n) outputs the complete factorization of n in the same format as factor

rho1(n)=
{
  local(x,y);

  x=2; y=5;
  while(gcd(y-x,n)==1,
    x=(x^2+1)%n;
    y=(y^2+1)%n; y=(y^2+1)%n
  );
  gcd(n,y-x)
}

rho2(n)=
{
  local(m);

  m=rho1(n);
  if (isprime(m),
    print(m)
  ,
    rho2(m)
  );
  if (isprime(n/m),
    print(n/m)
  ,
    rho2(n/m)
  );
}

rho(n)=
{
  local(m);

  m=factor(n,0); print(m);
  n=m[length(m[,1]),1];
  if (!isprime(n),rho2(n));
}

rhobrent(n)=
{
  x=y=x1=2; k=l=p=1; c=0;
```

```
  while (1,
    x=(x^2+1)%n; p=(p*(x1-x))%n; c++;
    if (c==20,
      if (gcd(p,n)>1, break);
      y=x;c=0
    );
    k--;
    if (!k,
      if (gcd(p,n)>1, break);

      x1=x; k=l; l <<= 1; \\ l = 2*l
      for (j=1,k, x=(x^2+1)%n);
      y=x; c=0
    )
  );
  g=1;
  while (g==1,
    y=(y^2+1)%n;
    g=gcd(x1-y,n)
  );
  if (g==n,error("algorithm fails"));
  g
}
```

5. Lenstras elliptische Kurven Methode. Das folgende PARI-Programm, geschrieben von F. Rodriguez-Villegas und A.M. Pacetti implementiert Lenstras Algorithmus zur Faktorisierung einer natürlichen Zahl n mit der elliptischen Kurven Methode. Man finde rekursiv mit ECM die Faktorisierung der folgenden beiden Zahlen:

$$n = 1004916476648823359263251605226592753957 7$$
$$n = 22222222222222222222222222223$$

Man diskutiere das unterschiedliche Verhalten der beiden Zahlen bezüglich ECM.

```
\\---------------------------------------------------------------------
\\ This is Lenstra algorithm for computing some non-trivial factor of
\\ an integer n. The entries are the number n and the number of times to
\\ run the test. There is also an extra entry (optional), which decides
\\ the size of the prime factors of (p-1), where p is some prime factor
\\ of n.

lenstra(n,times,k)=
        {local(aux,ans,cont,P,E,a4);
        if(isprime(n),return("It's prime"));
        if(times==0,return("Put more times"));
        if((aux=gcd(n,6))!=1,return([aux,n/aux]));
        cont=1;
        if(k,,k=multilcm(vector(precprime(truncate(log(n))))));
```

```
        if(k==1,k=multilcm(vector(precprime(n))));
        while(aux=n^(1/cont)>=2,if(round(aux)^cont==n, return (
        vector( cont,x,aux))); cont++);
        for(k=1,times,
        P=[random(n),random(n)]; a4=random(n);
        E=[0,0,0,a4,P[2]^2-P[1]^3-a4*P[1]];
        if((aux=gcd(4*E[4]^3+27*E[5]^2,n))!=1,if(aux!=n,return([aux,n/aux]),
        return(lenstra(n,times-1))));
        ellpowmod(E,P,k,n));
        return("I failed, try more times")}

\\=====================================================================
\\ This is a routine for computing the addition of 2 points in an
\\ elliptic curve, but looking at the equation reduced modulo some
\\ number n. If the addition cannot be done, outputs some non-trivial
\\ factors of n.
elladdmod(e,z1,z2,n)=
        {local(aux,ans,lambda);
        if(z1==[0],return(z2),if(z2==[0],return(z1)));
        if(z1!=z2,
        aux=bezout(z2[1]-z1[1],n);
        if(aux[3]!=1,return([aux[3],n/aux[3]]),lambda=(z2[2]-z1[2]) *
        aux[1]);
        [aux=((lambda^2-z1[1]-z2[1])%n),(-lambda*aux-(z1[2]-lambda*z1[1]))%n],
        aux=4*z1[1]^3+4*E[4]*z1[1]+4*E[5];
        aux=bezout(aux,n);
        if(aux[3]!=1,return([aux[3],n/aux[3]]),lambda=aux[1]);
        [ans=((z1[1]^4-2*E[4]*z1[1]^2-8*E[5]*z1[1]+E[4]^2)*lambda)%n,
        -((ans-z1[1])*bezout(2*z1[2],n)[1]*(3*z1[1]^2+E[4])+z1[2])%n])}

\\=====================================================================
\\ This is the classical algorithm for adding finding kP for a point P.

ellpowmod(E,P,k,n)=
        {local(aux,ans,l,temp);
        aux=binary(k);
        ans=[0];l=length(aux);temp=P;
        for(k=1,l,if(aux[l-k+1]==1,ans=elladdmod(E,ans,temp,n));
                temp=elladdmod(E,temp,temp,n));
        ans}

\\=====================================================================
\\ This is an auxiliary routine for given a vector with integers to
\\ compute the g.c.d. of all of them.

multigcd(v)=
        {local(ans);
        ans=gcd(v[1],v[2]);
        for(k=1,length(v)-2,ans=gcd(ans,v[k+2]));
```

```
        ans}

\\==================================================================
\\ This is an auxuliary routine for given a vector with integers to
\\ compute the l.c.m. of all of them.

multilcm(v)=
        {local(ans);
        ans=lcm(v[1],v[2]);
        for(k=1,length(v)-2,ans=lcm(ans,v[k+2]));
        ans}
```