```
#
# This is a commented version of the Maple pr
ocedure
# ifactor for analysis of the methods.
#
# The input parameter n in general a positive
 integer.
#
# After a parameter cheking other cases: nega
tive
# integer, rational numbers, etc. are treated
.
#
# The first step is remove factors 2, 3, 5,7,
 11, 13.
# gcd(n,2^4*3^2*5*7*11*13) calculated until b
ecame 1.
# If it is greater then 1, then this gcd is f
actored by
# the small routine ifactor/ifact235.
#
# The second step is remove somewhat larger p
rime factors below
# from 17 to 1699. This also happens with the
 gcd trick. Until the
# gcd is larger then 1, the routine ifactor/i
fact1st find these
# factors from the gcd.
#
# If the remainder still greater then 1, then
 the procedure
# corresponding to the second parameter is lo
aded until the name
# ‘ifactor/bottom‘ and the procedure ifactor/
ifact0th called
# with passing thierd and further parameters.
# This procedure use ‘ifactor/bottom‘.
#
# If no second parameter is passed, then the
‘ifactor/morrbril‘
```

```
# procedure loaded and the procedure ifactor/
ifact0th called.
#

ifactor:=proc(n)
local sol,r;
global ‘ifactor/bottom‘;
options remember,system,‘Copyright 1993 by Wa
terloo Maple Software‘;
    if nargs < 1 or 1 < nargs and not type(ar
gs[2],name) then
        ERROR(‘invalid arguments‘)
    fi;
    if type(n,integer) then
        if 0 < n then sol := 1; r := n
        elif n < 0 then sol := -1; r := -n
        else RETURN(0)
        fi
    elif type(n,fraction) then RETURN(ifactor
(op(1,n))/ifactor(op(2,n)))
    elif type(n,{‘*‘,set,list,relation}) then
 RETURN(map(ifactor,n))
    elif type(n,‘^‘) and type(op(2,n),integer
) then
        RETURN(ifactor(op(1,n))^op(2,n))
    elif type(n,‘‘(integer)) then RETURN(ifac
tor(op(1,n)))
    else ERROR(‘invalid arguments‘)
    fi;
    igcd(r,720720);
    while " <> 1 do
        sol := sol*‘ifactor/ifact235‘("); r :
= iquo(r,""); igcd("""",r)
    od;
    igcd(r,
    116542951143701442118256425539411806278705
5518107369035248436272442928865510\
    970895780845374261270209629822427348440139
2345696701325406686013877435618 3\
    909763696443067069059862065192038988478 41
```

```
821908294387922301947266843024378\
    112299030349298539707741678899215627540549809976535794519247972221385722721200060812856017119765934942419611716722790220172291221277043483457977929715054446536085051102596639253777494582990197958881314319454754358382573986765277067029998348079733587464341461034343515018798972815499657761920880748938214759811175844174797182002767061333889689987815611105878789730794721580125932049284399286767844431923823398185219113121941888475787066031000754040432081136673342490211021568504524132325028970923390
    );
    while " <> 1 do
        sol := sol*‘ifactor/ifact1st‘("); r := iquo(r,""); igcd("",r)
    od;
    if r <> 1 then
        if nargs = 1 then
            ‘ifactor/bottom‘ := readlib(’‘ifactor/morrbril‘’);
            ‘ifactor/ifact0th‘(r)
        else
            ‘ifactor/bottom‘ := readlib(‘ifactor/‘.args[2]);
            ‘ifactor/ifact0th‘(r,args[3 .. nargs])
        fi;
        if " <> FAIL then sol*" else FAIL fi
    else sol
    fi
end;


#
# This small routine find the factors of
# a positive integer known to have the
```

```
# gcd of 2^4*3^2*5*7*11*13 and the number
# to factor. The factorization is given back.
#

ifactor/ifact235:=proc(n)
local q;
options remember,system,‘Copyright 1993 by Wa
terloo Maple Software‘;
    if n = 1 then 1
    elif irem(n,13,’q’) = 0 then ‘ifactor/ifa
ct235‘(q)*‘‘(13)
    elif irem(n,11,’q’) = 0 then ‘ifactor/ifa
ct235‘(q)*‘‘(11)
    elif irem(n,7,’q’) = 0 then ‘ifactor/ifac
t235‘(q)*‘‘(7)
    elif irem(n,2,’q’) = 0 then ‘‘(2)*‘ifacto
r/ifact235‘(q)
    elif irem(n,3,’q’) = 0 then ‘‘(3)*‘ifacto
r/ifact235‘(q)
    else ‘‘(5)
    fi
end;


#
# This routine find the factors from 17 to 16
99. The
# input is a positive integer containing only
 factors
# between 17 and 1699.
#
# The routine divides first the factors into
# smaller groups 17--19, 23--37, 41--67,
# 71--137, 139-281, 283--659, 661--1699 using
 the gcd trick.
# Then, depending on the gcd obtained, a sear
ch routine
# ifactor/wheelfact is started. The search li
mit is
# choosen supposing two factors find: 17*19,
```

```
23*29, 41*43,
# 71*73, 139*149, 283*293, 661*673.
#

ifactor/ifact1st:=proc(n)
local i,ig,r,sol;
options remember,system,‘Copyright 1993 by Wa
terloo Maple Software‘;
    sol := 1;
    r := n;
    for i to 7 while r <> 1 do
        ig := igcd(r,(323,765049,10589676401
89,916834684886440380235864 1659,
        342104831177853836844000918542910563
842436194397212591435983889,
        792709889190322888122151467214412384
719082407423339586081109312119937 5\
        916897659993560023369901756756352260
640974174268277959137341840564 3105\
        622634495020112463 89
        ,
        179121216345076044215350912828530796
966835016565315142254527442696 6329\
        577088681163508088417026575611445251
431965793198643542697159606743 6404\
        292477929139917856403646738338454540
141395254935616183453275644191 8356\
        224214016169612042405530047600574836
123437652470013895386096822766 9770\
        861596156638350957592746942206445542
762469489659896584754267190755 8236\
        641492154574341354887668197201554136
418826500667651775965123772746 1023\
        124369799987063929201186649
        )[i]);
        if ig <> 1 then
            r := iquo(r,ig);
            while [323,667,1763,5183,20711,82
919,444853][i] <= ig do
                ‘ifactor/wheelfact‘(ig,[17,23
```

```
,41,71,139,283,661][i]);
                sol := sol*‘‘(");
                ig := iquo(ig,"")
            od;
            sol := sol*‘‘(ig)
        fi
    od;
    sol
end;


#
# This routine find the prime factors with se
arcing.
# The input is a positive integer known to ha
ve only factor
# between 17 and 1699.
#
# The "large searching" here goes using gcd t
oo, using
# intervals with length 30. The find gcd’s la
rger then 1
# finally factored by a smaller search trying
 odd numbers.
#

ifactor/wheelfact:=proc(n,s)
local i,ii;
options ‘Copyright 1993 by Waterloo Maple Sof
tware‘;
    for i from 30*iquo(s,30)+15 by 30 do
        i^2;
        if igcd(n,("-4)*("-16)*("-64)*("-196)
) <> 1 then
            for ii from i-14 by 2 do
                if igcd(ii,n) <> 1 then RETUR
N(ii) fi
            od
        fi
    od
```

```
end;


#
# This is the general end-factoring routine.
# The input is a positive integer containing
no
# factor below 1700.
#
# If the number is less then 1709^2, then it
is
# a prime and simple returned. Otherwise the
number is
# tested and if it is found to be prime, retu
rned.
#
# The second step is to try whether the numbe
rs is
# a power using the routine ifactor/power.
#
# The thierd step is to try Pollard's p-1 met
hod.
# If there are chanches then tried again.
# After the second failer or if no chances
# another version called ifactor/pp100000
# called to find factors p for which p-1
# has only factors below 100000.
#
# The last try is the procedure ifactor/botto
m,
# which depends on the second input parameter
.
#
# Finally, if no factor is found, then the
# routine returns with fail. Otherwise if is
called
# for the factors found and returns with the
product.
#
```

```
ifactor/ifact0th:=proc(n)
local sol,p;
options ‘Copyright 1993 by Waterloo Maple Sof
tware‘;
    if n < 2920681 then RETURN(‘‘(n)) fi;
    if isprime(n) then RETURN(‘‘(n)) fi;
    sol := ‘ifactor/power‘(n,’p’);
    if sol <> FAIL then RETURN(‘ifactor/ifact
0th‘(sol)^p) fi;
    sol := ‘ifactor/pollp1‘(n,13003);
    if sol = _tryagain then sol := ‘ifactor/p
ollp1‘(n,13004) fi;
    if sol = _tryagain then sol := FAIL fi;
    if sol = FAIL then sol := ‘ifactor/pp1000
00‘(n) fi;
    if sol = FAIL then
        sol := ‘ifactor/bottom‘(args);
        if not type(sol,integer) then RETURN(
sol) fi
    fi;
    ‘ifactor/ifact0th‘(n/sol,args[2 .. nargs]
)*
        ‘ifactor/ifact0th‘(sol,args[2 .. narg
s])
end;


#
# This small procedure check whether the inpu
t
# positive number is a prime power. It is kno
wn
# that there are no too small factors. The fa
ctor
# is given back and the exponent in the param
eter
# ’p’.
#

ifactor/power:=proc(n,p)
```

```
local i,r;
options 'Copyright 1993 by Waterloo Maple Sof
tware';
    r := isqrt(n);
    if r^2 = n then p := 2; RETURN(r) fi;
    for i from 3 by 2 to iquo(length(n)+2,3)
do
        if not isprime(i) then next fi;
        r := readlib('iroot')(n,i);
        if r^i = n then p := i; RETURN(r) fi
    od;
    FAIL
end;


#
# This procedure try to find factors of the i
nput
# positive integer n using Pollard's p-1 meth
od.
# The second parameter is the base to find th
e
# factors.
#
# The base is in a cycle powered gradually to
 2^9,
# 3^6*5^4*7^2*11^2, 7*11*13^2*31^2*1229,
# 17^2*19^2*23^2*37*41*263, 29^2*43*47*53*83*
443,
# ... 1699*1733*1741*1811*1867*1907*1913.
# After each step the gcd of the power-1 and
n
# is calculated. If the power was not 1, then
# this gcd is returned. If the power was 1,
# i.e. more factor steps in, then the power
# calculation is repeated in smaller steps.
# If this separates the factors, then a facto
r
# is given back. Otherwise we try again.
#
```

```
ifactor/pollp1:=proc(n,seed)
local d,i,k,primes,w;
options 'Copyright 1993 by Waterloo Maple Sof
tware';
    primes := [512,2701400625,15369250897,220
19225847811,3312226271377,
        573380756513,9895915431937,9901032144
937,26758334120513,
        221345652736259,35189633528637 1,60253
2741650737,705905813463569,
        1146860257543171,1197923438167619,410
1182822350853,1809363216948 9029,
        1578734133234 9199,3054169853683 4113,3
227286551043 2777,
        9019322338594 8289,9485927963231 9593,2
2095785694294 8429,
        2174267795928 4993,3538185221811 90721
,41939994546288 4489,
        5772441890673 13457,5765664480276 1223
,6350104393166 35813,
        1163208606573 930629,1274527543260 71315
3,9642014834428 64677,
        1189549054560 841981,1730661029236 70353
3,2324525996860 803761,
        1749379074918 976547,2379318098186 47476
1,3906310006235 021863,
        4400036932968 546433,3486357872137 29184
1,4051550852530 311307,
        7062924201913 552913,9946204387896 72865
7,9800419932800 824021,
        1401641645313 8321531,1845154705663 4890
273,2077083919129 6803529,
        2326223787388 0485889,2828292246527 8428
769,2377942002981 6701443,
        2900642736000 4220003,6322941213293 968
6429249];
    seed;
    for i to nops(primes) do
        modp(power(",primes[i]),n);
```

```
        if igcd("-1,n) <> 1 then
            if " <> 1 then RETURN(igcd("-1,n)
) fi;
            w := "";
            d := numtheory[factorset](primes[
i]);
            for k in d do
                w := modp(power(w,k),n);
                if w = 1 then
                    if 1 < i then RETURN(proc
name(n,modp(power(seed,k),n)))
                    fi
                elif igcd(w-1,n) <> 1 then RE
TURN(igcd(w-1,n))
                fi
            od;
            RETURN(_tryagain)
        fi
    od;
    FAIL
end;


#
# This procedure is a conterpart of the
# procedure above. Using the huge integers
# _prpr4000, _prpr10000, ... , _prpr94000
# which are products of primes from 4000 to 1
0000,
# from 10000 to 16000, etc. for which p-1
# is not smooth, with the gcd method and usin
g
# ifactor/wheelfract find these factors too.
#

ifactor/pp100000:=proc(n)
local i;
options 'Copyright 1993 by Waterloo Maple Sof
tware';
    for i from 4000 by 6000 to 94000 do
```

```
        igcd(_prpr.i,n);
        if " <> 1 then
            if " <> n then RETURN(")
            else RETURN(‘ifactor/wheelfact‘(n
,i+1))
            fi
        fi
    od;
    FAIL
end;


#
# The smallest constant used in ifactor/pp100
000,
# as an example.
#

_prpr4000:=996505806050489446326568828996145 6
35724099720836198366105348467073 4\
83901267943\
339014801270129828448962441064073276266494846
16165544727688937196670540674 7 2\
39720527290803677864361380156879525642460435 2
66474010399107017576607857274310 5\
3299423;


#
# In the ‘easy‘ case no more further effort d
one.
#

ifactor/easy:=proc(n)
        options ‘Copyright 1993 by Waterloo M
aple Software‘;
        _c.(length(n))
end;

#
```

```
# This is the default procedure for
# factorization of complicated cases.
# It use a variation of the Morrison--Brillha
rd
# algorithm with continued fractions (??).
#

ifactor/morrbril:=proc(n)
local i,j,A0,A1,Q0,Q1,PQ,iPQ,iPQc,g,r0,r1,qn,
p,X,primes;
global _sq,_X2,_XX2,_Nprimes4,_Heap4,_Prpri;
options 'Copyright 1993 by Waterloo Maple Sof
tware';
    _sq := '_sq';
    PQ := [];
    iPQ := 1;
    iPQc := 1;
    userinfo(1,ifactor,'ifactor final stage:'
,n,
        lprint('Using a variation of the Morr
ison-Brillhart algorithm'));
    g := isqrt(n);
    if n < g^2 then g := g-1 fi;
    X := isqrt(isqrt(isqrt(isqrt(10^(isqrt(iq
uo(599*length(n),4))-11)))));
    _X2 := min(X^2,20*X);
    _XX2 := X*_X2;
    userinfo(9,ifactor,'n' = n,'X' = X,'_X2'
= _X2);
    primes[1] := 2;
    for _Nprimes4 while primes[_Nprimes4] < X
 do
        primes[_Nprimes4];
        do  nextprime("); if modp(power(n,1/2
*"-1/2),") = 1 then break fi od;
        primes[_Nprimes4+1] := "
    od;
    p := 1;
    while p <= _Nprimes4 do  p := 2*p od;
    p := p-1;
```

```
    for i from _Nprimes4-1 by -1 to 0 do
        1;
        j := 2*i+1;
        while j < _Nprimes4 do  ""*_Heap4[j];
 j := 2*j od;
        if p < j then primes[j-p] else primes
[j-p+_Nprimes4] fi;
        _Heap4[i] := """*"
    od;
    _Prpri := 1440*_Heap4[0];
    A0 := 0;
    A1 := 1;
    Q1 := n;
    Q0 := 1;
    r1 := g;
    do
        qn := iquo(2*g-r1,Q0,'r0');
        A0 := modp(qn*A1+A0,n);
        Q1 := Q1+qn*(r0-r1);
        if [qn,A0,r1] = PQ then RETURN(readli
b('ifactor/pollard')(n))
        elif iPQc = iPQ then iPQ := 2*iPQ; PQ
 := [qn,A0,r1]; iPQc := 1
        else iPQc := iPQc+1
        fi;
        analyze_resid(A0,-Q1,n);
        if " <> FAIL then RETURN(") fi;
        qn := iquo(2*g-r0,Q1,'r1');
        A1 := modp(qn*A0+A1,n);
        Q0 := Q0+qn*(r1-r0);
        analyze_resid(A1,Q0,n);
        if " <> FAIL then RETURN(") fi
    od
end;


#
# This subroutine works for the Morrison--Bri
llhard
# algorithm.
```

```
#

analyze_resid:=proc(a,a2,n)
local lrgpr,resid,base,g;
global _sq;
options ‘Copyright 1993 by Waterloo Maple Sof
tware‘;
    abs(a2);
    igcd(",_Prpri);
    while " <> 1 do
        iquo("","); if _XX2 < " then RETURN(F
AIL) fi; igcd(","")
    od;
    if _X2 < "" then RETURN(FAIL) fi;
    lrgpr := "";
    base := a;
    resid := a2;
    do
        igcd(resid,_Heap4[0]);
        g := igcd(iquo(resid,"),");
        while g <> 1 do
            resid := iquo(resid,g^2);
            base := modp(base/g,n);
            igcd(g,resid);
            g := igcd(",iquo(resid,"))
        od;
        if resid = 1 then
            if base <> 1 and base <> n-1 then
 RETURN(igcd(base-1,n))
            else userinfo(9,ifactor,‘Bad luck
 (+-1)^2=1‘); RETURN(FAIL)
            fi
        fi;
        if lrgpr = 1 then lrgpr := lrgst_fact
or(resid) fi;
        userinfo(9,ifactor,‘*** factorable re
sidue ***‘,base,resid,lrgpr);
        if assigned(_sq[lrgpr]) then
            userinfo(9,ifactor,‘----- collisi
on at‘,lrgpr);
```

```
            resid := resid*_sq[lrgpr][2]/lrgp
r^2;
            base := modp(base*_sq[lrgpr][1]/l
rgpr,n);
            lrgpr := 1
        else _sq[lrgpr] := [base,resid]; RETU
RN(FAIL)
        fi
    od
end;


#
# This is D. Shanks' undocumented square-free
 factorization.
#

ifactor/squfof:=proc(a)
   local l,p,q,r,s,w;
   options 'Copyright 1993 by Waterloo Maple
Software';
       p := isqrt(a);
       if a < p^2 then p := p-1 fi;
       q := a-p^2;
       s := p;
       if q = 0 then RETURN(p) elif q = 1 the
n RETURN(squfof(2*a)) fi;
       l := 2*isqrt(2*s);
       do
           if q <= l then w[q/igcd(q,2)] := 1
 fi;
           p := s-modp(s+p,q);
           q := (a-p^2)/q;
           r := isqrt(q);
           if q = r^2 and w[r] <> 1 then brea
k fi;
           if q <= l then w[q/igcd(q,2)] := 1
 fi;
           p := s-modp(s+p,q);
           q := (a-p^2)/q
```

```
        od;
        p := p+r*iquo(s-p,r);
        q := (a-p^2)/r;
        do
            s-modp(s+"",","); if " = """ then RE
TURN(""/igcd("",2)) fi; (a-"^2)/""
        od
    end;


#
# This is J. M. Pollard's rho method.
#

ifactor/pollard:=proc(n,ex)
    local EX;
    options 'Copyright 1993 by Waterloo Maple
 Software';
        if nargs = 1 then EX := 2
        elif nargs <> 2 or not type(ex,intege
r) or ex < 2 then
            ERROR('invalid arguments')
        else EX := ex
        fi;
        2;
        modp(power(",EX)+1,n);
        do
            modp(power("",EX)+1,n);
            modp(power(power("",EX)+1,EX)+1,n
);
            if 1 < igcd(""-",n) then
                igcd(""-",n); if " = n then R
ETURN(FAIL) else RETURN(") fi
            fi
        od;
        FAIL
    end;


#
```

```
# This is Lenstra's elliptic curve method.
#

ifactor/lenstra:=proc(n)
    local i,s,prime,f,A,X,Z,rgen,sp,a,r,curve
s,B1,kg,kgg;
    options 'Copyright 1993 by Waterloo Maple
 Software';
        if nargs < 1 or 3 < nargs then ERROR(
'wrong number of arguments.') fi;
        if nargs < 3 then B1 := 1000000 else
B1 := args[3] fi;
        if nargs < 2 then curves := 30 else c
urves := args[2] fi;
        if modp(n,2) = 0 then RETURN(2) fi;
        if modp(n,3) = 0 then RETURN(3) fi;
        s := evalf(1/2*sqrt(5)-1/2,30);
        A := array(1 .. curves);
        X := array(1 .. curves);
        Z := array(1 .. curves,[1 $ curves]);
        rgen := rand(1 .. n-1);
        for i to curves do
            a := 0;
            while modp(a*(a^2-1)*(9*a^2-1),n)
 = 0 do
                r := rgen();
                kg := r^2+6;
                kgg := igcd(kg,n);
                if kgg <> 1 then RETURN(kgg)
fi;
                a := modp(6*r/kg,n)
            od;
            A[i] := modp(1/16*(-3*a^4-6*a^2+1
)/a^3+1/2,n);
            X[i] := modp(3/4*a,n)
        od;
        prime := 2;
        while prime <= B1 do
            sp := round(s*prime);
            'ifactor/lenstra/mulpp'(1,n,A,sp,
```

```
prime,B1,X,Z);
            f := Z[1];
            for i from 2 to curves do
                ‘ifactor/lenstra/mulpp‘(i,n,A
,sp,prime,B1,X,Z);
                f := modp(f*Z[i],n)
            od;
            f := igcd(f,n);
            if f <> 1 then RETURN(f) fi;
            prime := nextprime(prime)
        od;
        FAIL
    end;


#
# This subroutine the multiplications on a gi
ven
# elliptic curve.
#

ifactor/lenstra/mulpp:=proc(i,n,A,mm,nn,B1,X,
Z)
local pow,ax,az;
options ‘Copyright 1993 by Waterloo Maple Sof
tware‘;
    ax := X[i];
    az := Z[i];
    pow := nn;
    while pow <= B1 do
        ‘ifactor/lenstra/ellmul‘(n,A[i],mm,nn
,ax,az,’ax’,’az’); pow := pow*nn
    od;
    X[i] := ax;
    Z[i] := az
end;


#
# This procedure do some multiplications
```

```
# on an elliptic curve.
#

ifactor/lenstra/ellmul:=proc(n,A,mm,nn,px,pz,
aax,aaz)
local ax,az,bx,bz,cx,cz,tmpx,tmpz,d,e,t1,t2;
options ‘Copyright 1993 by Waterloo Maple Sof
tware‘;
    cx := px;
    cz := pz;
    e := mm;
    d := nn-mm;
    if e < d then
        ‘ifactor/lenstra/elldoub‘(n,A,px,pz,’
bx’,’bz’);
        ax := px;
        az := pz;
        d := d-e
    else
        ‘ifactor/lenstra/elldoub‘(n,A,px,pz,’
ax’,’az’);
        bx := px;
        bz := pz;
        e := e-d
    fi;
    while e <> 0 do
        if e < d then
            tmpx := bx;
            tmpz := bz;
            t1 := modp((ax-az)*(bx+bz),n);
            t2 := modp((ax+az)*(bx-bz),n);
            bx := modp(cz*modp((t1+t2)^2,n),n
);
            bz := modp(cx*modp((t1-t2)^2,n),n
);
            d := d-e
        else
            tmpx := ax;
            tmpz := az;
            t1 := modp((ax-az)*(bx+bz),n);
```

```
            t2 := modp((ax+az)*(bx-bz),n);
            ax := modp(cz*modp((t1+t2)^2,n),n
);
            az := modp(cx*modp((t1-t2)^2,n),n
);
            e := e-d
        fi;
        cx := tmpx;
        cz := tmpz
    od;
    aax := ax;
    aaz := az
end;


#
# This procedure do a doubling on the ellipti
c curve.
#

ifactor/lenstra/elldoub:=proc(n,A,ax,az,cx,cz
)
local t1,t2;
options ‘Copyright 1993 by Waterloo Maple Sof
tware‘;
    t1 := modp((ax+az)^2,n);
    t2 := modp((ax-az)^2,n);
    cx := modp(t1*t2,n);
    cz := modp((t1-t2)*modp(A*(t1-t2)+t2,n),n
)
end;
```