```
> restart;
```

Algorithme et Multiplication de Karatsuba

```
> Karatsuba := proc(f, g, n)
      local m, F1, G1, A, B, C;
      m := 1 / 2 * n;
      if n = 1 then
          return f * g
      end if;
      F1 := quo(f, x^m, x, 'F0');
      G1 := quo(g, x^m, x, 'G0');
      A := F0 * G0;
      B := F1 * G1;
      C := Karatsuba(F1 + F0, G0 + G1, m) - A - B;
      return expand(B * x^n + C * x^m + A)
  end proc;
      1
```

$$\mathbf{proc}(f, g, n) \tag{1.1}$$

```
    local m, F1, G1, A, B, C;
    m := 1 / 2 * n;
    if n = 1 then
        return f * g
    end if;
    F1 := quo(f, x^m, x, 'F0');
    G1 := quo(g, x^m, x, 'G0');
    A := F0 * G0;
    B := F1 * G1;
    C := Karatsuba(F1 + F0, G0 + G1, m) - A - B;
    return expand(B * x^n + C * x^m + A)
end proc
```

```
> f := x^2 + 2 x + 1; 1; g := x^4 + 5 x + 3; 1; k := expand(f g); 1
```

$$x^2 + 2\,x + 1$$

$$x^4 + 5\,x + 3$$

$$x^6 + 5\,x^3 + 13\,x^2 + 2\,x^5 + 11\,x + x^4 + 3 \tag{1.2}$$

```
> Karatsuba(f, g, 4); 1
```

$$x^6 + 5\,x^3 + 13\,x^2 + 2\,x^5 + 11\,x + x^4 + 3 \tag{1.3}$$

```
>
> Karatsuba1 := proc(f, g)
      local m, n, l;
```

```
        n := degree(f, x);
        l := degree(g, x);
        if n < l then
            n := l;
            l := degree(f, x)
        end if;
        Karatsuba(f, g, n)
    end proc;
        1
```

```
proc(f, g)                                                    (1.4)
    local m, n, l;
    n := degree(f, x);
    l := degree(g, x);
    if n < l then
        n := l;
        l := degree(f, x)
    end if;
    Karatsuba(f, g, n)
end proc
```

```
> Karatsuba1(f, g); 1
```

$$x^6 + 5\,x^3 + 13\,x^2 + 2\,x^5 + 11\,x + x^4 + 3 \qquad (1.5)$$

```
>
> Karatsuba2 := proc(f, g)
        local m, n, l, k;
        n := degree(f, x);
        l := degree(g, x);
        if n < l then
            n := l;
            l := degree(f, x)
        end if;
        k := 2^(floor(log[2](n)) + 1);
        Karatsuba(f, g, k)
    end proc;
        1
```

```
proc(f, g)                                                    (1.6)
    local m, n, l, k;
    n := degree(f, x);
    l := degree(g, x);
    if n < l then
        n := l;
        l := degree(f, x)
    end if;
```

```
    k := 2^(floor(log[2](n)) + 1);
    Karatsuba(f, g, k)
end proc
> p := Karatsuba2(f, g); 1
```

$$x^6 + 5\,x^3 + 13\,x^2 + 2\,x^5 + 11\,x + x^4 + 3 \tag{1.7}$$

```
>
```

L'algorithme de Karastuba est en $O\left(n^{\log[2](3)}\right)$, *c'est à dire en* $O\left(n^{1.59}\right)$, *il est donc mieux que la multiplication naïve en* $O\left(n^2\right)$ *(quadratique).*

Transformée de Fourier discrète sur ℂ

```
> with(PolynomialTools);
      -1;
      with(ListTools);
      -1;
      with(DiscreteTransforms);
      -1
>
> Fourier := proc(f, n, w)
      local i, R, L, m, F, R0, R1;
      L := [seq(coeff(f, x, i), i = 0..n - 1)];
      if n = 1 then
          return [L[1]]
      end if;
      m := 1 / 2 * n;
      R[0] := evalf(sum((L[j] + L[j + m]) * x^(j - 1), j = 1..m));
      R[1] := evalf(sum((L[j] - L[j + m]) * w^(j - 1) * x^(j - 1), j = 1..m));
      R0 := Fourier(R[0], m, w^2);
      R1 := Fourier(R[1], m, w^2);
      for i from 0 to m - 1 do
          L[2 * i + 1] := R0[i + 1];
          L[2 * i + 2] := R1[i + 1]
      end do;
      return L
  end proc;
      1
proc(f, n, w)                                                    (2.1)
    local i, R, L, m, F, R0, R1;
    L := [seq(coeff(f, x, i), i = 0..n - 1)];
    if n = 1 then
        return [L[1]]
    end if;
    m := 1 / 2 * n;
```

```
    R[0] := evalf(sum((L[j] + L[j+m])*x^(j-1), j = 1..m));
    R[1] := evalf(sum((L[j] - L[j+m])*w^(j-1)*x^(j-1), j = 1..m));
    R0 := Fourier(R[0], m, w^2);
    R1 := Fourier(R[1], m, w^2);
    for i from 0 to m - 1 do
        L[2*i+1] := R0[i+1];
        L[2*i+2] := R1[i+1]
    end do;
    return L
end proc
```

```
> w := exp(I*Pi, 1/2); 1; w^4; 1
```

$$I$$

$$1 \tag{2.2}$$

n doit être une puissance de 2

```
> Fourier(f, 4, w); 1
```

$$[4., 2.\,I, 0., -2.\,I] \tag{2.3}$$

```
> h := 2*x^3 + x^2 + 6*x + 4; 1
```

$$2\,x^3 + x^2 + 6\,x + 4 \tag{2.4}$$

```
> Fourier(h, 4, w); 1
```

$$[13., 3. + 4.\,I, -3., 3. - 4.\,I] \tag{2.5}$$

```
>
```

On peut vérifier si notre procédure est juste avec la fonction FFT et la fonction iFFT :

```
> X := array([1, 2, 1, 0]); 1; Y := array([0, 0, 0, 0]); 1; FFT(2, X, Y); 1
```

$$table([(1) = 1, (2) = 2, (3) = 1, (4) = 0])$$

$$table([(1) = 0, (2) = 0, (3) = 0, (4) = 0])$$

$$4 \tag{2.6}$$

```
> print(X); 1
```

$$X \tag{2.7}$$

```
> print(Y); 1
```

$$Y \tag{2.8}$$

```
> iFFT(1, X, Y); 1; print(X); 1; print(Y); 1
```

$$2$$

$$X$$

$$Y \tag{2.9}$$

```
> X := array([4, 6, 1, 2]); 1; Y := array([0, 0, 0, 0]); 1; FFT(2, X, Y); 1
```

$$table([(1) = 4, (2) = 6, (3) = 1, (4) = 2])$$

$$table([(1) = 0, (2) = 0, (3) = 0, (4) = 0])$$

$$4 \tag{2.10}$$

```
> print(X); 1; print(Y); 1
```

$$X$$

$$Y \tag{2.11}$$

```
> iFFT(2, X, Y); 1; print(X); 1; print(Y); 1
```

$$4$$

$$X$$

$$Y \tag{2.12}$$

```
>
```

On peut "s'amuser" à faire une procédure qui ne dépend que de f : qui calcule la puissance de 2 nécessaire au bon fonctionnement de FFT, notée ici m, et qui utilise une racine mième simple :
exp(2iπ/m) :

```
> Fourierbis := proc(f)
      local n, m, w;
      n := degree(f, x);
      m := 2^(floor(log[2](n)) + 1);
      w := exp((2*I*Pi)/m);
      Fourier(f, m, w)
  end proc;
      1
```

```
proc(f)
    local n, m, w;
    n := degree(f, x);
    m := 2^(floor(log[2](n)) + 1);
    w := exp((2*I*Pi)/m);
    Fourier(f, m, w)
end proc
```

(2.13)

```
> Fourierbis(f); 1
```

$$[4., 2.\,I, 0., -2.\,I] \tag{2.14}$$

```
> Fourierbis(h); 1
```

$$[13., 3. + 4.\,I, -3., 3. - 4.\,I] \tag{2.15}$$

```
> Fourierbis(p); 1
```

$$[36., 4.828427126 + 21.89949492\,I, -10. + 7.999999996\,I, -0.828427125 - 2.100505063\,I, 0., -0.828427126 + 2.100505058\,I, -10. - 7.999999996\,I, 4.828427125 - 21.89949492\,I] \tag{2.16}$$

```
>
```

Grâce à la FFT, on peut faire un algorithme permettant de calculer le produit de deux polynômes, avec une complexité plus petite que par Karatsuba : cela nous amène au chapitre suivant, multiplication rapide par FFT.

Multiplication rapide par FFT *(je n'ai pas réussi à écrire l'algo correctement : trouve un résultat différent de ce que je pensais)*

```
> w := e^(*(I π, 1/4)); 1
```

$$\frac{1}{2}\sqrt{2} + \frac{1}{2}\,\mathrm{I}\sqrt{2} \tag{3.1}$$

```
> Mult := proc(f, g, n, w)
      local F, G, h, K, i, v;
      v := combine(1 / w, exp);
      F := Fourier(f, n, w);
      G := Fourier(g, n, w);
      h := add(F[i] * G[i] * x^(i - 1), i = 1 ..n);
      K := Fourier(h, n, v);
      K := sum((K[i] * x^(i - 1)) / n, i = 1 ..n);
      return K
  end proc;
      1
```

```
proc(f, g, n, w)
    local F, G, h, K, i, v;
    v := combine(1 / w, exp);
    F := Fourier(f, n, w);
    G := Fourier(g, n, w);
    h := add(F[i] * G[i] * x^(i - 1), i = 1 ..n);
    K := Fourier(h, n, v);
    K := sum((K[i] * x^(i - 1)) / n, i = 1 ..n);
    return K
end proc
```

(3.2)

```
> Mult(f, h, 8, w); 1
```

$$4.000000006 + 2.500000000\,10^{-9}\,\mathrm{I} + (14.00000000 + 0.\,\mathrm{I})\,x + \left(17.00000000 + 2.000000001\,10^{-9}\,\mathrm{I}\right)x^2 + (10.00000000 + 0.\,\mathrm{I})\,x^3 + \left(4.999999999 - 2.500000000\,10^{-9}\,\mathrm{I}\right)x^4 + (2.000000000 - 0.\,\mathrm{I})\,x^5 + \left(1.250000000\,10^{-9} - 2.000000001\,10^{-9}\,\mathrm{I}\right)x^6 \tag{3.3}$$

```
> p; 1
```

$$x^6 + 5\,x^3 + 13\,x^2 + 2\,x^5 + 11\,x + x^4 + 3 \tag{3.4}$$

La complexité de cette méthode de calcul est en $O(n \cdot \log(n))$.
Il existe encore un autre algorithme encore plus efficace, l'algorithme de Shönhage et Strassen, qui est en $O(n \cdot \log(n) \cdot \log(\log(n)))$

Division rapide (ajout proposé par M. Jehanne)

La division euclidienne classique dans un anneau euclidien demande $O(n^2)$ *opérations pour des entrées de taille n. En fait, on peut la faire en* $O(M(n))$, *où M est un temps*

de multiplication, ce qui est bien mieux. Nous verrons le cas du polynôme, qui est le cas le plus simple.

```
> Rev:=proc(f,n)
  local g,G;
  G:=seq(coeff(f,x,n-i)*x^i,i=0..n);
  g:=sum(G[i],i=1..n+1);
  end;
```

```
proc(f, n)                                                    (4.1)
    local g, G;
    G := seq(coeff(f, x, n - i) * x^i, i = 0 ..n);
    g := sum(G[i], i = 1 ..1 + n)
end proc
```

```
> Inv:=proc(f,l)
  local g,r,i;
  g[0]:=1;
  r:=ceil(log(l));
  for i from 1 to r do
  g[i]:=rem((2*g[i-1]-f*g[i-1]^2),x^(2*i),x);
  od;
  return(g[r]);
  end;
```

```
proc(f, l)                                                    (4.2)
    local g, r, i;
    g[0] := 1;
    r := ceil(log(l));
    for i to r do
        g[i] := rem(2 * g[i - 1] - f * g[i - 1]^2, x^(2 * i), x)
    end do;
    return g[r]
end proc
```

```
>
```

```
> a:=5*x^5+4*x^4+3*x^3+2*x^2+x;
```

$$5\,x^5 + 4\,x^4 + 3\,x^3 + 2\,x^2 + x \qquad (4.3)$$

```
> b:=x^2+2*x+3;
```

$$x^2 + 2\,x + 3 \qquad (4.4)$$

```
> Div:=proc(a,b)
  local n,m,A,B,Q,q,r,k;
  n:=degree(a);k:=degree(b);
  if n<k then do q:=0;r:=a; return('q'=q,'r'=r);od;fi;
  m:=n-k;
  A:=Rev(a,n);
```

```
    B:=Inv(Rev(b,k),m+1);
    Q:=expand(rem(A·B,x^(m+1),x));
    q:=Rev(Q,m);
    r:=expand(a-q·b);
    return('q'=q,'r'=r);
    end;
proc(a, b)                                                    (4.5)
    local n, m, A, B, Q, q, r, k;
    n := degree(a);
    k := degree(b);
    if n < k then
        do
            q := 0;
            r := a;
            return 'q' = q, 'r' = r
        end do
    end if;
    m := n - k;
    A := Rev(a, n);
    B := Inv(Rev(b, k), 1 + m);
    Q := expand(rem(A* B, x^(1 + m), x));
    q := Rev(Q, m);
    r := expand(a - q* b);
    return 'q' = q, 'r' = r
end proc
> Div(a,b);
                    q = 20 - 6 x^2 + 5 x^3, r = -39 x - 60        (4.6)
>
>
```

L'algorithme d'inversion modulo x^l (*procédure* Inv(f,n)), *si l est égal à* 2^r (*puissance de* 2) , *utilise au plus* $3\,M(l)+l\in O(M(l))$ *opérations arithmétiques dans l'anneau euclidien.*
L'algorithme de division avec reste de a de degré n $+m$ *par un polynôme unitaire b de degré n, avec la relation* $n\geq m$, *peut être faite en* $4\,M(m)+M(n)+O(n)$ *opérations d'anneau.*