# Multiplication de grands entiers

par Alexandre Goyer et Nicholas Rumiz

Février 2020

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

# Multiplication de grands entiers

par Alexandre Goyer et Nicholas Rumiz

Février 2020

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

# Plan



UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Racines principales de l'unité

On se place dans un anneau $\mathbb{A}$ unitaire, commutatif et dans lequel 2 est inversible.

## Définition

Un élément $\alpha \in \mathbb{A}$ est *régulier* si pour tout $\beta$, $\alpha\beta = 0$ entraîne $\beta = 0$. Un élément $\omega \in \mathbb{A}$ est une *racine principale n-ième de l'unité* si $\omega^n = 1$ et si $\omega^t - 1$ est régulier pour tout $1 \leq t < n$.

## Lemme

Soient $\omega \in \mathbb{A}$ et $n \geq 2$ un puissance de deux. On a le résultat suivant.

$$\omega \text{ est une racine principale } n\text{-ième de l'unité} \iff \omega^{\frac{n}{2}} = -1.$$

université PARIS-SACLAY

# Transformée de Fourier discrète

Ainsi, dans la suite, on se place dans un anneau $\mathbb{A}$ unitaire, commutatif, dans lequel 2 est inversible et dans lequel on connait $\omega$ une racine $n$-ième principale de l'unité dans $\mathbb{A}$.

## La transformée de Fourier discrète

La *transformée de Fourier discrète* (abrégé **DFT** pour **D**iscrete **F**ourier **T**ransform en anglais) relativement à $\omega$ est l'opération d'évaluations suivante.

$$DFT_\omega : \left\{ \begin{array}{ccc} \mathbb{A}[X] & \longrightarrow & \mathbb{A}^n \\ P & \longmapsto & (P(1), P(\omega), ..., P(\omega^{n-1})) \end{array} \right.$$

# Transformée de Fourier discrète

En munissant $\mathbb{A}^n$ de la multiplication coordonnée par coordonnée, la transformée de Fourier est un morphisme surjectif de $\mathbb{A}$-algèbres. Moyennant les identifications classiques $\mathbb{A}[X]/(X^n - 1) \simeq \mathbb{A}[X]_{<n} \simeq \mathbb{A}^n$, on fait correspondre à l'opération $DFT_\omega$ l'automorphisme de $\mathbb{A}^n$ dont la matrice dans la base canonique est la suivante.

$$V(\omega, n) := \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{pmatrix}$$

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Transformée de Fourier discrète

## Théorème de Cooley-Tukey

Notons $d$ tel que $n = 2d$ et introduisons la matrice $D(\omega, d) := \mathrm{diag}(1, \omega, \ldots, \omega^{d-1}) \in \mathcal{M}_d(\mathbb{A})$. Alors on a l'identité matricielle suivante.

$$V(\omega, n) = \left(\begin{array}{c|c} I_d & D(\omega, d) \\ \hline I_d & -D(\omega, d) \end{array}\right) \times \left(\begin{array}{c|c} V(\omega^2, d) & 0_d \\ \hline 0_d & V(\omega^2, d) \end{array}\right) \times \Pi(n)$$

où $\Pi(n) \times {}^t(a_0, a_1, \ldots, a_{n-1}) = {}^t(a_0, a_2, \ldots, a_{n-2}, a_1, a_3, \ldots, a_{n-1})$.

## Théorème

$$DFT_\omega^{-1} = \frac{1}{n} DFT_{\omega^{-1}}$$

université PARIS-SACLAY

# Plan



UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Comment calculer efficacement le produit de deux entiers ?

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# En utilisant la FFT !

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Algorithme de Schönhage-Strassen

On fait ici le lien entre notre objectif initial, la multiplication de grands entiers, et la tranformée de Fourier rapide décrite à la section précédente.

## Multiplication de polynômes - squelette de l'algorithme (1/2)

**Entrées :**

- $n > 0$ un entier
- $\omega \in \mathbb{A}$ une racine principale $n$-ième de l'unité.
- $P, Q \in \mathbb{A}[X]$ tels que $deg(PQ) < n$

**Sortie :** $PQ$

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

## Multiplication de polynômes - squelette de l'algorithme (2/2)

**Étape 1** *Précalcul :* Calculs de $\omega^2, \omega^3, \ldots, \omega^{n-1}$

**Étape 2** *Évaluations :*

$$DFT_\omega(P) := (P(\omega^i))_{i=0}^{n-1}$$
$$DFT_\omega(Q) := (Q(\omega^i))_{i=0}^{n-1}$$

C'est dans cette étape qu'intervient l'algorithme de Cooley-Tukey (FFT).

**Étape 3** *Produits point à point :*

$$\begin{aligned} DFT_\omega(PQ) &= DFT_\omega(P) \times DFT_\omega(Q) \\ &= (P(\omega^i) \times Q(\omega^i))_{i=0}^{n-1} \end{aligned}$$

**Étape 4** *Interpolation :* $(PQ(\omega^i))_{i=0}^{n-1} \overset{DFT_\omega^{-1}}{\longmapsto} PQ$

$\longrightarrow$ *Cette étape revient à faire une évaluation sur les puissances successives de* $\omega^{-1}$.

## Algorithme de Schönhage-Strassen sur les entiers (1/3)

**Entrées :** $a, b \in \mathbb{N}$

**Sortie :** $ab$

**Les étapes :**

**Étape 1 :** *Calculs préliminaires :* Calculs de $u, v, k, K, M$ tels que :

- $u$ et $v$ nombres de bits respectifs de $a$ et $b$
- $k$ minimal tel que $2^k > \sqrt{u+v}$
- $K = 2^k$
- $M = \lceil \frac{u+v}{K} \rceil$ (partie entière supérieure)

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Algorithme de Schönhage-Strassen sur les entiers (2/3)

**Étape 2 :** *Décompositions en base* $2^M$ *:*

$$a = \sum_{i=0}^{K-1} a_i \left(2^M\right)^i$$

$$b = \sum_{i=0}^{K-1} b_i \left(2^M\right)^i$$

puis on considère les polynômes associés dans l'anneau $A_K[X]$ où $A_K = \mathbb{Z}/(2^{3K}+1)\mathbb{Z}$ :

$$A(X) = \sum_{i=0}^{K-1} \overline{a_i} X^i$$

$$B(X) = \sum_{i=0}^{K-1} \overline{b_i} X^i$$

## Algorithme de Schönhage-Strassen sur les entiers (3/3)

**Étape 3 :** *Calcul de* $C = AB$ *dans* $A_K[X]$ *:*
On exploite l'algorithme étudié précédemment avec $\omega = 2^6$ qui est une racine principale d'ordre $K$ de l'unité dans $A_K$
**Étape 4 :** *Évaluation :* $ab = C(2^M)$

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

# Plan



UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Pseudo-code de l'algorithme de Schönhage-Strassen

**multiplication** $(a, b)$

**Entrées** $a, b \in \mathbb{N}$
**Sortie** $ab \in \mathbb{N}$

* * * * *

$k, M \leftarrow$ **parametres** $(a, b)$
**Si** $k < 7$, **alors :**
    **Retourner** $a \times b$ (calculé avec un autre algorithme)
**Sinon :**
    $K \leftarrow 2^k$
    $L_a \leftarrow$ **decomposition** $(a, K, M)$
    $L_b \leftarrow$ **decomposition** $(b, K, M)$
    $\omega \leftarrow 2^6$
    $\text{mod} \leftarrow 2^{3K} + 1$
    $P_\omega, P_{\omega^{-1}} \leftarrow$ **precalcul** $(\omega, K, \text{mod})$
    $F_a \leftarrow \text{FFT}(L_a, K, \text{mod}, P_\omega, 0)$
    $F_b \leftarrow \text{FFT}(L_b, K, \text{mod}, P_\omega, 0)$
    $F_{ab} \leftarrow$ **point_par_point** $(F_a, F_b, K, \text{mod})$
    $L \leftarrow \text{FFT}(F_{ab}, K, \text{mod}, P_{\omega^{-1}}, 0)$
    $L_{ab} \leftarrow$ **div_par_K** $(L, K, \text{mod})$
    $ab \leftarrow$ **evaluation** $(L_{ab}, K, M)$
    **Retourner** $ab$

# Algorithme de Cooley-Tukey

## Théorème de Cooley-Tukey

Notons $d$ tel que $n = 2d$ et introduisons la matrice $D(\omega, d) := \mathrm{diag}(1, \omega, \ldots, \omega^{d-1}) \in \mathcal{M}_d(\mathbb{A})$. Alors on a l'identité matricielle suivante.

$$V(\omega, n) = \left(\begin{array}{c|c} I_d & D(\omega, d) \\ \hline I_d & -D(\omega, d) \end{array}\right) \times \left(\begin{array}{c|c} V(\omega^2, d) & 0_d \\ \hline 0_d & V(\omega^2, d) \end{array}\right) \times \Pi(n)$$

où $\Pi(n) \times {}^t(a_0, a_1, \ldots, a_{n-1}) = {}^t(a_0, a_2, \ldots, a_{n-2}, a_1, a_3, \ldots, a_{n-1})$.

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Pseudo-code de la FFT

**FFT** $(L, K, m, P, e)$

**Entrées** $K$ une puissance de 2

$m = 2^{3K} + 1$

$e \in \mathbb{N}$ tel que $N := \frac{K}{2^e} \geq 1$

$L = [q_0, q_1, \ldots, q_{N-1}]$ à coefficients dans $\mathbb{Z}/m\mathbb{Z}$

$P = [1, \omega, \ldots, \omega^{K-1}]$ où $w \in \mathbb{Z}/m\mathbb{Z}$ racine principale $K$-ième de l'unité

**Sortie** $[Q(1), Q(\alpha), \ldots, Q(\alpha^{N-1})]$ où $\alpha = \omega^{2^e}$ et $Q = q_0 + q_1 X + \cdots + q_{N-1} X^{N-1}$

* * * * *

**Si** $2^e < K$, **alors :**

$L \leftarrow \texttt{pi}(L, K, e)$

$L_1 \leftarrow$ « première moitié de $L$ »

$L_2 \leftarrow$ « deuxième moitié de $L$ »

$L_1 \leftarrow \text{FFT}(L_1, K, m, P, e+1)$

$L_2 \leftarrow \text{FFT}(L_2, K, m, P, e+1)$

$L_2 \leftarrow \texttt{diag}(L_2, K, m, P, e)$

$L \leftarrow \texttt{papillon}(L_1, L_2, m, e)$

# Plan



UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Coder « in place » : le choix des listes chaînées

L'idée est de gérer une seule liste tout au long de l'algorithme et d'appliquer des transformations au sein même de cette liste. On parle dans ce cas de « in place transform ».

On définit alors une nouvelle structure qu'on nomera « element » :

```
1 typedef struct element
2 {
3     mpz_t valeur;
4     struct liste * suivant;
5 }element;
```

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Un exemple : la fonction `pi` (1/2)

Montrons maintenant comment gérer ce genre de listes sur un exemple : l'implémentation de la fonction `pi` dont la transformation souhaitée est la suivante :

$$[a_0, a_1, a_2, a_3, \ldots, a_{K-2}, a_{K-1}] \longmapsto [a_0, a_2, \ldots, a_{K-2}, a_1, a_3, \ldots, a_{K-1}] .$$

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Un exemple : la fonction `pi` (2/2)

En langage C, on obtient :

```c
element *pi (element *liste, int K, int e)
/* La commande pi(L,K,e) sépare les K/(2^e) premiers elements de L
d indices pairs de ceux d indices impairs.                    */
{
    element *p1 = liste ;
    element *p2 = p1->suivant ;
    element *p_sauvegarde = p2 ;
    int i ;
    for (i=2 ; i < K/(1<<e) ; i++)
    {
        p1->suivant = p2->suivant ;
        p1 = p2 ;
        p2 = p1->suivant ;
    }
    p1->suivant = p_sauvegarde ;
}
```

# Plan



UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Complexité théorique de la fonction `fft`

Notons $C_K$ le coût total de la fonction `fft` sur un polynôme de degré $< K$ qui dénombrera le nombre d'opérations dans $\mathbb{A}$ et le nombre de modifications de pointeurs.

## Complexité de la fonction `fft`

Le nombre $C_K$ vérifie : $C_K = \mathcal{O}(K \log_2 K)$.

## Lemme

Soit $(u_n)_{n \geq 0}$ une suite croissante à termes positifs.

$$\text{Si } u_n \leq 2u_{\lceil \frac{n}{2} \rceil} + \mathcal{O}(n), \text{ alors } u_n = \mathcal{O}(n \log_2 n).$$

ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Complexité théorique de la fonction `fft`

## Démonstration.

`pi` : $< 3K$ modifications de pointeurs

`papillon` :

- $K$ additions/soustractions dans $\mathbb{A}$
- $< 2K$ modifications de pointeurs

`diag` :

- $K - 1$ multiplications dans $\mathbb{A}$
- $K - 1$ modifications de pointeurs

D'où finalement, l'inégalité suivante :

$$C_K \leq 2C_{\frac{K}{2}} + \mathcal{O}(K).$$

# Complexité théorique de la fonction `multiplication`

Notons $C(n)$ le nombre d'opérations binaires (ou modification de pointeurs) nécessaires pour multiplier deux entiers codés sur au plus $n$ bits avec la fonction `multiplication`.

Sachant que $K = \mathcal{O}(\sqrt{n})$, on montre que $\exists C > 0$ telle que :

$$\boxed{C(n) \leq C\sqrt{n}\,(\log n)^2\, C(24\sqrt{n}) + \mathcal{O}(n \log n)\,.}$$

On peut finalement énoncer le résultat général suivant sur la complexité de notre algorithme :

## Complexité de la fonction `multiplication`

Avec les notations précédentes, nous avons, pour tout $\epsilon > 0$ :

$$C(n) = \mathcal{O}(n^{1+\epsilon})\,.$$

# Plan



UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Cadre

Notre machine :

- Vitesse du processeur : $4 \times 1.8$ GHz
- Mémoire : 8Go DDR4 (2.4 GHz)

$\hookrightarrow$ On ne peut espérer pouvoir appliquer la fonction `fft` à un polynôme de degré $\geq 2^{28}$.

$\hookrightarrow$ On se place dans $\mathbb{A} = \mathbb{Z}/p\mathbb{Z}$ avec $p = 7 \times 2^{26} + 1$ premier.
On a $\omega = 30$ qui est une racine $2^{26}$-ième principale de l'unité dans $\mathbb{A}$ donc on peut appliquer `fft` dans $\mathbb{A}$ pour des polynômes de degré $< K$ jusqu'à $K = 2^{26}$ (au moins en théorie).

**Expérience** Appliquer `fft` à des polynômes de degrés $d = 2^n$ pour $n$ allant jusqu'à 26, et évaluer les temps d'exécution.

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

# Résultat

Notons $T(d)$ le temps moyen (sur 5 tests) que met notre machine pour calculer la FFT d'un polynôme de degré $< d$ dans $\mathbb{A}$.

| n = | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|
| T($2^n$) (en sec.) | 0.0088 | 0.0194 | 0.0426 | 0.0894 | 0.1984 | 0.4234 |

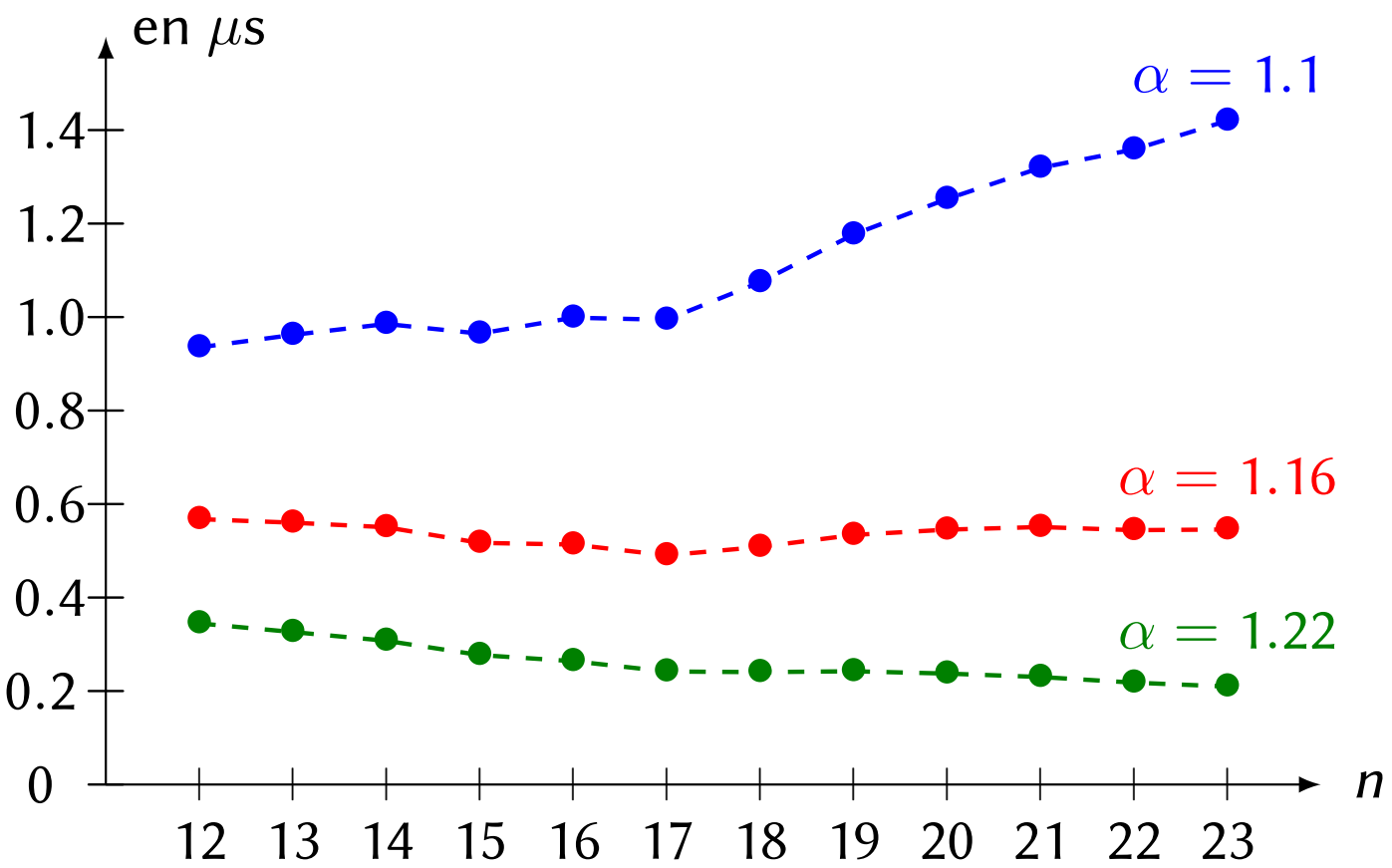
| n = | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|
| T($2^n$) (en sec.) | 0.9802 | 2.3022 | 5.2538 | 11.8614 | 26.2804 | 58.6598 |

$\hookrightarrow$ Le rapport entre le temps pour $2^{n+1}$ et celui pour $2^n$ est presque constant et vaut en moyenne 2.23.
Posons $\alpha_0 = 1.16 \simeq \log_2(2.23)$.

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

# Analyse du comportement

Graphe de la courbe $n \longmapsto \dfrac{T(2^n)}{(2^n)^\alpha}$ pour différentes valeurs de $\alpha$.




UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

# Annexe

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

# Complexité de la fonction `multiplication`

- `decomposition` : $\mathcal{O}(MK) = \mathcal{O}(n)$ opérations
- `parametres` : $\mathcal{O}(\log_2(n)^2 \log_2(\log_2(n)))$ opérations
- `precalcul` : $< C_1\ K \log(24\sqrt{n})\ C(24\sqrt{n})$ opérations
- `point_par_point` : $< C_2\ K \log(K)\ C(24\sqrt{n})$ opérations
- `evaluation` : $\mathcal{O}(MK) = \mathcal{O}(n)$ opérations
- `div_par_K` : $< C_3\ K \log(24\sqrt{n})\ C(24\sqrt{n})$ opérations
- `fft` : $\mathcal{O}(K \log(K) \log(24\sqrt{n})\ C(24\sqrt{n}))$ opérations

$$\Longrightarrow \boxed{C(n) \leq C\ \sqrt{n}\ (\log n)^2\ C(24\sqrt{n}) + \mathcal{O}(n)}$$

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY