# Diviser pour Régner, II

## I. Le tri fusion

Rappelons que pour trier une liste selon l'algorithme du tri fusion, on coupe une liste en deux, on trie chaque sous-liste, puis on fusionne les deux sous-listes triées. Une implémentation en OCaml est la suivante :

```
let rec partitionner l =
  match l with
  | [] -> ([], [])
  | [ a ] -> ([ a ], [])
  | a :: b :: l' ->
      let l1, l2 = partitionner l in
      (a :: l1, b :: l2)
;;

let rec fusionner l1 l2 =
  match (l1, l2) with
  | [], _ -> l2
  | _, [] -> l1
  | a :: l1', b :: l2' ->
      if a <= b then
        a :: fusionner l1' l2
      else
        b :: fusionner l1 l2'
;;

let rec tri_fusion l =
  match l with
  | [] -> []
  | [ _ ] -> l
  | _ ->
      let l1, l2 = partitionner l in
      fusionner (tri_fusion l1) (tri_fusion l2)
;;
```

Il est clair que la complexité de `partitionner` est en $\Theta(n)$, et que `fusionner` est en $\Theta(n_1 + n_2)$.

Ainsi, si $c_n$ désigne la complexité de `tri_fusion` pour une liste de longueur $n$, alors la suite $(c_n)$ vérifie la relation de récurrence :

$$\forall\, n \geq 2,\ c_n = \underbrace{\Theta(n)}_{\texttt{partitionner}} + c_{\lceil \frac{n}{2} \rceil} + c_{\lfloor \frac{n}{2} \rfloor} + \underbrace{\Theta(n)}_{\texttt{fusionner}}$$

On peut la mettre sous la forme

$$c_n = a c_{\lceil \frac{n}{2} \rceil} + b c_{\lfloor \frac{n}{2} \rfloor} + \Theta(n^\alpha)$$

avec $a + b = 2$ et $\alpha = 1$. Comme $\log_2(a+b) = \alpha$, on en déduit que :

$$c_n = \Theta(n \log n)$$

# II. Produits efficaces

## 1. Produit d'entiers, algorithme de Karatsuba

L'algorithme naïf pour multiplier deux entiers de taille $n$ (la taille étant le logarithme du nombre, c'est-à-dire à un facteur près le nombres de chiffres dans une base $b \geq 2$ fixée) est en $O(n^2)$.

On peut tenter une approche « diviser pour régner » en coupant les entiers en deux. Autrement dit, écrire un entier de taille $2n$ sous la forme $n = a \times 10^n + b$ (je suis ici en base 10). On a alors :

$$\left(a \times 10^n + b\right) \cdot \left(c \times 10^n + d\right) = a \underset{\uparrow}{\cdot} c \times 10^{2n} + (a \underset{\uparrow}{\cdot} d + b \underset{\uparrow}{\cdot} c) \times 10^n + b \underset{\uparrow}{\cdot} d$$

On a alors 4 produits de taille moitié, d'où un algorithme dont la complexité vérifie :

$$c_n = 4 c_{\frac{n}{2}} + \Theta(n)$$

d'où $c_n = \Theta(n^2)$, puisque $\log_2 4 = 2 > 1$. On ne gagne rien.

Mais comme $(a-b)(c-d) = ac - ad - bc + bd$, on peut écrire :

$$\begin{aligned} &\left(a \times 10^n + b\right) \cdot \left(c \times 10^n + d\right) = \\ &\qquad a \underset{\uparrow}{\cdot} c \times 10^{2n} + \left(a \cdot c + b \cdot d - (a-b) \underset{\uparrow}{\cdot} (c-d)\right) \times 10^n + b \underset{\uparrow}{\cdot} d \end{aligned}$$

On n'a plus que 3 produits de taille moitié à faire. La relation de récurrence devient alors :

$$c_n = 3 c_{\frac{n}{2}} + b_n \qquad \text{avec} \qquad b_n = \Theta(n^1)$$

On compare donc cette fois $\log_2 3$ et 1. Comme $log_2 3 > 1$, on en déduit que

$$c_n = \Theta\left(n^{\log_2 3}\right)$$

On passe donc d'un exposant 2 à un exposant $\log_2 3 \simeq 1.5849625$. Pour $n = 10\,000$, on passe de $n^2 = 100\,000\,000$ opérations à $n^{1.5849625} \simeq 2\,187\,000$.

## 2. Produits de matrices, algorithme de Strassen

Pour un produit de deux matrices de tailles respectifs $m \times n$ et $n \times p$, on ne connaît pas d'algorithme qui fait mieux que l'algorithme naïf en $\Theta(mnp)$.

Pour des matrices carrées, on peut faire mieux. Décomposons

$$A = \left(\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right) \qquad B = \left(\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \end{array}\right) \qquad C = \left(\begin{array}{c|c} C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \end{array}\right)$$

Si $C = AB$, alors on a :

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & \qquad C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & \qquad C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

On a 8 produits de taille moitié, d'où une relation de complexité de la forme

$$c_n = 8c_{\frac{n}{2}} + \Theta(n^2)$$

qui se résoud en $c_n = \Theta(n^3)$.

Or, Volker Strassen a remarqué, en 1969, que :

$$\begin{aligned} C_{1,1} &= M_1 + M_4 - M_5 + M_7 & \qquad C_{1,2} &= M_3 + M_5 \\ C_{2,1} &= M_2 + M_4 & \qquad C_{2,2} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

où l'on définit

$$\begin{aligned} M_1 &= \big(A_{1,1} + A_{2,2}\big)\big(B_{1,1} + B_{2,2}\big) & \qquad M_2 &= \big(A_{2,1} + A_{2,2}\big)B_{1,1} \\ M_3 &= A_{1,1}\big(B_{1,2} - B_{2,2}\big) & \qquad M_4 &= A_{2,2}\big(B_{2,1} - B_{1,1}\big) \\ M_5 &= \big(A_{1,1} + A_{1,2}\big)B_{2,2} & \qquad M_6 &= \big(A_{2,1} - A_{1,1}\big)\big(B_{1,1} + B_{1,2}\big) \\ M_7 &= \big(A_{1,2} - A_{2,2}\big)\big(B_{2,1} + B_{2,2}\big) \end{aligned}$$

Chacun des $M_i$ nécessite un seul produit, donc avec cette décomposition, la relation de complexité devient :

$$c_n = 7c_{\frac{n}{2}} + b_n \qquad \text{avec} \qquad b_n = \Theta(n^2)$$

Ainsi, la complexité obtenue est :

$$c_n = \Theta\big(n^{\log_2 7}\big)$$

avec $\log_2 7 \simeq 2,807$. C'est mieux que l'algorithme naïf.

Notons que la constante multiplicative fait que cet algorithme ne devient plus efficace que pour des matrices de taille de l'ordre de plusieurs centaines.