INF4705 — Analyse et conception d'algorithmes

CHAPITRE 4 : ALGORITHMES DIVISER-POUR-RÉGNER

Nous en avons déjà vu. Voici deux rappels :

Exemple 1 *Fouille dichotomique*
Posons la récurrence puis résolvons directement grâce à la formule ($\star$) *de la section 3.3.1 :*

$$T(n) = \begin{cases} c_1 & \text{si } n = 1;\ c_1 > 0 \\ T(n/2) + c_2 & \text{sinon};\ c_2 > 0,\ n \text{ puissance de 2} \end{cases}$$

$T(n) \in \Theta(n^0 \lg n) = \Theta(\lg n)$, *puisque* $\ell = 1, b = 2, k = 0$ *et* $T(n)$ *éventuellement non décroissante.*

(Note : À strictement parler, il s'agit d'un algorithme de simplification *puisqu'on ramène la résolution d'un exemplaire à celle d'*un seul *exemplaire de plus petite taille.)*

Exemple 2 *Multiplication d'entiers*
Posons la récurrence puis résolvons directement grâce à la formule ($\star$) *de la section 3.3.1 :*

$$T(n) = \begin{cases} h(n) & \text{si } n \leq n_0 \\ 3T(n/2) + g(n) & \text{sinon; } n \text{ puissance de 2} \end{cases}$$

où $h(n) \in \Theta(n^2)$ *et* $g(n) \in \Theta(n)$.

$$T(n) \in \Theta(n^{\lg 3} \simeq n^{1.58} \mid n \text{ puissance de 2}), \quad \text{puisque } \ell = 3, b = 2, k = 1$$

Principe :
On ramène la résolution d'un exemplaire de problème à celle de plusieurs exemplaires de plus petite taille dont les solutions servent à obtenir une solution à l'exemplaire originel.

Schéma général :

fonction diviser-pour-regner(x {exemplaire}) : y {solution}
  si x est petit ou simple **alors retourner** algo-simple(x) ;
  décomposer x en sous-exemplaires $x_1, \ldots, x_\ell$;
  pour $i = 1$ **à** ℓ **faire**
    $y_i \leftarrow$ diviser-pour-regner(x_i) ;
  recombiner les y_i en une solution y pour x ;
  retourner y

Trois décisions à prendre :

• **Quand est x "petit ou simple" ?** détermination d'un seuil adéquat

Exemple 3
Considérons

$$T(n) = \begin{cases} h(n) & \text{si } n \leq n_0 \\ 3T(\lceil n/2 \rceil) + g(n) & \text{sinon} \end{cases}$$

Si $h(n) = n^2\ \mu s$ *et* $g(n) = 16n\ \mu s$, *la multiplication d'entiers à 5000 chiffres prend :*
– $\sim 25s$ *de façon classique ;*
– $\sim 41s$ *en diviser-pour-régner avec* $n_0 = 1$ *;*
– $\sim 6s$ *en diviser-pour-régner avec* $n_0 = 64$.

Il y a pire : "classique" est préférable pour $n = 67$ alors que "diviser-pour-régner" est préférable pour $n = 66$. On voudrait donc un seuil $n_0 = 67$ pour les exemplaire de taille 67 mais plutôt un seuil, disons, $n_0 = 64$ pour les exemplaires de taille 66 ! La leçon : même pour une implantation particulière, il n´y a pas de seuil optimal *pour toute taille d´exemplaire.*
Soit en général $h(n) = (an^2 + bn + c)\mu s$ et $g(n) = (dn + e)\mu s$.

Deux approches possibles :

Méthode hybride : 1. *Évaluer empiriquement les constantes de l'équation de récurrence (ici a, b, c, d, e).*
2. *Déterminer un seuil n_0 pour lequel $T(n_0)$ est sensiblement le même qu'on utilise "algo-simple" directement ou après un niveau de récursivité :*

$$an_0^2 + bn_0 + c \;\simeq\; 3(a\lceil n_0/2\rceil^2 + b\lceil n_0/2\rceil + c) + dn_0 + e$$

Méthode essai-erreur *On cherche un seuil qui donne un meilleur temps de calcul que les seuils voisins (par exemple, que sa moitié et son double).*

- **Comment décomposer x ?** nombre de morceaux, de taille équilibrée

Exemple 4

$$T(n) = T(\lambda n) + T((1-\lambda)n) + cn, \;\; 0 < \lambda < 1$$

– *si $\lambda = 1/2$ (parfaitement équilibré) :*

$$T(n) = 2T(n/2) + cn \longrightarrow T(n) \in \Theta(n \lg n)$$

– *si $\lambda = 1/n$ (parfaitement déséquilibré) :*

$$T(n) = T(1) + T(n-1) + cn \longrightarrow T(n) \in \Theta(n^2)$$

- **Comment recombiner les solutions des sous-exemplaires ?** une façon efficace de profiter du travail accompli

Exemple 5

Diviser-pour-régner appliqué au problème du sac a dos ?
On divise les objets en deux groupes ? On considère pour chacun un sac de mi-capacité ?
Comment se servir des sous-solutions obtenues ? ?

4.1 Tri

Soit un tableau de n entiers (en général des éléments sur lesquels est défini un ordre total). Nous voulons arranger ces entiers en ordre non décroissant dans ce tableau.

Il existe de nombreux algorithmes pour résoudre ce problème très étudié parce que si répandu. Vous en connaissez d´ailleurs plusieurs parmi : tri en bulle, tri par insertion, tri par sélection, tri en monceau (de Williams, ¨heapsort¨), ¨shellsort¨ et même, lorsque la taille des nombres est bornée, le tri du pigeonnier, ¨bin sort¨ et ¨radix sort¨.

En voici deux autres, que vous connaissez probablement déjà, mais qui sont de bons exemples d´application fort différente de la technique diviser-pour-régner au même problème.

4.1.1 Tri par fusion

idée : On sépare les entiers en deux parties à peu près égales qu'on trie récursivement, puis on fusionne les suites triées en préservant l'ordre. Si n est petit, on utilise par exemple un tri par insertion.

Ici la décomposition est facile et elle est en plus équilibrée ; la recombinaison, elle, est moyennement facile. Le gros du travail est fait *après* les appels récursifs : on applique un algorithme (vorace ?) de fusion de deux listes triées en temps linéaire.

analyse de la complexité temps :

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + g(n), \;\; g(n) \in \Theta(n)$$

$$\vdots$$

$$T(n) \in \Theta(n \lg n) \;\; \text{en pire cas}$$

analyse de la complexité espace : La fusion utilise un espace mémoire supplémentaire de taille n ; la complexité totale est donc $\Theta(n)$.

4.1.2 Tri de Hoare (¨quicksort¨)

idée : On sépare les entiers de part et d'autre d'un pivot et on trie récursivement chaque partie, puis . . .c'est tout ! Si n est petit, on utilise par exemple un tri par insertion.

Ici la décomposition efficace est moyennement facile mais la recombinaison facile.
Une décomposition équilibrée est par contre difficile.
Le gros du travail est fait *avant* les appels récursifs : on peut séparer les éléments de part et d´autre du pivot efficacement (en temps linéaire) et *in situ*.

Comment choisir le pivot :

choix	équilibre	coût
quelconque	aucune garantie	¢
médiane	parfait	$$$
médiane parmi K, K petit	un compromis	$
moyenne	un autre compromis	$$

Le tri de Hoare choisit arbitrairement le premier élément du (sous-)tableau comme pivot.

Équilibre des sous-problèmes :
Ironiquement, le pire cas survient lorsque le tableau est déjà trié !

$$T(n) = T(0) + T(n-1) + g(n), \;\; g(n) \in \Theta(n) \;\Rightarrow\; T(n) \in \Theta(n^2)$$

Donc tri de Hoare $\in \Omega(n^2)$ en pire cas.

Et en moyenne ?
Évaluons cela, sous deux hypothèses :
- Les n éléments du tableau sont distincts.
- Les $n!$ permutations initiales sont équiprobables.
  ($\Rightarrow$ les permutations dans les sous-tableaux le sont aussi.)

Soit $T(m)$ le temps d'exécution moyen de `quicksort`(A[i..j]), où $m = j - i + 1$, le nombre d'éléments du sous-tableau.
Par nos hypothèses, toutes les positions ℓ possibles du pivot après l'appel `pivot`(A[1..n],ℓ) sont équiprobables (puisque toutes les identités du pivot le sont).
Donc,

$$T(n) = \frac{1}{n} \sum_{\ell=1}^{n} (g(n) + T(\ell - 1) + T(n - \ell)), \quad g(n) \in \Theta(n) \;\; \text{et } n \text{ suffisamment grand.}$$

Soient n_0 notre seuil et d une constante réelle positive telle que $g(n) \leq dn$, $\forall n > n_0$:

$$T(n) \leq dn + \frac{1}{n} \sum_{\ell=1}^{n} (T(\ell - 1) + T(n - \ell)), \quad n > n_0$$

mais $\sum_{\ell=1}^{n}(T(\ell-1)+T(n-\ell)) = (T(0)+T(n-1))+(T(1)+T(n-2))+\cdots+(T(n-2)+T(1))+(T(n-1)+T(0))$

$$T(n) \leq dn + \frac{2}{n}\sum_{k=0}^{n-1} T(k), \quad n > n_0$$

puisque chaque $T(k),\ 0 \leq k \leq n-1$ apparaît exactement deux fois dans la sommation.
Les techniques de résolution de relations de récurrence que nous avons vues ne s'appliquent pas ici. On peut espérer que sa complexité en moyenne sera dans l'ordre de $n \lg n$, comme pour le tri par fusion.

preuve par induction constructive :
On prouve par induction mathématique un énoncé partiellement spécifié ; en cours de preuve, nous accumulerons des contraintes sur cette spécification qui nous permettront finalement de préciser l'énoncé initial.

Théorème 1 *Le tri de Hoare sur n entiers prend en moyenne un temps dans l'ordre de $n \lg n$ (c.-à-d. $T(n) \leq c \cdot n \lg n,\ n \geq 2$ pour un certain c non spécifié).*

Démonstration 1 *Supposons s.p.d.g. que $n_0 \geq 2$.*

base de l'induction : $2 \leq n \leq n_0$

On doit montrer que $T(n) \leq c \cdot n \lg n$. Nous n'avons qu'un nombre fini de cas, d'où on tire la première contrainte sur c :

$$c \geq \frac{T(n)}{n \lg n} \ \ \forall\, 2 \leq n \leq n_0$$

hypothèse d'induction généralisée : $T(k) \leq c \cdot k \lg k \ \ \forall\, 2 \leq k < n$.

pas d'induction : *Considérons $n > n_0$ et soit $a = T(0) + T(1)$.*

$$\begin{aligned}
T(n) &\leq dn + \frac{2}{n}\sum_{k=0}^{n-1} T(k) \\
&= dn + \frac{2}{n}\left(T(0) + T(1) + \sum_{k=2}^{n-1} T(k)\right) \\
&\leq dn + \frac{2a}{n} + \frac{2}{n}\sum_{k=2}^{n-1} c \cdot k \lg k \quad \textit{par l'H.I.} \\
&\leq dn + \frac{2a}{n} + \frac{2c}{n}\int_2^n x \lg x \, dx \\
&= dn + \frac{2a}{n} + \frac{2c}{n}\left[\frac{x^2 \lg x}{2} - \frac{x^2}{4}\right]_{x=2}^{n} \\
&= dn + \frac{2a}{n} + \frac{2c}{n}\left(\frac{n^2 \lg n}{2} - \frac{n^2}{4} - \frac{4 \lg 2}{2} + \frac{4}{4}\right) \\
&< dn + \frac{2a}{n} + \frac{2c}{n}\left(\frac{n^2 \lg n}{2} - \frac{n^2}{4}\right) \quad \textit{puisque } \lg 2 > 0.69 \\
&= dn + \frac{2a}{n} + cn \lg n - \frac{cn}{2} \\
&= cn \lg n - \left(-d - \frac{2a}{n^2} + \frac{c}{2}\right) n.
\end{aligned}$$

Donc $T(n) \leq cn \lg n$ en autant que $-d - \frac{2a}{n^2} + \frac{c}{2} \geq 0$ c.-à-d. $c \geq 2d + \frac{4a}{n^2}$.
Puisqu'on considérait $n > n_0$, on obtient comme seconde contrainte sur c :

$$c \geq 2d + \frac{4a}{(n_0+1)^2}$$

Il suffit donc de prendre par exemple

$$c = \max\left(2d + \frac{4(T(0)+T(1))}{(n_0+1)^2}, \max\{\frac{T(n)}{n\lg n} \mid 2 \leq n \leq n_0\}\right)$$

pour conclure que $T(n) \leq c \cdot n\lg n \; \forall n \geq 2$ *et donc que* $T(n) \in \mathcal{O}(n\lg n)$.

$\surd$

Note : On l'appelle "quicksort" parce qu'en pratique la constante cachée est plus petite que celles des tri de Williams (heapsort) et tri par fusion.

analyse de la complexité espace : On serait tenté de croire que ce tri s´effectue *in situ* et n´a donc pas besoin d´espace supplémentaire mais il faut quand même conserver des marqueurs délimitant les sous-tableaux qui n´ont pas encore été triés, ce qui demande en pire cas $\Theta(n)$ espace mémoire supplémentaire. (Note : On peut toutefois réduire celui-ci à $\Theta(\lg n)$ par élimination de la récursivité de queue.) De toute façon, la complexité totale demeure dans $\Theta(n)$.

4.2 Médiane et sélection

Étant donné un tableau de n entiers, on sait facilement trouver leur somme, leur moyenne, leur plus grand ou plus petit élément en temps $\Theta(n)$.

Et si on cherche le $k^{\text{ème}}$ plus grand ? C´est le problème de *sélection*, dont un cas particulier ($k = \lfloor n/2 \rfloor$) est le problème de la *médiane*.

- On pourrait d´abord trier le tableau puis le parcourir en ordre ascendant jusqu´au $k^{\text{ème}}$: $\Theta(n\lg n)$ en pire cas.
- On pourrait construire un monceau puis retirer k éléments : $\Theta(n) + \Theta(k\lg n)$ en pire cas donc $\Theta(n)$ si $k < n/\lg n$ mais $\Theta(n\lg n)$ si $k = \lfloor n/2 \rfloor$.

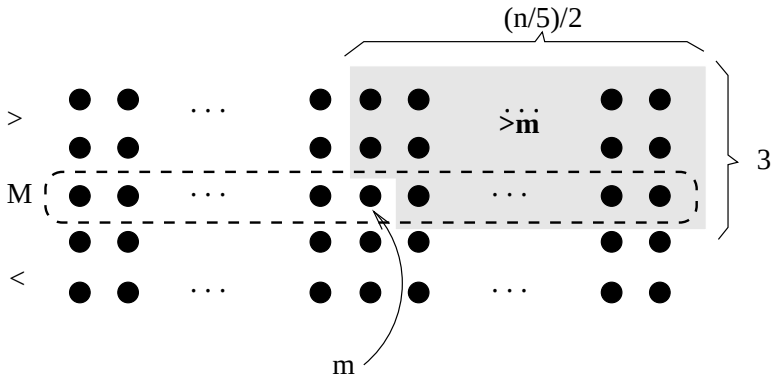
Voyons comment faire en temps $\Theta(n)$ en pire cas, ce qui est optimal. Si à chaque appel on pouvait se débarrasser d´au moins une fraction constante du tableau (comme pour la fouille dichotomique), menant par exemple à une récurrence du genre

$$T(n) = \begin{cases} c_1 & \text{si } n \leq n_0 \\ T(9n/10) + c_2 n & \text{sinon} \end{cases},$$

on obtiendrait le résultat voulu, $T(n) \in \Theta(n)$.

Ébauche de l´algorithme sélection(S,k) **:** (nous supposerons pour simplifier la présentation que les éléments sont distincts)

1. Diviser les éléments en groupes de cinq (avec possiblement un dernier groupe contenant moins d'éléments).
2. Trier chacun des groupes puis rassembler l´élément médian de chacun (le troisième) dans M.
3. $m \leftarrow$ sélection$(M, \lceil\lceil n/5\rceil/2\rceil)$ (on cherche la médiane des médianes).
4. Séparer les éléments de part et d´autre de m en deux ensembles $S_<$ (plus petits que m) et $S_>$ (plus grands).
5. (a) Si $k = |S_<| + 1$ alors retourner m.
   (b) Si $k < |S_<| + 1$ alors retourner sélection$(S_<, k)$.
   (c) Si $k > |S_<| + 1$ alors retourner sélection$(S_>, k - (|S_<| + 1))$.



analyse de la complexité temps :

1. $\Theta(n)$
2. $\lceil n/5 \rceil \times \Theta(1) \Rightarrow \Theta(n)$
3. $T(\lceil n/5 \rceil)$
4. $\Theta(n)$. On aura au plus approximativement $7n/10$ éléments dans $S_<$ et dans $S_>$.
5. $\sim T(7n/10)$

Nous obtenons donc

$$T(n) \simeq T(\lceil n/5 \rceil) + T(7n/10) + \Theta(n) \Rightarrow \ldots \Rightarrow T(n) \simeq T(9n/10) + cn.$$

Note : En utilisant cet algorithme pour choisir la médiane comme pivot dans le tri de Hoare, on obtient une décomposition équilibrée[1] et donc un comportement $\mathcal{O}(n \lg n)$ en pire cas. Par contre en pratique la constante multiplicative cachée associée au calcul de la médiane fait préférer un choix de pivot plus simple.

4.3 Dominance dans un ensemble de points

Soient deux points $p = (x_1^p, \ldots, x_k^p)$ et $q = (x_1^q, \ldots, x_k^q)$ dans un espace à k dimensions. On dit que p *domine* q si $x_i^p > x_i^q \ \forall i \in \{1, \ldots, k\}$. Le *problème de dominance* pour un ensemble S de n points consiste à déterminer pour chaque point de S combien d´autres points de cet ensemble il domine. Ce problème a des applications importantes en statistiques pour vérifier des hypothèses sur des distributions.

$\mathbf{k = 1}$:

En une dimension, il suffit de trier les points puis de les parcourir en ordre ascendant : le premier domine 0 point, le second domine 1 point, ..., le $i$ème domine $i - 1$ points, ...

$\mathbf{k > 1}$:

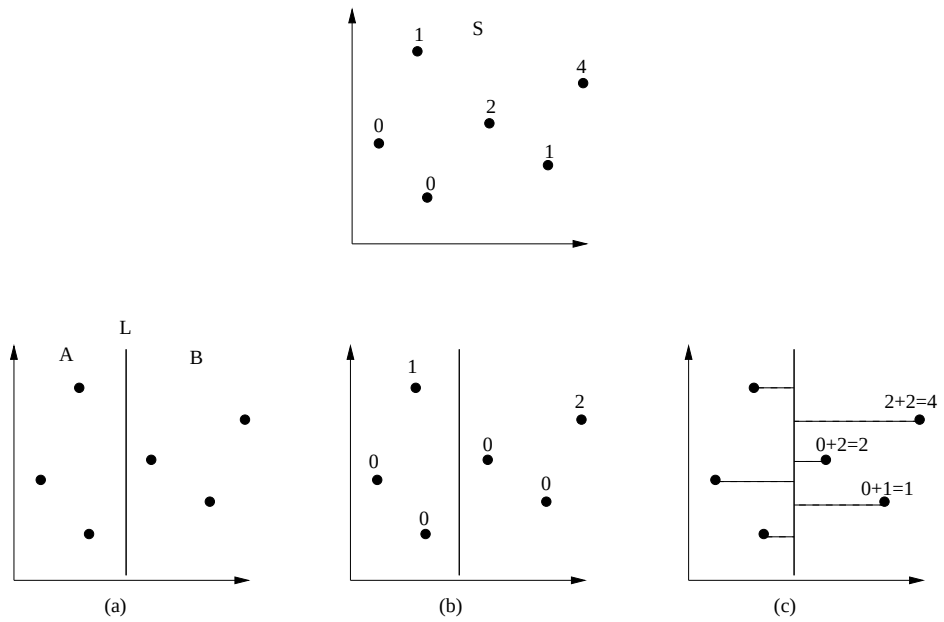
Nous résoudrons récursivement deux problèmes en k dimensions mais de taille $n/2$ puis un problème en $k - 1$ dimensions de taille n.

Voyons d´abord en deux dimensions. Nous décomposons le problème (S) en deux sous-problèmes (A et B) parfaitement équilibrés en calculant la médiane en x puis en départageant les points de part et d´autre de celle-ci (a). Ces sous-problèmes sont résolus récursivement (b). Comment recombiner les sous-solutions ?

- Clairement aucun point de A ne domine un point de B : donc les valeurs calculées pour les points de A sont celles de la solution de S.
- Pour tout point de B, les points de A qu´il domine sont exactement ceux qui ont une coordonnée en y qui lui est inférieure : il suffit donc de projeter tous les points sur la ligne L puis de la parcourir de bas en haut en notant le nombre de points de A qu´on a rencontré et en ajoutant ce nombre au compte de chaque point de B au moment où on le rencontre (c).

[1] Il faut cependant modifier légèrement la procédure répartissant les éléments de part et d´autre du pivot. On les sépare plutôt en trois parties : $<$, $=$ et $>$ que le pivot. Pourquoi ?

Exemple 6



analyse de la complexité temps : Nous savons calculer la médiane en un temps dans $\Theta(n)$. Départager les points est également dans $\Theta(n)$. Si $T(n)$ est le temps pris par tout l´algorithme sur S, les deux sous-problèmes prennent chacun un temps $T(n/2)$. Afin d´accélérer la recombinaison, il est avantageux de trier S en prétraitement selon la coordonnée en y puis de maintenir cet ordonnancement tout au long de l´algorithme. Nous payons un coût initial de $\Theta(n \lg n)$ mais la recombinaison s´effectue alors en $\Theta(n)$.

Le coût total de l´algorithme est donc $\Theta(n \lg n) + T(n)$ où

$$T(n) \leq 2T(n/2) + cn.$$

Puisque $T(n) \in \mathcal{O}(n \lg n)$, on en conclut que la complexité en temps de calcul est dans $\Theta(n \lg n)$.

Cet algorithme se généralise en k dimensions. L´étape de recombinaison correspond alors à un problème de dominance en $k-1$ dimensions, que nous savons déjà résoudre[2]. On peut prouver par induction que sa complexité en temps de calcul est dans $\mathcal{O}(n \lg^{k-1} n)$.

4.4 Multiplication de matrices

Soient $A = (a_{ij})$ et $B = (b_{ij})$ deux matrices $n \times n$ et $C = (c_{ij})$ leur produit. La façon usuelle de calculer C,

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj},$$

mène à une complexité dans $\Theta(n^3)$, en supposant que la multiplication et l'addition scalaires sont des opérations élémentaires.

L´algorithme de Strassen arrive à le faire en $\Theta(n^{\lg_2 7})$ ($\lg_2 7 \simeq 2.81$). Comme chez la multiplication de grands entiers, on arrive à ¨sauver¨ des multiplications : on peut multiplier deux matrices 2×2 en effectuant 7 multiplications scalaires plutôt que 8 :

[2] Il faut cependant modifier un peu l´algorithme car on ne s´intéresse qu´à la dominance des points de B sur ceux de A.

$$
\begin{aligned}
c_{11} &= m_2 + m_3 \\
c_{12} &= m_1 + m_2 + m_5 + m_6 \\
c_{21} &= m_1 + m_2 + m_4 - m_7 \\
c_{22} &= m_1 + m_2 + m_4 + m_5
\end{aligned}
\qquad \text{où} \qquad
\begin{aligned}
m_1 &= (a_{21} + a_{22} - a_{11}) \times (b_{22} - b_{12} + b_{11}) \\
m_2 &= a_{11} \times b_{11} \\
m_3 &= a_{12} \times b_{21} \\
m_4 &= (a_{11} - a_{21}) \times (b_{22} - b_{12}) \\
m_5 &= (a_{21} + a_{22}) \times (b_{12} - b_{11}) \\
m_6 &= (a_{12} - a_{21} + a_{11} - a_{22}) \times b_{22} \\
m_7 &= a_{22} \times (b_{11} + b_{22} - b_{12} - b_{21})
\end{aligned}
$$

L´algorithme découpe donc chaque matrice en quatre parties égales (donnant une pseudo-matrice 2×2), résout récursivement sept sous-problèmes puis recombine les solutions par plusieurs additions prenant un temps dans $\Theta(n^2)$:

$$T(n) \leq 7T(n/2) + cn^2.$$

Par la suite, on arriva à faire mieux (asymptotiquement) en ¨sauvant¨ des multiplications pour des matrices 70×70, donnant une complexité $\Theta(n^{\log_{70} 143640})$ plutôt que $\Theta(n^{\log_{70} 343000}) = \Theta(n^3)$ ($\log_{70} 143640 \simeq 2.795$). D´autres suivirent . . .nous en sommes à un exposant d´environ 2.376.