Partie 2. Techniques algorithmiques

5. Diviser pour régner

Bruno Grenet

Université Grenoble Alpes – IM²AG
L3 Mathématiques et Informatique
UE Algorithmique

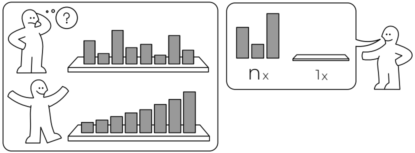
https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html

Table des matières

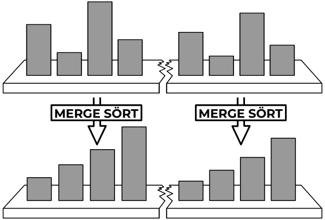
Table des matières

MERGE SÖRT

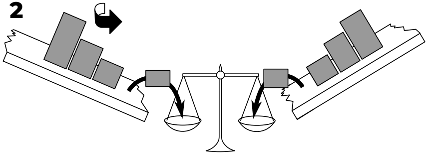
idea-instructions.com/merge-sort/
v1.1, CC by-nc-sa 4.0



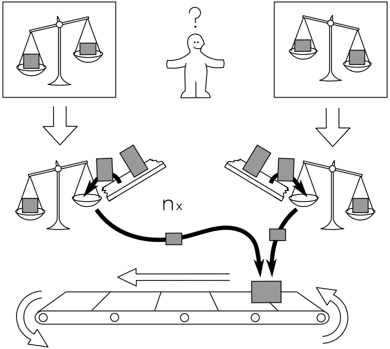
1



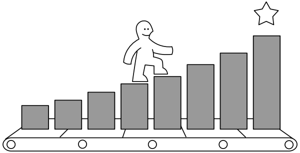
2



3



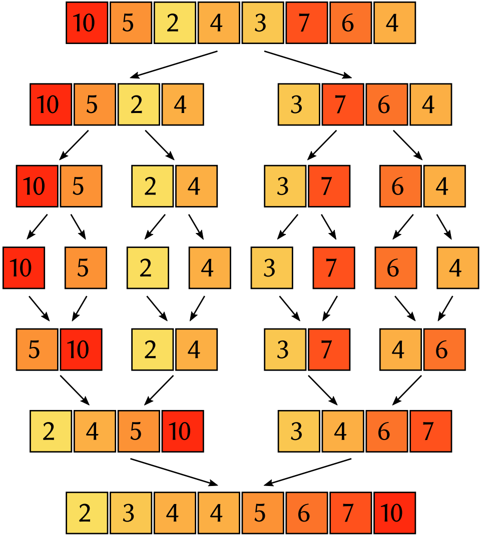
4



Algorithme du TRIFUSION

TRIFUSION(T):

1. $n \leftarrow \#T$
2. Si $n \leq 1$: Renvoyer T
3. Sinon :
4. $T_1 \leftarrow$TRIFUSION($T_{[0,\lfloor n/2 \rfloor[}$)
5. $T_2 \leftarrow$TRIFUSION($T_{[\lfloor n/2 \rfloor, n[}$)
6. Renvoyer FUSION(T_1, T_2)



Algorithme du TriFusion

TriFusion(T):

1. $n \leftarrow \#T$
2. Si $n \leq 1$: Renvoyer T
3. Sinon :
4. $T_1 \leftarrow$TriFusion$(T_{[0,\lfloor n/2 \rfloor[})$
5. $T_2 \leftarrow$TriFusion$(T_{[\lfloor n/2 \rfloor, n[})$
6. Renvoyer Fusion(T_1, T_2)

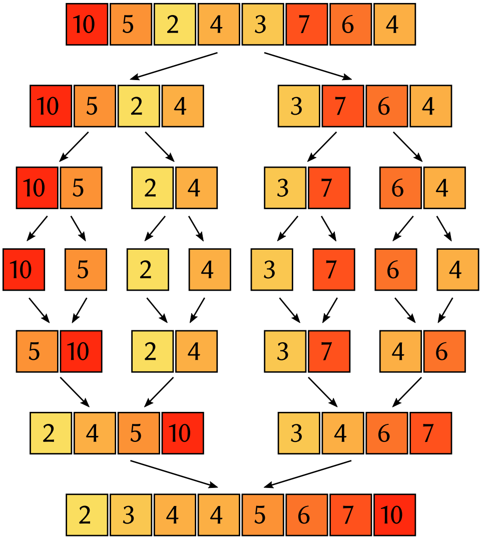
Lemme

Soit $t(n)$ la complexité de TriFusion et $f(n)$ la complexité de Fusion. Alors

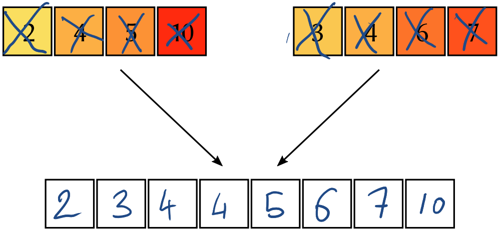
$$t(n) = t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + f(n) + O(1)$$

pour $n > 1$, et $t(n) = 0$ sinon

(complexité = nombre de comparaisons)



Algorithme de Fusion



Idée de l'algorithme

- T_1 et T_2 vus comme des **piles**
- S vu comme une **file**
- À chaque itération,
  - on dépile la plus petite des deux têtes
    - si une pile est vide, on dépile l'autre
  - on enfile dans S

Algorithme de Fusion

Fusion(T_1, T_2):

$n_1 \leftarrow \#T_1$; $n_2 \leftarrow \#T_2$
$S \leftarrow$ tableau de taille $n = n_1 + n_2$
$i_1 \leftarrow 0$; $i_2 \leftarrow 0$
Pour $i_S = 0$ à $n - 1$:
    Si $i_1 \geq n_1$: *(T_1 vide)*
        $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
    Sinon si $i_2 \geq n_2$: *(T_2 vide)*
        $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
    Sinon si $T_{1[i_1]} \leq T_{2[i_2]}$:
        $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
    Sinon :
        $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
Renvoyer S

Idée de l'algorithme

- T_1 et T_2 vus comme des **piles**
- S vu comme une **file**
- À chaque itération,
  - on dépile la plus petite des deux têtes
    - si une pile est vide, on dépile l'autre
  - on enfile dans S

Algorithme de Fusion

Fusion(T_1, T_2):

$n_1 \leftarrow \#T_1$; $n_2 \leftarrow \#T_2$
$S \leftarrow$ tableau de taille $n = n_1 + n_2$
$i_1 \leftarrow 0$; $i_2 \leftarrow 0$
Pour $i_S = 0$ à $n-1$:
    Si $i_1 \geq n_1$: *(T_1 vide)*
        $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
    Sinon si $i_2 \geq n_2$: *(T_2 vide)*
        $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
    Sinon si $T_{1[i_1]} \leq T_{2[i_2]}$:
        $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
    Sinon :
        $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
Renvoyer S

Lemme

La complexité $f(n)$ de Fusion est $O(n)$.

Preuve :

- Boucle de 0 à $n-1$
- Chaque itération coûte $O(1)$

Algorithme de Fusion

Fusion(T_1, T_2):

```
n₁ ← #T₁; n₂ ← #T₂
S ← tableau de taille n = n₁ + n₂
i₁ ← 0; i₂ ← 0
Pour i_S = 0 à n − 1:
    Si i₁ ≥ n₁ :                      (T₁ vide)
        S[i_S] ← T₂[i₂]; i₂ ← i₂ + 1
    Sinon si i₂ ≥ n₂ :                (T₂ vide)
        S[i_S] ← T₁[i₁]; i₁ ← i₁ + 1
    Sinon si T₁[i₁] ≤ T₂[i₂]:
        S[i_S] ← T₁[i₁]; i₁ ← i₁ + 1
    Sinon :
        S[i_S] ← T₂[i₂]; i₂ ← i₂ + 1
Renvoyer S
```

Lemme

La complexité $f(n)$ de Fusion est $O(n)$.

Lemme

Si T_1 et T_2 sont deux tableaux triés (par ordre croissant), Fusion(T_1, T_2) renvoie un tableau trié contenant l'union des éléments de T_1 et T_2.

Preuve : Invariant :

$S[0, i_S[$ est trié et contient les i_S plus petits éléments de $T_1 \sqcup T_2$.

Retour sur le TriFusion

Théorème

L'algorithme TriFusion trie tout tableau de taille n en temps $O(n \log n)$.

Preuve de correction par récurrence

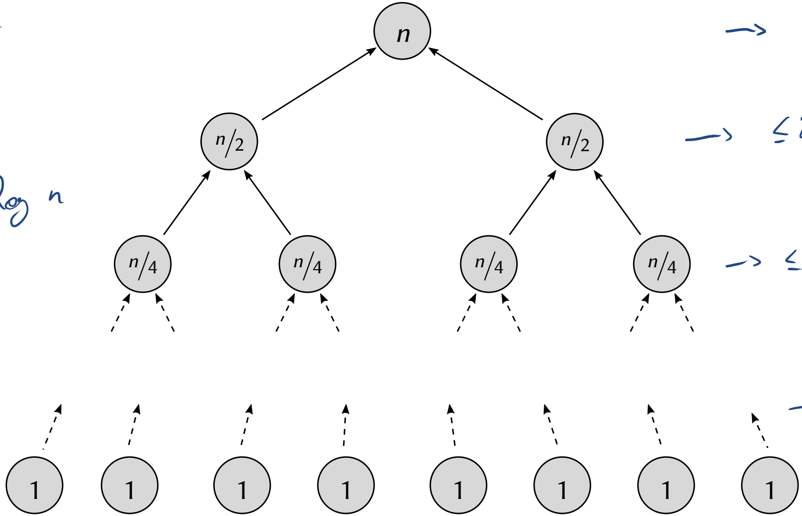
- $n \leq 1$: OK car rien à faire
- $n > 1$: Par hyp. de réc. T_1 et T_2 sont triés car $\lceil n/2 \rceil < n$ pour $n \geq 2$.
  et l'algo Fusion est correct.

Preuve de complexité

$$t(n) = t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + \Theta(n)$$

$$\rightsquigarrow \quad t(n) = \Theta(n \log n)$$: à démontrer.

Intuition de la complexité de TriFusion



$\log n$

$\longrightarrow \leq c \cdot n$

$\longrightarrow \leq 2 \cdot c\, n/2 = c\,n$

$\longrightarrow \leq 4 \cdot c\, n/4 = c \cdot n$

$\longrightarrow \leq \frac{n}{2} \cdot c\, n/(n/2) = c \cdot n$

$(f(n) \leq cn)$

$\longrightarrow \leq c \cdot n \log n = \Theta(n \log n)$

Table des matières

La stratégie « diviser pour régner »

1. Diviser le problème en sous-problèmes
2. Résoudre récursivement ces sous-problèmes
3. Combiner les solutions pour reconstruire la solution du problème original.

- Stratégie principalement utilisée pour obtenir de meilleures complexités que celles données par un algorithme moins évolué.
- Exemple : la recherche dichotomique

Exemple du tri fusion

1. Diviser le tableau en 2 sous-tableaux de tailles environ égales
2. Trier récursivement chaque sous-tableau
3. Fusionner les sous-tableaux triés

Analyse d'un algorithme « diviser pour régner »

Récurrence(s) sur la taille du problème

Correction

- Hypothèse de récurrence : les appels récursifs sont corrects
- Preuve d'hérédité : *diviser* et/ou *combiner* sont correctes
- Preuve de correction

Complexité

1. Établir l'équation de récurrence
2. Résoudre la récurrence :
   - soit estimation (arbre de récursion, ...) puis preuve par récurrence
   - soit utilisation du *master theorem*

Une version du « *master theorem* »

Théorème

Soit $T : \mathbb{N} \to \mathbb{R}$ vérifiant pour tout $n \geq n_0$,

$$T(n) \leq aT(\lceil n/b \rceil) + O(n^d)$$

où a, $d \geq 0$ et $b > 1$. Alors

$$T(n) = \begin{cases} O(n^d) & \text{si } b^d > a \quad (d > \log_b a) \\ O(n^d \log n) & \text{si } b^d = a \quad (d = \log_b a) \\ O(n^{\log_b a}) & \text{si } b^d < a \quad (d < \log_b a) \end{cases}$$

Exemple du tri fusion

$t(n) \leq t(\lceil n/2 \rceil]) + t(\lfloor n/2 \rfloor) + O(n)$

$\leq 2\, t(\lceil n/2 \rceil) + \Theta(n)$

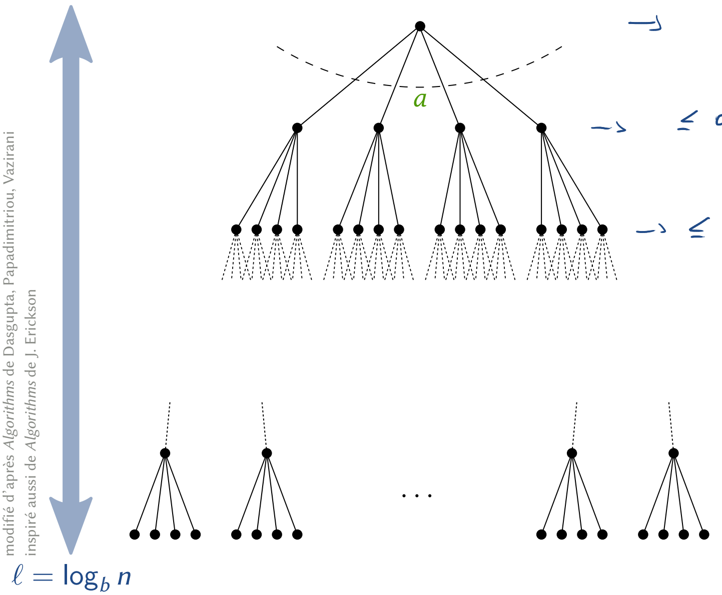
$a = 2$

$b = 2$

$d = 1$

$b^d = 2^1 = a \rightsquigarrow \Theta(n \log n)$

Intuition graphique



$\rightarrow \leq c \cdot n^d$

$\rightarrow \leq a \times c\,(n/b)^d$

$\rightarrow \leq a^2 \cdot c\,(n/b^2)^d$

$\vdots$

$\leq a^\ell \cdot c \cdot (n/b^\ell)^d$

modifié d'après *Algorithms* de Dasgupta, Papadimitriou, Vazirani
inspiré aussi de *Algorithms* de J. Erickson

Cœur de la preuve

Résoudre $T(n) \leq aT(\lceil n/b \rceil) + c \cdot n^d$ avec $n = b^\ell$

Lemme

Pour $\ell \geq 0$, $T(b^\ell) \leq a^\ell T(1) + cb^{d\ell} \sum_{i=0}^{\ell-1} (a/b^d)^i$

$\ell = 0 \qquad T(b^0) \leq a^0 \, T(1)$

$\ell \geq 0 \qquad T(b^{\ell+1}) \leq a \cdot T(b^\ell) + c \cdot b^{\ell d} \leq a^{\ell+1} T(1) + a c b^{d\ell} \sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i + c b^{(\ell+1)d}$

$\leq a^{\ell+1} T(1) + c b^{d\ell} \sum_{i=0}^{\ell-1} \frac{a^{i+1} b^d}{(b^d)^{i+1}} + c b^{(\ell+1)d}$

$\leq a^{\ell+1} T(1) + c b^{d(\ell+1)} \times \sum_{i=1}^{\ell} \left(a/b^d\right)^i + c b^{(\ell+1)d}$

$\leq a^{\ell+1} T(1) + c b^{(\ell+1)d} \sum_{i=0}^{\ell} \left(a/b^d\right)^i$

Cœur de la preuve

Résoudre $T(n) \leq aT(\lceil n/b \rceil) + c \cdot n^d$ avec $n = b^\ell$

Lemme

Pour $\ell \geq 0$, $T(b^\ell) \leq a^\ell T(1) + cb^{d\ell} \sum_{i=0}^{\ell-1} (a/b^d)^i$

Lemme

$$\sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i = \begin{cases} O(1) & \text{si } b^d > a \\ O(\ell) & \text{si } b^d = a \\ O((a/b^d)^\ell) & \text{si } b^d < a \end{cases}$$

$b^d > a \Rightarrow a/b^d < 1$: $\sum_{i=0}^{\ell-1} (a/b^d)^i < \sum_{i=0}^{+\infty} (a/b^d)^i = \frac{1}{1 - a/b^d} = \Theta(1)$

$b^d = a \Rightarrow a/b^d = 1$: $\sum_{i=0}^{\ell-1} 1^i = \ell$

$b^d > a \Rightarrow a/b^d > 1$: $\sum_{i=0}^{\ell-1} (a/b^d)^i = \frac{(a/b^d)^\ell - 1}{a/b^d - 1} = \Theta((a/b^d)^\ell)$

Cœur de la preuve

Résoudre $T(n) \leq aT(\lceil n/b \rceil) + c \cdot n^d$ avec $n = b^\ell$

Lemme

Pour $\ell \geq 0$, $T(b^\ell) \leq a^\ell T(1) + cb^{d\ell} \displaystyle\sum_{i=0}^{\ell-1} (a/b^d)^i$

Lemme

$$\sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i = \begin{cases} O(1) & \text{si } b^d > a \\ O(\ell) & \text{si } b^d = a \\ O((a/b^d)^\ell) & \text{si } b^d < a \end{cases}$$

Corollaire

$$T(b^\ell) = \begin{cases} O(\cancel{a^\ell} + b^{d\ell}) & \text{si } b^d > a \\ O(\cancel{a^\ell} + b^{d\ell} \cdot \ell) & \text{si } b^d = a \\ O(a^\ell + b^{d\ell} \cancel{(a/b^d)}^\ell) & \text{si } b^d < a \end{cases}$$

Fin de la preuve

Cas $n = b^\ell$ $\rightarrow$ $\ell = \log_b n$

$$T(b^\ell) = \begin{cases} O(b^{d\ell}) & \text{si } b^d > a \\ O(b^{d\ell} \cdot \ell) & \text{si } b^d = a \\ O(a^\ell) & \text{si } b^d < a \end{cases}$$

$\rightarrow \Theta(b^{\log_b n \cdot d}) = \Theta(n^d)$

$\rightarrow \Theta(n^d \log n)$

$\rightarrow \Theta(a^{\log_b n}) = \Theta(b^{\log_b a \log_b n}) = \Theta(n^{\log_b a})$

Cas général

Hypothèse : $T(n)$ est croissant.

On prend ℓ minimal tq $b^\ell \geq n$ (donc $b^{\ell-1} < n$)

Alors $b^\ell < b \cdot n$ et $T(n) \leq T(b^\ell)$

$$T(n) \leq T(b^\ell) = \begin{cases} \Theta((bn)^d) = \Theta(n^d) \\ \Theta((bn)^d \log(bn)) = \Theta(n^d \log n) \\ \Theta\left((bn)^{\log_b a}\right) = \Theta\left(n^{\log_b a}\right). \end{cases}$$

Conclusion

« Diviser pour régner »

1. Diviser ; 2. Résoudre récursivement ; 3. Combiner

Conception : seuls 1. et 3. demandent de la réflexion

Correction : preuve par récurrence

Complexité : *master theorem* (en général) *à apprendre !*

Autres versions du *master theorem*

- Récurrences plus générales *ex.:* $T(n) = aT(\lceil n/b \rceil) + O(n^d \log^c n)$
- Résultats plus précis
  - Constantes dans le « grand O »
  - Termes de plus bas degré

Objectifs du chapitre

- Reconnaître un algo. « diviser pour régner »
- Prouver sa correction et analyser sa complexité
- Tenter une stratégie « diviser pour régner » sur un nouveau problème

Table des matières

Retour à l'école primaire

Multiplication d'entiers

Entrée Deux entiers A et B écrits en base 10

Sortie L'entier $C = A \times B$, en base 10

$$
\begin{array}{rrrrrrrr}
 & & & & 1 & 3 & 8 & 2 \\
 & & & \times & 7 & 6 & 3 & 4 \\
\hline
 & & & & 5 & 5 & 2 & 8 \\
 & & & 4 & 1 & 4 & 6 & \\
 & & 8 & 2 & 9 & 2 & & \\
 & 9 & 6 & 7 & 4 & & & \\
\hline
1 & 0 & 5 & 5 & 0 & 1 & 8 & 8
\end{array}
$$

Complexité

- Combien de *multiplications chiffre à chiffre* sont effectuées ?
- Combien d'*additions chiffre à chiffre* sont effectuées ?

$\Big\}\ \Theta(n^2)$

Première tentative

Entrées $A = \sum_{i=0}^{n-1} a_i 10^i$ et $B = \sum_{i=0}^{n-1} b_i 10^i$

Diviser $A = A_0 + 10^{\lfloor n/2 \rfloor} A_1$
$B = B_0 + 10^{\lfloor n/2 \rfloor} B_1$

Récursion $C_{00} = A_0 \times B_0 \qquad C_{01} = A_0 \times B_1$
$C_{10} = A_1 \times B_0 \qquad C_{11} = A_1 \times B_1$

Combiner $C = C_{00} + 10^{\lfloor n/2 \rfloor}(C_{01} + C_{10}) + 10^{2\lfloor n/2 \rfloor} C_{11}$

$$\begin{array}{rr}
 & 1\,3\,8\,2 \\
 & \times\,7\,6\,3\,4 \\
\hline
C_{00} = & 2\,7\,8\,8 \\
C_{01} = & 6\,2\,3\,2 \\
C_{10} = & 4\,4\,2 \\
C_{11} = & 9\,8\,8 \\
\hline
 & = 1\,0\,5\,5\,0\,1\,8\,8
\end{array}$$

Correction

$$AB = (A_0 + 10^{\lfloor n/2 \rfloor} A_1) \times (B_0 + 10^{\lfloor n/2 \rfloor} B_1)$$
$$= A_0 B_0 + 10^{\lfloor n/2 \rfloor}(A_0 B_1 + A_1 B_0) + 10^{2\lfloor n/2 \rfloor} A_1 B_1$$

Complexité

$T(n) \leq 4T(\lceil n/2 \rceil) + O(n)$

$a = 4$
$b = 2$
$d = 1$

$a = 4 > b^d = 2$

$\rightsquigarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

Idée de Karatsuba (version Knuth)

$$A_0 B_1 + A_1 B_0 = A_0 B_0 + A_1 B_1 - (A_0 - A_1)(B_0 - B_1)$$

- $A_0 B_0$ et $A_1 B_1$ sont calculés de toute façon
  $\rightsquigarrow$ un seul produit en plus !
- $A_0 - A_1$ et $B_0 - B_1$ ont $\simeq n/2$ chiffres mais peuvent être négatifs $\rightsquigarrow$ règle des signes

Diviser $A = A_0 + 10^{\lfloor n/2 \rfloor} A_1$
$B = B_0 + 10^{\lfloor n/2 \rfloor} B_1$

Récursion $C_{00} = A_0 \times B_0 \qquad C_{11} = A_1 \times B_1$
$D = (A_0 - A_1) \times (B_0 - B_1)$

Combiner $C = C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - D) + 10^{2\lfloor n/2 \rfloor} C_{11}$

$$\begin{array}{rrr}
 & & 1\,3\,8\,2 \\
 & & \times\,7\,6\,3\,4 \\
\hline
A_0 - A_1 = & 6\,9 & \\
B_0 - B_1 = & -\,4\,2 & \\
\hline
C_{00} = & & 2\,7\,8\,8 \\
C_{11} = & 9\,8\,8 & \\
-D = & 2\,8\,9\,8 & \\
C_{00} = & 2\,7\,8\,8 & \\
C_{11} = & 9\,8\,8 & \\
\hline
 & = 1\,0\,5\,5\,0\,1\,8\,8 &
\end{array}$$

Algorithme de Karatsuba (1962)

Karatsuba(A, B):

1. Si A et B n'ont qu'un chiffre : Renvoyer $a_0 b_0$
2. Écrire A sous la forme $A_0 + 10^{\lfloor n/2 \rfloor} A_1$
3. Écrire B sous la forme $B_0 + 10^{\lfloor n/2 \rfloor} B_1$
4. $C_{00} \leftarrow$ Karatsuba(A_0, B_0)
5. $C_{11} \leftarrow$ Karatsuba(A_1, B_1)
6. $D \leftarrow$ Karatsuba$(|A_0 - A_1|, |B_0 - B_1|)$
7. $s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$
8. Renvoyer $C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$

Correction

$$A \times B = C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$$

Complexité

Soit $K(n)$ le temps de calcul de Karatsuba pour des entrées de taille n. Alors

$K(n) \leq 3K(\lceil n/2 \rceil) + O(n)$ $a = 3$, $b = 2$, $d = 1$ $b^d < a$ $K(n) = \Theta(n^{\log_2 3}) \simeq \Theta(n^{1,58})$

Dans la *vraie vie*

Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, …

- Grands entiers : tableaux d'entiers de w bits $\iff$ entiers en base 2^w
- Exemples : `gmp` (C/C++), `BigInteger` (Java), `int` (Python), …
- Autre utilisation : polynômes

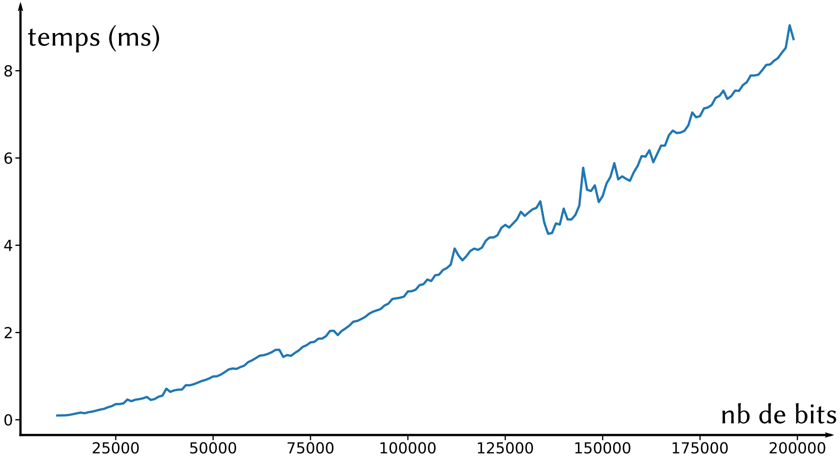
Quel TAD utiliser ?

- TAD entier : opérations en temps $O(1)$ — dont multiplication
  - Réaliste pour des entiers raisonnables — indices de tableau, …
  - Irréaliste pour de grands entiers — cryptographie, …
- TAD entier borné : opérations en temps $O(1)$ pour des entiers $< 2^w$

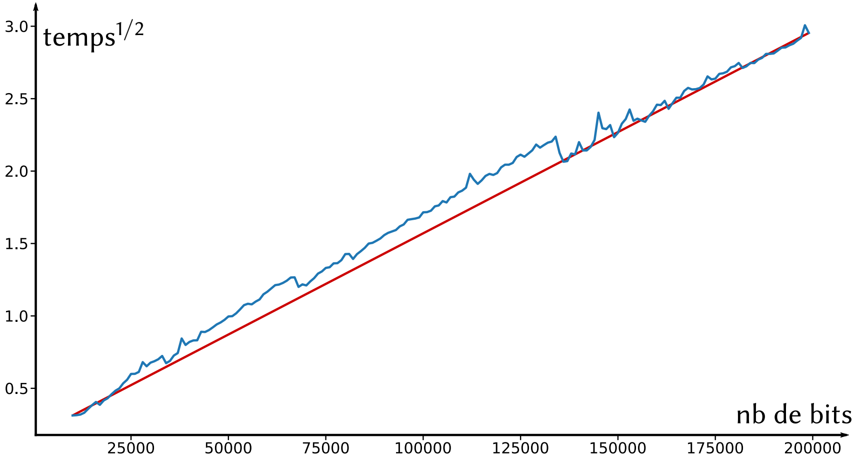
Algorithmes plus rapides

- Toom-3 (1963) : découpe en 3 morceaux — $O(n^{1,465})$
- Toom-Cook (1966): découpe en r morceaux — $O(n^{1+\epsilon})$
- Schönhage-Strassen (1971) : basé sur la FFT — $O(n \log n \log\log n)$
- …
- Harvey-Hoeven (2021) : utilise aussi la FFT — $\boldsymbol{O(n \log n)}$

Pour finir : multiplication d'entiers en Python



Pour finir : multiplication d'entiers en Python



Pour finir : multiplication d'entiers en Python