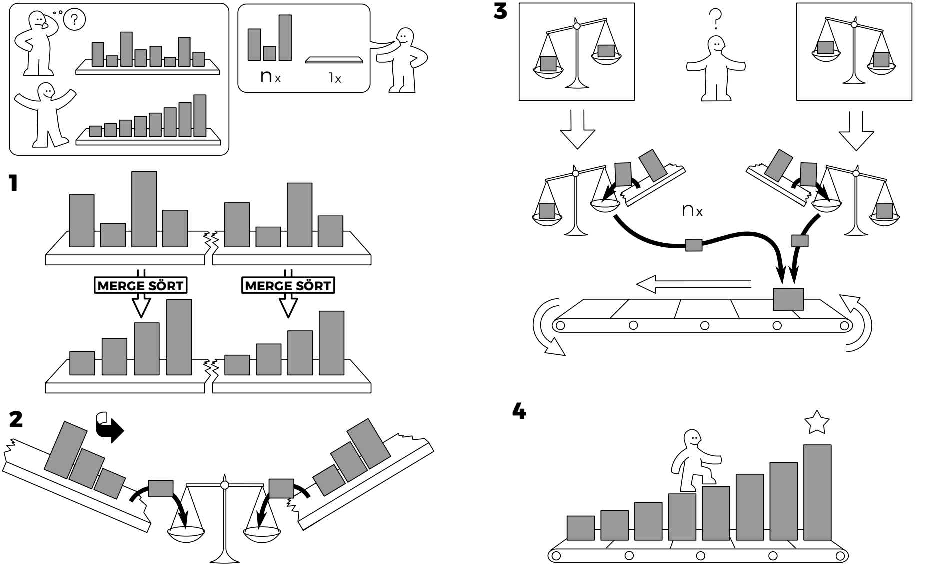
Chapitre 5
Diviser pour régner

HLIN401 : Algorithmique et complexité

Université de Montpellier
2018 – 2019

MERGE SÖRT

IDEA



Algorithme du TriFusion

Algorithme : TriFusion(T)

$n \leftarrow$ taille(T)
si $n = 1$ **alors**
    Retourner T
sinon
    $T_1 \leftarrow$TriFusion($T[0, \lfloor n/2 \rfloor - 1]$)
    $T_2 \leftarrow$TriFusion($T[\lfloor n/2 \rfloor, n]$)
    Retourner Fusion(T_1, T_2)

Algorithme du TriFusion

Algorithme : TriFusion(T)

$n \leftarrow$ taille(T)
si $n = 1$ **alors**
  Retourner T
sinon
  $T_1 \leftarrow$TriFusion($T[0, \lfloor n/2 \rfloor - 1]$)
  $T_2 \leftarrow$TriFusion($T[\lfloor n/2 \rfloor, n]$)
  Retourner Fusion(T_1, T_2)

Lemme

Soit $T(n)$ la complexité de TriFusion *et $F(n)$ la complexité de* Fusion*. Alors*

$$T(n) = \begin{cases} O(1) & \text{si } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + F(n) & \text{sinon} \end{cases}$$

Algorithme de Fusion

Algorithme : Fusion(T_1, T_2)

$n_1 \leftarrow$ taille(T_1) ; $n_2 \leftarrow$ taille(T_2)
$S \leftarrow$ tableau de taille $n_1 + n_2$
$i_1 \leftarrow 0$; $i_2 \leftarrow 0$
pour $i_S = 0$ *à* $n_1 + n_2 - 1$ **faire**
  si $i_1 \geq n_1$ **alors** $S[i_S] \leftarrow T_2[i_2]$; $i_2 \leftarrow i_2 + 1$
  sinon si $i_2 \geq n_2$ **alors** $S[i_S] \leftarrow T_1[i_1]$; $i_1 \leftarrow i_1 + 1$
  sinon si $T_1[i_1] < T_2[i_2]$ **alors** $S[i_S] \leftarrow T_1[i_1]$; $i_1 \leftarrow i_1 + 1$
  sinon $S[i_S] \leftarrow T_2[i_2]$; $i_2 \leftarrow i_2 + 1$

retourner S

Algorithme de Fusion

Algorithme : Fusion(T_1, T_2)

$n_1 \leftarrow$ taille(T_1) ; $n_2 \leftarrow$ taille(T_2)
$S \leftarrow$ tableau de taille $n_1 + n_2$
$i_1 \leftarrow 0$; $i_2 \leftarrow 0$
pour $i_S = 0$ *à* $n_1 + n_2 - 1$ **faire**
    si $i_1 \geq n_1$ **alors** $S[i_S] \leftarrow T_2[i_2]$; $i_2 \leftarrow i_2 + 1$
    sinon si $i_2 \geq n_2$ **alors** $S[i_S] \leftarrow T_1[i_1]$; $i_1 \leftarrow i_1 + 1$
    sinon si $T_1[i_1] < T_2[i_2]$ **alors** $S[i_S] \leftarrow T_1[i_1]$; $i_1 \leftarrow i_1 + 1$
    sinon $S[i_S] \leftarrow T_2[i_2]$; $i_2 \leftarrow i_2 + 1$

retourner S

Lemme

La complexité $F(n)$ de Fusion *est $O(n)$.*

Preuve évidente !

Correction de la FUSION

Lemme

Si T_1 et T_2 sont deux tableaux triés (par ordre croissant), FUSION(T_1, T_2) *renvoie un tableau trié.*

Preuve au tableau

$\mathcal{P}_{i_S}$: à l'entrée de l'itération i_S de la boucle **pour**,

1. $S[0, i_S - 1]$ contient les i_S plus petits éléments de $T_1 \cup T_2$ en ordre croissant
2. i_1 est l'indice du plus petit élément de T_1 non présent dans S
3. i_2 est l'indice du plus petit élément de T_2 non présent dans S

Retour sur le TriFusion

Théorème

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

Algorithme : TriFusion(T)

$n \leftarrow$ taille(T)
si $n = 1$ **alors**
  Retourner T
sinon
  $T_1 \leftarrow$TriFusion($T[0, \lfloor n/2 \rfloor - 1]$)
  $T_2 \leftarrow$TriFusion($T[\lfloor n/2 \rfloor, n]$)
  Retourner Fusion(T_1, T_2)

Retour sur le TriFusion

Algorithme : TriFusion(T)

$n \leftarrow$ taille(T)
si $n = 1$ **alors**
  Retourner T
sinon
  $T_1 \leftarrow$TriFusion($T[0, \lfloor n/2 \rfloor - 1]$)
  $T_2 \leftarrow$TriFusion($T[\lfloor n/2 \rfloor, n]$)
  Retourner Fusion(T_1, T_2)

Théorème

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

Preuve de correction par récurrence (facile !)

- Si $n = 1$, OK
- Si $n > 1$, $\lfloor n/2 \rfloor \leq \lceil n/2 \rceil < \boldsymbol{n}$, donc T_1 et T_2 triés après appels récursifs. La correction de Fusion suffit à conclure.

Retour sur le TriFusion

Algorithme : TriFusion(T)

$n \leftarrow$ taille(T)
si $n = 1$ **alors**
  Retourner T
sinon
  $T_1 \leftarrow$TriFusion($T[0, \lfloor n/2 \rfloor - 1]$)
  $T_2 \leftarrow$TriFusion($T[\lfloor n/2 \rfloor, n]$)
  Retourner Fusion(T_1, T_2)

Théorème

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

Preuve de correction par récurrence (facile !)

- Si $n = 1$, OK
- Si $n > 1$, $\lfloor n/2 \rfloor \leq \lceil n/2 \rceil < \boldsymbol{n}$, donc T_1 et T_2 triés après appels récursifs. La correction de Fusion suffit à conclure.

Preuve de complexité au tableau

- **Équation de récurrence** : $T(n) \leq 2T(\lceil n/2 \rceil) + O(n)$
- **Arbre de récursion** $\rightsquigarrow$ estimation du temps de calcul
- **Preuve par récurrence** de l'estimation

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.

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.

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 (un peu naïf...) : la recherche dichotomique

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 (un peu naïf...) : 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

Analyse d'un algorithme « diviser pour régner »

Récurrence(s) sur la taille du problème

- Preuve de correction

Analyse d'un algorithme « diviser pour régner »

Récurrence(s) sur la taille du problème

- Preuve de correction
- Complexité :
  1. Établir l'équation de récurrence
  2. Estimer le résultat (arbre de récursion, déroulement de la récurrence, habitude, ...)
  3. Preuve par récurrence

Analyse d'un algorithme « diviser pour régner »

Récurrence(s) sur la taille du problème

- Preuve de correction
- Complexité :
  1. Établir l'équation de récurrence
  2. Estimer le résultat (arbre de récursion, déroulement de la récurrence, habitude, ...)
  3. Preuve par récurrence

  ou

  2-3. Utiliser le « *master theorem* » !

Une version du « *master theorem* »

Théorème

S'il existe trois entiers $a \geq 0$, $b > 1$, $d \geq 0$ *et* $n_0 > 0$ *tels que pour tout* $n \geq n_0$,

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

Alors

$$\boldsymbol{T(n) = \begin{cases} O(n^d) & \textit{si } b^d > a \\ O(n^d \log n) & \textit{si } b^d = a \\ O(n^{\frac{\log a}{\log b}}) & \textit{si } b^d < a \end{cases}}$$

Une version du « *master theorem* »

Théorème

S'il existe trois entiers $a \geq 0$, $b > 1$, $d \geq 0$ *et* $n_0 > 0$ *tels que pour tout* $n \geq n_0$,

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

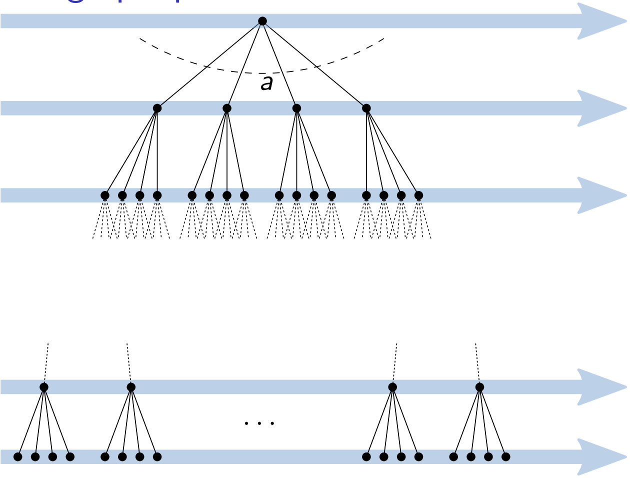
Alors

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

Exemple du tri fusion

- $T(n) \leq 2T(\lceil n/2 \rceil]) + O(n) : a = 2,\ b = 2,\ d = 1$
- $b^d = a \rightsquigarrow T(n) = O(n^d \log n) = O(n \log n)$

Preuve graphique



$$
\begin{aligned}
&1 \times n^d \\
&+ \\
&a \times (n/b)^d \\
&+ \\
&a^2 \times (n/b^2)^d \\
&+ \\
&\vdots \\
&+ \\
&a^{\ell-1} \times (n/b^{\ell-1})^d \\
&+ \\
&a^{\ell} \times (n/b^{\ell})^d
\end{aligned}
$$

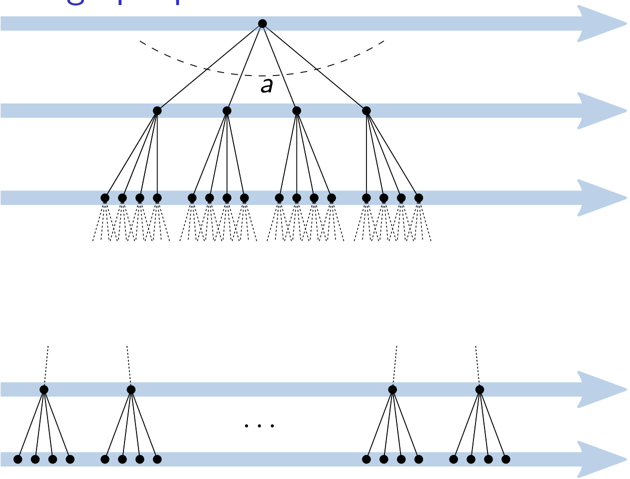
modifié d'après *Algorithms* de Dasgupta, Papadimitriou, Vazirani

inspiré aussi de *Algorithms* de J. Erickson

Preuve graphique

modifié d'après *Algorithms* de Dasgupta, Papadimitriou, Vazirani

inspiré aussi de *Algorithms* de J. Erickson



$$1 \times n^d$$
$$+$$
$$a \times (n/b)^d$$
$$+$$
$$a^2 \times (n/b^2)^d$$
$$+$$
$$\vdots$$
$$+$$
$$a^{\ell-1} \times (n/b^{\ell-1})^d$$
$$+$$
$$a^{\ell} \times (n/b^{\ell})^d$$

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

En pratique

- Versions plus générales du « *master theorem* »
  - Récurrences plus générales
  - Constantes des « grands O »
  - Termes de plus bas degré
- Dans ce cours : étude de plusieurs exemples
  - Utilisation autorisée du « *master theorem* »
  - ... donc à apprendre !

Objectifs :

- Savoir tenter un « diviser pour régner »
- Savoir analyser sa complexité
- Reconnaître un algo. « diviser pour régner »

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

Retour à l'école primaire

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

Multiplication d'entiers

Entrée Deux entiers A et B écrits en base 10

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

Retour à l'école primaire

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

Multiplication d'entiers

Entrée Deux entiers A et B écrits en base 10

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

Question

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

Retour à l'école primaire

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

Multiplication d'entiers

Entrée Deux entiers A et B écrits en base 10

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

Question

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

$\leadsto$ $O(n^2)$ *multiplications et additions*

Retour à l'école primaire

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

Multiplication d'entiers

Entrée Deux entiers A et B écrits en base 10

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

Question

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

$\rightsquigarrow O(n^2)$ *multiplications et additions*

Peut-on faire mieux ?

Première tentative

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

$A = 1382, \; B = 7634$

Première tentative

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

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

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

$A = 1382,\ B = 7634$
$A_1 = 13,\ A_0 = 82$
$B_1 = 76,\ B_0 = 34$

Première tentative

Entrée $A = \sum_{i=0}^{n-1} a_i 10^i$ 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 $C_{00} = A_0 \times B_0, \; C_{01} = A_0 \times B_1$

$C_{10} = A_1 \times B_0, \; C_{11} = A_1 \times B_1$

$A = 1382, \; B = 7634$
$A_1 = 13, \; A_0 = 82$
$B_1 = 76, \; B_0 = 34$
$C_{00} = 2788, \; C_{01} = 6232$
$C_{10} = 442, \; C_{11} = 988$

Première tentative

Entrée $A = \sum_{i=0}^{n-1} a_i 10^i$ 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 $C_{00} = A_0 \times B_0,\ C_{01} = A_0 \times B_1$

$C_{10} = A_1 \times B_0,\ C_{11} = A_1 \times B_1$

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

$A = 1382,\ B = 7634$
$A_1 = 13,\ A_0 = 82$
$B_1 = 76,\ B_0 = 34$
$C_{00} = 2788,\ C_{01} = 6232$
$C_{10} = 442,\ C_{11} = 988$
$C = 2788 + 100 \cdot (6232 + 442) + 10000 \cdot 988 = 10550188$

Première tentative

$A = 1382,\ B = 7634$
$A_1 = 13,\ A_0 = 82$
$B_1 = 76,\ B_0 = 34$
$C_{00} = 2788,\ C_{01} = 6232$
$C_{10} = 442,\ C_{11} = 988$
$C = 2788 + 100 \cdot (6232 + 442) + 10000 \cdot 988 = 10550188$

Entrée $A = \sum_{i=0}^{n-1} a_i 10^i$ 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 $C_{00} = A_0 \times B_0,\ C_{01} = A_0 \times B_1$
$C_{10} = A_1 \times B_0,\ C_{11} = A_1 \times B_1$

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

Preuve de correction :

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

Première tentative

$A = 1382,\ B = 7634$
$A_1 = 13,\ A_0 = 82$
$B_1 = 76,\ B_0 = 34$
$C_{00} = 2788,\ C_{01} = 6232$
$C_{10} = 442,\ C_{11} = 988$
$C = 2788 + 100 \cdot (6232 + 442) + 10000 \cdot 988 = 10550188$

Entrée $A = \sum_{i=0}^{n-1} a_i 10^i$ 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 $C_{00} = A_0 \times B_0,\ C_{01} = A_0 \times B_1$
$C_{10} = A_1 \times B_0,\ C_{11} = A_1 \times B_1$

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

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

Preuve de complexité : $T(n) \leq 4T(\lceil n/2 \rceil) + O(n)$

- $a = 4,\ b = 2,\ d = 1 : b^d < a$
- $T(n) = O(n^{\log a / \log b}) = O(n^{\log 4 / \log 2}) = O(n^2)$...

Idée de Karatsuba (version Knuth)

$$\boxed{A_0B_1 + A_1B_0 = A_0B_0 + A_1B_1 - (A_0 - A_1)(B_0 - B_1)}$$

Idée de Karatsuba (version Knuth)

$$\boxed{\boldsymbol{A_0 B_1 + A_1 B_0 = A_0 B_0 + A_1 B_1 - (A_0 - A_1)(B_0 - B_1)}}$$

- $A_0 B_0$ et $A_1 B_1$ sont calculés de toute façon
- un seul produit en plus !

Idée de Karatsuba (version Knuth)

$$\boxed{\boldsymbol{A_0 B_1 + A_1 B_0 = A_0 B_0 + A_1 B_1 - (A_0 - A_1)(B_0 - B_1)}}$$

- $A_0 B_0$ et $A_1 B_1$ sont calculés de toute façon
- un seul produit en plus !
- $A_0 - A_1$ possède (env.) $n/2$ chiffres... mais peut être négatif
- on utilise la règle des signes

$\times$	$+$	$-$
$+$	$+$	$-$
$-$	$-$	$+$

Algorithme de Karatsuba (1962)

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

$A = 1382,\ B = 7634$

Algorithme de Karatsuba (1962)

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

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

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

$A = 1382,\ B = 7634$
$A_1 = 13,\ A_0 = 82$
$B_1 = 76,\ B_0 = 34$

Algorithme de Karatsuba (1962)

Entrée $A = \sum_{i=0}^{n-1} a_i 10^i$ 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 $C_{00} = A_0 \times B_0,\ C_{11} = A_1 \times B_1$

$D = (A_0 - A_1) \times (B_0 - B_1)$

$A = 1382,\ B = 7634$

$A_1 = 13,\ A_0 = 82$

$B_1 = 76,\ B_0 = 34$

$C_{00} = 2788,\ C_{11} = 988$

$D = 69 \times (-42) = -2898$

Algorithme de Karatsuba (1962)

Entrée $A = \sum_{i=0}^{n-1} a_i 10^i$ 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 $C_{00} = A_0 \times B_0,\ C_{11} = A_1 \times B_1$

$D = (A_0 - A_1) \times (B_0 - B_1)$

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

$A = 1382,\ B = 7634$

$A_1 = 13,\ A_0 = 82$

$B_1 = 76,\ B_0 = 34$

$C_{00} = 2788,\ C_{11} = 988$

$D = 69 \times (-42) = -2898$

$C = 2788 + 100 \cdot (2788 + 988 + 2898) + 10000 \cdot 988 = 10550188$

Algorithme de Karatsuba (1962)

Entrée $A = \sum_{i=0}^{n-1} a_i 10^i$ 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 $C_{00} = A_0 \times B_0, \; C_{11} = A_1 \times B_1$
$D = (A_0 - A_1) \times (B_0 - B_1)$

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

$A = 1382, \; B = 7634$
$A_1 = 13, \; A_0 = 82$
$B_1 = 76, \; B_0 = 34$
$C_{00} = 2788, \; C_{11} = 988$
$D = 69 \times (-42) = -2898$
$C = 2788 + 100 \cdot (2788$
$+ 988 + 2898) + 10000 \cdot 988$
$= 10550188$

Algorithme : Karatsuba(A, B)
si *A et B n'ont qu'un chiffre* **alors retourner** $a_0 b_0$
Écrire A sous la forme $A_0 + 10^{\lfloor n/2 \rfloor} A_1$
Écrire B sous la forme $B_0 + 10^{\lfloor n/2 \rfloor} B_1$
$C_{00} \leftarrow$ Karatsuba(A_0, B_0)
$C_{11} \leftarrow$ Karatsuba(A_1, B_1)
$D \leftarrow$ Karatsuba$(|A_0 - A_1|, |B_0 - B_1|)$
$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$
retourner $C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$

Terminaison de l'algorithme de Karatsuba

Algorithme : $\textsc{Karatsuba}(A, B)$

si *A et B n'ont qu'un chiffre* **alors retourner** $a_0 b_0$
Écrire A sous la forme $A_0 + 10^{\lfloor n/2 \rfloor} A_1$
Écrire B sous la forme $B_0 + 10^{\lfloor n/2 \rfloor} B_1$
$C_{00} \leftarrow \textsc{Karatsuba}(A_0, B_0)$
$C_{11} \leftarrow \textsc{Karatsuba}(A_1, B_1)$
$D \leftarrow \textsc{Karatsuba}(|A_0 - A_1|, |B_0 - B_1|)$
$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$
retourner $C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$

Lemme (terminaison)

Pour $n > 1$, $|A_0 - A_1|$ et $|B_0 - B_1|$ ont $< n$ chiffres.

Preuve : $-10^{\lceil n/2 \rceil} \leq A_0 - A_1 \leq 10^{\lfloor n/2 \rfloor}$ et $\lceil n/2 \rceil < n$ pour $n > 1$.

Terminaison de l'algorithme de Karatsuba

Algorithme : $\textsc{Karatsuba}(A, B)$

si *A et B n'ont qu'un chiffre* **alors retourner** $a_0 b_0$
Écrire A sous la forme $A_0 + 10^{\lfloor n/2 \rfloor} A_1$
Écrire B sous la forme $B_0 + 10^{\lfloor n/2 \rfloor} B_1$
$C_{00} \leftarrow \textsc{Karatsuba}(A_0, B_0)$
$C_{11} \leftarrow \textsc{Karatsuba}(A_1, B_1)$
$D \leftarrow \textsc{Karatsuba}(|A_0 - A_1|, |B_0 - B_1|)$
$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$
retourner $C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$

Lemme (terminaison)

Pour $n > 1$, $|A_0 - A_1|$ et $|B_0 - B_1|$ ont $< n$ chiffres.

Preuve : $-10^{\lceil n/2 \rceil} \leq A_0 - A_1 \leq 10^{\lfloor n/2 \rfloor}$ et $\lceil n/2 \rceil < n$ pour $n > 1$.

Corollaire

L'algorithme $\textsc{Karatsuba}$ termine.

Preuve : appels récursifs sur des entiers **strictement plus petits**

Correction de l'algorithme de Karatsuba

Algorithme : $\textsc{Karatsuba}(A, B)$

si *A et B n'ont qu'un chiffre* **alors retourner** $a_0 b_0$
Écrire A sous la forme $A_0 + 10^{\lfloor n/2 \rfloor} A_1$
Écrire B sous la forme $B_0 + 10^{\lfloor n/2 \rfloor} B_1$
$C_{00} \leftarrow \textsc{Karatsuba}(A_0, B_0)$
$C_{11} \leftarrow \textsc{Karatsuba}(A_1, B_1)$
$D \leftarrow \textsc{Karatsuba}(|A_0 - A_1|, |B_0 - B_1|)$
$s \leftarrow \mathrm{signe}(A_0 - A_1) \times \mathrm{signe}(B_0 - B_1)$
retourner $C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$

Lemme (complexité)

Soit $K(n)$ le temps de calcul de Karatsuba pour des entrées de taille n. Alors $K(n) \leq \mathbf{3}K(\lceil n/2 \rceil) + O(n)$

Correction de l'algorithme de Karatsuba

Algorithme : $\textsc{Karatsuba}(A, B)$

si *A et B n'ont qu'un chiffre* **alors retourner** $a_0 b_0$
Écrire A sous la forme $A_0 + 10^{\lfloor n/2 \rfloor} A_1$
Écrire B sous la forme $B_0 + 10^{\lfloor n/2 \rfloor} B_1$
$C_{00} \leftarrow \textsc{Karatsuba}(A_0, B_0)$
$C_{11} \leftarrow \textsc{Karatsuba}(A_1, B_1)$
$D \leftarrow \textsc{Karatsuba}(|A_0 - A_1|, |B_0 - B_1|)$
$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$
retourner $C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$

Lemme (complexité)

Soit $K(n)$ le temps de calcul de Karatsuba pour des entrées de taille n. Alors $K(n) \leq \mathbf{3}K(\lceil n/2 \rceil) + O(n)$

Corollaire (*master theorem*)

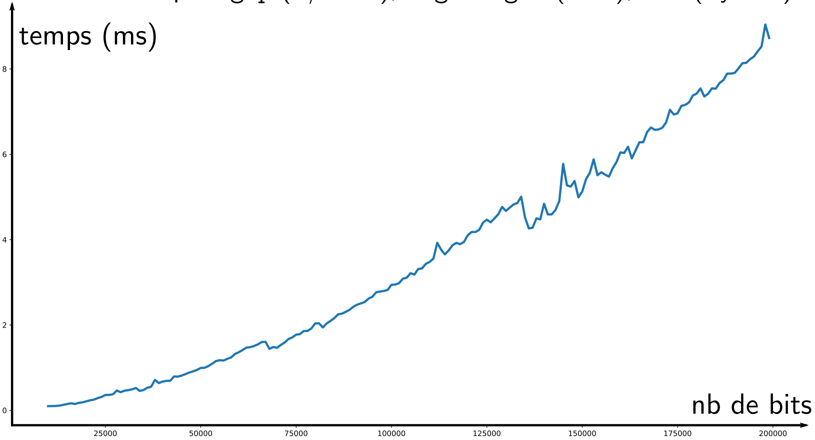
$$K(n) = O(n^{\log 3/\log 2}) = \boldsymbol{O(n^{\log 3})} \simeq O(n^{1,58})$$

Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, ...
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)

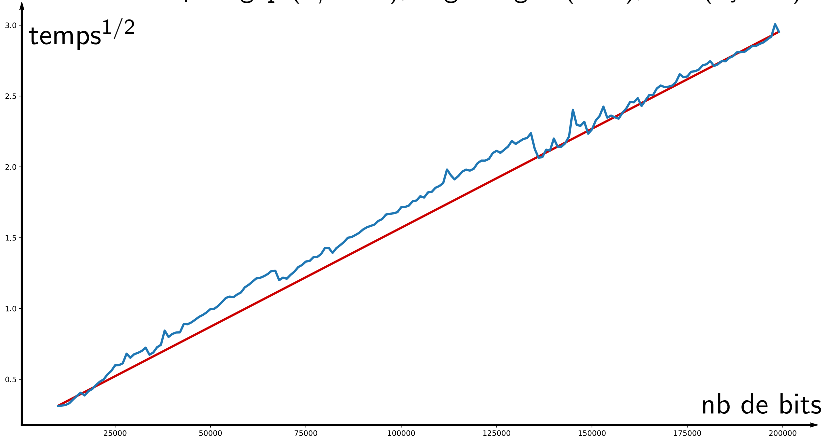
Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, ...
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)



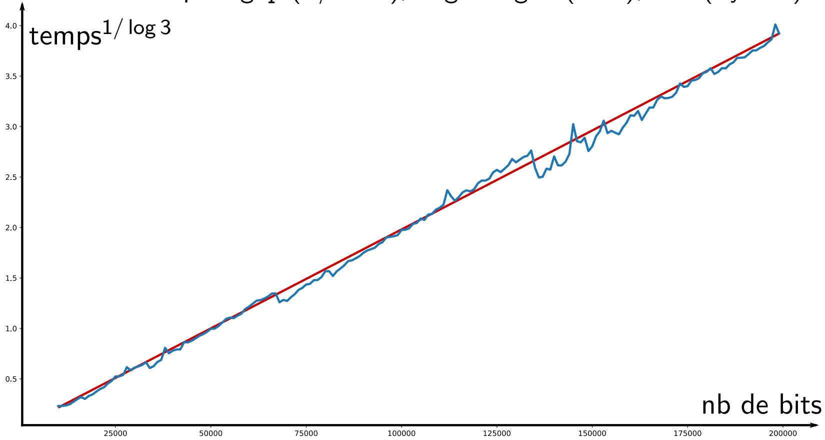
Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, ...
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)



Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, ...
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)



Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, …
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)
- Remarque par rapport au modèle
  - Modèle du cours : multiplication en temps $O(1)$
  - Irréaliste pour de grands entiers
  - Modèle souvent utilisé : taille d'un registre $= O(\log n)$

Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, ...
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)
- Remarque par rapport au modèle
  - Modèle du cours : multiplication en temps $O(1)$
  - Irréaliste pour de grands entiers
  - Modèle souvent utilisé : taille d'un registre $= O(\log n)$
- Autre utilisation : polynômes

Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, ...
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)
- Remarque par rapport au modèle
  - Modèle du cours : multiplication en temps $O(1)$
  - Irréaliste pour de grands entiers
  - Modèle souvent utilisé : taille d'un registre $= O(\log n)$
- Autre utilisation : polynômes
- Algorithmes plus rapides (1960's)
  - Toom-3 : découpe en 3 morceaux $O(n^{1,465})$
  - Toom-Cook : découpe en r morceaux $O(n^{1+\epsilon})$
  - Algorithmes basés sur la FFT $O(n \log n \log\log n)$

Dans la *vraie vie*

- Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, ...
  - Grands entiers représentés comme liste d'entiers de taille k bits $\iff$ entiers écrits en base 2^k !
  - Exemple : `gmp` (C/C++), `BigInteger` (Java), `int` (Python)
- Remarque par rapport au modèle
  - Modèle du cours : multiplication en temps $O(1)$
  - Irréaliste pour de grands entiers
  - Modèle souvent utilisé : taille d'un registre $= O(\log n)$
- Autre utilisation : polynômes
- Algorithmes plus rapides (1960's)
  - Toom-3 : découpe en 3 morceaux $O(n^{1,465})$
  - Toom-Cook : découpe en r morceaux $O(n^{1+\epsilon})$
  - Algorithmes basés sur la FFT $O(n \log n \log \log n)$
  - Record actuel (2018), utilise la FFT $O(n \log n 2^{2 \log^*(n)})$

Définition et algorithmes $\pm$ naïfs

Entrée Un tableau T de n nombres, et un entier $k \in \{1, \ldots, n\}$

Sortie le $k^{\text{ème}}$ plus petit élément de T, noté $\mathbf{rang}(\boldsymbol{k}, \boldsymbol{T})$

Définition et algorithmes $\pm$ naïfs

Entrée Un tableau T de n nombres, et un entier $k \in \{1, \ldots, n\}$

Sortie le $k^{\text{ème}}$ plus petit élément de T, noté $\mathbf{rang}(\boldsymbol{k}, \boldsymbol{T})$

Algorithme en $O(n^2)$:

```
pour i = 0 à n − 1 faire
    c = 0
    pour j = 0 à n − 1 faire
        si T[j] ≤ T[i] alors
            c ← c + 1
    si c = k alors retourner T[i]
```

Définition et algorithmes $\pm$ naïfs

Entrée Un tableau T de n nombres, et un entier $k \in \{1, \ldots, n\}$

Sortie le $k^{\text{ème}}$ plus petit élément de T, noté $\mathbf{rang}(\boldsymbol{k}, \boldsymbol{T})$

Algorithme en $O(n^2)$:

```
pour i = 0 à n − 1 faire
    c = 0
    pour j = 0 à n − 1 faire
        si T[j] ≤ T[i] alors
            c ← c + 1
    si c = k alors retourner T[i]
```

Algorithme en $O(n \log n)$:

```
Trier T
retourner T[k − 1]
```

Définition et algorithmes ± naïfs

Entrée Un tableau T de n nombres, et un entier $k \in \{1, \ldots, n\}$

Sortie le $k^{\text{ème}}$ plus petit élément de T, noté **rang(k, T)**

Algorithme en $O(n^2)$:

```
pour i = 0 à n − 1 faire
    c = 0
    pour j = 0 à n − 1 faire
        si T[j] ≤ T[i] alors
            c ← c + 1
    si c = k alors retourner T[i]
```

Algorithme en $O(n \log n)$:

```
Trier T
retourner T[k − 1]
```

Algorithme en $O(n)$?

Stratégie « diviser pour régner »

Diviser Choisir un **pivot** $p \in T$ pour séparer T en trois :

- $\boldsymbol{T}_{\text{inf}}$ qui contient les éléments x de T vérifiant $x < p$
- $\boldsymbol{T}_{\text{eq}}$ qui contient les éléments x de T vérifiant $x = p$
- $\boldsymbol{T}_{\text{sup}}$ qui contient les éléments x de T vérifiant $x > p$

Stratégie « diviser pour régner »

Diviser Choisir un **pivot** $p \in T$ pour séparer T en trois :

- $\boldsymbol{T}_{\text{inf}}$ qui contient les éléments x de T vérifiant $x < p$
- $\boldsymbol{T}_{\text{eq}}$ qui contient les éléments x de T vérifiant $x = p$
- $\boldsymbol{T}_{\text{sup}}$ qui contient les éléments x de T vérifiant $x > p$

Récursion Trouver $\text{rang}(k, T)$ dans T_{inf}, T_{eq} ou T_{sup}

$$\text{rang}(k, T) = \begin{cases} \text{rang}(k, T_{\text{inf}}) & \text{si } k \leq n_{\text{inf}} \\ p & \text{si } n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}} \\ \text{rang}(k - n_{\text{inf}} - n_{\text{eq}}, T_{\text{sup}}) & \text{si } n_{\text{inf}} + n_{\text{eq}} < k \end{cases}$$

où $n_{\text{inf}} = |T_{\text{inf}}|$, $n_{\text{eq}} = |T_{\text{eq}}|$, $n_{\text{sup}} = |T_{\text{sup}}|$

Stratégie « diviser pour régner »

Diviser Choisir un **pivot** $p \in T$ pour séparer T en trois :

- $\boldsymbol{T}_{\text{inf}}$ qui contient les éléments x de T vérifiant $x < p$
- $\boldsymbol{T}_{\text{eq}}$ qui contient les éléments x de T vérifiant $x = p$
- $\boldsymbol{T}_{\text{sup}}$ qui contient les éléments x de T vérifiant $x > p$

Récursion Trouver $\text{rang}(k, T)$ dans T_{inf}, T_{eq} ou T_{sup}

$$\text{rang}(k, T) = \begin{cases} \text{rang}(k, T_{\text{inf}}) & \text{si } k \leq n_{\text{inf}} \\ p & \text{si } n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}} \\ \text{rang}(k - n_{\text{inf}} - n_{\text{eq}}, T_{\text{sup}}) & \text{si } n_{\text{inf}} + n_{\text{eq}} < k \end{cases}$$

où $n_{\text{inf}} = |T_{\text{inf}}|$, $n_{\text{eq}} = |T_{\text{eq}}|$, $n_{\text{sup}} = |T_{\text{sup}}|$

Combiner Rien à faire...

Stratégie « diviser pour régner »

Diviser Choisir un **pivot** $p \in T$ pour séparer T en trois :

- $\boldsymbol{T_{\text{inf}}}$ qui contient les éléments x de T vérifiant $x < p$
- $\boldsymbol{T_{\text{eq}}}$ qui contient les éléments x de T vérifiant $x = p$
- $\boldsymbol{T_{\text{sup}}}$ qui contient les éléments x de T vérifiant $x > p$

Récursion Trouver $\text{rang}(k, T)$ dans T_{inf}, T_{eq} ou T_{sup}

$$\text{rang}(k, T) = \begin{cases} \text{rang}(k, T_{\text{inf}}) & \text{si } k \leq n_{\text{inf}} \\ p & \text{si } n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}} \\ \text{rang}(k - n_{\text{inf}} - n_{\text{eq}}, T_{\text{sup}}) & \text{si } n_{\text{inf}} + n_{\text{eq}} < k \end{cases}$$

où $n_{\text{inf}} = |T_{\text{inf}}|$, $n_{\text{eq}} = |T_{\text{eq}}|$, $n_{\text{sup}} = |T_{\text{sup}}|$

Combiner Rien à faire...

Question importante : quel choix pour le pivot ?

L'algorithme

Algorithme : $\textsc{Rang}(T, k)$

si $k = 1$ **alors retourner** $T[0]$
$p \leftarrow \textsc{ChoixPivot}(T)$
$n_{\text{inf}} \leftarrow 0,\ n_{\text{eq}} \leftarrow 0$
pour $i = 0$ *à* $n - 1$ **faire** `// Calcul de` $n_{\texttt{inf}}$ `et` $n_{\texttt{eq}}$
  si $T[i] < p$ **alors** $n_{\text{inf}} \leftarrow n_{\text{inf}} + 1$
  sinon si $T[i] = p$ **alors** $n_{\text{eq}} \leftarrow n_{\text{eq}} + 1$
si $k \leq n_{inf}$ **alors**
  Calculer T_{inf} et **retourner** $\textsc{Rang}(T_{inf}, k)$
sinon si $n_{inf} < k \leq n_{inf} + n_{eq}$ **alors**
  retourner p
sinon
  Calculer T_{sup} et **retourner** $\textsc{Rang}(T_{sup}, k - n_{inf} - n_{eq})$

Correction de l'algorithme

Lemme

$$\text{rang}(k, T) = \begin{cases} \text{rang}(k, T_{inf}) & si\ k \leq n_{inf} \\ p & si\ n_{inf} < k \leq n_{inf} + n_{eq} \\ \text{rang}(k - n_{inf} - n_{eq}, T_{sup}) & si\ n_{inf} + n_{eq} < k \end{cases}$$

Correction de l'algorithme

Lemme

$$\text{rang}(k, T) = \begin{cases} \text{rang}(k, T_{inf}) & si\ k \leq n_{inf} \\ p & si\ n_{inf} < k \leq n_{inf} + n_{eq} \\ \text{rang}(k - n_{inf} - n_{eq}, T_{sup}) & si\ n_{inf} + n_{eq} < k \end{cases}$$

Preuve

Cas 1 ($k \leq n_{\text{inf}}$) : soit $r = \text{rang}(k, T_{\text{inf}})$
Si $x \in T_{\text{eq}} \cup T_{\text{sup}}$, $x > r$; et il y a k éléments $\leq r$ dans T_{inf} ;
donc il y a k éléments $\leq r$ dans T

Correction de l'algorithme

Lemme

$$\text{rang}(k, T) = \begin{cases} \text{rang}(k, T_{inf}) & si\ k \leq n_{inf} \\ p & si\ n_{inf} < k \leq n_{inf} + n_{eq} \\ \text{rang}(k - n_{inf} - n_{eq}, T_{sup}) & si\ n_{inf} + n_{eq} < k \end{cases}$$

Preuve

Cas 1 $(k \leq n_{\text{inf}})$: soit $r = \text{rang}(k, T_{\text{inf}})$
Si $x \in T_{\text{eq}} \cup T_{\text{sup}}$, $x > r$; et il y a k éléments $\leq r$ dans T_{inf} ; donc il y a k éléments $\leq r$ dans T

Cas 2 $(n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}})$: soit $r = p$
Si $x \in T_{\text{sup}}$, $x > r$; et il y a $n_{\text{inf}} < k$ éléments $< r$ dans T_{inf}, et n_{eq} éléments égaux à r dans T_{eq} ; donc $\text{rang}(k, T) \in T_{\text{eq}}$

Correction de l'algorithme

Lemme

$$\text{rang}(k, T) = \begin{cases} \text{rang}(k, T_{inf}) & si\ k \leq n_{inf} \\ p & si\ n_{inf} < k \leq n_{inf} + n_{eq} \\ \text{rang}(k - n_{inf} - n_{eq}, T_{sup}) & si\ n_{inf} + n_{eq} < k \end{cases}$$

Preuve

Cas 1 $(k \leq n_{\text{inf}})$: soit $r = \text{rang}(k, T_{\text{inf}})$
Si $x \in T_{\text{eq}} \cup T_{\text{sup}}$, $x > r$; et il y a k éléments $\leq r$ dans T_{inf} ; donc il y a k éléments $\leq r$ dans T

Cas 2 $(n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}})$: soit $r = p$
Si $x \in T_{\text{sup}}$, $x > r$; et il y a $n_{\text{inf}} < k$ éléments $< r$ dans T_{inf}, et n_{eq} éléments égaux à r dans T_{eq} ; donc $\text{rang}(k, T) \in T_{\text{eq}}$

Cas 3 $(n_{\text{inf}} + n_{\text{eq}} < k)$ soit $r = \text{rang}(k - n_{\text{inf}} - n_{\text{eq}}, T_{\text{sup}})$
il y a $n_{\text{inf}} + n_{\text{eq}} < k$ éléments $< r$ dans $T_{\text{inf}} \cup T_{\text{eq}}$; il y a $k - n_{\text{inf}} - n_{\text{eq}}$ éléments $\leq r$ dans T_{sup} ; donc $k - n_{\text{inf}} - n_{\text{eq}} + (n_{\text{inf}} + n_{\text{eq}}) = k$ éléments $\leq r$ au total

Complexité

Algorithme : $\text{Rang}(T, k)$

si $k = 1$ **alors retourner** $T[0]$
$p \leftarrow \text{ChoixPivot}(T)$
Calculer n_{inf} et n_{eq}
si $k \leq n_{inf}$ **alors** **retourner** $\text{Rang}(T_{inf}, k)$
si $n_{inf} < k \leq n_{inf} + n_{eq}$ **alors retourner** p
sinon **retourner** $\text{Rang}(T_{sup}, k - n_{inf} - n_{eq})$

Complexité

Algorithme : $\textsc{Rang}(T, k)$

si $k = 1$ **alors retourner** $T[0]$
$p \leftarrow \textsc{ChoixPivot}(T)$
Calculer n_{inf} et n_{eq}
si $k \leq n_{inf}$ **alors** **retourner** $\textsc{Rang}(T_{inf}, k)$
si $n_{inf} < k \leq n_{inf} + n_{eq}$ **alors retourner** p
sinon **retourner** $\textsc{Rang}(T_{sup}, k - n_{inf} - n_{eq})$

- Calculer n_{inf}, n_{eq}, T_{inf}, T_{sup} : $O(n)$

Complexité

Algorithme : $\textsc{Rang}(T, k)$
si $k = 1$ **alors retourner** $T[0]$
$p \leftarrow \textsc{ChoixPivot}(T)$
Calculer n_{inf} et n_{eq}
si $k \leq n_{inf}$ **alors retourner** $\textsc{Rang}(T_{inf}, k)$
si $n_{inf} < k \leq n_{inf} + n_{eq}$ **alors retourner** p
sinon retourner $\textsc{Rang}(T_{sup}, k - n_{inf} - n_{eq})$

- Calculer n_{inf}, n_{eq}, T_{inf}, T_{sup} : $O(n)$
- Un appel récursif (au pire) sur T_{inf} ou T_{sup}, de taille n' :

$$t(n) = t(n') + O(n)$$

Complexité

Algorithme : $\textsc{Rang}(T, k)$

si $k = 1$ **alors retourner** $T[0]$
$p \leftarrow \textsc{ChoixPivot}(T)$
Calculer n_{inf} et n_{eq}
si $k \leq n_{inf}$ **alors** **retourner** $\textsc{Rang}(T_{inf}, k)$
si $n_{inf} < k \leq n_{inf} + n_{eq}$ **alors retourner** p
sinon **retourner** $\textsc{Rang}(T_{sup}, k - n_{inf} - n_{eq})$

- Calculer n_{inf}, n_{eq}, T_{inf}, T_{sup} : $O(n)$
- Un appel récursif (au pire) sur T_{inf} ou T_{sup}, de taille n' :

$$t(n) = t(n') + O(n)$$

  - Cas idéal : $n' = n/2 \rightsquigarrow t(n) = O(n)$ (*master theorem*)
  - Pire cas : $n' = n - 1 \rightsquigarrow t(n) = O(n^2)$ (*à la main*)

Complexité

Algorithme : $\textsc{Rang}(T, k)$
si $k = 1$ **alors retourner** $T[0]$
$p \leftarrow \textsc{ChoixPivot}(T)$
Calculer n_{inf} et n_{eq}
si $k \leq n_{inf}$ **alors** **retourner** $\textsc{Rang}(T_{inf}, k)$
si $n_{inf} < k \leq n_{inf} + n_{eq}$ **alors retourner** p
sinon **retourner** $\textsc{Rang}(T_{sup}, k - n_{inf} - n_{eq})$

- Calculer n_{inf}, n_{eq}, T_{inf}, T_{sup} : $O(n)$
- Un appel récursif (au pire) sur T_{inf} ou T_{sup}, de taille n' :

$$t(n) = t(n') + O(n)$$

  - Cas idéal : $n' = n/2 \rightsquigarrow t(n) = O(n)$ (*master theorem*)
  - Pire cas : $n' = n - 1 \rightsquigarrow t(n) = O(n^2)$ (*à la main*)

But : choix de pivot pour minimiser n' !

Choix du pivot (1)

Algorithme : $\textsc{ChoixPivot}(T)$

retourner $T[0]$

Choix du pivot (1)

Algorithme : $\textsc{ChoixPivot}(T)$
retourner $T[0]$

Complexité

- Cas le pire : tableau initialement trié

$\rightsquigarrow$ $n' = n - 1$, donc $t(n) = O(n^2)$

Choix du pivot (1)

Algorithme : ChoixPivot(T)

retourner $T[0]$

Complexité

- Cas le pire : tableau initialement trié

$\rightsquigarrow$ $n' = n - 1$, donc $t(n) = O(n^2)$

- Si tableau aléatoire : on peut montrer que $\mathbb{E}[n'] = n/2$

$\rightsquigarrow$ $t(n) = O(n)$ « en moyenne »

Choix du pivot (1)

Algorithme : ChoixPivot(T)
retourner $T[0]$

Complexité

- Cas le pire : tableau initialement trié

$\rightsquigarrow$ $n' = n - 1$, donc $t(n) = O(n^2)$

- Si tableau aléatoire : on peut montrer que $\mathbb{E}[n'] = n/2$

$\rightsquigarrow$ $t(n) = O(n)$ « en moyenne »

Choix correct si les tableaux sont aléatoires, mais en pratique ce n'est rarement le cas !

Choix du pivot (2)

Algorithme : ChoixPivot(T)

$j \leftarrow$ entier aléatoire entre 0 et $n-1$
retourner $T[j]$

Choix du pivot (2)

Algorithme : ChoixPivot(T)

$j \leftarrow$ entier aléatoire entre 0 et $n-1$
retourner $T[j]$

Complexité

- Cas le pire : si on manque de chance

$\rightsquigarrow$ $n' = n - 1$, donc $t(n) = O(n^2)$

Choix du pivot (2)

Algorithme : ChoixPivot(T)

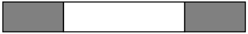
$j \leftarrow$ entier aléatoire entre 0 et $n-1$
retourner $T[j]$

Complexité

- Cas le pire : si on manque de chance

$\rightsquigarrow$ $n' = n-1$, donc $t(n) = O(n^2)$

- Mais avec probabilité $1/2$: $n' \geq n/4$



$\rightsquigarrow$ $t(n) = O(n)$ avec « bonne probabilité »

Choix du pivot (2)

Algorithme : ChoixPivot(T)

$j \leftarrow$ entier aléatoire entre 0 et $n-1$

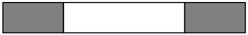
retourner $T[j]$

Complexité

- Cas le pire : si on manque de chance

$\rightsquigarrow$ $n' = n - 1$, donc $t(n) = O(n^2)$

- Mais avec probabilité $1/2$: $n' \geq n/4$



$\rightsquigarrow$ $t(n) = O(n)$ avec « bonne probabilité »

Bon choix, quelque soit le tableau, mais difficile à analyser

Choix du pivot (3)

Algorithme : $\textsc{ChoixPivot}(T)$

$j \leftarrow$ entier aléatoire entre 0 et $n-1$
Calculer n_{inf} et n_{sup} avec pivot $T[j]$
si $n_{inf} \leq 3n/4$ *et* $n_{sup} \leq 3n/4$ **alors retourner** $T[j]$
sinon retourner $\textsc{ChoixPivot}(T)$

Choix du pivot (3)

Algorithme : ChoixPivot(T)

$j \leftarrow$ entier aléatoire entre 0 et $n-1$
Calculer n_{inf} et n_{sup} avec pivot $T[j]$
si $n_{inf} \leq 3n/4$ *et* $n_{sup} \leq 3n/4$ **alors retourner** $T[j]$
sinon retourner ChoixPivot(T)

Complexité

- Cas le pire : $n' = 3n/4$
- Coût de ChoixPivot : $O(n)$ fois le nombre d'essais
- En moyenne 2 tentatives pour j car réussite avec proba. $1/2$

Choix du pivot (3)

Algorithme : ChoixPivot(T)

$j \leftarrow$ entier aléatoire entre 0 et $n-1$
Calculer n_{inf} et n_{sup} avec pivot $T[j]$
si $n_{inf} \leq 3n/4$ *et* $n_{sup} \leq 3n/4$ **alors retourner** $T[j]$
sinon retourner ChoixPivot(T)

Complexité

- Cas le pire : $n' = 3n/4$
- Coût de ChoixPivot : $O(n)$ fois le nombre d'essais
- En moyenne 2 tentatives pour j car réussite avec proba. $1/2$

- $\mathbb{E}[t(n)] \leq \mathbb{E}[t(3n/4)] + O(n) \rightsquigarrow \mathbb{E}[t(n)] = O(n)$ (*master thm*)

Choix du pivot (3)

Algorithme : ChoixPivot(T)

$j \leftarrow$ entier aléatoire entre 0 et $n-1$
Calculer n_{inf} et n_{sup} avec pivot $T[j]$
si $n_{inf} \leq 3n/4$ *et* $n_{sup} \leq 3n/4$ **alors retourner** $T[j]$
sinon retourner ChoixPivot(T)

Complexité

- Cas le pire : $n' = 3n/4$
- Coût de ChoixPivot : $O(n)$ fois le nombre d'essais
- En moyenne 2 tentatives pour j car réussite avec proba. $1/2$

- $\mathbb{E}[t(n)] \leq \mathbb{E}[t(3n/4)] + O(n) \rightsquigarrow \mathbb{E}[t(n)] = O(n)$ (*master thm*)

Bon choix, *facile* à analyser. En pratique, préférer le précédent !

Choix du pivot (bonus !)

Algorithme : $\textsc{ChoixPivot}(T)$

pour $i = 0$ *à* $\lceil n/5 \rceil - 1$ **faire**

 $m_i \leftarrow \textsc{Mediane}(T[5i, 5i+4])$

retourner $\textsc{Mediane}([m_0, \ldots, m_{\lceil n/5 \rceil - 1}])$

$(\textsc{Mediane}(T) = \textsc{Rang}(T, \lfloor n/2 \rfloor))$

Choix du pivot (bonus !)

Algorithme : $\textsc{ChoixPivot}(T)$

pour $i = 0$ *à* $\lceil n/5 \rceil - 1$ **faire**

  $m_i \leftarrow \textsc{Mediane}(T[5i, 5i+4])$

retourner $\textsc{Mediane}([m_0, \ldots, m_{\lceil n/5 \rceil - 1}])$

$(\textsc{Mediane}(T) = \textsc{Rang}(T, \lfloor n/2 \rfloor))$

Complexité

- Cas le pire : on peut montrer que $n' \leq 7n/10 + 6$ *(pas si facile)*
- Coût de $\textsc{ChoixPivot}$: $O(n) + t(\lceil n/5 \rceil)$

Choix du pivot (bonus !)

Algorithme : $\textsc{ChoixPivot}(T)$

pour $i = 0$ *à* $\lceil n/5 \rceil - 1$ **faire**
  $m_i \leftarrow \textsc{Mediane}(T[5i, 5i+4])$
retourner $\textsc{Mediane}([m_0, \ldots, m_{\lceil n/5 \rceil - 1}])$

$(\textsc{Mediane}(T) = \textsc{Rang}(T, \lfloor n/2 \rfloor))$

Complexité

- Cas le pire : on peut montrer que $n' \leq 7n/10 + 6$ *(pas si facile)*
- Coût de $\textsc{ChoixPivot}$: $O(n) + t(\lceil n/5 \rceil)$

- $t(n) \leq t(7n/10 + 6) + t(\lceil n/5 \rceil) + O(n)$

$\rightsquigarrow$ $t(n) = O(n)$ *(par récurrence pas si facile)*

Choix du pivot (bonus !)

Algorithme : $\textsc{ChoixPivot}(T)$

pour $i = 0$ *à* $\lceil n/5 \rceil - 1$ **faire**
  $m_i \leftarrow \textsc{Mediane}(T[5i, 5i+4])$

retourner $\textsc{Mediane}([m_0, \ldots, m_{\lceil n/5 \rceil - 1}])$

$(\textsc{Mediane}(T) = \textsc{Rang}(T, \lfloor n/2 \rfloor))$

Complexité

- Cas le pire : on peut montrer que $n' \leq 7n/10 + 6$ *(pas si facile)*
- Coût de $\textsc{ChoixPivot}$: $O(n) + t(\lceil n/5 \rceil)$

- $t(n) \leq t(7n/10 + 6) + t(\lceil n/5 \rceil) + O(n)$

$\rightsquigarrow$ $t(n) = O(n)$ *(par récurrence pas si facile)*

Algorithme déterministe : bien en théorie, moins en pratique !

Récapitulatif

Théorème

RANG(T, k) *retourne le* $k^{\text{ème}}$ *élément de* T.

En fonction de ChoixPivot, *sa complexité peut être*

- $O(n^2)$ *dans le pire des cas mais* $O(n)$ « *en moyenne* »
- $O(n)$ *avec bonne probabilité, pour tout tableau*
- $O(n)$ *de manière déterministe, pour tout tableau*

Récapitulatif

Théorème

$\textsc{rang}(T, k)$ retourne le $k^{\text{ème}}$ élément de T.
En fonction de $\textsc{ChoixPivot}$, sa complexité peut être

- *$O(n^2)$ dans le pire des cas mais $O(n)$ « en moyenne »*
- *$O(n)$ avec bonne probabilité, pour tout tableau*
- *$O(n)$ de manière déterministe, pour tout tableau*

Remarques

- Le choix de pivot $T[0]$ fonctionne avec bonne probabilité si on mélange aléatoirement T au début
- Version plus complexe de cet algorithme : tri $\textsc{QuickSort}$