

# Chapitre 5

## Diviser pour régner

HLIN401 : Algorithmique et complexité

Université de Montpellier  
2018 – 2019

1. Premier exemple : tri fusion

2. Qu'est-ce que « diviser pour régner » ?

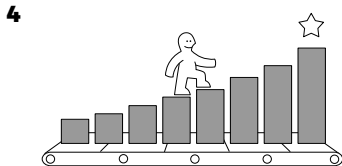
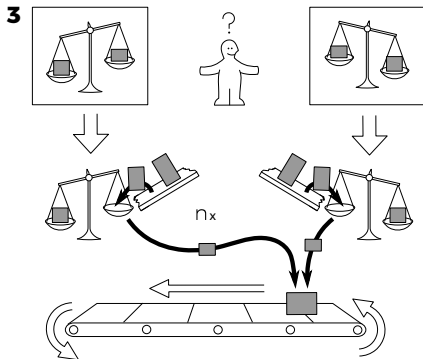
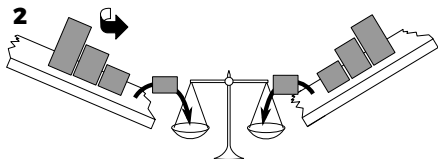
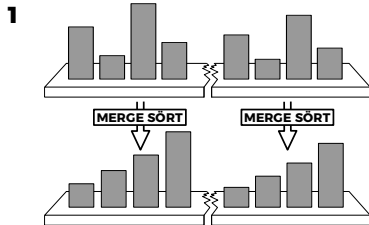
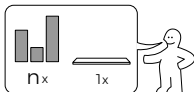
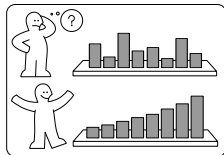
3. Deuxième exemple : multiplication d'entiers

4. Exemple spécial : Calcul de rang

# MERGE SÖRT

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

IDEA



# Algorithme du TRIFUSION

**Algorithme :** TRIFUSION( $T$ )

$n \leftarrow \text{taille}(T)$

**si**  $n = 1$  **alors**

    Retourner  $T$

**sinon**

$T_1 \leftarrow \text{TRIFUSION}(T[0, \lfloor n/2 \rfloor - 1])$

$T_2 \leftarrow \text{TRIFUSION}(T[\lfloor n/2 \rfloor, n])$

    Retourner FUSION( $T_1, T_2$ )

# Algorithme du TRIFUSION

**Algorithme :** TRIFUSION( $T$ )

$n \leftarrow \text{taille}(T)$

**si**  $n = 1$  **alors**

    Retourner  $T$

**sinon**

$T_1 \leftarrow \text{TRIFUSION}(T[0, \lfloor n/2 \rfloor - 1])$

$T_2 \leftarrow \text{TRIFUSION}(T[\lfloor n/2 \rfloor, n])$

    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( $T_1, T_2$ )

$n_1 \leftarrow \text{taille}(T_1)$ ;  $n_2 \leftarrow \text{taille}(T_2)$

$S \leftarrow$  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( $T_1, T_2$ )

$n_1 \leftarrow \text{taille}(T_1)$ ;  $n_2 \leftarrow \text{taille}(T_2)$

$S \leftarrow$  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),  $\text{FUSION}(T_1, T_2)$  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( $T$ )

$n \leftarrow \text{taille}(T)$

**si**  $n = 1$  **alors**

    Retourner  $T$

**sinon**

$T_1 \leftarrow \text{TRIFUSION}(T[0, \lfloor n/2 \rfloor - 1])$

$T_2 \leftarrow \text{TRIFUSION}(T[\lfloor n/2 \rfloor, n])$

    Retourner FUSION( $T_1, T_2$ )

# Retour sur le TRIFUSION

## Théorème

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

**Algorithme :** TRIFUSION( $T$ )

$n \leftarrow \text{taille}(T)$

**si**  $n = 1$  **alors**

    Retourner  $T$

**sinon**

$T_1 \leftarrow \text{TRIFUSION}(T[0, \lfloor n/2 \rfloor - 1])$

$T_2 \leftarrow \text{TRIFUSION}(T[\lfloor n/2 \rfloor, n])$

    Retourner FUSION( $T_1, T_2$ )

**Preuve de correction** par récurrence (facile!)

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

# Retour sur le TRIFUSION

## Théorème

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

**Algorithme :** TRIFUSION( $T$ )

$n \leftarrow \text{taille}(T)$

**si**  $n = 1$  **alors**

    Retourner  $T$

**sinon**

$T_1 \leftarrow \text{TRIFUSION}(T[0, \lfloor n/2 \rfloor - 1])$

$T_2 \leftarrow \text{TRIFUSION}(T[\lfloor n/2 \rfloor, n])$

    Retourner FUSION( $T_1, T_2$ )

**Preuve de correction** par récurrence (facile!)

- ▶ Si  $n = 1$ , OK
- ▶ Si  $n > 1$ ,  $\lfloor n/2 \rfloor \leq \lceil n/2 \rceil < n$ , 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

1. Premier exemple : tri fusion
2. Qu'est-ce que « diviser pour régner » ?
3. Deuxième exemple : multiplication d'entiers
4. Exemple spécial : Calcul de rang

# 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 »

Réurrence(s) sur la taille du problème

# Analyse d'un algorithme « diviser pour régner »

Réurrence(s) sur la taille du problème

- Preuve de correction

# Analyse d'un algorithme « diviser pour régner »

## Récurrance(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 »

## Réurrence(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  $n \geq n_0$ ,*

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

*Alors*

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

# 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  $n \geq n_0$ ,*

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

*Alors*

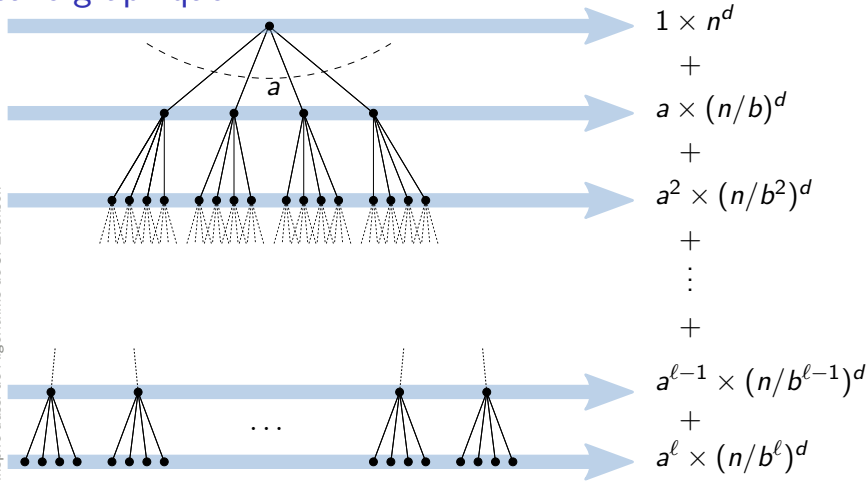
$$T(n) = \begin{cases} O(n^d) & \text{si } b^d > a \\ O(n^d \log n) & \text{si } b^d = a \\ O(n^{\frac{\log a}{\log b}}) & \text{si } b^d < a \end{cases}$$

## Exemple du tri fusion

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

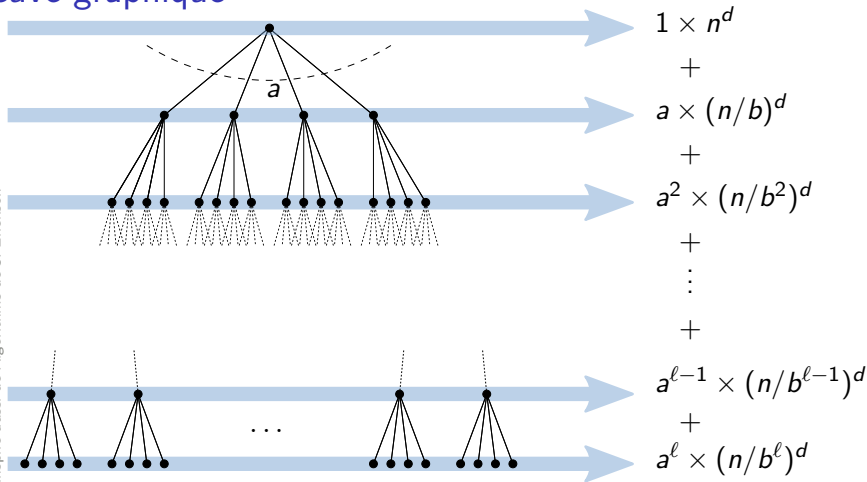
# Preuve graphique

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



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



# 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 »

1. Premier exemple : tri fusion
2. Qu'est-ce que « diviser pour régner » ?
3. Deuxième exemple : multiplication d'entiers
4. Exemple spécial : Calcul de rang

# 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

## Multiplication d'entiers

**Entrée** Deux entiers  $A$  et  $B$  écrits en base 10

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

$$\begin{array}{r} 1382 \\ \times 7634 \\ \hline 5528 \\ + 4146 \\ + 8292 \\ + 9674 \\ \hline 10550188 \end{array}$$

# 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

$$\begin{array}{r} \phantom{\times} 1382 \\ \times \phantom{00} 7634 \\ \hline \phantom{00} 5528 \\ + \phantom{000} 4146 \\ + \phantom{0000} 8292 \\ + \phantom{00000} 9674 \\ \hline 10550188 \end{array}$$

## Question

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

# 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

$$\begin{array}{r} \phantom{\times} 1382 \\ \times \phantom{00} 7634 \\ \hline \phantom{00} 5528 \\ + \phantom{000} 4146 \\ + \phantom{0000} 8292 \\ + \phantom{00000} 9674 \\ \hline 10550188 \end{array}$$

## Question

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

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

# 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

$$\begin{array}{r} 1382 \\ \times 7634 \\ \hline 5528 \\ + 4146 \phantom{0} \\ + 8292 \phantom{00} \\ + 9674 \phantom{000} \\ \hline 10550188 \end{array}$$

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

$$A = 1382, B = 7634$$

$$A_1 = 13, A_0 = 82$$

$$B_1 = 76, B_0 = 34$$

$$C_{00} = 2788, C_{01} = 6232$$

$$C_{10} = 442, C_{11} = 988$$

$$\begin{aligned} C &= 2788 + 100 \cdot (6232 \\ &\quad + 442) + 10000 \cdot 988 \\ &= 10550188 \end{aligned}$$

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

Preuve de correction :

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

$$A = 1382, B = 7634$$

$$A_1 = 13, A_0 = 82$$

$$B_1 = 76, B_0 = 34$$

$$C_{00} = 2788, C_{01} = 6232$$

$$C_{10} = 442, C_{11} = 988$$

$$\begin{aligned} C &= 2788 + 100 \cdot (6232 \\ &\quad + 442) + 10000 \cdot 988 \\ &= 10550188 \end{aligned}$$

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

$$A = 1382, B = 7634$$

$$A_1 = 13, A_0 = 82$$

$$B_1 = 76, B_0 = 34$$

$$C_{00} = 2788, C_{01} = 6232$$

$$C_{10} = 442, C_{11} = 988$$

$$C = 2788 + 100 \cdot (6232 + 442) + 10000 \cdot 988 \\ = 10550188$$

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

►  $a = 4, b = 2, d = 1 : b^d < a$

►  $T(n) = O(n^{\log a / \log b}) = O(n^{\log 4 / \log 2}) = O(n^2) \dots$

## Idée de Karatsuba (version Knuth)

$$A_0B_1 + A_1B_0 = A_0B_0 + A_1B_1 - (A_0 - A_1)(B_0 - B_1)$$

## Idée de Karatsuba (version Knuth)

$$A_0B_1 + A_1B_0 = A_0B_0 + A_1B_1 - (A_0 - A_1)(B_0 - B_1)$$

- ▶  $A_0B_0$  et  $A_1B_1$  sont calculés de toute façon
- ▶ un seul produit en plus !

# Idée de Karatsuba (version Knuth)

$$A_0B_1 + A_1B_0 = A_0B_0 + A_1B_1 - (A_0 - A_1)(B_0 - B_1)$$

- ▶  $A_0B_0$  et  $A_1B_1$  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

×	+	-
+	+	-
-	-	+



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

$$A = 1382, B = 7634$$

$$A_1 = 13, A_0 = 82$$

$$B_1 = 76, B_0 = 34$$

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

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

$$\begin{aligned} C &= 2788 + 100 \cdot (2788 \\ &\quad + 988 + 2898) + 10000 \cdot 988 \\ &= 10550188 \end{aligned}$$

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

$C_{00} \leftarrow \text{KARATSUBA}(A_0, B_0)$

$C_{11} \leftarrow \text{KARATSUBA}(A_1, B_1)$

$D \leftarrow \text{KARATSUBA}(|A_0 - A_1|, |B_0 - B_1|)$

$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$

**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 :** KARATSUBA( $A, B$ )

**si**  $A$  et  $B$  n'ont qu'un chiffre **alors retourner**  $a_0b_0$

Écrire  $A$  sous la forme  $A_0 + 10^{\lfloor n/2 \rfloor} A_1$

Écrire  $B$  sous la forme  $B_0 + 10^{\lfloor n/2 \rfloor} B_1$

$C_{00} \leftarrow \text{KARATSUBA}(A_0, B_0)$

$C_{11} \leftarrow \text{KARATSUBA}(A_1, B_1)$

$D \leftarrow \text{KARATSUBA}(|A_0 - A_1|, |B_0 - B_1|)$

$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$

**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 :** KARATSUBA( $A, B$ )

**si**  $A$  et  $B$  n'ont qu'un chiffre **alors retourner**  $a_0b_0$

Écrire  $A$  sous la forme  $A_0 + 10^{\lfloor n/2 \rfloor} A_1$

Écrire  $B$  sous la forme  $B_0 + 10^{\lfloor n/2 \rfloor} B_1$

$C_{00} \leftarrow \text{KARATSUBA}(A_0, B_0)$

$C_{11} \leftarrow \text{KARATSUBA}(A_1, B_1)$

$D \leftarrow \text{KARATSUBA}(|A_0 - A_1|, |B_0 - B_1|)$

$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$

**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 KARATSUBA termine.

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

# Correction de l'algorithme de Karatsuba

**Algorithme :** KARATSUBA( $A, B$ )

**si**  $A$  et  $B$  n'ont qu'un chiffre **alors retourner**  $a_0b_0$

Écrire  $A$  sous la forme  $A_0 + 10^{\lfloor n/2 \rfloor} A_1$

Écrire  $B$  sous la forme  $B_0 + 10^{\lfloor n/2 \rfloor} B_1$

$C_{00} \leftarrow \text{KARATSUBA}(A_0, B_0)$

$C_{11} \leftarrow \text{KARATSUBA}(A_1, B_1)$

$D \leftarrow \text{KARATSUBA}(|A_0 - A_1|, |B_0 - B_1|)$

$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$

**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  $K(n) \leq 3K(\lceil n/2 \rceil) + O(n)$



# Correction de l'algorithme de Karatsuba

**Algorithme :** KARATSUBA( $A, B$ )

**si**  $A$  et  $B$  n'ont qu'un chiffre **alors retourner**  $a_0b_0$

Écrire  $A$  sous la forme  $A_0 + 10^{\lfloor n/2 \rfloor} A_1$

Écrire  $B$  sous la forme  $B_0 + 10^{\lfloor n/2 \rfloor} B_1$

$C_{00} \leftarrow \text{KARATSUBA}(A_0, B_0)$

$C_{11} \leftarrow \text{KARATSUBA}(A_1, B_1)$

$D \leftarrow \text{KARATSUBA}(|A_0 - A_1|, |B_0 - B_1|)$

$s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$

**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  $K(n) \leq 3K(\lceil n/2 \rceil) + O(n)$

## Corollaire (*master theorem*)

$$K(n) = O(n^{\log 3 / \log 2}) = \mathbf{O}(n^{\log 3}) \simeq O(n^{1.58})$$

# 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)
- ▶ 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  $O(n \log n 2^{2^{\log^*(n)}})$

1. Premier exemple : tri fusion
2. Qu'est-ce que « diviser pour régner » ?
3. Deuxième exemple : multiplication d'entiers
4. Exemple spécial : Calcul de rang

# Définition et algorithmes $\pm$ naïfs

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

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



# Définition et algorithmes $\pm$ naïfs

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

**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  $T[j] \leq T[i]$  alors  
       $c \leftarrow c + 1$   
  si  $c = k$  alors retourner  $T[i]$ 
```

# Définition et algorithmes $\pm$ naïfs

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

**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  $T[j] \leq T[i]$  alors  
       $c \leftarrow c + 1$   
  si  $c = k$  alors retourner  $T[i]$ 
```

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

```
Trier  $T$   
retourner  $T[k - 1]$ 
```

# Définition et algorithmes $\pm$ naïfs

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

**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  $T[j] \leq T[i]$  alors  
       $c \leftarrow c + 1$   
  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 :

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

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

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

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

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 :

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

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

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

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 :

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

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

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

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 :**  $\text{RANG}(T, k)$

**si**  $k = 1$  **alors retourner**  $T[0]$

$p \leftarrow \text{CHOIXPIVOT}(T)$

$n_{\text{inf}} \leftarrow 0, n_{\text{eq}} \leftarrow 0$

**pour**  $i = 0$  **à**  $n - 1$  **faire**

**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**  $k \leq n_{\text{inf}}$  **alors**

    Calculer  $T_{\text{inf}}$  et **retourner**  $\text{RANG}(T_{\text{inf}}, k)$

**sinon si**  $n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}}$  **alors**

**retourner**  $p$

**sinon**

    Calculer  $T_{\text{sup}}$  et **retourner**  $\text{RANG}(T_{\text{sup}}, k - n_{\text{inf}} - n_{\text{eq}})$



# Correction de l'algorithme

## Lemme

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

# Correction de l'algorithme

## Lemme

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

## 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  $T_{\text{inf}}$ ;  
donc il y a  $k$  éléments  $\leq r$  dans  $T$

# Correction de l'algorithme

## Lemme

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

## 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  $T_{\text{inf}}$ ;  
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  $T_{\text{inf}}$ ,  
et  $n_{\text{eq}}$  éléments égaux à  $r$  dans  $T_{\text{eq}}$ ; donc  $\text{rang}(k, T) \in T_{\text{eq}}$

# Correction de l'algorithme

## Lemme

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

## 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  $T_{\text{inf}}$ ; 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  $T_{\text{inf}}$ , et  $n_{\text{eq}}$  éléments égaux à  $r$  dans  $T_{\text{eq}}$ ; 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  $T_{\text{sup}}$ ; 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  $n_{\text{inf}}$  et  $n_{\text{eq}}$

**si**  $k \leq n_{\text{inf}}$  **alors retourner**  $\text{RANG}(T_{\text{inf}}, k)$

**si**  $n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}}$  **alors retourner**  $p$

**sinon retourner**  $\text{RANG}(T_{\text{sup}}, k - n_{\text{inf}} - n_{\text{eq}})$

# Complexité

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

**si**  $k = 1$  **alors retourner**  $T[0]$

$p \leftarrow \text{CHOIXPIVOT}(T)$

Calculer  $n_{\text{inf}}$  et  $n_{\text{eq}}$

**si**  $k \leq n_{\text{inf}}$  **alors retourner**  $\text{RANG}(T_{\text{inf}}, k)$

**si**  $n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}}$  **alors retourner**  $p$

**sinon retourner**  $\text{RANG}(T_{\text{sup}}, k - n_{\text{inf}} - n_{\text{eq}})$

► Calculer  $n_{\text{inf}}$ ,  $n_{\text{eq}}$ ,  $T_{\text{inf}}$ ,  $T_{\text{sup}}$  :  $O(n)$

# Complexité

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

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

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

# Complexité

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

- Calculer  $n_{\text{inf}}, n_{\text{eq}}, T_{\text{inf}}, T_{\text{sup}} : O(n)$
- Un appel récursif (au pire) sur  $T_{\text{inf}}$  ou  $T_{\text{sup}}$ , 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 : RANG( $T, k$ )  
si  $k = 1$  alors retourner  $T[0]$   
 $p \leftarrow \text{CHOIXPIVOT}(T)$   
Calculer  $n_{\text{inf}}$  et  $n_{\text{eq}}$   
si  $k \leq n_{\text{inf}}$  alors retourner RANG( $T_{\text{inf}}, k$ )  
si  $n_{\text{inf}} < k \leq n_{\text{inf}} + n_{\text{eq}}$  alors retourner  $p$   
sinon retourner RANG( $T_{\text{sup}}, k - n_{\text{inf}} - n_{\text{eq}}$ )
```

- ▶ Calculer  $n_{\text{inf}}, n_{\text{eq}}, T_{\text{inf}}, T_{\text{sup}} : O(n)$
- ▶ Un appel récursif (au pire) sur  $T_{\text{inf}}$  ou  $T_{\text{sup}}$ , 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 :** CHOIXPIVOT( $T$ )

**retourner**  $T[0]$

# Choix du pivot (1)

**Algorithme :** CHOIXPIVOT( $T$ )  
**retourner**  $T[0]$

## Complexité

► Cas le pire : tableau initialement trié

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

# Choix du pivot (1)

**Algorithme :** CHOIXPIVOT( $T$ )  
**retourner**  $T[0]$

## Complexité

- ▶ Cas le pire : tableau initialement trié

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

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

↪  $t(n) = O(n)$  « en moyenne »

## Choix du pivot (1)

**Algorithme :** CHOIXPIVOT( $T$ )

**retourner**  $T[0]$

### Complexité

► Cas le pire : tableau initialement trié

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

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

↪  $t(n) = O(n)$  « 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

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

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

↪  $t(n) = O(n)$  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

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

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

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



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

## Choix du pivot (3)

**Algorithme :** CHOIXPIVOT( $T$ )

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

Calculer  $n_{\text{inf}}$  et  $n_{\text{sup}}$  avec pivot  $T[j]$

**si**  $n_{\text{inf}} \leq 3n/4$  **et**  $n_{\text{sup}} \leq 3n/4$  **alors retourner**  $T[j]$

**sinon retourner** CHOIXPIVOT( $T$ )

## Choix du pivot (3)

**Algorithme :** CHOIXPIVOT( $T$ )

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

Calculer  $n_{\text{inf}}$  et  $n_{\text{sup}}$  avec pivot  $T[j]$

**si**  $n_{\text{inf}} \leq 3n/4$  et  $n_{\text{sup}} \leq 3n/4$  **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  $n_{\text{inf}}$  et  $n_{\text{sup}}$  avec pivot  $T[j]$

**si**  $n_{\text{inf}} \leq 3n/4$  et  $n_{\text{sup}} \leq 3n/4$  **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  $n_{\text{inf}}$  et  $n_{\text{sup}}$  avec pivot  $T[j]$

**si**  $n_{\text{inf}} \leq 3n/4$  et  $n_{\text{sup}} \leq 3n/4$  **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 :** CHOIXPIVOT( $T$ )

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

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

**retourner**  $\text{MEDIANE}([m_0, \dots, m_{\lceil n/5 \rceil - 1}])$

( $\text{MEDIANE}(T) = \text{RANG}(T, \lfloor n/2 \rfloor)$ )

## Choix du pivot (bonus!)

**Algorithme :** CHOIXPIVOT( $T$ )

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

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

**retourner**  $\text{MEDIANE}([m_0, \dots, m_{\lceil n/5 \rceil - 1}])$

( $\text{MEDIANE}(T) = \text{RANG}(T, \lfloor n/2 \rfloor)$ )

### Complexité

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

## Choix du pivot (bonus!)

**Algorithme :** CHOIXPIVOT( $T$ )

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

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

**retourner**  $\text{MEDIANE}([m_0, \dots, m_{\lceil n/5 \rceil - 1}])$

( $\text{MEDIANE}(T) = \text{RANG}(T, \lfloor n/2 \rfloor)$ )

### Complexité

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

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

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



## Choix du pivot (bonus!)

**Algorithme :** CHOIXPIVOT( $T$ )

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

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

**retourner**  $\text{MEDIANE}([m_0, \dots, m_{\lceil n/5 \rceil - 1}])$

( $\text{MEDIANE}(T) = \text{RANG}(T, \lfloor n/2 \rfloor)$ )

### Complexité

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

$$\text{▶ } t(n) \leq t(7n/10 + 6) + t(\lceil n/5 \rceil) + O(n)$$

$$\rightsquigarrow t(n) = O(n) \qquad \text{(par récurrence pas si facile)}$$

Algorithme déterministe : bien en théorie, moins en pratique !

# Récapitulatif

## Théorème

$\text{RANG}(T, k)$  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

$\text{RANG}(T, k)$  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  $O(n) \ll$  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 QUICKSORT