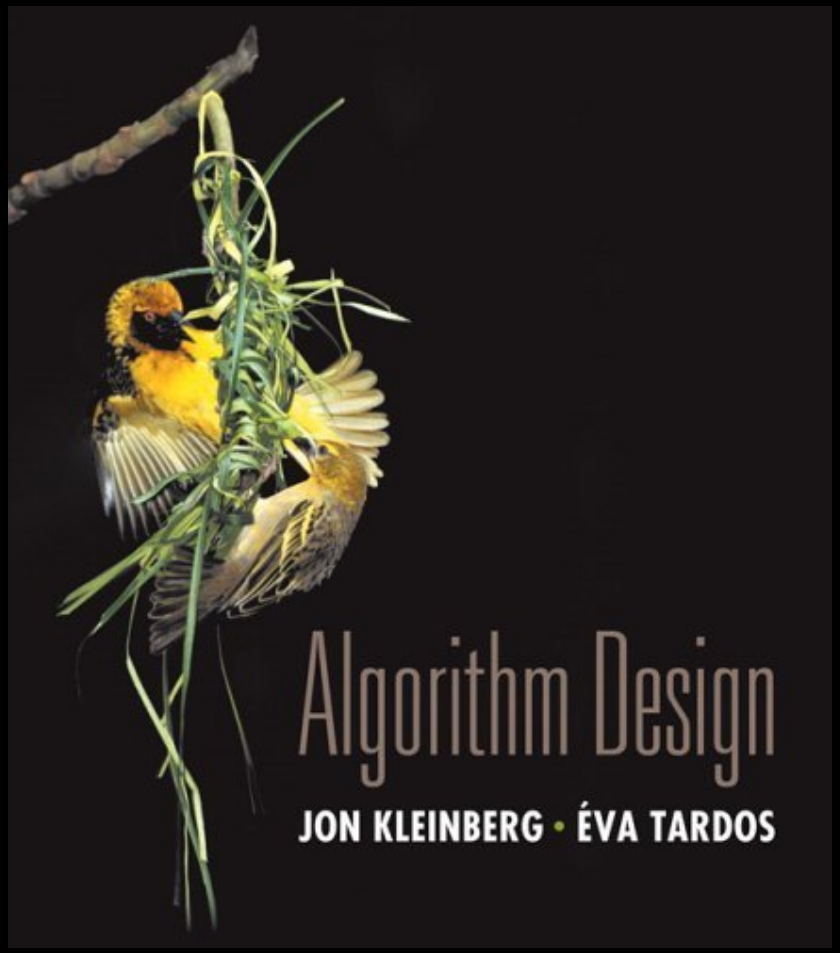


Chapitre 5

Diviser pour régner

Diviser pour régner (Divide-and-Conquer)

Diviser pour régner.

- "Casser" le problème en plusieurs parties.
- Résoudre chaque partie récursivement.
- Combiner les solutions des sous-problèmes en une solution globale.

Utilisation la plus fréquente.

- Casser un pb de taille n en deux parties de taille $\frac{1}{2}n$.
- Résoudre les deux parties récursivement.
- Combiner les deux solutions en une solution globale en temps linéaire.

Conséquence.

- Force brute : n^2.
- Diviser pour régner : $n \log n$.

Divide et impera.
Veni, vidi, vici.
- *Julius Caesar*

Tri fusion (mergesort)

Tri

Applications immédiates du tri.

- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.

Problèmes qui deviennent plus faciles une fois les objets triés.

- Find the median.
- Find the closest pair
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

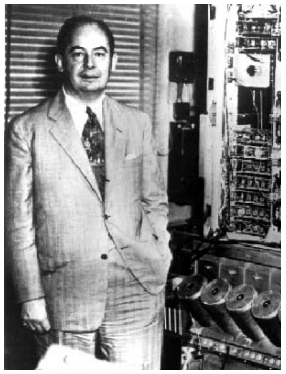
Applications moins immédiates du tri.

- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.
- . . .

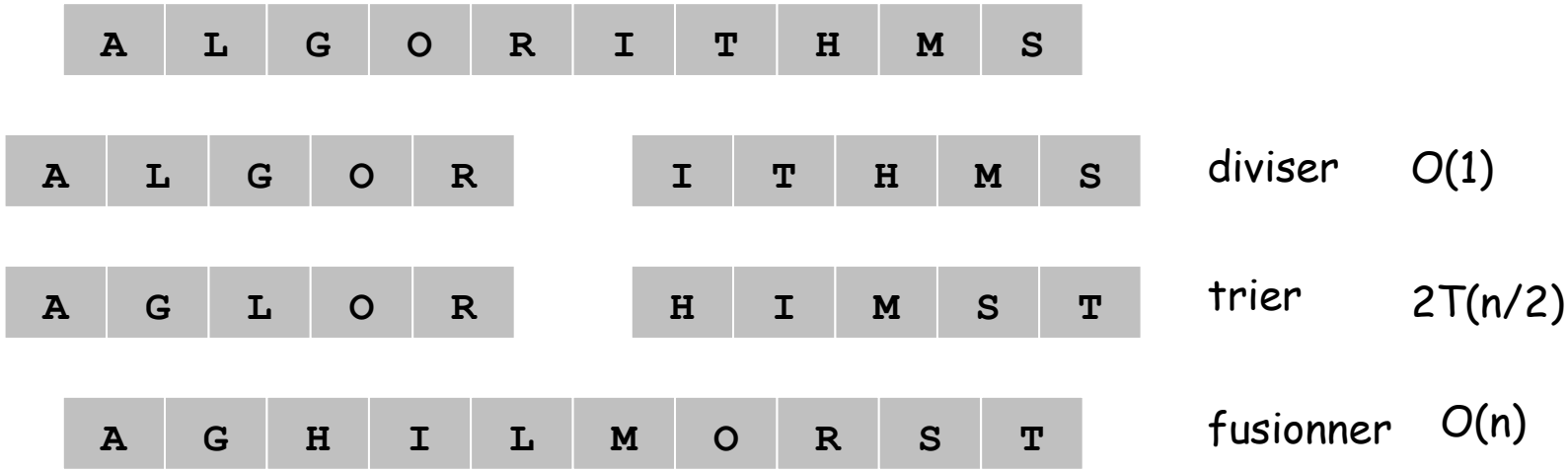
Tri fusion

Tri fusion.

- Diviser le tableau en deux moitiés.
- Trier récursivement chaque moitié.
- Fusionner les deux moitiés en un tableau complet trié.



Jon von Neumann (1945)

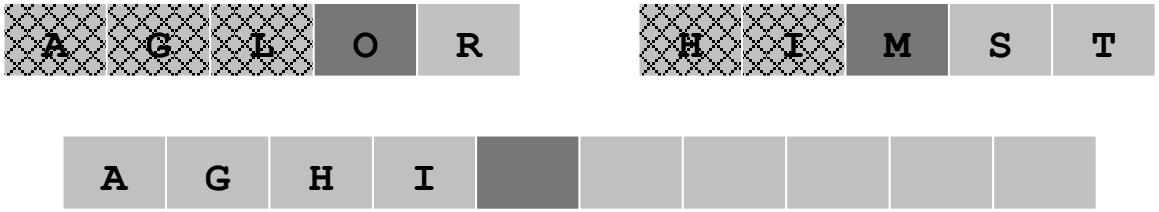


Fusion

Fusion. Combiner deux listes triées en une liste complète triée.

Comment fusionner efficacement?

- Nombre linéaire de comparaisons.
- Utilisation d'un tableau temporaire.



Défi pour les blasés. Fusion sur place (In-place merge.) [Kronrud, 1969]

Utilise un espace auxiliaire de taille constante

Une relation de récurrence utile

Déf. T(n) = nbre de comparaisons pour trier (par tri fusion) une liste de n éléments.

Récurrence du tri-fusion.

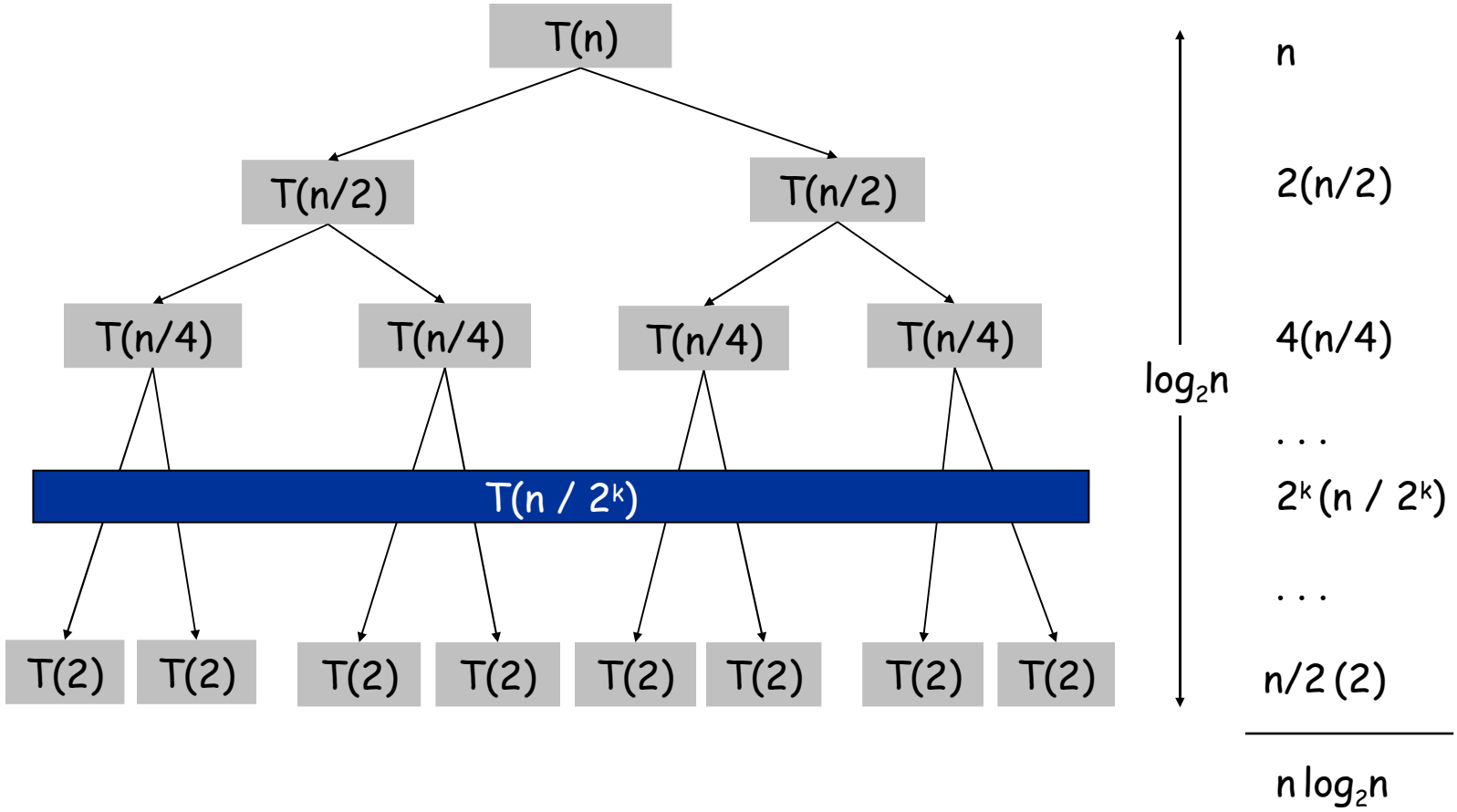
$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Solution. $T(n) = O(n \log_2 n)$.

Preuves variées. Plusieurs façons de résoudre cette récurrence. On commence par supposer que n est une puissance de 2 et on remplace $\leq$ par =.

Preuve (technique de l'arbre de récursion)

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



Preuve (technique du télescopage)

Propriété. Si T(n) satisfait cette récurrence, alors $T(n) = n \log_2 n$.

on suppose que n est une puissance de 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Preuve. Pour n > 1:

$$\begin{aligned} \frac{T(n)}{n} &\leq \frac{2T(n/2)}{n} + 1 \\ &\leq \frac{T(n/2)}{n/2} + 1 \\ &\leq \frac{T(n/4)}{n/4} + 1 + 1 \\ &\leq L \\ &\leq \frac{T(n/n)}{n/n} \underbrace{+ 1 + L + 1}_{\log_2 n} \\ &\leq \log_2 n \end{aligned}$$

Preuve par induction

Propriété. Si T(n) satisfait cette récurrence, alors $T(n) = n \log_2 n$.

↑ on suppose que n est une puissance de 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Preuve. (par induction sur n)

- Cas de base : n = 1.
- Hypothèse d'induction : $T(n) = n \log_2 n$.
- But : montrer que $T(2n) = 2n \log_2 (2n)$.

$$\begin{aligned} T(2n) &\stackrel{?}{=} 2T(n) + 2n \\ &\stackrel{?}{=} 2n \log_2 n + 2n \\ &\stackrel{?}{=} 2n\left(\log_2(2n) - 1\right) + 2n \\ &\stackrel{?}{=} 2n \log_2 (2n) \end{aligned}$$

Analyse de la récurrence du tri fusion

Propriété. si T(n) satisfait la récurrence suivante, alors $T(n) \le n\lceil \lg n \rceil$.

$\lceil \lg n \rceil$ ↑ $\log_2 n$

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Preuve. (par induction sur n)

- Cas de base : n = 1.
- On pose $n_1 = \lfloor n / 2 \rfloor$, $n_2 = \lceil n / 2 \rceil$.
- hypothèse d'induction : $T(k) \le k\lceil \lg k \rceil$ satisfait par k=1, 2, … , n–1.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1\lceil \lg n_1 \rceil + n_2\lceil \lg n_2 \rceil + n \\ &\le n_1\lceil \lg n_2 \rceil + n_2\lceil \lg n_2 \rceil + n \\ &\le n\lceil \lg n_2 \rceil + n \\ &\le n(\lceil \lg n \rceil - 1) + n \\ &\le n\lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &\le \lceil n/2 \rceil \\ &\le \lceil 2^{\lceil \lg n \rceil} / 2 \rceil \\ &\le 2^{\lceil \lg n \rceil} / 2 \end{aligned}$$

$$\Rightarrow \lg n_2 \le \lceil \lg n \rceil - 1$$

Nombre d'inversions

Inversions

Un site de musique en ligne cherche à comparer les goûts des internautes.

- Vous classez n titres.
- Le site consulte sa base de données à la recherche de personnes ayant des goûts similaires.

Mesure de similitude : nombre d'inversions entre deux classements.

- Votre classement : 1, 2, …, n.
- Un autre classement : $a_1, a_2, \ldots, a_n$.
- Les titres i et j sont inversés si i < j, mais $a_i > a_j$.

titres

	A	B	C	D	E
Vous	1	2	3	4	5
X	1	3	4	2	5

Inversions

3-2, 4-2

- Force brute : vérifier les $\Theta(n^2)$ couples (i,j).

Applications

Applications.

- Théorie du vote.
- Filtrage (Collaborative filtering.)
- Mesure du degré de "rangement" d'un tableau.
- Analyse de sensibilité de la fonction de ranking de Google.
- Agrégation de classements pour la méta-recherche sur le Web.
- Statistique non paramétrique (par exemple, distance T de Kendall.)

Nombre d'inversions : diviser pour régner

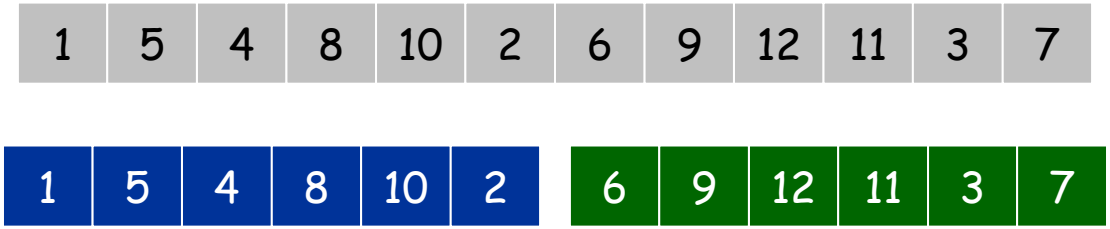
Diviser pour régner (divide and conquer.)

1	5	4	8	10	2	6	9	12	11	3	7

Nombre d'inversions : diviser pour régner

Diviser pour régner (divide and conquer.)

- Divide : séparer la liste en deux morceaux.



Divide : $O(1)$.

Nombre d'inversions : diviser pour régner

Diviser pour régner (divide and conquer.)

- Divide : séparer la liste en deux morceaux.
- Conquer : compter récursivement le nbre d'inversions dans chaque partie.

Divide : $O(1)$.

Conquer : $2T(n / 2)$

5 inversions bleu-bleu

8 inversions vert-vert

5-4, 5-2, 4-2, 8-2, 10-2

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Nombre d'inversions : diviser pour régner

Diviser pour régner.

- Divide : séparer la liste en deux morceaux.
- Conquer : compter récursivement les inversions dans chaque partie.
- Combiner : compter les inversions entre éléments de parties distinctes, et retourner la somme des trois quantités.

Divide : $O(1)$.

Conquer : $2T(n / 2)$

5 inversions bleu-bleu

8 inversions vert-vert

9 inversions bleu-vert
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combiner : ???

Total = 5 + 8 + 9 = 22.

Nombre d'inversions : combiner

Combiner : compter les inversions bleu-vert

- On suppose que chaque partie est triée.
- On compte les inversions entre i et j, où a_i et a_j sont dans des parties distinctes.
- On fusionne les deux parties triées en une liste triée.

afin de préserver l'invariant de tri

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 inversions bleu-vert : 6 + 3 + 2 + 2 + 0 + 0

Compter : $O(n)$

2	3	7	10	11	14	16	17	18	19	23	25

Fusionner : $O(n)$

$$T(n) \le T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

Nombre d'inversions : implémentation

Pré-condition. [Fusionner-et-Compter] A et B sont triées.

Post-condition. [Trier-et-Compter] L est triée.

```
Trier-et-Compter(L) {
   si la liste L a un élément
      retourner 0 et la liste L

   Diviser la liste en deux moitiés A et B
   (r_A, A) ← Trier-et-Compter(A)
   (r_B, B) ← Trier-et-Compter(B)
   (r , L) ← Fusionner-et-Compter(A, B)

   retourner r = r_A + r_B + r et la liste triée L
}
```

Paire de points les plus proches

Paire de points les plus proches

Closest pair. Étant donnés n points du plan, trouver une paire de points aussi proches l'un de l'autre que possible (pour la distance euclidienne.)

Primitive géométrique fondamentale.

- Applications graphiques, vision par ordinateur, systèmes d'information géographique, modélisation moléculaire, contrôle du trafic aérien.
- Cas particulier du pb du voisin le plus proche, MST euclidien, diagrammes de Voronoi.

des solutions efficaces pour closest pair ont inspiré des algo. efficaces pour ces pb.

Force brute. Tester toutes les paires (p,q) en $\Theta(n^2)$ comparaisons.

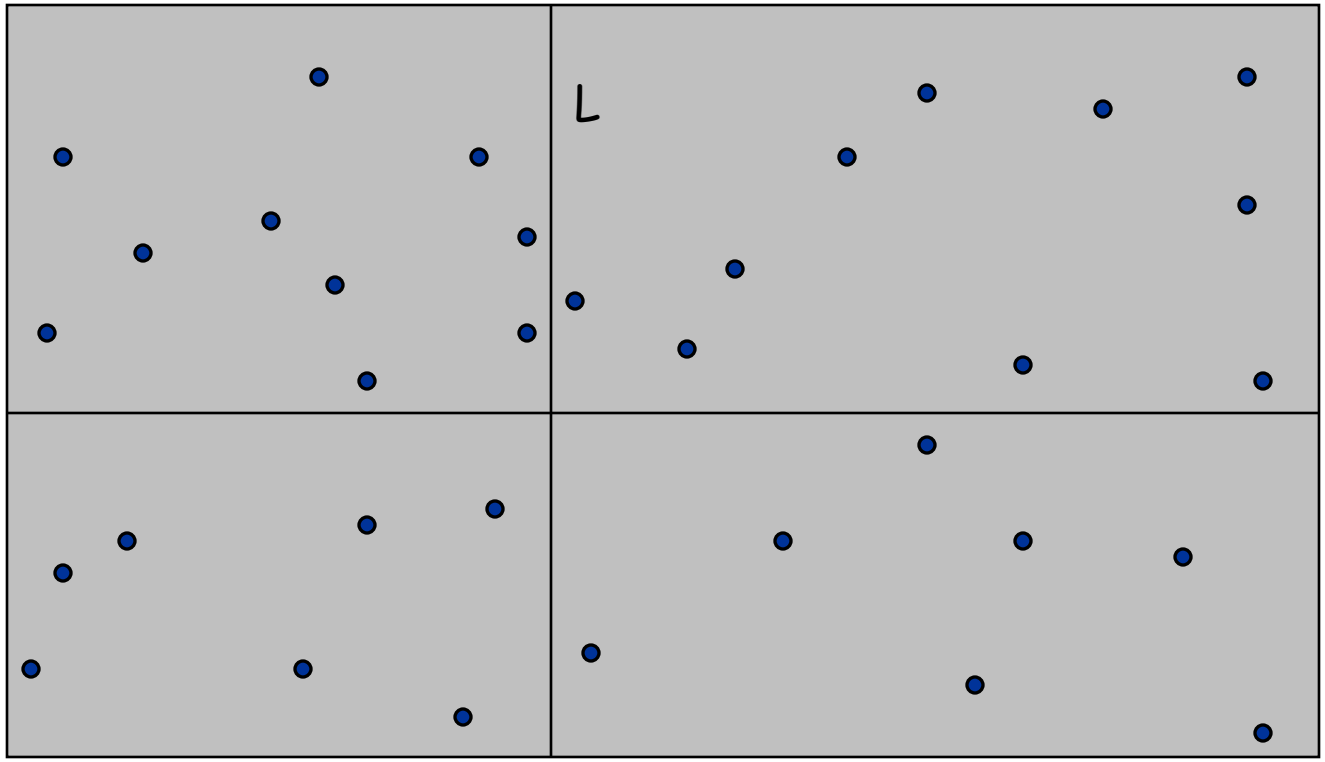
Version 1-D. $O(n \log n)$: c'est facile si tous les points sont alignés.

Hypothèse. Les points ont des abscisses (coor. x) deux à deux distinctes.

pour faciliter la présentation

Paire de points les plus proches : première approche

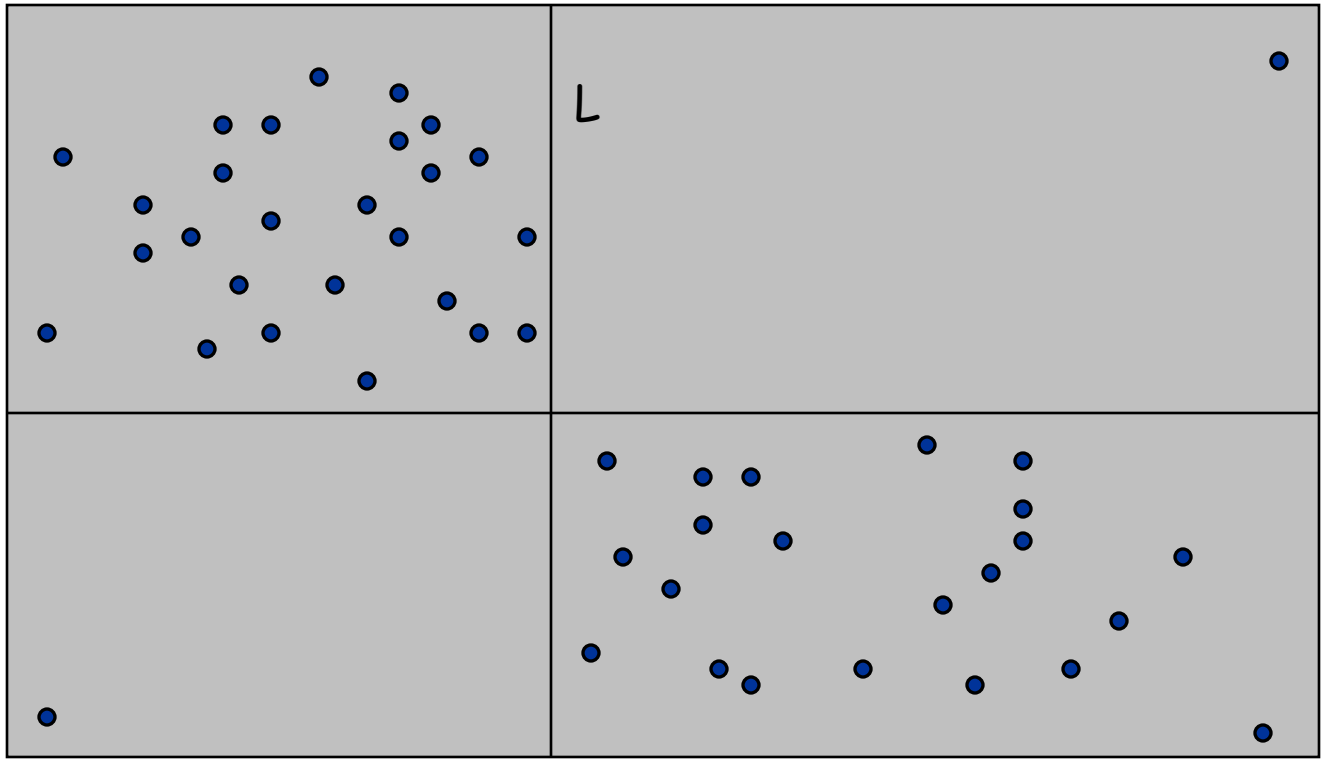
Divide. Diviser la région en 4 quadrants.



Paire de points les plus proches : première approche

Divide. Diviser la région en 4 quadrants.

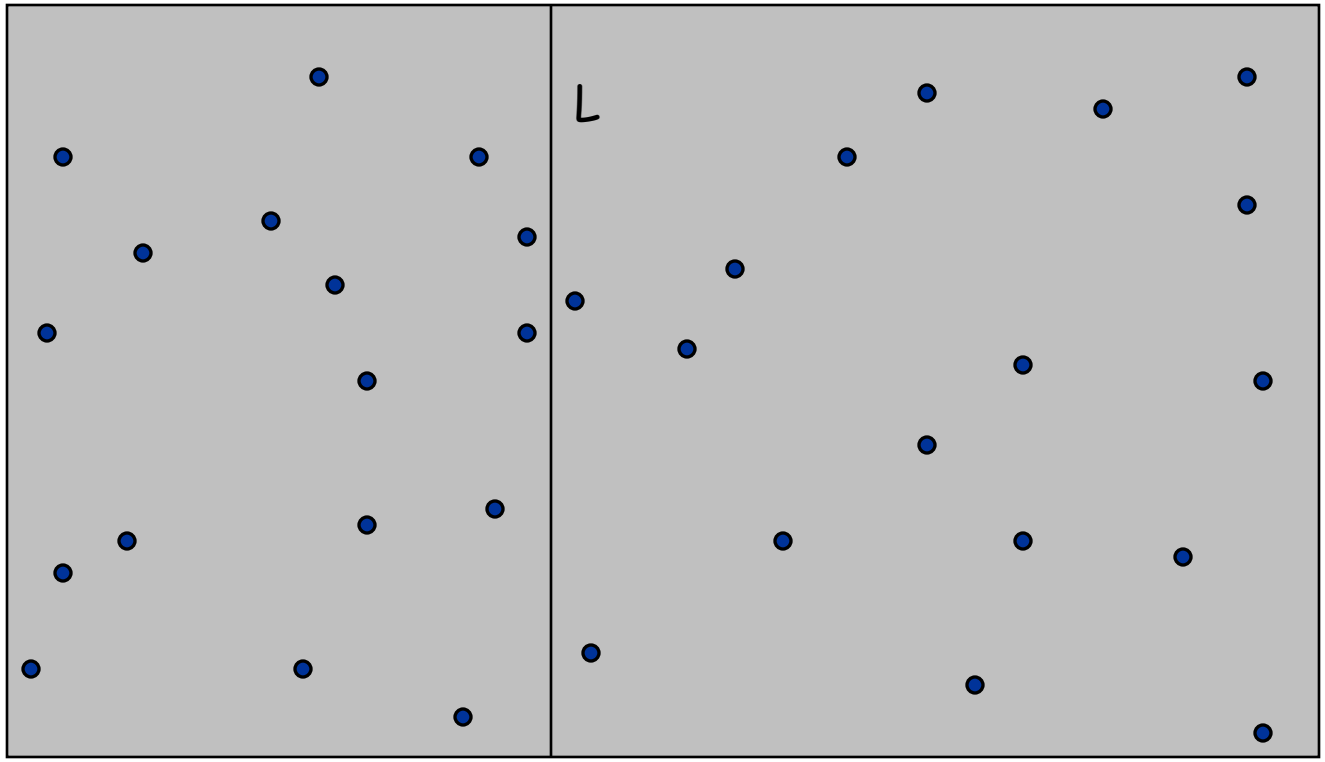
Obstacle. Impossible d'assurer qu'il y a n/4 points dans chaque quadrant.



Paire de points les plus proches

Algorithme.

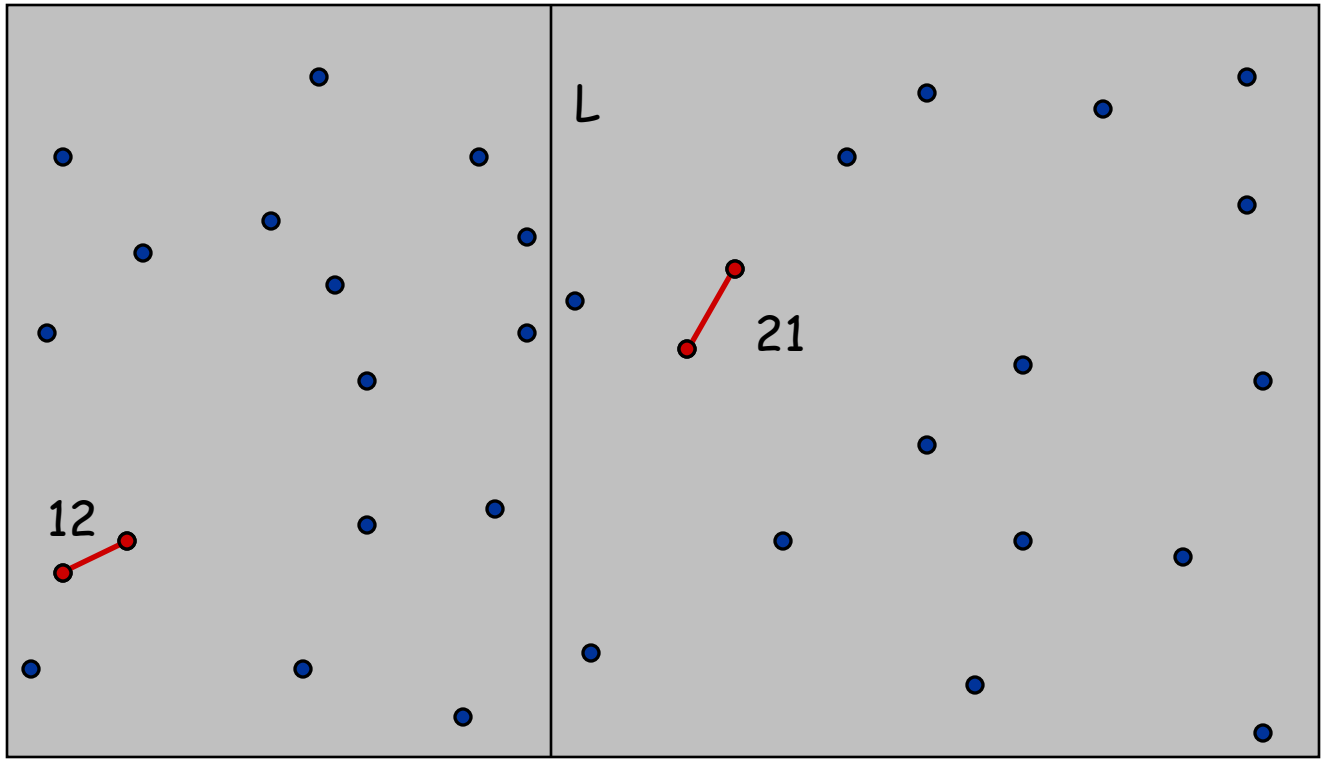
- Divide : tracer une ligne verticale L t.q. approx. $\frac{1}{2}$n points de chaque côté.



Paire de points les plus proches

Algorithme.

- Divide : tracer une ligne verticale L t.q. approx. $\frac{1}{2}$n points de chaque côté.
- Conquer : trouver une closest pair de chaque côté, récursivement.

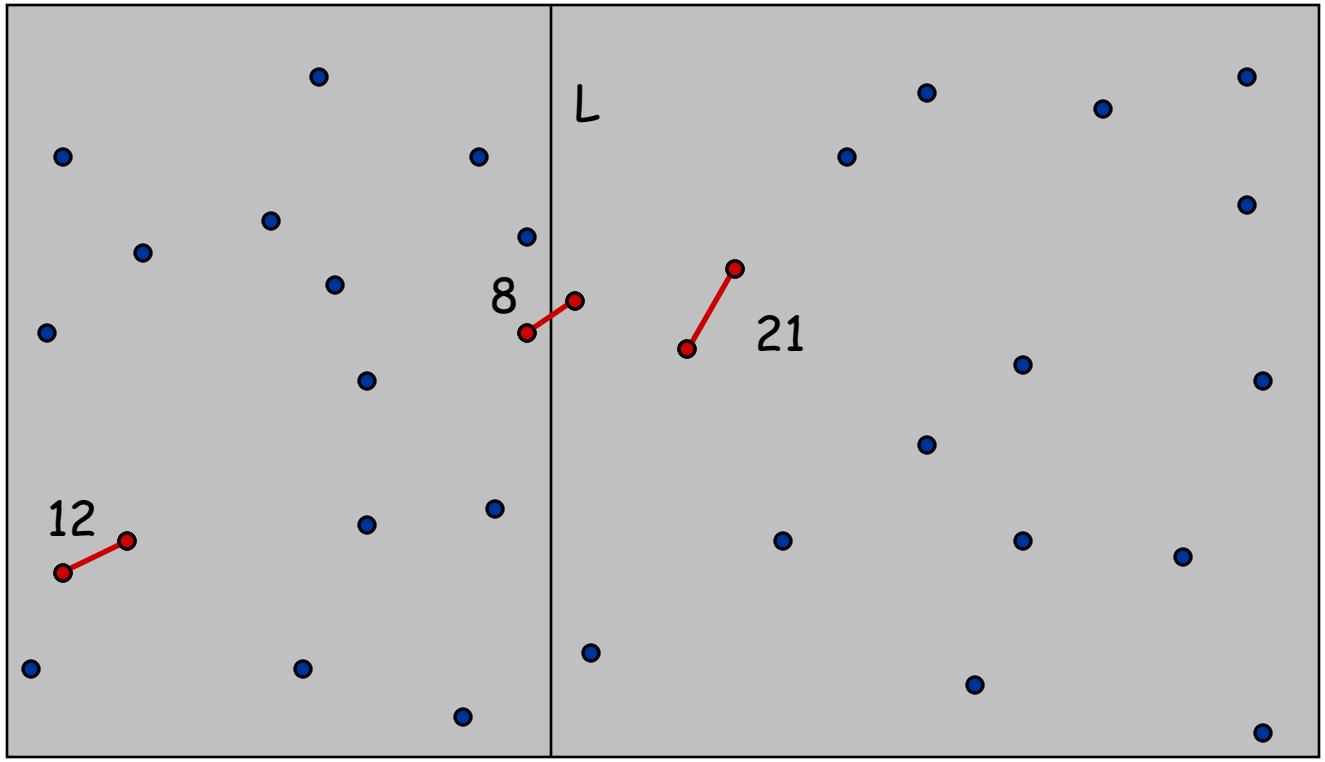


Paire de points les plus proches

Algorithme.

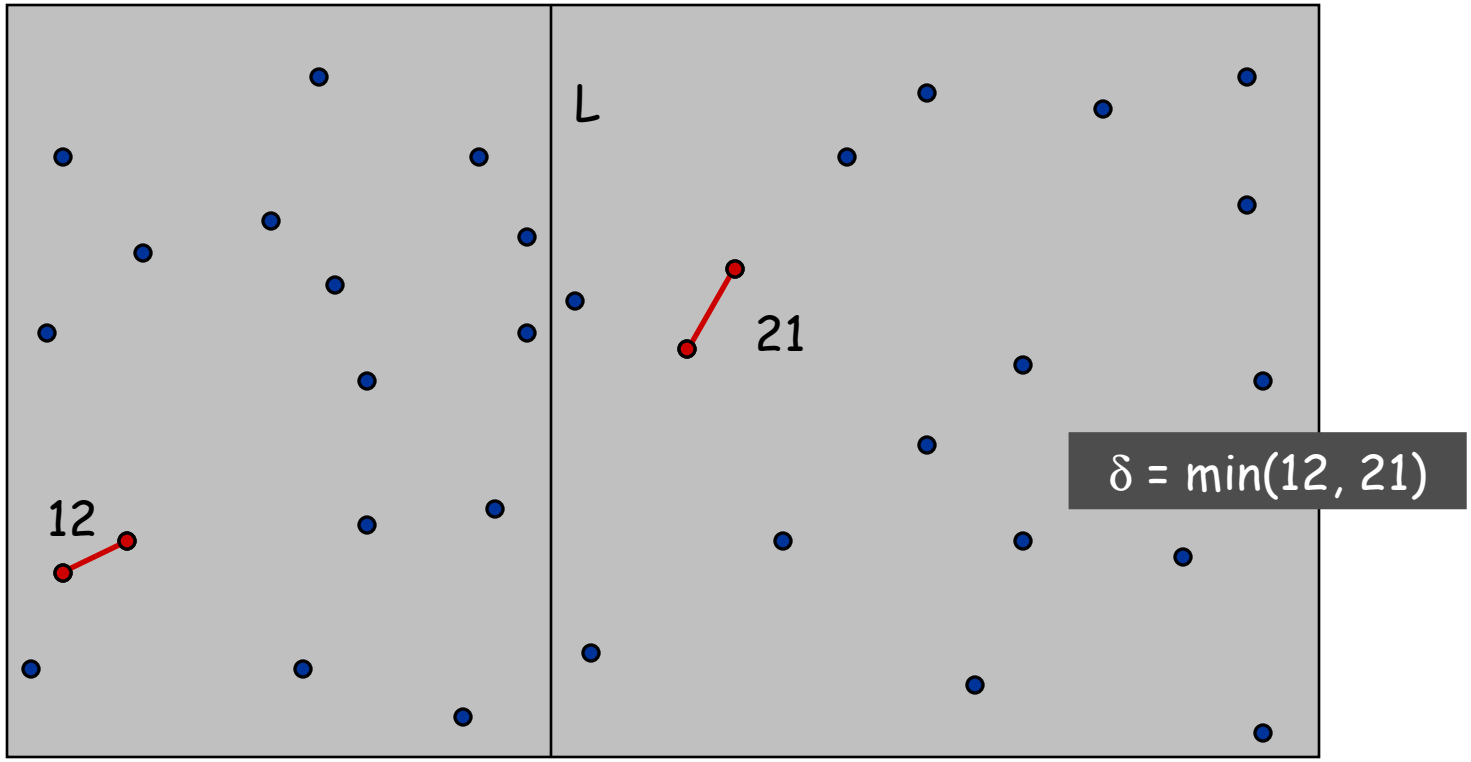
- Divide : tracer une ligne verticale L t.q. approx. $\frac{1}{2}n$ points de chaque côté.
- Conquer : trouver une closest pair de chaque côté, récursivement.
- Combiner : trouver une closest pair avec un point de chaque côté.
- Retourner la meilleure des 3 paires.

semble réclamer $\Theta(n^2)$ comparaisons



Paire de points les plus proches

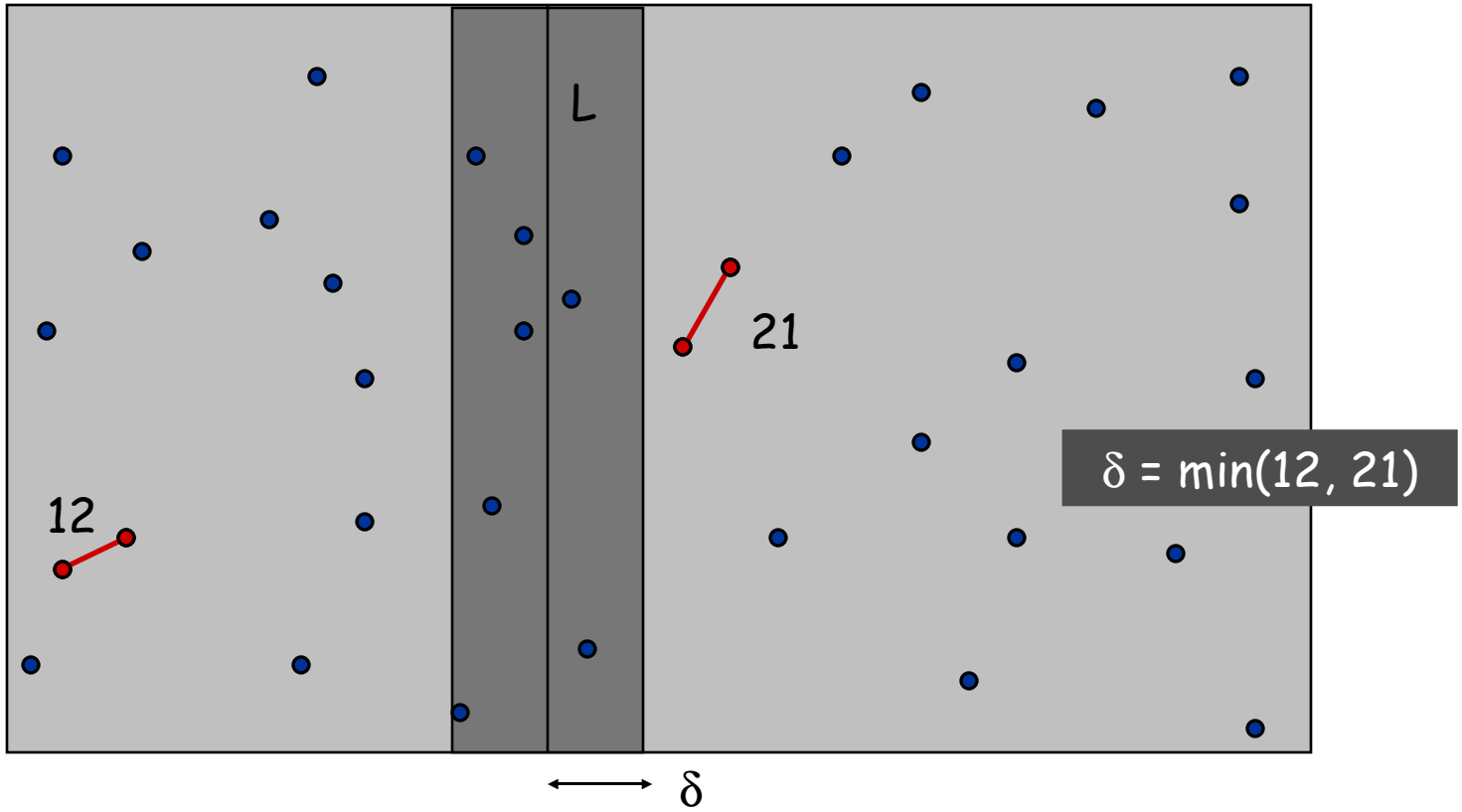
Trouver une closest pair avec un point de chaque côté, en supposant que la distance < δ.



Paire de points les plus proches

Trouver une closest pair avec un point de chaque côté, en supposant que la distance < δ.

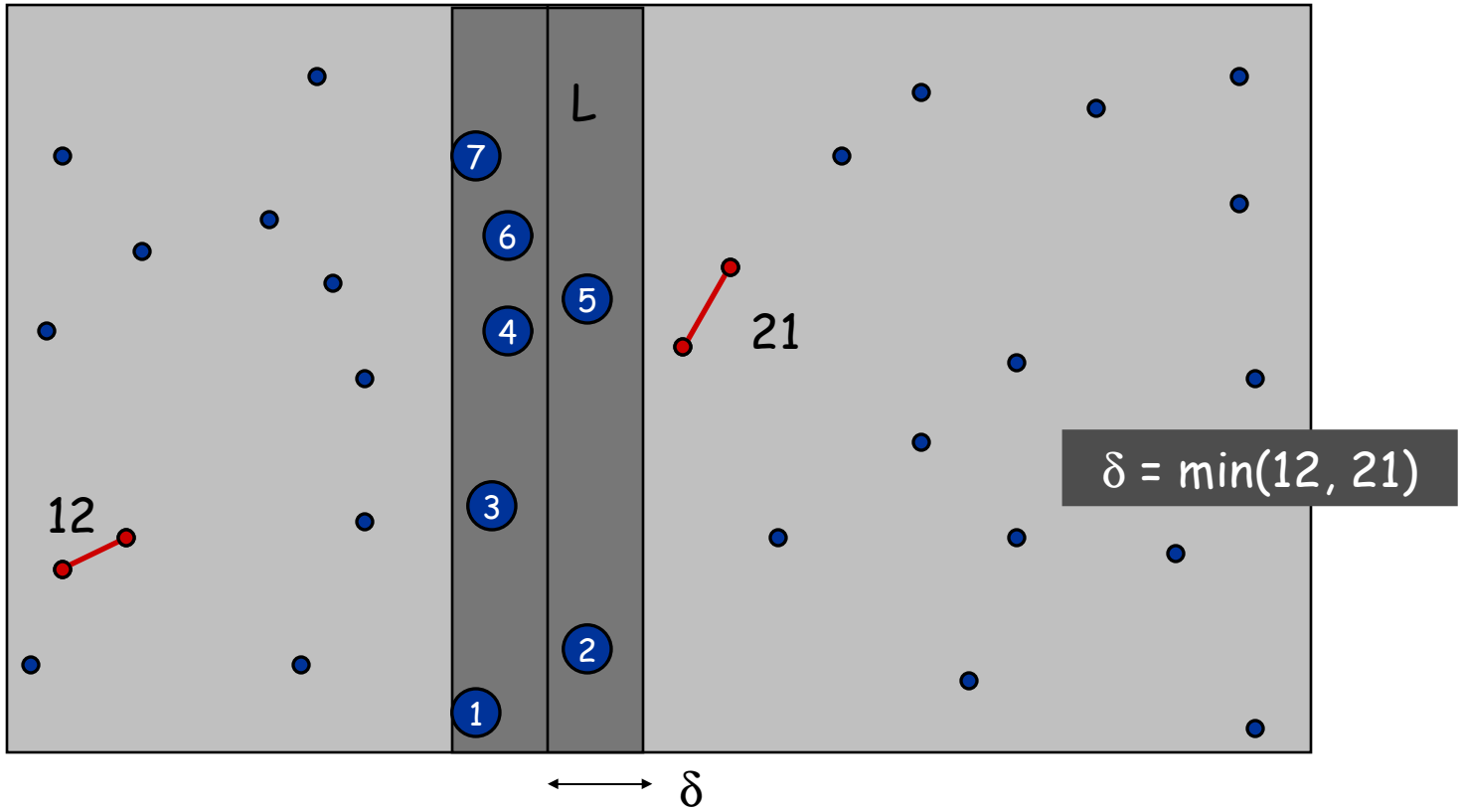
- Remarque : on se limite à une bande de largeur 2δ autour de L.



Paire de points les plus proches

Trouver une closest pair avec un point de chaque côté, en supposant que la distance < δ.

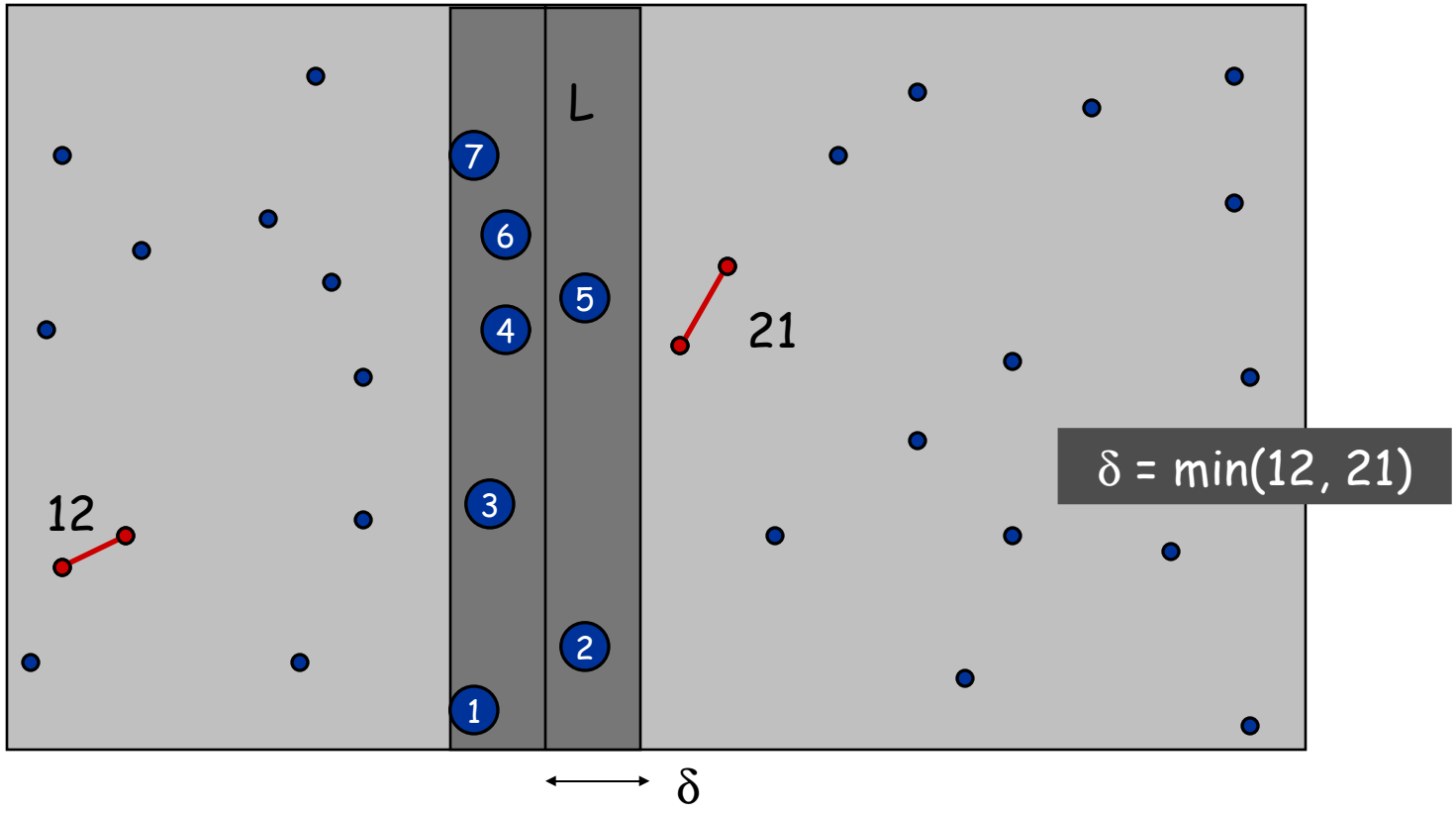
- Remarque : on se limite à une bande de largeur 2δ autour de L.
- On trie les points dans la bande de largeur 2δ- par leur ordonnée.



Paire de points les plus proches

Trouver une closest pair avec un point de chaque côté, en supposant que la distance < δ.

- Remarque : on se limite à une bande de largeur 2δ autour de L.
- On trie les points dans la bande de largeur 2δ- par leur ordonnée.
- Only check distances of those within 11 positions in sorted list!



Paire de points les plus proches

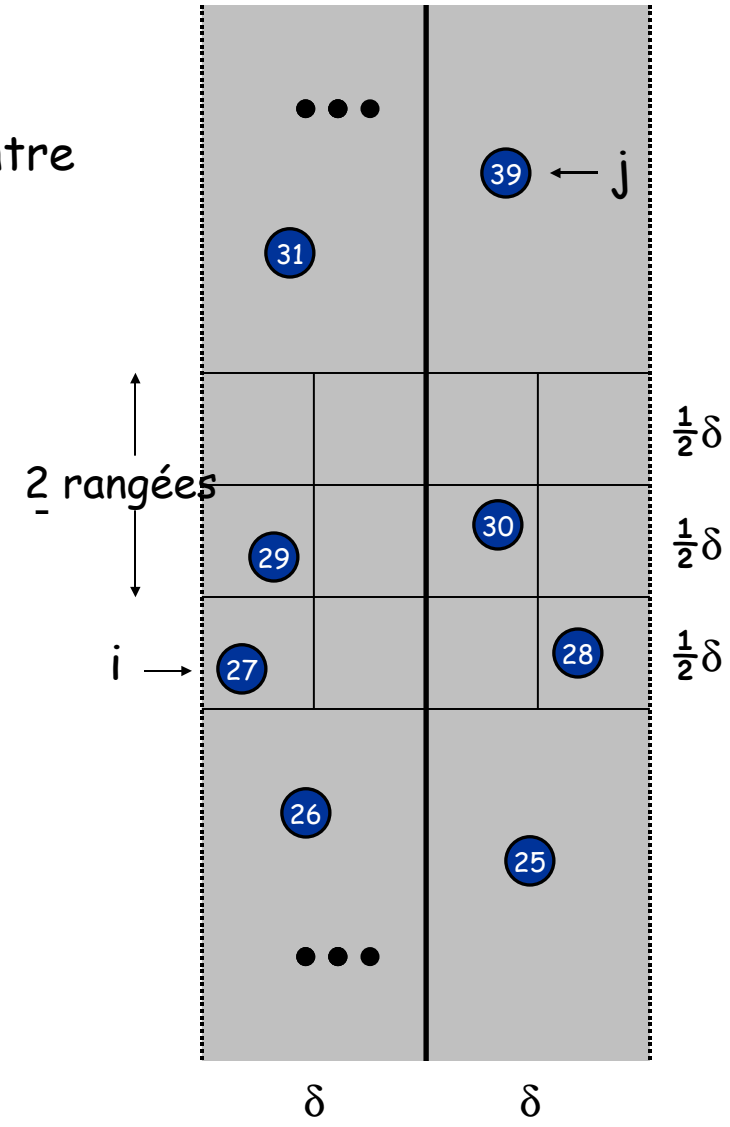
Déf. Soit s_i le point de i^e ordonnée dans la bande de largeur 2δ autour de la ligne L.

Propriété. Si $|i - j| \geq 12$, alors la distance entre s_i et s_j est supérieure ou égale à δ.

Preuve.

- Au plus un point par boîte $\frac{1}{2}\delta \times \frac{1}{2}\delta$.
- Deux points séparés par au moins 2 rangées sont distants d'au moins $2(\frac{1}{2}\delta)$.

Remarque . La propriété reste vraie si on remplace 12 par 7.



Paire de points les plus proches : algorithme

```
Closest-Pair(p1, …, pn) {
   Calculer une ligne de séparation L qui partage les
   points en dux parties égales.

   δ1 = Closest-Pair(partie gauche)
   δ2 = Closest-Pair(partie droite)
   δ  = min(δ1, δ2)

   Supprimer les points éloignés de plus de δ de L

   Trier les points restants selon leur ordonnée.

   Parcourir les points dans l'ordre des ordonnées et
   calculer la distance entre un point quelconque
   et ses 11 successeurs. Si une distance est plus
   petite que δ, mettre à jour δ.

   retourner δ.
}
```

$O(n \log n)$

$2T(n / 2)$

$O(n)$

$O(n \log n)$

$O(n)$

Paire de points les plus proches : analyse

Temps d'exécution.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

Q. Peut-on atteindre $O(n \log n)$?

R. Oui. On ne trie pas les points dans la bande *from scratch* à chaque fois.

- Chaque appel récursif retourne deux listes : tous les points triés slon leur ordonnée, et tous les points triés selon leur abscisse.
- On trie par fusion de deux listes déjà triées.

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$

Multiplication entière

Arithmétique entière

Addition. Étant donnés deux entiers de n chiffres a et b, calculer a + b.

- $O(n)$ opérations binaires (bit operations.)

Multiplication. Étant donnés deux entiers de n chiffres a et b, calculer a × b.

- Solution brutale : $\Theta(n^2)$ opérations binaires.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Addition

							1	1	0	1	0	1	0	1	
						*	0	1	1	1	1	1	0	1	
						1	1	0	1	0	1	0	1	0	
					0	0	0	0	0	0	0	0	0		
				1	1	0	1	0	1	0	1	0			
			1	1	0	1	0	1	0	1	0				
		1	1	0	1	0	1	0	1	0					
	1	1	0	1	0	1	0	1	0						
1	1	0	1	0	1	0	1	0							
0	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0

Multiplication

Multiplication "diviser pour régner" : échauffement

Pour multiplier deux entiers de n chiffres (binaires) :

- On multiplie quatre entiers de $\frac{1}{2}$n chiffres.
- On additionne deux entiers de $\frac{1}{2}$n chiffres, et on décale pour obtenir le résultat.

$$x \quad ¿ \quad 2^{n/2} \cdot x_1 + x_0$$

$$y \quad ¿ \quad 2^{n/2} \cdot y_1 + y_0$$

$$xy \quad ¿ \quad \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0$$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{Q(n)}_{\text{add, shift}} \Rightarrow T(n) = Q(n^2)$$

on suppose que n est une puissance de 2

Multiplication de Karatsuba

Pour multiplier deux entiers de n chiffres (binaires) :

- On additionne deux entiers de $\frac{1}{2}$n chiffres.
- On multiplie trois entiers de $\frac{1}{2}$n chiffres.
- On additionne, soustrait et décale des entiers de $\frac{1}{2}$n chiffres pour obtenir le résultat.

$$x \;¿\; 2^{n/2}\cdot x_1 + x_0$$

$$y \;¿\; 2^{n/2}\cdot y_1 + y_0$$

$$xy \;¿\; \underset{A}{2^n\cdot x_1 y_1} + 2^{n/2}\cdot\left(\underset{B}{x_1 y_0 + x_0 y_1}\right) + \underset{C}{x_0 y_0}$$

$$¿\; 2^n\cdot x_1 y_1 + 2^{n/2}\cdot\left(\underset{A}{(x_1+x_0)(y_1+y_0)} - x_1 y_1 - \underset{C}{x_0 y_0}\right) + x_0 y_0$$

Théorème. [Karatsuba-Ofman, 1962] On peut multiplier deux entiers de n chiffres en $O(n^{1.585})$ opérations binaires.

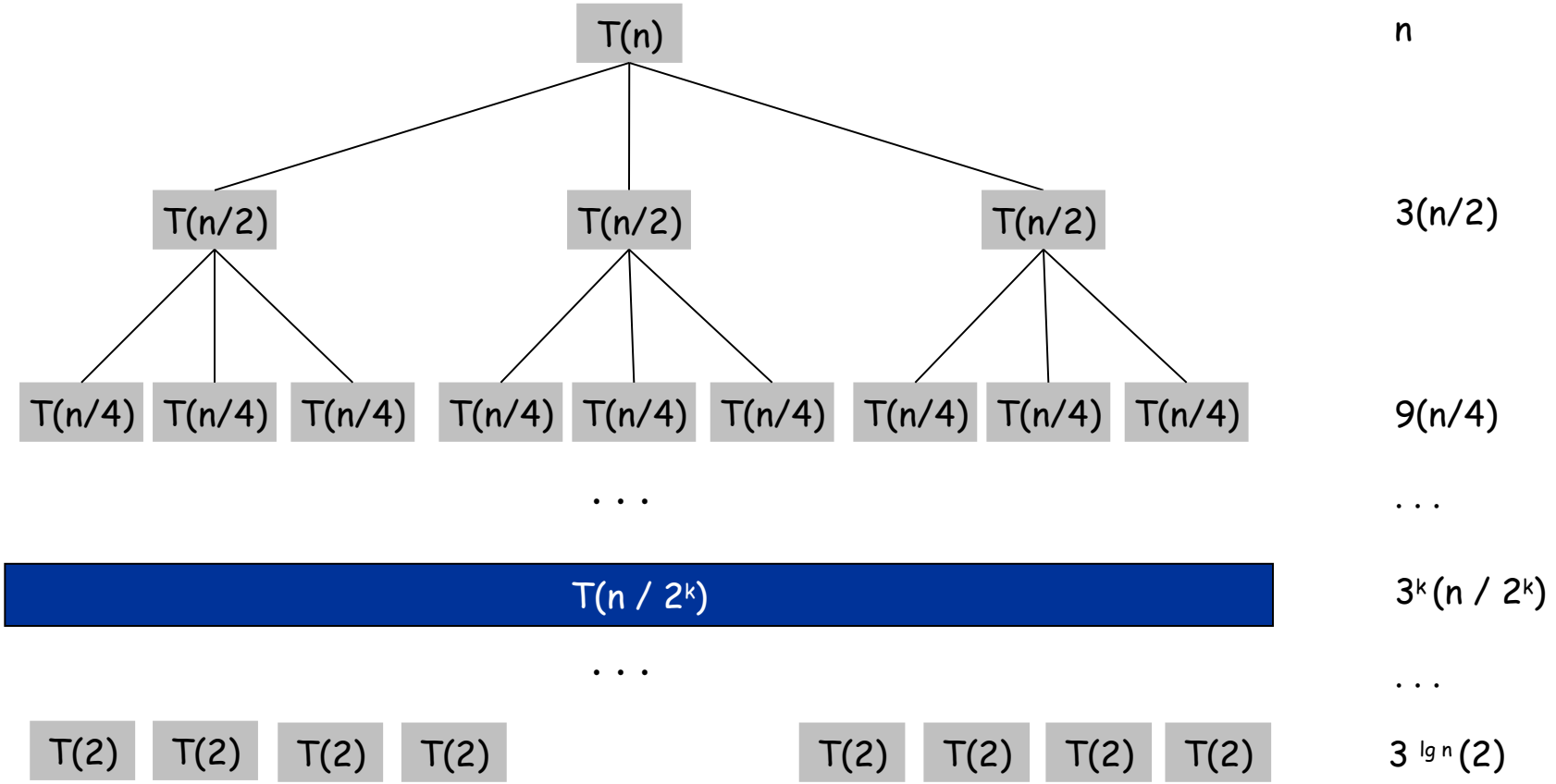
$$T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1+\lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{Q(n)}_{\text{add, subtract, shift}}$$

$$\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})$$

Multiplication de Karatsuba : arbre de récursion

$$T(n)=\begin{cases} 0 & \text{if } n=1 \\ 3T(n/2)+n & \text{otherwise} \end{cases}$$

$$T(n)=\sum_{k=0}^{\log_2 n} n\left(\frac{3}{2}\right)^k = \frac{\left(\frac{3}{2}\right)^{1+\log_2 n}-1}{\frac{3}{2}-1} = 3n^{\log_2 3}-2$$



Multiplication matricielle

Multiplication matricielle

Multiplication matricielle. Étant données deux matrices n x n A et B, calculer $C = AB$.

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

$$\begin{bmatrix} c_{11} & c_{12} & L & c_{1n} \\ c_{21} & c_{22} & L & c_{2n} \\ M & M & O & M \\ c_{n1} & c_{n2} & L & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & L & a_{1n} \\ a_{21} & a_{22} & L & a_{2n} \\ M & M & O & M \\ a_{n1} & a_{n2} & L & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & L & b_{1n} \\ b_{21} & b_{22} & L & b_{2n} \\ M & M & O & M \\ b_{n1} & b_{n2} & L & b_{nn} \end{bmatrix}$$

Force brute. $\Theta(n^3)$ opérations arithmétiques.

Question fondamentale. Peut-on faire mieux que la force brute ?

Multiplication matricielle : échauffement

Diviser pour régner.

- Divide : partitionner A et B en blocs $\frac{1}{2}n \times \frac{1}{2}n$.
- Conquer : faire récursivement 8 multiplications de matrices $\frac{1}{2}n \times \frac{1}{2}n$.
- Combiner : faire les 4 additions des produits appropriés.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \cdot B_{11}) + (A_{12} \cdot B_{21}) \\ C_{12} &= (A_{11} \cdot B_{12}) + (A_{12} \cdot B_{22}) \\ C_{21} &= (A_{21} \cdot B_{11}) + (A_{22} \cdot B_{21}) \\ C_{22} &= (A_{21} \cdot B_{12}) + (A_{22} \cdot B_{22}) \end{aligned}$$

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{Q(n^2)}_{\text{add, form submatrices}} \quad \Rightarrow T(n) = Q(n^3)$$

Multiplication matricielle : idée clé

Idée clé. Multiplier des matrices 2 x 2 avec seulement 7 multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &\leftarrow P_5 + P_4 - P_2 + P_6 \\ C_{12} &\leftarrow P_1 + P_2 \\ C_{21} &\leftarrow P_3 + P_4 \\ C_{22} &\leftarrow P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \cdot (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \cdot B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \cdot B_{11} \\ P_4 &\leftarrow A_{22} \cdot (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \cdot (B_{11} + B_{12}) \end{aligned}$$

- 7 multiplications.
- 18 = 10 + 8 additions (ou soustractions).

Multiplication matricielle rapide

Multiplication matricielle rapide. (Strassen, 1969)

- Diviser : partitionner A et B en blocs $\frac{1}{2}n \times \frac{1}{2}n$.
- Calculer : 14 matrices $\frac{1}{2}n \times \frac{1}{2}n$ via 10 additions matricielles.
- Conquérir : faire récursivement 7 multiplications de matrices $\frac{1}{2}n \times \frac{1}{2}n$.
- Combiner : 7 produits en 4 termes en utilisant 8 additions matricielles.

Analyse.

- On suppose que n est une puissance de 2.
- T(n) = nbre d'opérations arithmétiques.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{Q(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = Q(n^{\log_2 7}) = O(n^{2.81})$$

Multiplication matricielle rapide en pratique

Les problèmes de l'implémentation.

- Matrices creuses.
- Effets de cache.
- Stabilité numérique.
- Dimensions impaires.
- Croisement avec l'algorithme classique autour de $n = 128$.

Erreur courante : "l'algo de Strassen n'est qu'une curiosité théorique."

- L'Advanced Computation Group d'Apple fait état d'une accélération de facteur 8 sur leur G4 Velocity Engine quand $n \sim 2{,}500$.
- L'ensemble des instances pour lesquelles l'algo de Strassen est utile est un sujet de controverse.

Remarque. On peut "Strasseniser" la résolution de $Ax=b$, le calcul du déterminant et des valeurs propres, et d'autres opérations matricielles.

Multiplication matricielle rapide en théorie

Q. Peut-on multiplier deux mat. 2 x 2 avec seulement 7 mult. scalaires ?

R. Oui ! [Strassen, 1969] $Q(n^{\log_2 7}) = O(n^{2.81})$

Q. Peut-on multiplier deux mat. 2 x 2 avec seulement 6 mult. scalaires ?

R. Impossible. [Hopcroft and Kerr, 1971] $Q(n^{\log_2 6}) = O(n^{2.59})$

Q. Peut-on multiplier deux mat. 3 x 3 avec seulement 21 mult. scalaires ?

R. Également impossible. $Q(n^{\log_3 21}) = O(n^{2.77})$

Q. Peut-on mult. deux mat. 70 x 70 avec seulement 143.640 mult. scalaires ?

R. Oui ! [Pan, 1980] $Q(n^{\log_{70} 143640}) = O(n^{2.80})$

La guerre des décimales.

- Décembre 1979 : $O(n^{2.521813})$.
- Janvier 1980 : $O(n^{2.521801})$.

Multiplication matricielle rapide en théorie

Meilleur résultat connu. $O(n^{2.376})$ [Coppersmith-Winograd, 1987.]

Conjecture. $O(n^{2+\varepsilon})$ pour tout $\varepsilon > 0$.

Réserve. Les améliorations théoriques de l'algorithme de Strassen apparaissent de moins en moins praticables.