

2025-2026

- ▶ Quand on parle de la complexité sans préciser, c'est souvent de la **complexité temporelle dans le pire des cas**
- ▶ **Praticable = polynomial**: un algorithme "en 2^n " qui s'exécute en 1 seconde pour une donnée de taille $n = 10$, nécessite plus d'une semaine pour $n = 30$
- ▶ L'analyse asymptotique de la complexité ne révèle pas ce qui se passe sur des "petites" données et n'exempte pas de tester et d'expérimenter
- ▶ **Attention à la taille de la donnée!**

- ▶ G1 : TD en M5 A8 - TP en M5 A16 **vendredi 13h15**
- ▶ G2 : TD en B09 Uranus - TP en M5 A13
- ▶ G3 : TD en B09 Saturne - TP en M5 A14
- ▶ G4 : TD en M5 A3 - TP en M5 A15 **vendredi 13h15**
- ▶ G5 : TD en B09 Jupiter - TP en M5 A16
- ▶ G6 : TD in B09 Mars - TP en M5 A11

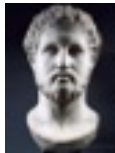
- Partie 1. Quelques schémas ou **paradigmes** d'algorithmes
- Partie 2. Un peu de complexité de problèmes
- Partie 3. Heuristiques et méta-heuristiques

DILBERT COMICS



Ici paradigme = approche pour résoudre un problème

DIVIDE AND CONQUER / DIVIDE UT REGNES / DIVIDE ET IMPERA



Philippe II de Macédoine

- ▶ cas de base: savoir résoudre les "petits" problèmes.
- ▶ Diviser le problème en sous-problèmes de taille "vraiment" plus petite: typiquement la taille est **divisée** par une constante.
- ▶ Résoudre les sous-problèmes par induction
- ▶ Combiner les solutions des sous-problèmes

Simplifier et déléguer

ALGORITHMIC DESIGN PATTERN

- ▶ Diviser pour régner
- ▶ Programmation Dynamique
- ▶ Algorithmes Gloutons
- ▶ Backtracking
- ▶ Branch and Bound
- ▶ ...

DIVISER POUR RÉGNER: LE SCHÉMA

```

DivReg (Probleme P)
if P est "petit" then
  | CasBase(P);
else
  | Diviser P en P1, ..Pk "vraiment" plus petits ;
  | CombinerSolution(DivReg(P1), DivReg(P2), ...DivReg(Pk));
end


```

Algorithm 1: Le schéma diviser pour régner

L'algorithme est en $O(\log n)$

Un autre exemple?

UN AUTRE EXEMPLE: LE TRI FUSION- MERGE SORT

Donnée	D	O	N	N	E	E	A	T	R	I	E	R	
Diviser	D	O	N	N	E	E		A	T	R	I	E	R
Récursion	D	E	E	N	N	O		A	E	I	R	R	T
Fusion	A	D	E	E	E	I	N	N	O	R	R	T	

LE TRI FUSION

Input: L Liste

Output: L triée

Liste $triF(L)$

if $(L.length \leq 1)$ **then**
| **return** L

else

| $mil = L.length/2;$

| **return** fusion($triF(L[0..mil - 1])$, $triF(T[mil..L.length - 1])$)

end

Avec *fusion* qui fusionne deux listes triées en une liste triée.

Preuve :par induction

Arrêt: longueur de la liste à trier décroît strictement à chaque appel

COMPLEXITÉ?

Comptons le nombre de comparaisons d'éléments:

La fusion de deux listes triées de k éléments nécessite **au plus $2k - 1$ comparaisons d'éléments.**

Soit $Comp(n)$ le nombre de comparaisons faites pour trier n éléments. Supposons n pair:

$$Comp(n) \leq 2 * Comp(n/2) + n$$

$$Comp(1) = 0$$

On est dans le cas 2, et $T(n) = O(\log n)$

LE MASTER THEOREM POUR LE MERGE SORT

Master Theorem -version allégée

Si $T(n) = aT(\lfloor n/b \rfloor) + O(n^d)$, avec $a > 0, b > 1, d \geq 0$ alors

- ▶ si $d > \log_b a, T(n) = O(n^d)$
- ▶ si $d = \log_b a, T(n) = O(n^d \log n)$
- ▶ si $d < \log_b a, T(n) = O(n^{\log_b a})$

$$\text{Comp}(n) = 2 * \text{Comp}(n/2) + O(n)$$

Donc $a = b = 2, d = 1$

On est encore dans le cas 2, la complexité est ici $O(n \log n)$.

Un troisième exemple

LA MULTIPLICATION D'ENTRIERS

Donnée: x, y entiers > 0 à n chiffres

Sortie: $x * y$

On compte le nombre d'opérations élémentaires sur les chiffres.

Algorithme de l'école primaire

$$\begin{array}{r} 384132 \\ \times 23556 \\ \hline 2304792 \\ 1920660 \\ 1920660 \\ 1152396 \\ 768264 \\ \hline 9048613392 \end{array}$$

combien d'opérations élémentaires? environ n^2 multiplications de chiffres, et environ $2n^2$ additions de chiffres

DIVISER POUR MULTIPLIER

Supposons $n = 2^k$

Soit x_1 (reps. y_1) formé des $n/2$ premiers chiffres (les plus à gauche) de x (resp. de y), x_2 (resp. y_2) formé des $n/2$ derniers.

Donc: $x = 10^{n/2}x_1 + x_2, y = 10^{n/2}y_1 + y_2$

et: $x * y = (10^{n/2}x_1 + x_2) * (10^{n/2}y_1 + y_2)$

Soit: $x * y = 10^n(x_1 * y_1) + 10^{n/2}(x_1 * y_2 + x_2 * y_1) + x_2 * y_2$

On obtient un algorithme dont la complexité est donnée par:

$$C(n) = 4 * C(n/2) + O(n)$$

Master Theorem: $a = 4, b = 2, d = 1 < 2 = \log_2 4$

On obtient encore un algorithme en $O(n^2)$. ☹

L'ASTUCE DE KARATSUBA

$$x * y = 10^n(x_1 * y_1) + 10^{n/2}(x_1 * y_2 + x_2 * y_1) + x_2 * y_2$$

Il suffit de faire trois multiplications d'entiers à au plus $n/2$ chiffres:

1. $x_1 * y_1$
2. $x_2 * y_2$
3. $(x_1 - x_2) * (y_2 - y_1) = x_1 * y_2 + x_2 * y_1 - x_1 * y_1 - x_2 * y_2$

On obtient un algorithme dont la complexité est donnée par:

$$C(n) = 3C(n/2) + O(n)$$

Par le Master Theorem avec $a = 3, b = 2, d = 1$,

On obtient un algorithme en $O(n^{\log_2 3})$ ($\log_2 3 \approx 1.58$). 😊

C'est l'algorithme de Karatsuba.

UN DERNIER EXEMPLE: LA PAIRE DE POINTS LES PLUS PROCHES

Le problème :

Soit P un ensemble de n points du plan.

On cherche la paire de points les plus proches (pour la distance euclidienne).

Pour simplifier, on supposera que tous les points ont une abscisse différente (pas nécessaire mais plus simple pour la présentation)

LA MULTIPLICATION D'ENTIERS

Donnée:

x, y entiers < 0 à n chiffres

Sortie: $x * y$

On compte le nombre d'opérations élémentaires sur les chiffres.

- ▶ Algo de l'école primaire: $O(n^2)$
- ▶ Multiplication à l'égyptienne plus efficace mais aussi en $O(n^2)$, si on compte les opérations sur les chiffres.
- ▶ Algorithme de Karatsuba en $O(n^{\log_2 3})$
- ▶ Algorithme de Toom-Cook, toujours par "Diviser pour régner", en $O(n^{\log_3 5})$ en divisant en trois parties.
- ▶ Algorithme Schoenage-Strassen (1971), basé sur la FFT en $O(n \log n \log(\log n))$
- ▶ Algorithme de Furer (2007) en $O(n \log n K^{O(\log^* n)})$
- ▶ Algorithmes en $O(n \log n)$ en 2019, pas encore utilisables en pratique
- ▶ On ne sait si on peut faire mieux! (voir article de gazette IM 07/21: <https://www.gdr-im.fr/gazette/>)

EN DIMENSION 1?

Le problème :

Soient n points d'une droite.

On cherche la paire de points les plus proches (pour la distance euclidienne).

Il suffit de trier par une coordonnée et de calculer la distance entre un point et son successeur!

On obtient un algorithme en $O(n \log n)$

EN DIMENSION 2?

Une méthode exhaustive serait en $O(n^2)$.

DIVISER POUR RÉGNER?

Idée: Diviser le plan en deux parties, chacune contenant la moitié des points.

Par exemple, on peut le couper par la droite $x = med$ avec med la valeur médiane des x .

Remarque: pourquoi a-t-on supposé les abscisses des points sont toutes différentes?

DIVISER L'ENSEMBLE DES POINTS EN DEUX

Input: P_x la liste des points de P triée par x croissant
 $cpt = 0$;
for x dans P_x **do**
 if $(cpt \leq n/2)$ **then**
 ajouter x à G
 else
 ajouter x à D ;
 end
 $cpt++$;
end

Remarque: G et D sont triés aussi par x croissant.

DIVISER L'ENSEMBLE DES POINTS EN DEUX

Input: P_x la liste des points de P triée par x croissant
 $cpt = 0$;
for x dans P_x **do**
 if $(cpt \leq n/2)$ **then**
 ajouter x à G
 else
 ajouter x à D ;
 end
 $cpt++$;
end

Coût= celui du tri soit en $O(n \log n)$.

DIVISER POUR RÉGNER?

- ▶ Diviser le plan en deux parties, chacune contenant la moitié des points.
- ▶ On calcule la paire des points plus proches dans chacune des deux parties.
- ▶ Problème? Peut-être que la paire est composée d'un point de chaque partie!
- ▶ Comment faire?

RÉCAPITULATIF

1. Trier les points selon x
2. Partager en deux parties de même cardinal (à 1 près) par une droite verticale $x = med, G, D$.
3. Résoudre à gauche et à droite. cela donne une distance minimale min_G à gauche, min_D à droite.
4. Trouver, si il existe un point de G et un point de D , à distance inférieure à $\delta = \min(min_G, min_D)$; si oui, soit la distance minimale min_{GD} entre un point de G et un point de D
5. La distance minimale est donc le minimum de min_G, min_D , ou min_{GD} selon la réponse du 4.

Coût du point 4?

UN LEMME...

Remarque: si deux points sont à distance $< \delta$ et de part et d'autre de la droite $x = med$, ils sont à distance au plus δ de la droite.

On peut donc limiter la recherche aux points de S l'ensemble des points (x, y) tels que $med - \delta \leq x \leq med + \delta$.

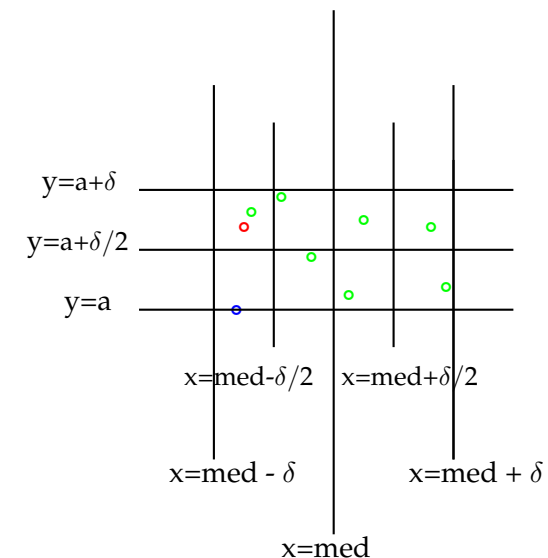
Lemma

Soit $\delta = \min(min_G, min_D)$

Soit S trié par y croissant.

Alors si deux points de chaque côté de la droite sont séparés par au moins 7 points dans la liste, ils sont à distance $> \delta$.

Au plus un point par "carré"



1. Initialisation

- 1.1 trier les points selon x pour construire S_x :
- 1.2 trier les points selon y pour construire S_y
2. Partager en deux parties de même cardinal (à 1 près) par une droite verticale $x = med$: on obtient $G(G_x, G_y), D(D_x, D_y)$.
3. Résoudre à gauche et à droite. Cela donne une distance minimale min_G à gauche, min_D à droite.
4. Trouver, si il existe un point de G et un point de D , à distance inférieure à $\delta = \min(min_G, min_D)$; on notera alors la distance minimale min_{GD} entre un point de G et un point de D
5. La distance minimale est donc le minimum de min_G, min_D , resp. min_{GD} .

La complexité est donnée par l'équation $C(n) = 2C(n/2) + O(n)$

L'algorithme sera en $O(n \log n)$.

- ▶ les algorithmes de recherche dichotomique, recherche de la médiane, ..
- ▶ le tri fusion, le tri rapide
- ▶ les algorithmes de multiplication: entiers (Karatsuba), matrices (Strassen)
- ▶ La transformée de Fourier rapide, FFT
- ▶ Géométrie algorithmique: Enveloppe convexe.

Le problème

Soit P un ensemble de n points du plan.

On a obtenu un algorithme en $O(n \log n)$.

L'algorithme peut être généralisé en dimension > 2 .

- ▶ Diviser un problème en problèmes de taille nettement plus petite i.e. a priori divisée par une constante.
- ▶ Attention à la recombinaison!
- ▶ On peut éventuellement appliquer le Master Theorem pour la complexité
- ▶ Souvent plus facile à écrire de façon récursive
- ▶ Peut être intéressant à paralléliser (surtout si division et recombinaison peu coûteuses).