902 DIVISER POUR RÉGNER et dans les ténèbres les lier.

Certains problèmes algorithmiques peuvent se résoudre efficacement en découpant le problème en plusieurs instances plus petites du même problème. Le paradigme "Diviser pour régner" traite de ce type de problèmes.

I] Description du paradigme. [Cormen], 2.3.1, p.26

Étant donné un problème, on cherche à le résoudre en se ramenant à des instances plus petites du même problème afin d'obtenir une résolution récursive. Il y a donc 3 étapes:

① Diviser: Décomposer une instance de taille n en a instances de taille inférieure* à n. *STRICTE

② Régner: Résoudre le problème sur ces instances

③ Combiner: Résoudre l'instance initiale à partir des solutions obtenues en ②.

Exemple 1: Exponentiation rapide :
Dans un monoïde quelconque, calculer x^n par la méthode naïve se fait $O(n)$ opérations.

* Diviser: $x^n = \begin{cases} (x^{\lfloor \frac{n}{2} \rfloor})^2 \times x & \text{si } n \text{ impair} \\ (x^{\frac{n}{2}})^2 & \text{si } n \text{ pair} \end{cases}$

* Régner: Calculer $x^{\lfloor \frac{n}{2} \rfloor} = y$ avec la même méthode.

* Combiner: Calculer y^2 ou $y^2 \times 1$.

↳ Complexité: $O(\log_2 n)$ produits.

Calculs de complexité: [Papa], 2.2, Master Theorem, p.49.
Si le problème est divisé en sous-problèmes de même taille (ou même seulement de même taille moyenne, cf pb de sélection et tri rapide), alors la complexité temporelle $\mathcal{C}(n)$ satisfait une relation de récurrence $\mathcal{C}(n) \leqslant a\,\mathcal{C}\left(\left\lceil \frac{n}{d} \right\rceil\right) + O(f(n))$ → temps de division et recombinaison
(nb de sous-problèmes ; taille des sous-problèmes)

Le théorème suivant permet alors de calculer $\mathcal{C}(n)$ dans de nombreux cas :

Thm 2: Soit $\mathcal{C}: \mathbb{N} \to \mathbb{R}$ croissante telle que $\mathcal{C}(n) \leqslant a\,\mathcal{C}(\lceil \frac{n}{2} \rceil) + O(n^b)$.
Alors:
* Si $b < \log_2(a)$, $\mathcal{C}(n) = O(n^{\log_2(a)})$. (Régner domine)
* Si $b = \log_2(a)$, $\mathcal{C}(n) = O(n^b \log_2(n))$
* Si $b > \log_2(a)$, $\mathcal{C}(n) = O(n^b)$ (Diviser/Combiner domine)

Exple: Exp. rapide: $\mathcal{C}(n) \leqslant \mathcal{C}(\lceil \frac{n}{2} \rceil) + O(1) \rightsquigarrow \mathcal{C}(n) = O(\log_2 n)$

II] Problèmes de tri et de recherche.

1] Recherche dichotomique dans un tableau trié: [Cormen], exo 2.3.5. ↑34.

Entrée: T tableau de n nombres trié par ordre croissant, et x un nombre.
Sortie: Indice de x dans T, ou 0 si $x \notin T$.

* Diviser: Comparer x à $T[\lfloor \frac{n}{2} \rfloor]$, $X = \begin{cases} T[1..\lfloor \frac{n}{2} \rfloor - 1] & \text{si } x < T[\lfloor \frac{n}{2} \rfloor] \\ T[\lfloor \frac{n}{2} \rfloor + 1..1] & \text{si } x > T[\lfloor \frac{n}{2} \rfloor] \end{cases}$
Retourner $\lfloor \frac{n}{2} \rfloor$ si $x = T[\lfloor \frac{n}{2} \rfloor]$.
* Régner: Recherche de x dans X.
* Combiner: Si x pas trouvé, renvoyer 0.

↳ Complexité: $\mathcal{C}(n) \leqslant \mathcal{C}(\lceil \frac{n}{2} \rceil) + O(1) \rightsquigarrow \mathcal{C}(n) = O(\log_2 n)$

2] Sélection de l'élément de rang k dans un tableau non-trié: [Cormen] 9.2 ↑179

Entrée: T tableau de nombres distincts de taille n, $k \in [\![1, n]\!]$.
Sortie: L'élément de rang k de T (i.e. il y a $k-1$ elts plus petits dans T)

* Diviser: On choisit aléatoirement (uniforme) $i \in [\![1, n]\!]$ et on sépare T en $T_< = [y \in T / y < T[i]]$, $T_= = T[i]$, $T_> = [y \in T / y > T[i]]$.
* Régner: Si $|T_<| = k-1$, renvoyer $T[i]$
$|T_<| < k-1$, chercher l'élt de rang k dans $T_<$
$|T_<| > k-1$, chercher l'élt de rang $(k - |T_<|)$ dans $T_>$.
* Combiner: Rien à faire.

↳ Complexité: - Au pire: $\mathcal{C}(n) = \mathcal{C}(n-1) + O(n) \rightsquigarrow \mathcal{C}(n) = O(n^2)$
- En moyenne: On peut raisonner sur l'espérance de la taille des sous-tableaux dans lesquels se déroule l'algorithme. Ceci donne une complexité en moyenne $\mathcal{C}(n) = O(n)$.

Préciser qu'on utilise un tableau → T' pour renvoyer le résultat

II] Tri fusion : [Cormen], 2.3.1 p. 26

Entrée : T tableau de n nombres
Sortie : T' contenant les éléments de T triés.

* Diviser : $T_- = T[1 .. \lfloor \frac{n}{2} \rfloor]$ et $T_+ = T[\lfloor \frac{n}{2} \rfloor + 1 .. n]$
* Régner : Trier T_- et T_+.
* Combiner : On lit T_- et T_+ simultanément de gauche à droite, et on ajoute à chaque étape dans T' le plus petit des deux éléments lus. (Voir annexe A)

↳ Complexité : $\mathcal{C}(n) \leq 2\mathcal{C}\left(\lceil \frac{n}{2} \rceil\right) + O(n) \rightsquigarrow \mathcal{C}(n) = O(n \log_2 n)$

Rq : Complexité optimale en théorie, assez long en pratique à cause du "Combiner".

4] Tri rapide : [Cormen], 7.3 p. 167.

Même problème, mais on cherche à améliorer le "Combiner" par une méthode probabiliste.

* Diviser : On choisit aléatoirement $i \in [\![1, n]\!]$, et on sépare T en $T_<$, $T_>$ et $T_=$ comme en II 2].
* Régner : Trier $T_<$ et $T_>$
* Combiner : Concaténer $T_<$, $T_=$ et $T_>$.

↳ Complexité : - Au pire : $\mathcal{C}(n) = O(n^2)$ (comme en II 2]).

- En moyenne : Similairement au problème de sélection II 2], on peut montrer que la complexité moyenne est $\mathcal{C}(n) = O(n \log_2 n)$

En pratique, l'algorithme est plus rapide que le tri fusion.

III] Algorithmes de multiplication :

1] Produit de nombres ; algorithme de Karatsuba : [Papa], 2.1 ↑ 45

Entrée : Deux nombres x et y écrits en binaire de taille n.
Sortie : Le produit $x \times y$ (avec sommes en temps négligeable).

* Diviser : On écrit $x = x_G + 2^{\lfloor \frac{n}{2} \rfloor} x_D$, $y = y_G + 2^{\lfloor \frac{n}{2} \rfloor} y_D$, avec x_G, x_D, y_G, y_D de taille $\lfloor \frac{n}{2} \rfloor$ ou $\lceil \frac{n}{2} \rceil$.
* Régner : Calculer $G = x_G y_G$, $D = x_D y_D$, $M = (x_G + x_D)(y_G + y_D)$
* Combiner : Renvoyer $G + 2^{\lfloor \frac{n}{2} \rfloor}(M - G - D) + D = x \times y$.

↳ Complexité : $\mathcal{C}(n) \leq 3\mathcal{C}\left(\lceil \frac{n}{2} \rceil\right) + O(1) \rightsquigarrow \mathcal{C}(n) = O(n^{\log_2 3}) \approx O(n^{1,58})$

Rq : L'idée est de ne calculer que 3 produits dans le "Régner" au lieu de 4 (ce qui donnerait la même complexité que l'algorithme naïf).

2] Produit de matrices ; algorithme de Strassen [Cormen], 28.2 p. 710

On décompose une matrice A en 4 blocs $A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix}$. En multipliant les matrices par blocs, on se ramène au produit de 8 matrices de taille au plus $\lceil \frac{n}{2} \rceil$ si les matrices initiales étaient de taille n. Par une astuce similaire à l'algo. de Karatsuba, on peut se ramener à 7 produits, d'où $\mathcal{C}(n) \leq 7\mathcal{C}\left(\lceil \frac{n}{2} \rceil\right) + O(n^2) \rightsquigarrow \mathcal{C}(n) = O(n^{\log_2 7}) \approx O(n^{2,8})$ [Cormen], 30.2 p. 803.

3] Transformée de Fourier rapide : (En supposant le produit en $\mathcal{C}$ constant) ← adapt

Pour multiplier les polynômes, l'algorithme naïf (par produit de Cauchy) est en $O(n^2)$. On peut améliorer ceci en utilisant la transformée de Fourier rapide (TFR) pour évaluer deux polynômes de degré n en $n+1$ points, faire le produit de ces évaluations (en temps linéaire), puis faire une interpolation par la TFR inverse, qui est la TFD à coefficient près, ce qui donne une complexité en $O(n \log n)$.

Entrée : $P = (a_0, \dots, a_n)$ $n+1$ coefficients, ω une racine prim. n^{e} de 1.
Sortie : $S = (s_0, \dots, s_n)$ la TFR de P, avec $s_i = \sum_{k=0}^{n} a_k (\omega^i)^k$.

* Diviser : $P_0 = (a_0, a_2, \dots, a_{2i}, \dots)$ et $P_1 = (a_1, a_3, \dots a_{2i+1}, \dots)$, afin que $P(X) = P_0(X^2) + X P_1(X^2)$.
* Régner : Calculer S_0 la TFR de P_0 en ω^2 et S_1 celle de P_1 en ω^2.
* Combiner : Renvoyer $S = (s_0, \dots, s_n)$ avec $s_i := S_0[i] + \omega^i S_1[i]$.

↳ Complexité : $\mathcal{C}(n) \leq 2\mathcal{C}\left(\lceil \frac{n}{2} \rceil\right) + O(n) \rightsquigarrow \mathcal{C}(n) = O(n \log_2 n)$

IV] Géométrie algorithmique :

1] Points les plus proches : (DÉV.) [Cormen], 33.4 p. 925

Entrée : Tableau R de points du plan de taille $n \geq 2$.
Sortie : Distance minimale entre deux points de R.

* Prétraitement : Trier R par abscisses croissantes, et par ordonnées croiss.
* Diviser : Partitionner R en R_G et R_D de même taille avec les abscisses de R_G inférieures ou égales à celles de R_D.
* Régner : Calculer d_G et d_D les distances min. dans R_G et R_D.

on suppose que la complexité est ? stds

* Combiner : $d := \min(d_G, d_D)$. On parcourt par ordonnées croissantes les points situés dans une bande verticale de largeur $2d$ centrée sur une droite séparant R_G et R_D : Pour chaque point P, on compare d et la distance de P aux 7 points suivants (c'est suffisant par un argument faisant intervenir le principe des tiroirs), et on renvoie le min de d et des distances trouvées dans ce parcours. (Voir annexe B)

↳ Complexité : Partie récursive : $\mathcal{C}(n) \leq 2\mathcal{C}(\lceil \frac{n}{2} \rceil) + O(n)$.
⤳ $\mathcal{C}(n) = O(n \log_2 n)$
Complexité totale : $\mathcal{C}_{t}(n) = O(n \log_2 n)$

2) Enveloppe convexe rapide (DÉV) [Prepa], 3.3.4 p 112

Entrée : Tableau R de points du plan, de taille $n \geq 2$.
Sortie : Liste C de points de R définissant le polygone conv(R).

* Diviser : Soient $g, d \in R$ tels que $\forall p \in R$, $\widehat{dgp} \in [0, \pi]$. On fixe h tel que l'aire du triangle hgp soit maximale, et tel que $\widehat{hgp}$ soit maximal parmi les h vérifiant cette propriété.
* Régner : Enveloppes convexes C_g de $\{x \in R / \widehat{hgx} \in [0, \pi]\}$
C_d de $\{x \in R / \widehat{dhx} \in [0, \pi]\}$
* Combiner : Concaténer C_g et $C_d \setminus \{h\}$.

↳ Complexité : - Au pire : $O(n^2)$
- Cas "non dégénérés" : $O(n \log n)$

(Voir annexe C)

V) Aspects théoriques :

1) Théorème de Savitch (DÉV) [Carton] Partie 4.3.1, Thm 4.28, p. 203

Déf : Si M est une machine de Turing, on définit, pour $n \in \mathbb{N}$, $t(n)$ et $s(n)$ le temps et l'espace maximal utilisé lors d'une exécution de M sur les entrées de taille n.

Lemme : Il existe une constante K_M telle que : $t(n) \leq 2^{K s(n)}$. De plus, $s(n) \leq \max(t(n), n)$.

admis par le jury → défaut de précision

Déf : SPACE(a) (resp NSPACE(a)) pour $a \in \mathbb{N}$, est l'ensemble des problèmes de décision résolubles par une machine déterministe (resp. non-déterministe) en complexité spatiale a.

On peut alors montrer, par une méthode "Diviser pour Régner", le théorème de Savitch :

Thm : Soit $a \in \mathbb{N}$. Alors NSPACE$(a) \subseteq$ SPACE(a)

Corollaire : PSPACE = NPSPACE

2) Complexité du produit et de l'inversion matricielle. [Cormen] 28.4 b) p 731

L'algorithme de Strassen pour multiplier des matrices n'est pas optimal, mais la complexité optimale n'est pas connue. On peut montrer par une méthode "Diviser pour Régner" que l'inversion matricielle a une complexité asymptotique temporelle équivalente à celle de la multiplication, ce qui implique que l'optimum évoqué peut être étudié via le produit comme via l'inversion :

Thm : Si on sait multiplier les matrices $n \times n$ en temps $M(n)$ tel que : ① $M(n) = \Omega(n^2)$.
② $\forall k \in [\![0, n]\!]$, $M(n+k) = O(M(n))$.
③ $\exists c < \frac{1}{2}$ tel que $M(\frac{n}{2}) \leq c\, M(n)$.
Alors on peut inverser une matrice inversible en temps $O(M(n))$. [Cormen], Thm 28.8 p 732

On peut démontrer une réciproque (avec des conditions similaires), mais sans utiliser de "Diviser pour Régner".

[Carton] Langages formels, Calculabilité et Complexité, Olivier Carton (Version beige !) (Vuibert)

[Cormen] Introduction à l'algorithmique, Cormen-Leiserson-Rivest-Stein, 2ème éd (Dunod)

[Papa] Algorithms, Dasgupta-Papadimitriou-Vazirani, (McGraw Hill)

[Prepa] Computational Geometry, Preparata-Shamos (Springer-Verlag)

On utilise ... la complexité ...

* Question : Trouver un contre-exemple quand C n'est pas croissante.
  - Soucis où on utilise l'hypothèse de croissance.

Annexe ⇝ Fusion = dessin inutile

Enveloppe convexe = C'est bien.

Question - Est-ce que le tri fusion est en place ?

↳ Comme il est présenté : NON

Mais il existe 1 version en place ⇝ COMPLIQUÉ

- Quand on calcule la complexité, on utilise sur le Thm (de complexité)
  MAIS de autres cas (Π 2 peu en / complexité en pire cas), NEW (réel)

⚠ Il y a plusieurs types de complexité

Dispo : - FFT
- Comparaison fin de tri fusion / tri rapide
- Programme de Vanna ?

↳ Tri réseau [Cormen]
↳ édité 2 (⚠ Pas de la 3e)

[Preparata, Shamos] Computational Geometry, an introduction.
3.3.4. QUICKHULL Techniques (p. 112).

Calcul de l'enveloppe convexe dans le plan

Algorithme diviser pour régner avec la partie astucieuse dans le "Diviser" (il en existe un autre, dont la complexité au pire est $O(n \log n)$, où l'astuce est dans le "combiner", cf [Preparata Shamos], 3.3.5). Il y a (beaucoup) d'autres algos en dehors du paradigme DpR (cf [Preparata Shamos], 3.3, ou bien [Cormen] 33.3).

Le problème :

Entrée : Tableau R de points distincts du plan de taille $n \geq 2$.

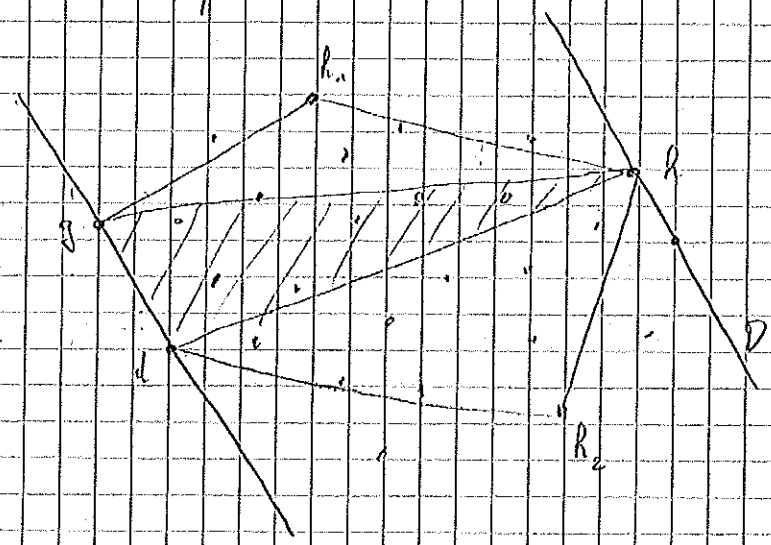
Sortie : Liste C de points de R définissant le polygone convexe conv(R) (env. convexe de R).

Déf : Si d, g, p sont des points, on dit que dgp est direct si l'angle $\widehat{dgp}$ appartient à $[0, \pi]$.
(Intuition : Si g et d sont sur l'axe des abscisses, avec g à gauche de d, l'ensemble des points p tels que dgp est direct est le demi-plan supérieur).

*Diviser : Si g, d sont deux points de R tels que $\forall p \in R$, dgp est direct, alors on choisit $h \in R$ tel que le triangle dgh soit d'aire maximale, et parmi ces points, celui tel que $\widehat{hgd}$ soit maximal

Lemme : Le point h est au bord de $conv(R)$.

Dém : La droite D parallèle à (gd) passant par h est telle que $\forall p \in R$, si dgp est direct, alors $p \in D$ (sinon, l'aire hgd ne serait pas max). De plus, tous les points de $R \cap D$ sont du même côté de h sur D (sinon, hgd ne serait pas max). Par conséquent, $h \notin conv(R \setminus \{h\})$, et h est un point extrémal de R.



Explication de la figure

Dans la figure ci dessus, g et d sont choisis à l'initialisation de l'algo, R (et la droite D du lemme) est choisi comme précédemment. Le triangle hd_1R_2 représente les points que l'on peut "oublier" pour la suite, et les points h_1 et h_2 sont ceux choisis au deuxième appel récursif, h_1 pour la partie du dessin au dessus de (gR), h_2 pour celle en dessous de (hd).

↳ Utiliser les caractérisations des points sur l'enveloppe convexe (admises / un max possible)

* <u>Régner</u> : Soient $R_g = \{p \in R \,/\, hgp \text{ est direct}\}$

$R_d = \{p \in R \,/\, dhp \text{ est direct}\}$

et C_g et C_d leurs enveloppes convexes (i.e. liste de points extrémaux).

* <u>Combiner</u> : C est la concaténation de C_g et $C_d \setminus \{h\}$. (on enlève h pour ne pas le compter deux fois ; comme C_d est une liste, on peut supprimer h en temps constant puisqu'il est en tête de liste).

Mieux expliquer ← Pq les pts des appels sont sur l'enveloppe convexe finale ?

* <u>Correction de l'algo</u> :

Si on initialise g et d comme points extrémaux alors le lemme montre que chaque point h construit sera dans conv(R), et la correction en découle.

* <u>Pseudocode</u> :

<u>EnvRapide</u> (R)

- g = pt d'abcisse min
- d = pt d'angle polaire min rel. à la demi-droite : $\overset{g}{\downarrow}$

} g et d ainsi choisis sont extrémaux. (Temps linéaire)

<u>Retourner</u> EnvRéc (R, g, d)

↑

cf page suivante

```
EnvRéc (R, g, d)
    Si |R| = 2
        Renvoyer (g; d)
    Sinon
        h = g
        Pour x ∈ R
            Si (aire(xgd) ≥ aire(Rgd))
                Si (x^gd > h^gd)
                    h = x
        R_g = [x ∈ R / hgx direct]
        R_d = [x ∈ R / dhx direct]
        Renvoyer Concat(EnvRéc(R_g, g, h), EnvRéc(R_d, h, d))
```

Le calcul des aires et des angles se fait en tps constant. Donc cette boucle se fait en temps $O(n)$

temps $O(n)$ (parcours du tableau R)

* Complexité :

1. Au pire : À chaque étape, R_g ou R_d est vide, et aucun point n'est "oublié" dans le triangle hachuré :

$$\mathcal{C}(n) = \mathcal{C}(n-1) + O(n) \rightsquigarrow \mathcal{C}(n) = O(n^2)$$

Cas non-dégénérés : Si à chaque étape, $\frac{|R_g|}{|R|}$ et $\frac{|R_d|}{|R|}$ sont tous deux $\leq \frac{1}{2}$, alors :

$$\mathcal{C}(n) \leq 2\,\mathcal{C}\left(\left\lceil \tfrac{n}{2} \right\rceil\right) + O(n) \rightsquigarrow \mathcal{C}(n) = O(n \log n)$$

(Phénomène comparable au tri rapide, au pire et en moyenne)

Rq : Calcul de l'aire d'un triangle avec un déterminant.

(contrôle)

[Cormen], 33.4 p 925

Recherche des points les plus proches

L'algo ci-dessous ne recherche en fait pas les points les plus proches, mais la distance entre eux ; il est facile de modifier l'algo pour récupérer, avec chaque distance, les deux points concernés.

Le problème :

Entrée : R un tableau de points du plan de taille $n \geq 2$.
Sortie : d_{min} la distance min entre deux points de R.

Remarque : Il y a un algorithme naïf : calculer la distance entre les $\binom{n}{2}$ paires de points possibles, et $\binom{n}{2} = O(n^2)$.

Pour l'algo "Diviser pour Régner", on va avoir besoin d'un prétraitement (i.e. avant d'entrer dans la partie récursive) : le tri de R.

$O(n \log_2 n)$ avec le tri fusion.

* Prétraitement : Soit $\preceq_a$ l'ordre lexicographique ordonnant les abcisses, puis les ordonnées, et $\preceq_o$ celui ordonnant les ordonnées puis les abcisses. On crée deux tableaux X et Y contenant R trié selon $\preceq_a$ et $\preceq_o$ respectivement.

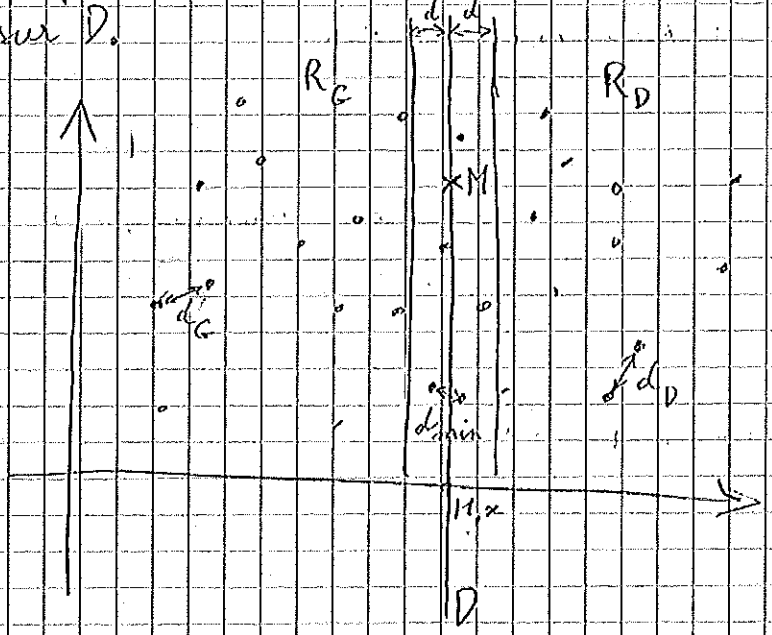
* Diviser : On trouve un point médian $M = (M.x, M.y)$ relativement à $\preceq_a$, et on pose :

$X_G = X[1..\lfloor \frac{n}{2} \rfloor]$, $Y_G = [p \in Y / p \preceq_a M]$

$X_D = X[\lfloor \frac{n}{2} \rfloor + 1 .. n]$, $Y_D = [p \in Y / M \prec_a p]$

R_G et R_D

Ceci sépare R en deux parties par une droite verticale D passant par M, telle que les points de R_G sont à gauche de ou sur D, et ceux de R_D sont à droite de ou sur D.

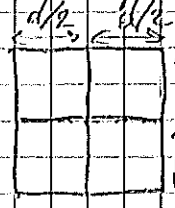


* Régner: Trouver les distances minimales d_G et d_D dans X_G et X_D (déjà triés, donc pas de prétraitement dans les appels récursifs).

* Combiner: Reste à déterminer si d_{min} est réalisée par une paire de points (P, Q) avec $P \in R_G$ et $Q \in R_D$. Si $d = \min(d_G, d_D)$, ceci ne peut se produire que dans la bande verticale de largeur $2d$ centrée en D. De plus, il n'est pas utile de parcourir toute la bande pour chaque point dans la bande:

Lemme: Un carré de côté d ne peut contenir plus de 4 points de R_G (resp: de R_D):

Dém: On découpe le carré comme suit:



Par le principe des tiroirs, si 6 points de R_σ sont dans le carré, alors il y en a 2 dans le même petit carré, et ils sont à distance $0 < \frac{d}{\sqrt{2}} < d \leq d_\sigma$, ce qui est absurde puisque $d_\sigma \leq D$.

On pose donc $Y' = [P \in Y \,/\, M.x - d \leq P.x \leq M.x + d]$, et on parcourt Y' (trié par ordonnées croissantes) en ne vérifiant que les distances de chaque point avec 7 points suivants. Le lemme montre que ceci fournit un algorithme correct.

* Pseudo-code :

```
PlusProcheRes(R)
    X = Tri_{≤_a}(R)
    Y = Tri_{≤_o}(R)                    } O(n log_2(n))
    d = PlusProcheResRéc(X, Y)
    Renvoyer d

PlusProcheResRéc(X, Y)
    Si |X| ≤ 3
        Algo naïf.
    Sinon
        M = X[⌊n/2⌋]
        X_G = X[1..⌊n/2⌋] ; Y_G = [P ∈ Y / P ≲_a M]
        X_D = X[⌊n/2⌋+1..n] ; Y_D = [P ∈ Y / M <_a P]      } O(n)
        d_G = PlusProcheResRéc(X_G, Y_G)
        d_D = PlusProcheResRéc(X_D, Y_D)
```

$d = \min(d_G, d_D)$

$Y' = [P \in Y \mid M_0.x - d \leq P_0.x \leq M_1.x + d]$ $\rightsquigarrow O(n)$

Pour i de 1 à $|Y'|$ } $O(n)$

  Pour j de $i+1$ à $\min(i+8, |Y'|)$ }

    $d = \min(d, \text{dist}(Y'[i], Y'[j]))$

Renvoyer d

* Complexité :

- Algo récursif : $\mathcal{C}_{réc}(n) \leq 2\,\mathcal{C}_{réc}\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)$

$\rightsquigarrow \mathcal{C}_{réc}(n) = O(n \log_2^2 n)$

- Algo total : $\mathcal{C}_{tot}(n) = \mathcal{C}_{réc}(n) + \underbrace{O(n \log_2 n)}_{\text{temps de tri}}$

$\rightsquigarrow \boxed{\mathcal{C}_{tot}(n) = O(n \log_2 n)}$

(Voir les exos qui suivent dans [Cormen] pour quelques trucs d'optimisation de l'algo).