Algorithmique — 2. Techniques algorithmiques

5. Diviser pour régner

Bruno Grenet

https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html

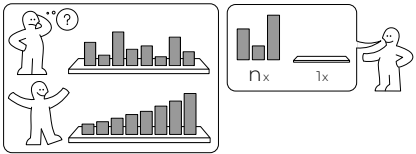
Université Grenoble Alpes – IM²AG
L3 Mathématiques et Informatique

Table des matières

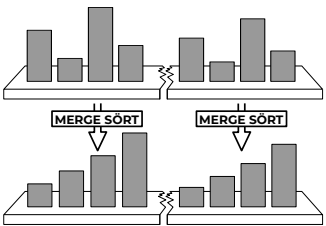
Table des matières

MERGE SÖRT

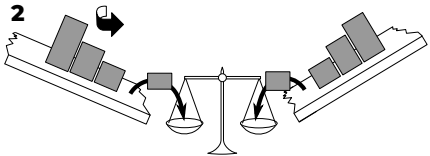
idea-instructions.com/merge-sort/
v1.1, CC by-nc-sa 4.0



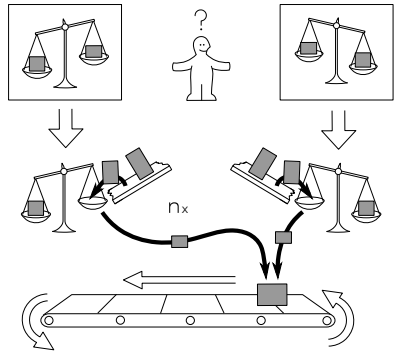
1



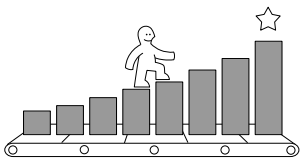
2



3



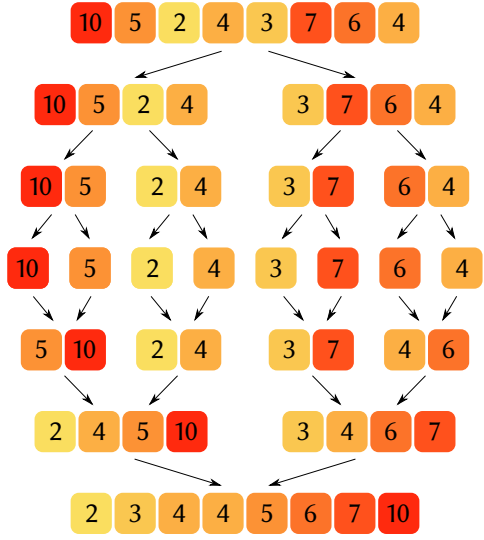
4



Algorithme du TriFusion

TriFusion(T):

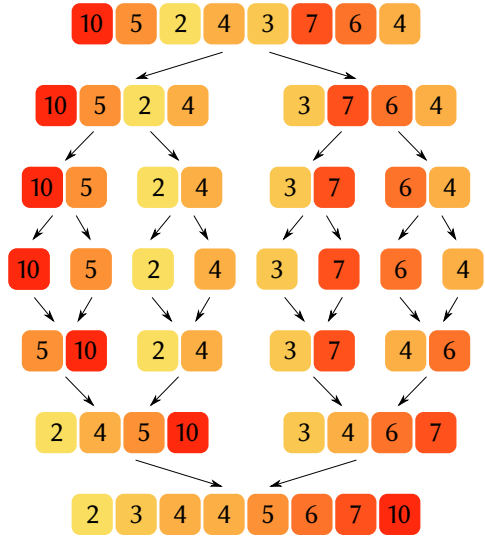
1. $n \leftarrow \#T$
2. Si $n \leq 1$: Renvoyer T
3. Sinon :
4. $\quad T_1 \leftarrow$TriFusion($T_{[0,\lfloor n/2 \rfloor[}$)
5. $\quad T_2 \leftarrow$TriFusion($T_{[\lfloor n/2 \rfloor,n[}$)
6. $\quad$ Renvoyer Fusion(T_1, T_2)



Algorithme du TriFusion

TriFusion(T):

1. $n \leftarrow \#T$
2. Si $n \leq 1$: Renvoyer T
3. Sinon :
4. $\quad T_1 \leftarrow$TriFusion($T_{[0,\lfloor n/2 \rfloor[}$)
5. $\quad T_2 \leftarrow$TriFusion($T_{[\lfloor n/2 \rfloor,n[}$)
6. $\quad$ Renvoyer Fusion(T_1, T_2)



Lemme

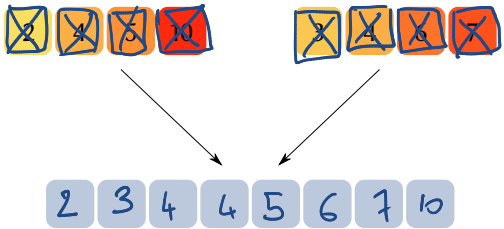
Soit $t(n)$ la complexité de TriFusion et $f(n)$ la complexité de Fusion. Alors

$$t(n) = t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + f(n) + O(1)$$

pour $n > 1$, et $t(n) = 0$ sinon

(complexité = nombre de comparaisons)

Algorithme de Fusion



Idée de l'algorithme

- T_1 et T_2 vus comme des **piles**
- S vu comme une **file**
- À chaque itération,
  - on dépile la plus petite des deux têtes
    - si une pile est vide, on dépile l'autre
  - on enfile dans S

Algorithme de Fusion

Fusion(T_1, T_2):

1. $n_1 \leftarrow \#T_1$; $n_2 \leftarrow \#T_2$
2. $S \leftarrow$ tableau de taille $n = n_1 + n_2$
3. $i_1 \leftarrow 0$; $i_2 \leftarrow 0$
4. Pour $i_S = 0$ à $n - 1$:
5. Si $i_1 \geq n_1$: *(T_1 vide)*
6. $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
7. Sinon si $i_2 \geq n_2$: *(T_2 vide)*
8. $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
9. Sinon si $T_{1[i_1]} \leq T_{2[i_2]}$:
10. $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
11. Sinon :
12. $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
13. Renvoyer S

Idée de l'algorithme

- T_1 et T_2 vus comme des **piles**
- S vu comme une **file**
- À chaque itération,
  - on dépile la plus petite des deux têtes
    - si une pile est vide, on dépile l'autre
  - on enfile dans S

Algorithme de Fusion

Fusion(T_1, T_2):

1. $n_1 \leftarrow \#T_1$; $n_2 \leftarrow \#T_2$
2. $S \leftarrow$ tableau de taille $n = n_1 + n_2$
3. $i_1 \leftarrow 0$; $i_2 \leftarrow 0$
4. Pour $i_S = 0$ à $n - 1$:
5. Si $i_1 \geq n_1$: *(T_1 vide)*
6. $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
7. Sinon si $i_2 \geq n_2$: *(T_2 vide)*
8. $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
9. Sinon si $T_{1[i_1]} \leq T_{2[i_2]}$:
10. $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
11. Sinon :
12. $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
13. Renvoyer S

Lemme

La complexité $f(n)$ de Fusion est $O(n)$.

Preuve :

- 1 itération de lgr n
- chaque itération coûte $\Theta(1)$

Algorithme de Fusion

Fusion(T_1, T_2):

1. $n_1 \leftarrow \#T_1$; $n_2 \leftarrow \#T_2$
2. $S \leftarrow$ tableau de taille $n = n_1 + n_2$
3. $i_1 \leftarrow 0$; $i_2 \leftarrow 0$
4. Pour $i_S = 0$ à $n - 1$:
5. Si $i_1 \geq n_1$: *(T_1 vide)*
6. $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
7. Sinon si $i_2 \geq n_2$: *(T_2 vide)*
8. $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
9. Sinon si $T_{1[i_1]} \leq T_{2[i_2]}$:
10. $S_{[i_S]} \leftarrow T_{1[i_1]}$; $i_1 \leftarrow i_1 + 1$
11. Sinon :
12. $S_{[i_S]} \leftarrow T_{2[i_2]}$; $i_2 \leftarrow i_2 + 1$
13. Renvoyer S

Lemme

La complexité $f(n)$ de Fusion est $O(n)$.

Lemme

Si T_1 et T_2 sont deux tableaux triés (par ordre croissant), Fusion(T_1, T_2) renvoie un tableau trié contenant l'union des éléments de T_1 et T_2.

Preuve : Invariant

$S_{[0,i_S[}$ est trié et contient les i_S plus petits éléments de $T_1 \cup T_2$

Retour sur le TriFusion

Théorème

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

Preuve de correction par récurrence

- $n < 2$: ✓
- $n \geq 2$: Hyp. : TriFusion trie correctement les tableaux de taille $< n$
  ↳ T_1 et T_2 sont triés (car $\lfloor n/2 \rfloor \leq \lceil n/2 \rceil < n$ pour $n \geq 2$)

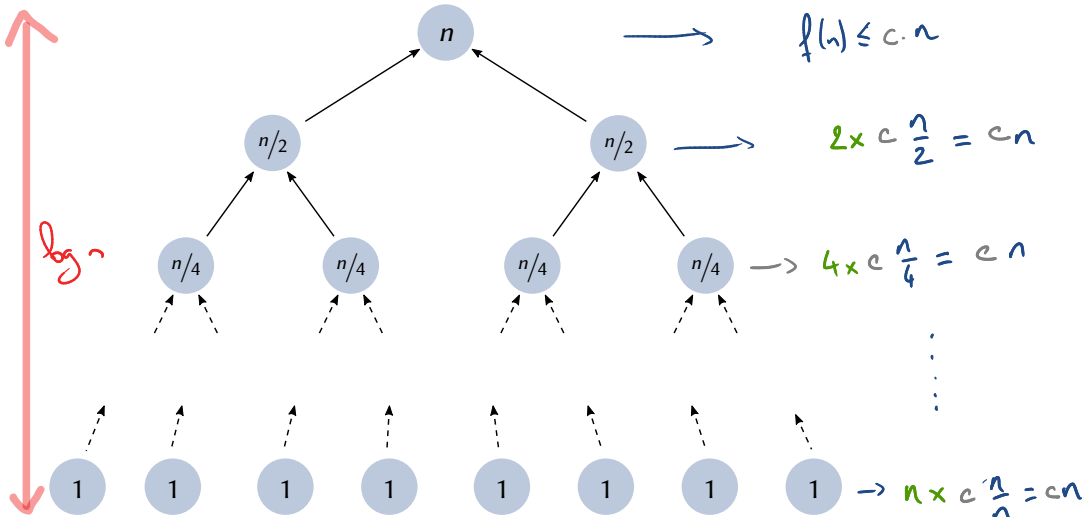
  La correction découle de celle de Fusion

Preuve de complexité

$$t(n) = t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + \Theta(n)$$

↳ $t(n) = \Theta(n \log n)$ (à démontrer)

Intuition de la complexité de TriFusion



$(f(n) \leq cn)$

$cn \log n = \mathcal{O}(n \log n)$

Table des matières

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 : la recherche dichotomique

Exemple du tri fusion

1. Diviser le tableau en 2 sous-tableaux de tailles environ égales
2. Trier récursivement chaque sous-tableau
3. Fusionner les sous-tableaux triés

Analyse d'un algorithme « diviser pour régner »

Récurrence(s) sur la taille du problème

Correction

- Hypothèse de récurrence : les appels récursifs sont corrects
- Preuve d'hérédité : *diviser* et/ou *combiner* sont correctes
- Preuve de correction

Complexité

1. Établir l'équation de récurrence
2. Résoudre la récurrence :
   - soit estimation (arbre de récursion, …) puis preuve par récurrence
   - soit utilisation du *master theorem*

Une version du « *master theorem* »

Théorème

Soit $T : \mathbb{N} \to \mathbb{R}$ vérifiant pour tout $n \geq n_0$,

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

où a, $d \geq 0$ et $b > 1$. Alors

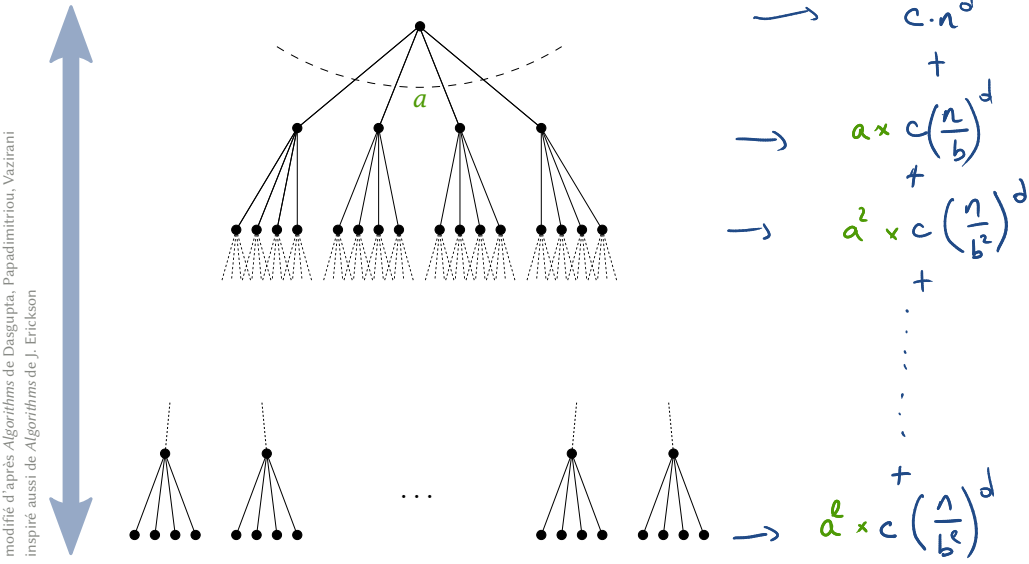
$$T(n) = \begin{cases} O(n^d) & \text{si } b^d > a \quad (d > \log_b a) \\ O(n^d \log n) & \text{si } b^d = a \quad (d = \log_b a) \\ O(n^{\log_b a}) & \text{si } b^d < a \quad (d < \log_b a) \end{cases}$$

Exemple du tri fusion

$t(n) \leq t(\lceil n/2 \rceil]) + t(\lfloor n/2 \rfloor) + O(n) \leq 2t(\lceil n/2 \rceil) + \Theta(n)$

$a = 2$, $b = 2$, $d = 1$ $\longrightarrow$ $t(n) = \Theta(n^d \log n) = \Theta(n \log n)$

Intuition graphique



Cœur de la preuve

Résoudre $T(n) \leq aT(\lceil n/b \rceil) + c \cdot n^d$ avec $n = b^\ell$

Lemme

Pour $\ell \geq 0$, $T(b^\ell) \leq a^\ell T(1) + cb^{d\ell} \sum_{i=0}^{\ell-1} (a/b^d)^i$

- $\ell = 0$ $\quad T(1) \leq T(1)$ ✓
- $\ell > 0$ $\quad T(b^{\ell+1}) \leq a \cdot T(b^\ell) + c\, b^{(\ell+1)d} \overset{\text{H.R.}}{\leq} a\left[a^\ell T(1) + c\, b^{d\ell} \sum_{i=0}^{\ell-1} (a/b^d)^i\right] + cb^{(\ell+1)d}$

$$\leq a^{\ell+1} T(1) + a \cdot cb^{d\ell} \sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i + cb^{(\ell+1)d}$$

$$= a^{\ell+1} T(1) + cb^{d\ell} \sum_{i=0}^{\ell-1} \frac{a^{i+1}}{(b^d)^{i+1}} b^d + cb^{(\ell+1)d} = a^{\ell+1} T(1) + cb^{d(\ell+1)} \sum_{i=1}^{\ell} \left(\frac{a}{b^d}\right)^i + cb^{(\ell+1)d}$$

$$= a^{\ell+1} T(1) + cb^{d(\ell+1)} \sum_{i=0}^{\ell} \left(a/b^d\right)^i$$ ✓

Cœur de la preuve

Résoudre $T(n) \leq aT(\lceil n/b \rceil) + c \cdot n^d$ avec $n = b^\ell$

Lemme

Pour $\ell \geq 0$, $T(b^\ell) \leq a^\ell T(1) + cb^{d\ell} \sum_{i=0}^{\ell-1} (a/b^d)^i$

Lemme

$$\sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i = \begin{cases} O(1) & \text{si } b^d > a \quad (1) \\ O(\ell) & \text{si } b^d = a \quad (2) \\ O((a/b^d)^\ell) & \text{si } b^d < a \quad (3) \end{cases}$$

(1)+(3) $\sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i = \frac{(a/b^d)^\ell - 1}{a/b^d - 1}$ $\longrightarrow \leq \frac{1}{1 - a/b^d}$ si $a < b^d$ $\longrightarrow \Theta(1)$

$\longrightarrow \Theta\left(\left(\frac{a}{b^d}\right)^\ell\right)$ si $a > b^d$

(2) Si $a = b^d$, $\sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i = \sum_{i=0}^{\ell-1} 1 = \ell$

Cœur de la preuve

Résoudre $T(n) \leq aT(\lceil n/b \rceil) + c \cdot n^d$ avec $n = b^\ell$

Lemme

Pour $\ell \geq 0$, $T(b^\ell) \leq a^\ell T(1) + cb^{d\ell} \sum_{i=0}^{\ell-1} (a/b^d)^i$

Lemme

$$\sum_{i=0}^{\ell-1} \left(\frac{a}{b^d}\right)^i = \begin{cases} O(1) & \text{si } b^d > a \\ O(\ell) & \text{si } b^d = a \\ O((a/b^d)^\ell) & \text{si } b^d < a \end{cases}$$

Corollaire

$$T(b^\ell) = \begin{cases} O(\cancel{a^\ell} + b^{d\ell}) & \text{si } b^d > a \\ O(\cancel{a^\ell} + b^{d\ell} \cdot \ell) & \text{si } b^d = a \\ O(a^\ell + \cancel{b^{d\ell}(a/b^d)^\ell}) & \text{si } b^d < a \end{cases}$$

Fin de la preuve

Cas $n = b^\ell$

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

$(\ell = \log_b n)$

$$= O(b^{\log_b n \cdot d}) = O(n^d)$$
$$= O(n^d \log n)$$
$$= O(a^{\log_b n}) = O(b^{\log_b(a)\log_b(n)}) = O(n^{\log_b a})$$

Cas général $(n \neq b^\ell)$

- Hypothèse : $T(n)$ croissant
- Soit ℓ min tq $b^\ell \geq n$. Alors $b^{\ell-1} < n$ donc $b^\ell \leq b \cdot n$

$$T(n) \leq T(b^\ell) = \begin{cases} \Theta((bn)^d) = \Theta(n^d) & \text{si } b^d > a \\ \Theta((bn)^d \log(bn)) = \Theta(n^d \log n) & \text{si } b^d = a \\ \Theta((bn)^{\log_b a}) = \Theta(n^{\log_b a}) & \text{si } b^d < a \end{cases}.$$

Conclusion

« Diviser pour régner »

1. Diviser ; 2. Résoudre récursivement ; 3. Combiner

Conception : seuls 1. et 3. demandent de la réflexion

Correction : preuve par récurrence

Complexité : *master theorem* (en général)

à apprendre !

Autres versions du *master theorem*

- Récurrences plus générales
- Résultats plus précis
  - Constantes dans le « grand O »
  - Termes de plus bas degré

ex.: $T(n) = aT(\lceil n/b \rceil) + O(n^d \log^c n)$

Objectifs du chapitre

- Reconnaître un algo. « diviser pour régner »
- Prouver sa correction et analyser sa complexité
- Tenter une stratégie « diviser pour régner » sur un nouveau problème

Table des matières

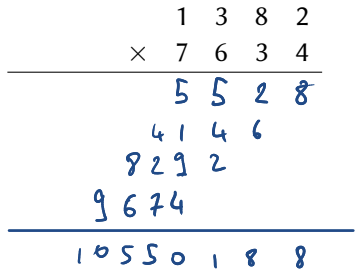
Retour à l'école primaire

Multiplication d'entiers

avec n chiffres

Entrée Deux entiers A et B écrits en base 10

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



Complexité

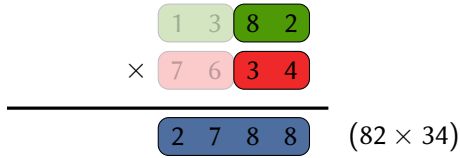
- Combien de *multiplications chiffre à chiffre* sont effectuées ? $\rightarrow \Theta(n^2)$
- Combien d'*additions chiffre à chiffre* sont effectuées ? $\rightarrow \Theta(n^2)$

Première tentative

$$\begin{array}{rcccc} & 1 & 3 & 8 & 2 \\ \times & 7 & 6 & 3 & 4 \\ \hline \end{array}$$

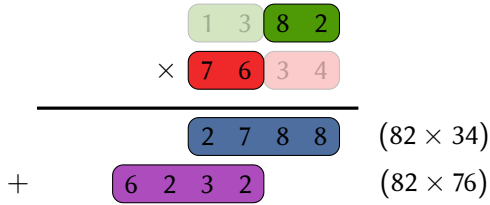
Première tentative

Première tentative

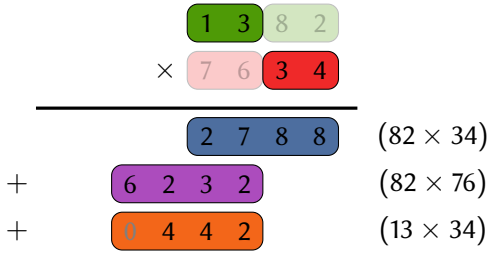


(82×34)

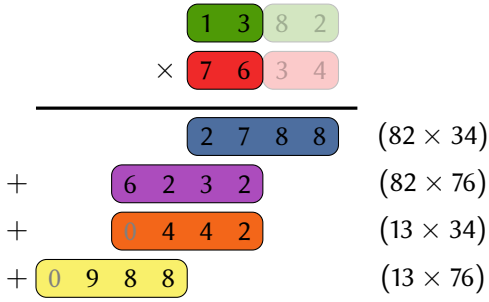
Première tentative



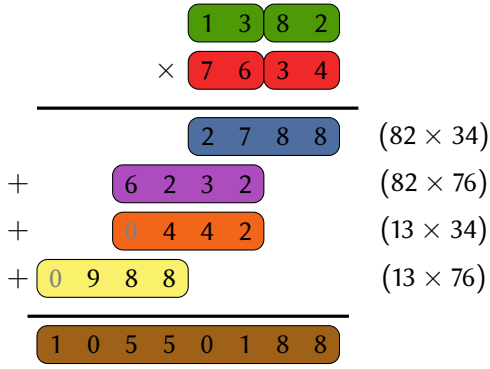
Première tentative



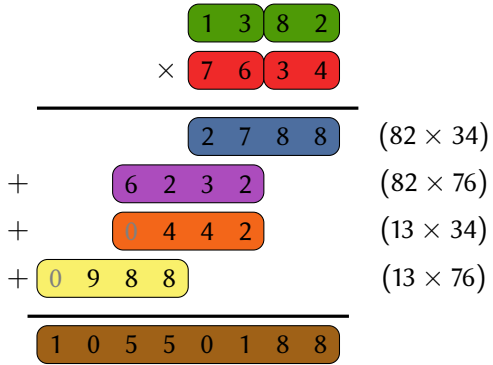
Première tentative



Première tentative



Première tentative



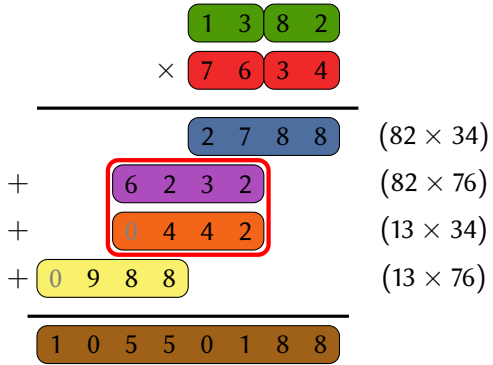
Complexité

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

$a = 4$
$b = 2$
$d = 1$

$\rightarrow T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$

Première tentative



Complexité

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

Idée de Karatsuba (version Knuth)

$$\begin{array}{rl} & (\,13 - 82\,) \\ \times & (\,76 - 34\,) \\ \hline \end{array}$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\ 1\ 3 - 8\ 2\ ) \\
\times & (\ 7\ 6 - 3\ 4\ ) \\
\hline
 & 0\ 9\ 8\ 8 \qquad (13 \times 76)
\end{array}
$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\,13 - 82\,) \\
\times & (\,76 - 34\,) \\
\hline
 & 0988 \qquad (13 \times 76) \\
- & 0442 \qquad (13 \times 34)
\end{array}
$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\ 1\ 3 - 8\ 2\ ) \\
\times & (\ 7\ 6 - 3\ 4\ ) \\
\hline
 & 0\ 9\ 8\ 8 \qquad (13 \times 76) \\
- & 0\ 4\ 4\ 2 \qquad (13 \times 34) \\
- & 6\ 2\ 3\ 2 \qquad (82 \times 76)
\end{array}
$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\,1\;3 - 8\;2\,) \\
\times & (\,7\;6 - 3\;4\,) \\
\hline
 & 0\;9\;8\;8 \qquad (13 \times 76) \\
- & 0\;4\;4\;2 \qquad (13 \times 34) \\
- & 6\;2\;3\;2 \qquad (82 \times 76) \\
+ & 2\;7\;8\;8 \qquad (82 \times 34)
\end{array}
$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\,13 - 82\,) \\
\times & (\,76 - 34\,) \\
\hline
 & 0988 \qquad (13 \times 76) \\
- & 0442 \qquad (13 \times 34) \\
- & 6232 \qquad (82 \times 76) \\
+ & 2788 \qquad (82 \times 34) \\
\hline
- & 2898
\end{array}
$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\,13 - 82\,) \\
\times & (\,76 - 34\,) \\
\hline
 & 0988 \qquad (13 \times 76) \\
- & 0442 \qquad (13 \times 34) \\
- & 6232 \qquad (82 \times 76) \\
+ & 2788 \qquad (82 \times 34) \\
\hline
- & 2898
\end{array}
$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\,13 - 82\,) \\
\times & (\,76 - 34\,) \\
\hline
 & 0988 \quad (13 \times 76) \\
- & 0442 \quad (13 \times 34) \\
- & 6232 \quad (82 \times 76) \\
+ & 2788 \quad (82 \times 34) \\
\hline
- & 2898
\end{array}
$$

$$
\begin{array}{rl}
 & 1382 \\
\times & 7634 \\
\hline
 & 2788 \\
+ & 2898 \\
+ & 2788 \\
+ & 0988 \\
+ & 0988 \\
\hline
 & 10550188
\end{array}
$$

Idée de Karatsuba (version Knuth)

$$
\begin{array}{rl}
 & (\,13 - 82\,) \\
\times & (\,76 - 34\,) \\
\hline
 & 0988 \quad (13 \times 76) \\
- & 0442 \quad (13 \times 34) \\
- & 6232 \quad (82 \times 76) \\
+ & 2788 \quad (82 \times 34) \\
\hline
- & 2898
\end{array}
$$

$$
\begin{array}{rl}
 & 1382 \\
\times & 7634 \\
\hline
 & 2788 \\
+ & 2898 \\
+ & 2788 \\
+ & 0988 \\
+ & 0988 \\
\hline
 & 10550188
\end{array}
$$

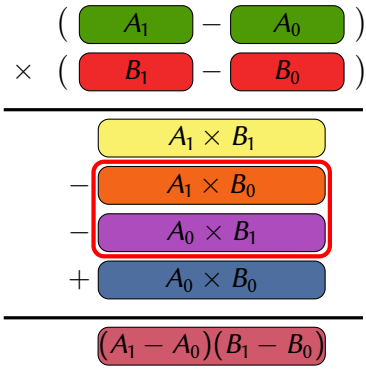
Complexité

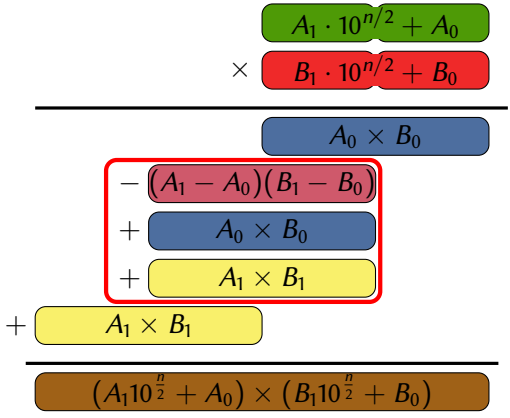
$T(n) \leq 3T(\lceil n/2 \rceil) + O(n)$

$a = 3$
$b = 2$
$d = 1$

↳ $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1,585})$

Algorithme de Karatsuba : en deux dessins





Algorithme de Karatsuba (1962)

KARATSUBA(A, B):

1. Si A et B n'ont qu'un chiffre : Renvoyer $a_0 b_0$
2. Écrire A sous la forme $A_0 + 10^{\lfloor n/2 \rfloor} \cdot A_1$
3. Écrire B sous la forme $B_0 + 10^{\lfloor n/2 \rfloor} \cdot B_1$
4. $C_{00} \leftarrow$ KARATSUBA(A_0, B_0)
5. $C_{11} \leftarrow$ KARATSUBA(A_1, B_1)
6. $D \leftarrow$ KARATSUBA$(|A_0 - A_1|, |B_0 - B_1|)$
7. $s \leftarrow \text{signe}(A_0 - A_1) \times \text{signe}(B_0 - B_1)$
8. Renvoyer $C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - s \cdot D) + 10^{2\lfloor n/2 \rfloor} C_{11}$

Correction

$$A \times B = C_{00} + 10^{\lfloor n/2 \rfloor}(C_{00} + C_{11} - sD) + 10^{2\lfloor n/2 \rfloor} C_{11}$$

Complexité

L'algorithme KARATSUBA a une complexité $O(n^{\log_2 3}) = O(n^{1{,}585})$

Dans la *vraie vie*

Base 10 $\rightsquigarrow$ bases 2^{32}, 2^{64}, …

- Grands entiers : tableaux d'entiers de w bits $\iff$ entiers en base 2^w
- Exemples : `gmp` (C/C++), `BigInteger` (Java), `int` (Python), …
- Autre utilisation : polynômes

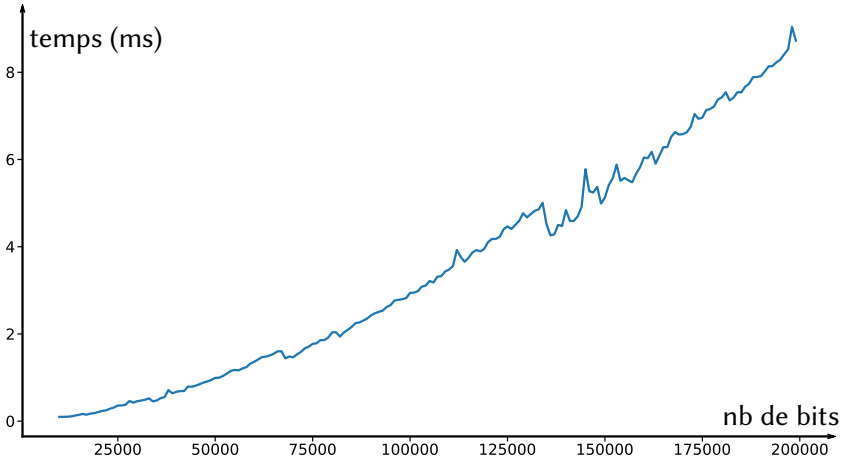
Quel TAD utiliser ?

- TAD entier : opérations en temps $O(1)$ dont multiplication
  - Réaliste pour des entiers raisonnables indices de tableau, …
  - Irréaliste pour de grands entiers cryptographie, …
- TAD entier borné : opérations en temps $O(1)$ pour des entiers $< 2^w$

Algorithmes plus rapides

- Toom-3 (1963) : découpe en 3 morceaux $O(n^{1,465})$
- Toom-Cook (1966): découpe en r morceaux $O(n^{1+\epsilon})$
- Schönhage-Strassen (1971) : basé sur la FFT $O(n \log n \log\log n)$
- …
- Harvey-Hoeven (2021) : utilise aussi la FFT **$O(n \log n)$**

Pour finir : multiplication d'entiers en Python



Pour finir : multiplication d'entiers en Python

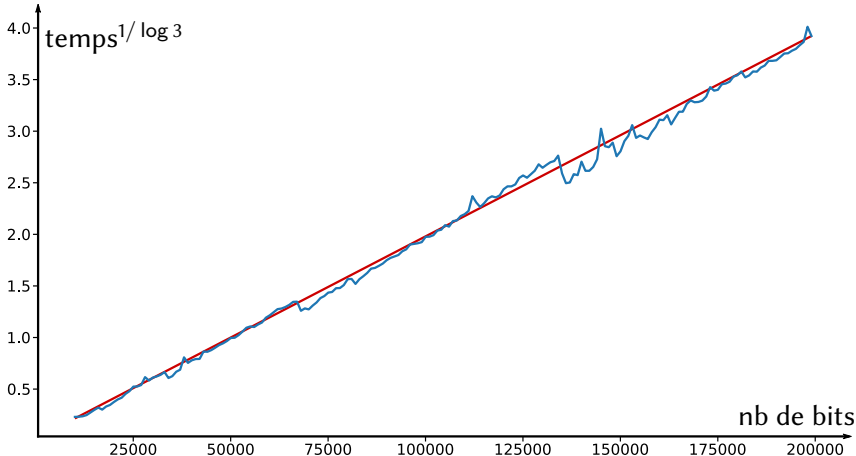
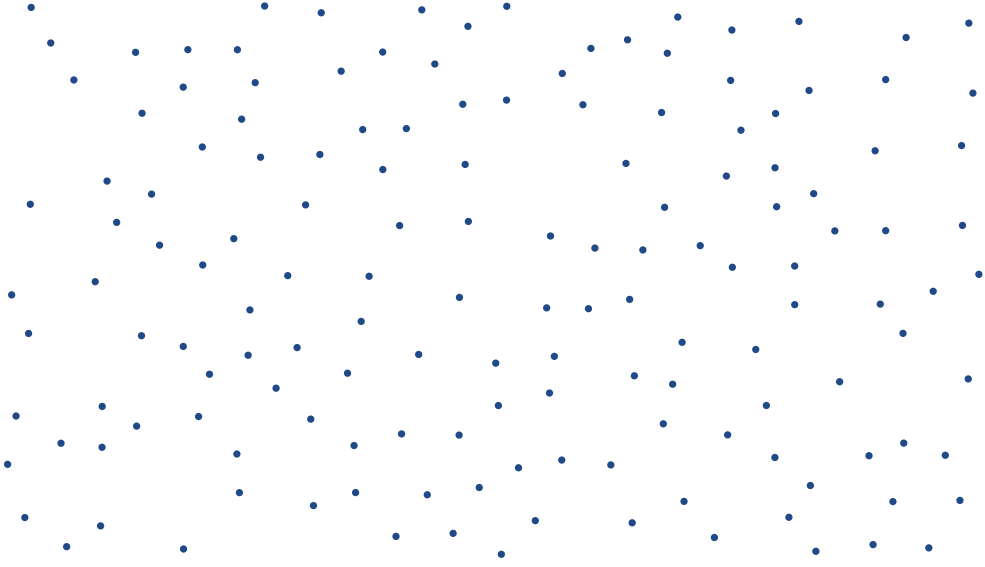
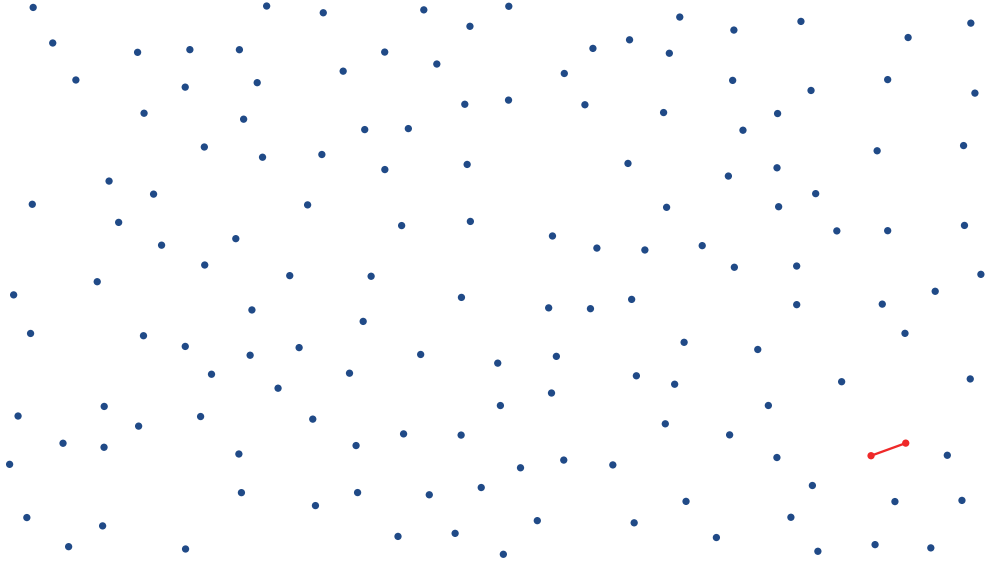


Table des matières

Quels sont les deux points les plus proches?



Quels sont les deux points les plus proches?



L'approche *naïve* : tester tous les couples de points

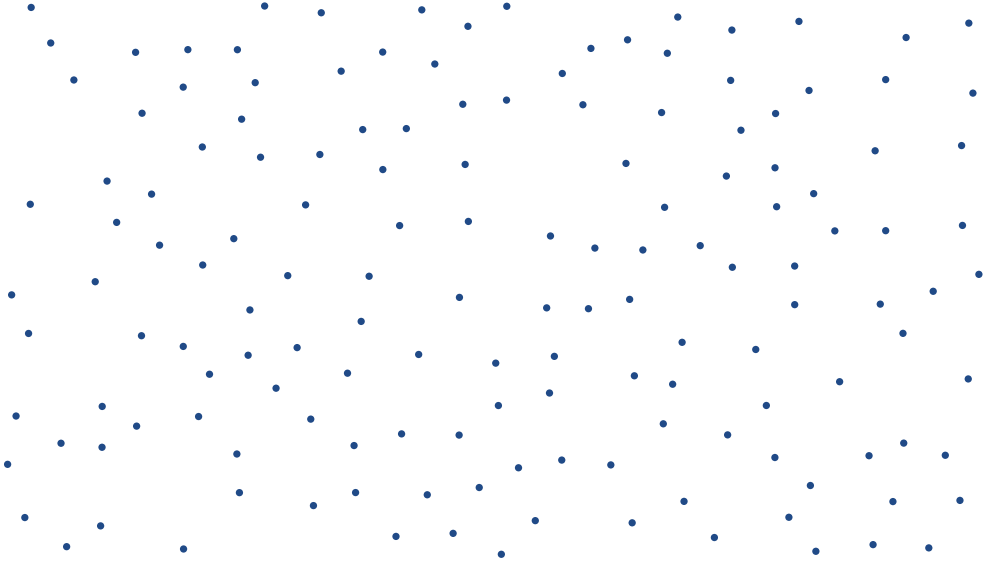
DistanceNaïve(T) : $n = \#T$

1. $d \leftarrow +\infty$
2. Pour $i = 0$ à $n-1$:
3. Pour $j = i+1$ à $n-1$:
4. $(x_i, y_i) \leftarrow T_{[i]}$
5. $(x_j, y_j) \leftarrow T_{[j]}$
6. $d_{ij} \leftarrow \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ distance($T_{[i]}$, $T_{[j]}$)
7. Si $d_{ij} < d$: $d \leftarrow d_{ij}$
8. Renvoyer d

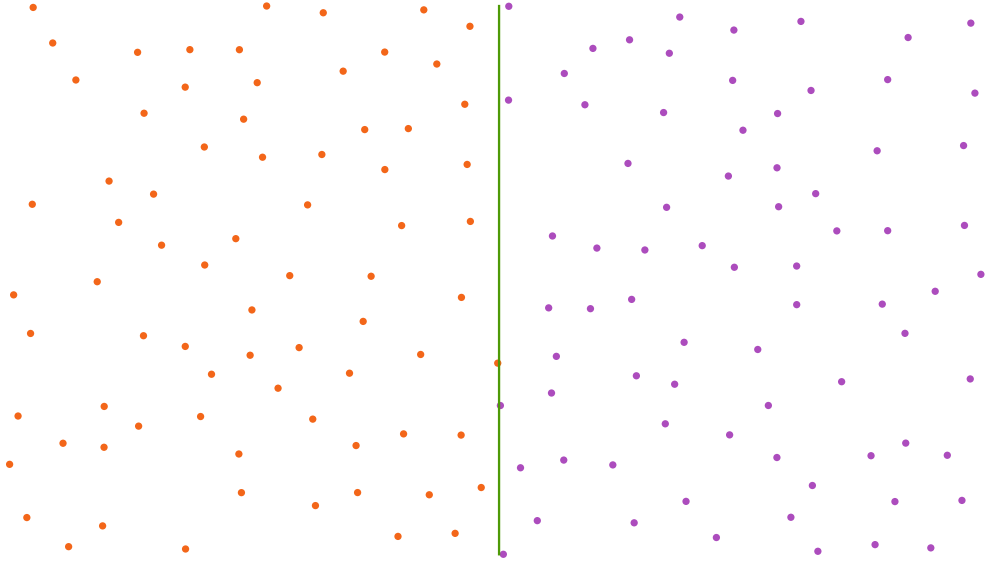
Correction et complexité

↳ ✓ ↳ $\Theta(n^2)$ [double boucle]

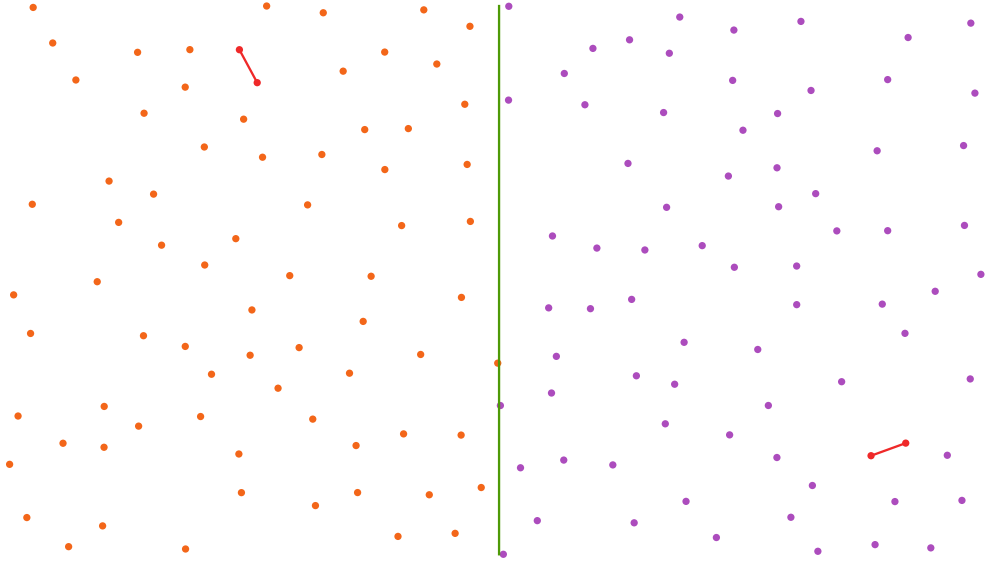
Une approche « diviser-pour-régner » — exemple 1



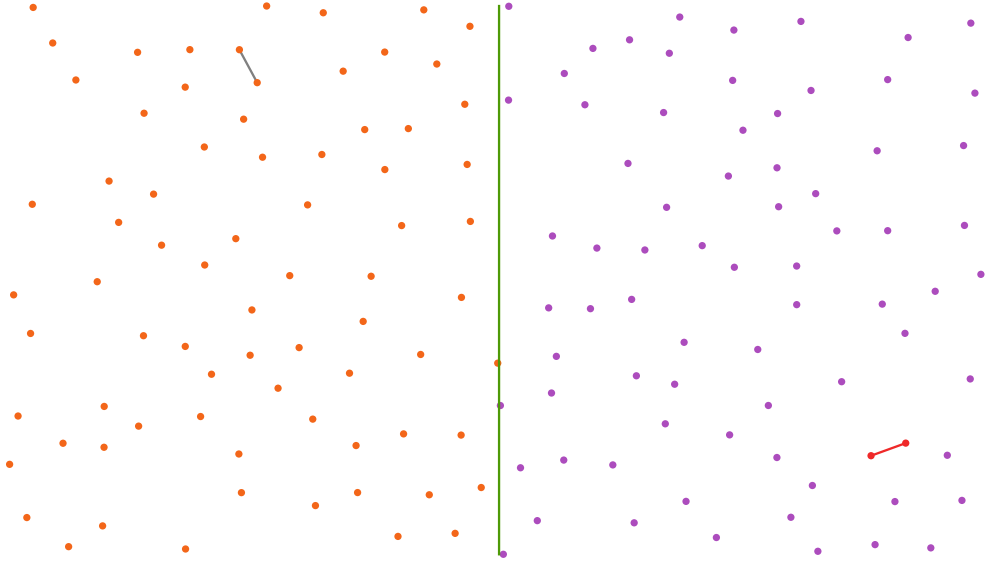
Une approche « diviser-pour-régner » — exemple 1



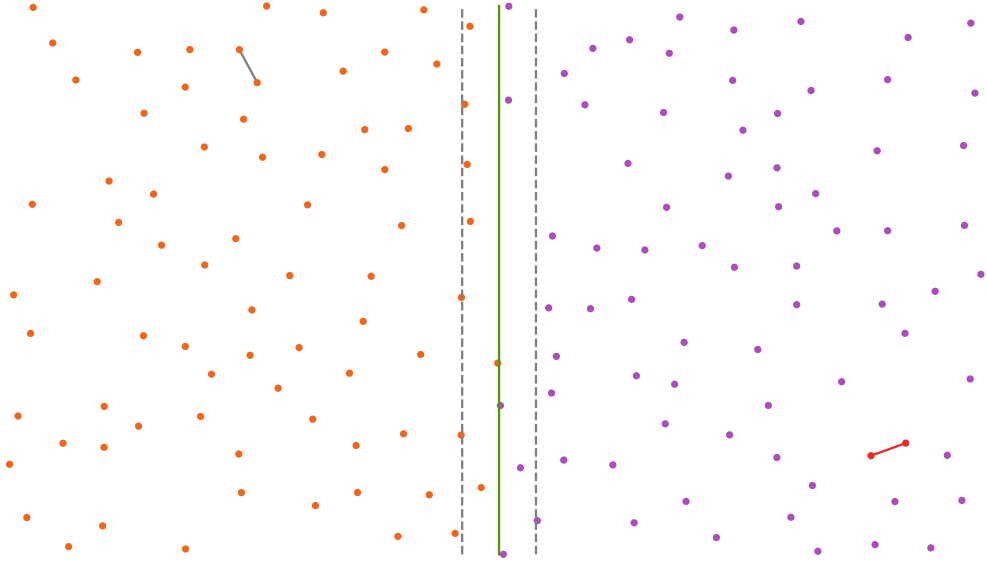
Une approche « diviser-pour-régner » — exemple 1



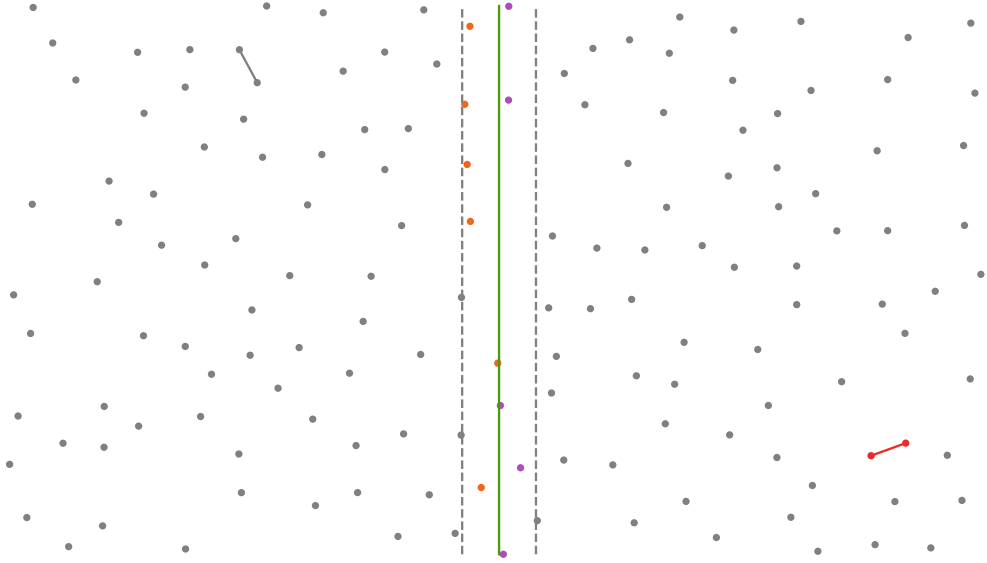
Une approche « diviser-pour-régner » — exemple 1



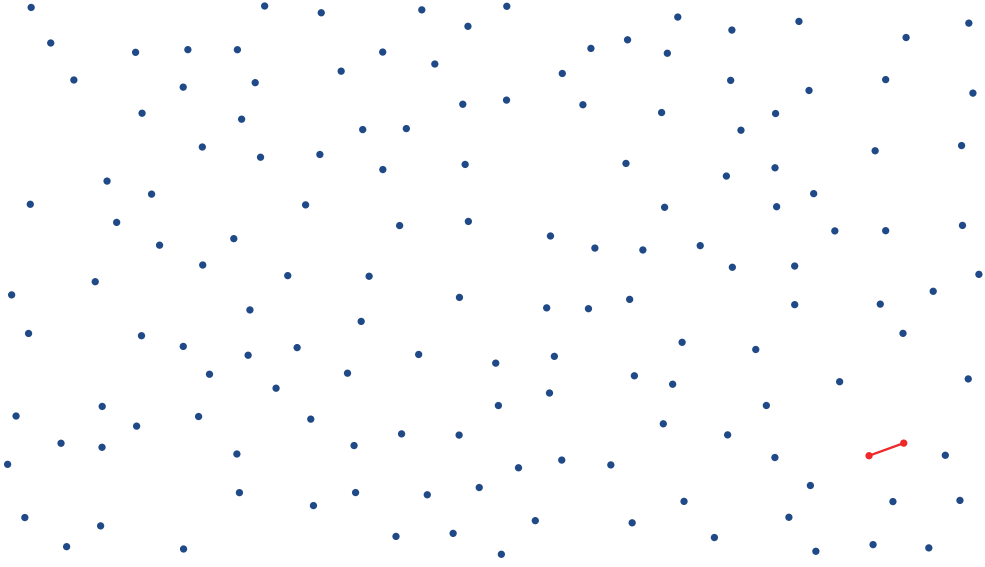
Une approche « diviser-pour-régner » — exemple 1



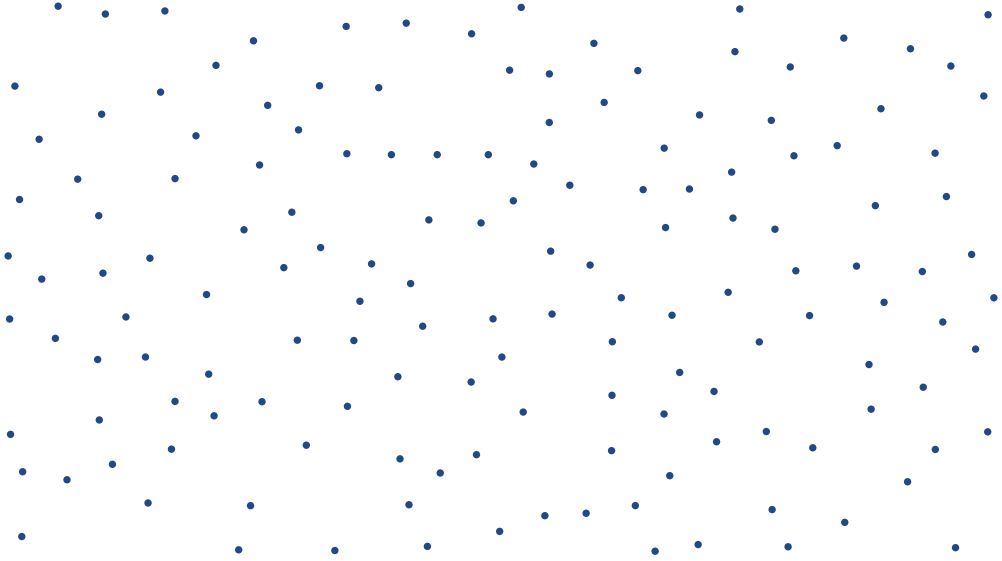
Une approche « diviser-pour-régner » — exemple 1



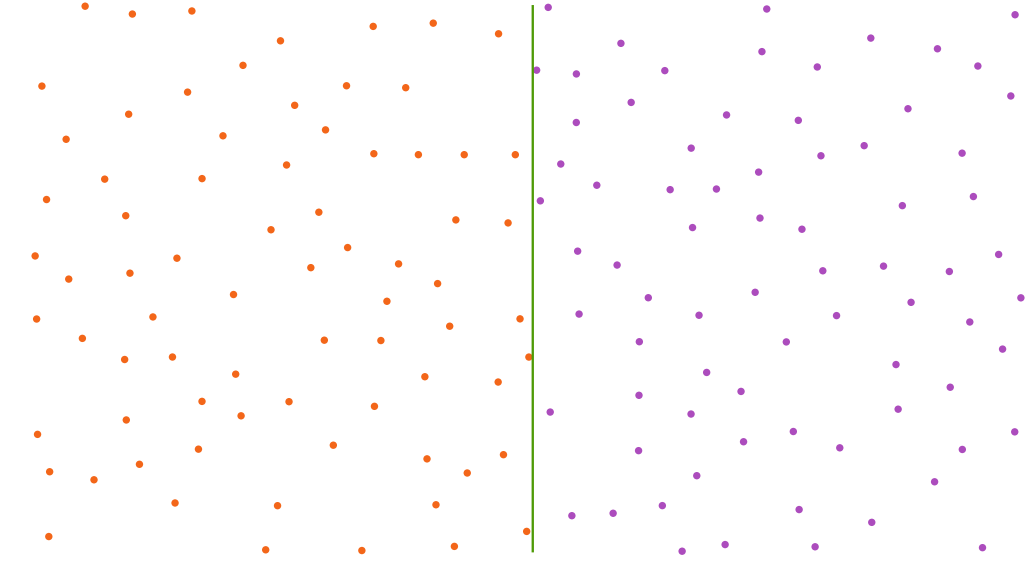
Une approche « diviser-pour-régner » — exemple 1



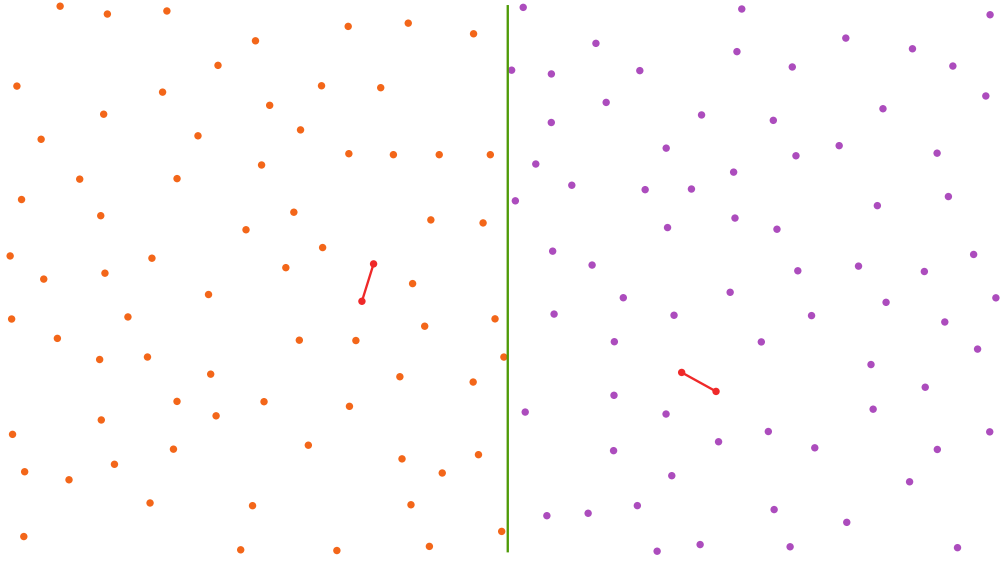
Une approche « diviser-pour-régner » — exemple 2



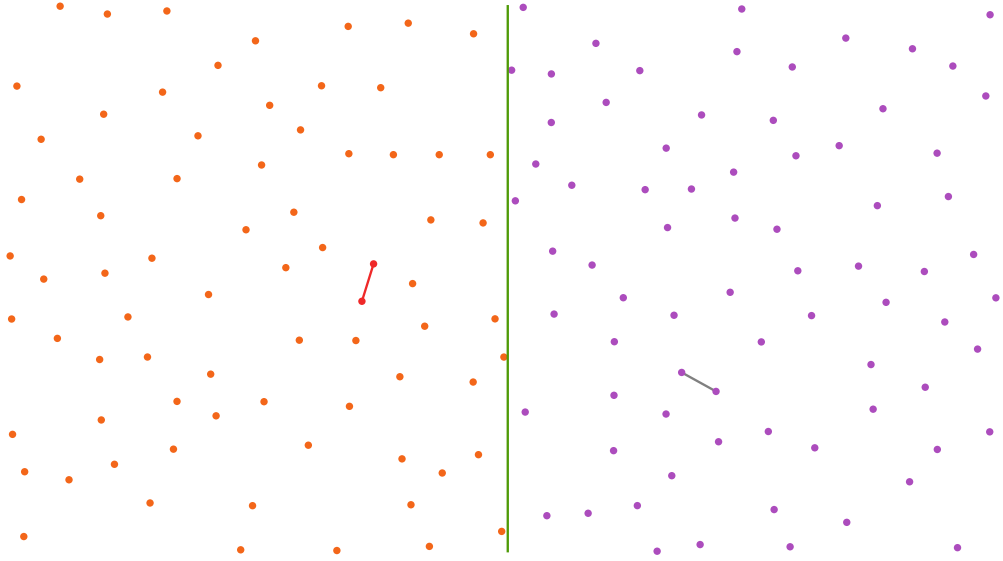
Une approche « diviser-pour-régner » — exemple 2



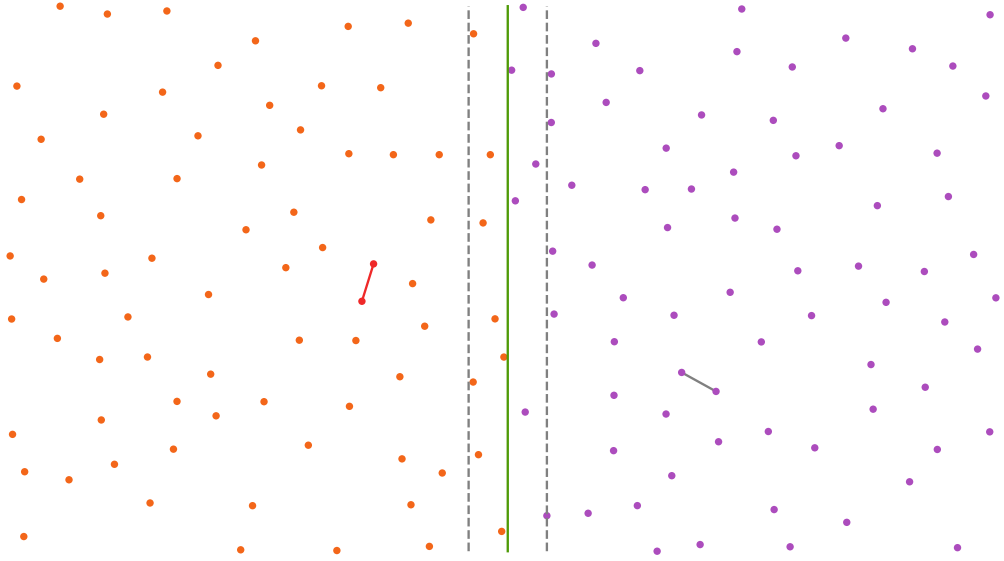
Une approche « diviser-pour-régner » — exemple 2



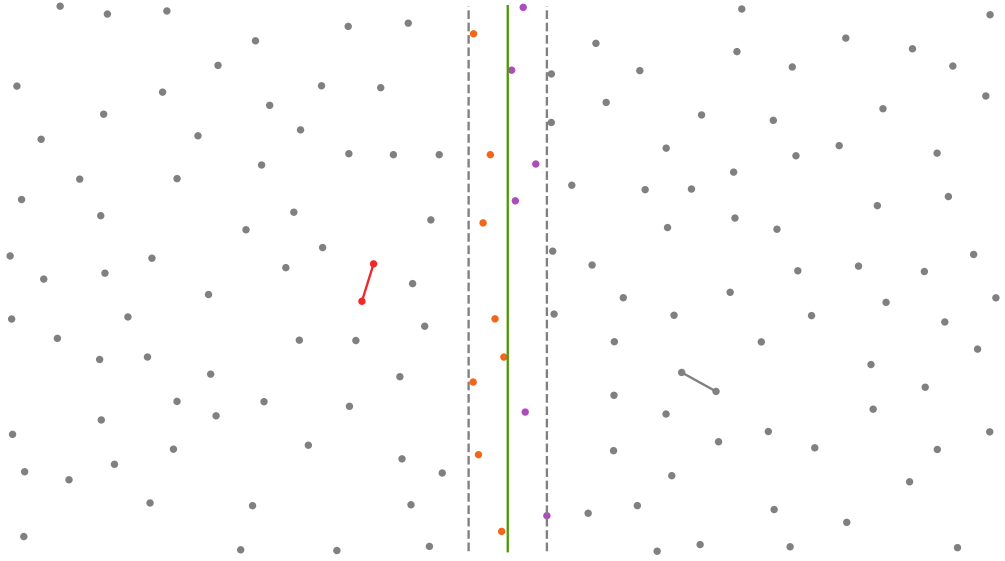
Une approche « diviser-pour-régner » — exemple 2



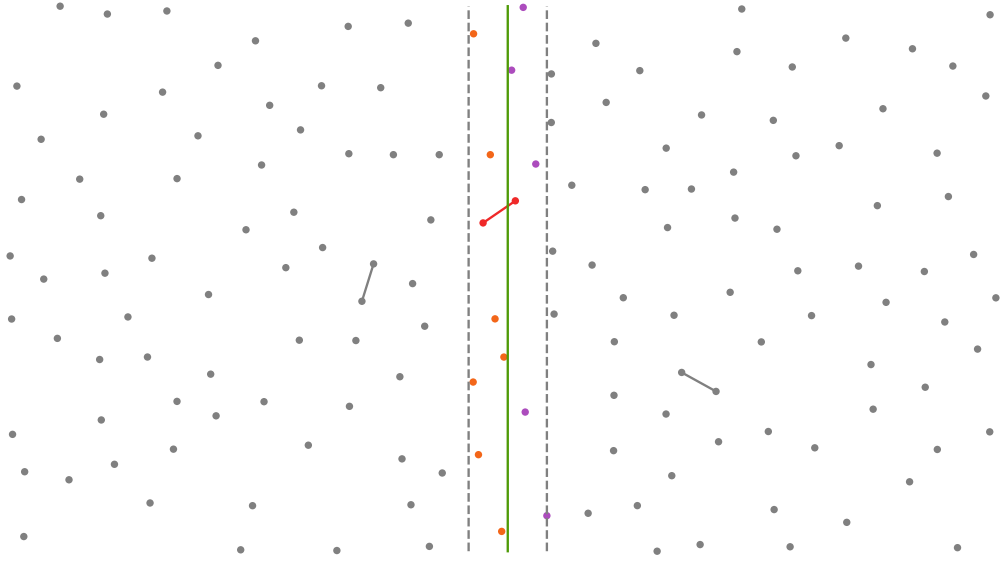
Une approche « diviser-pour-régner » — exemple 2



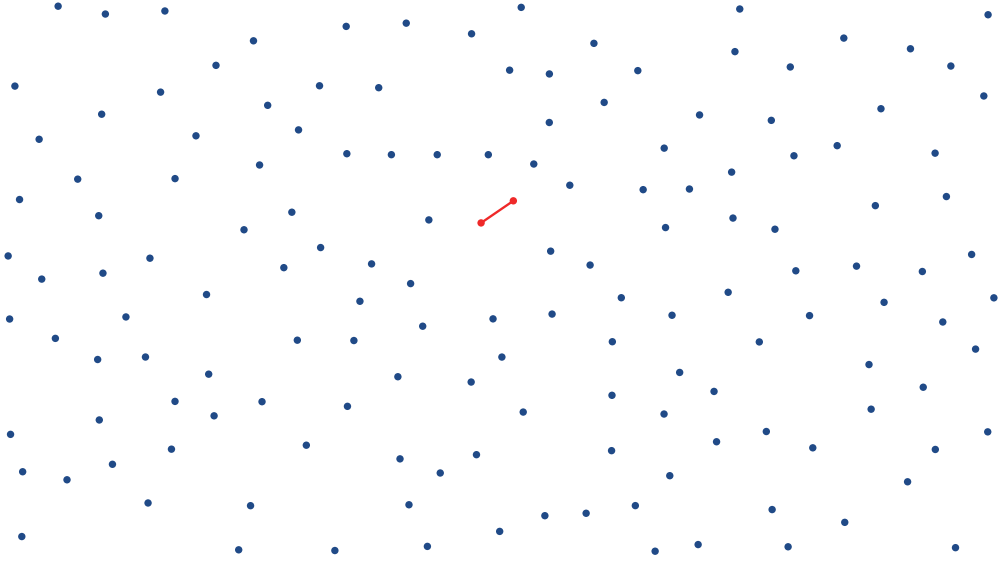
Une approche « diviser-pour-régner » — exemple 2



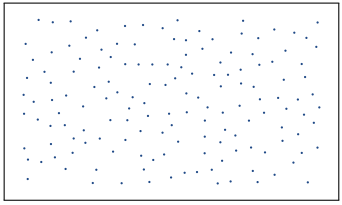
Une approche « diviser-pour-régner » — exemple 2



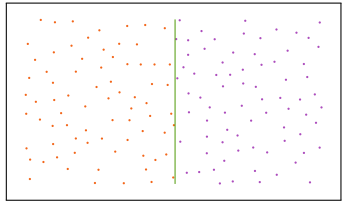
Une approche « diviser-pour-régner » — exemple 2



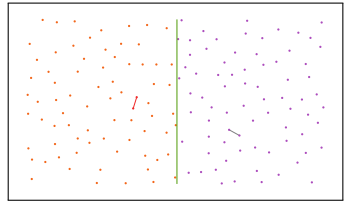
Idée de l'algorithme



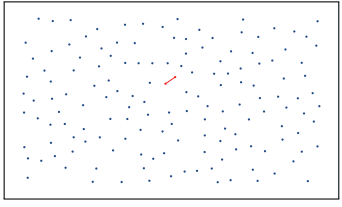
0. Entrée



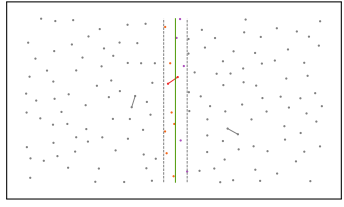
1. Diviser



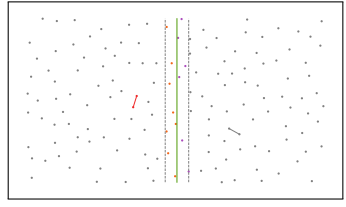
2. Résoudre récursivement



5. Sortie



4. Distance min dans la bande



3. Calcul de la bande

Algorithme : première version

DistanceMinimale(T) : $n = \#T$

0. Si $n \leq 4$: renvoyer DistanceNaïve(T)
1. Trier T par abscisses croissantes
   $(G, D) \leftarrow (T_{[0,\lceil n/2 \rceil[}, T_{[\lceil n/2 \rceil, n[})$
2. $(d_G, d_D) \leftarrow$ (DistanceMinimale(G), DistanceMinimale(D))
   $d \leftarrow \min(d_G, d_D)$
3. $m \leftarrow (T_{[\lceil n/2 \rceil - 1]} + T_{[\lceil n/2 \rceil]})/2$
   $B \leftarrow [(x, y) \in T : m - d \leq x \leq m + d]$
4. $d_B \leftarrow$ DistanceMinimale(B)
5. Renvoyer $\min(d, d_B)$

Complexité

$$t(n) \leq 2t(\lceil n/2 \rceil) + \Theta(n \log n) + t(\#B)$$

(t croissante) $\qquad \#B \leq n$

⚠ l'algo ne termine pas

Distance minimale dans la bande

Lemme

Soit $(x_0, y_0), (x_1, y_1) \in B$ à distance $< d$, $y_0 \leq y_1$
Alors il existe ≤ 8 points $(x_i, y_i) \in B$ tels que $y_0 \leq y_i \leq y_1$

Distance minimale dans la bande

Lemme

Soit $(x_0, y_0), (x_1, y_1) \in B$ vérifiant $y_0 \leq y_1 \leq y_0 + d$
Alors il existe ≤ 8 points $(x_i, y_i) \in B$ tels que $y_0 \leq y_i \leq y_1$

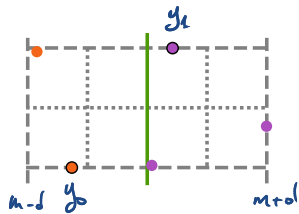
Distance minimale dans la bande

Lemme

Soit $(x_0, y_0), (x_1, y_1) \in B$ vérifiant $y_0 \leq y_1 \leq y_0 + d$
Alors il existe ≤ 8 points $(x_i, y_i) \in B$ tels que $y_0 \leq y_i \leq y_1$

Preuve

Rectangle de taille $2d \times d$
Ls chaque petit carré de $d/2 \times d/2$
ne contient qu'un seul point au plus
(car la diag. fait $\sqrt{\frac{d^2}{2}} = d/\sqrt{2} < d$)



Distance minimale dans la bande

Lemme

Soit $(x_0, y_0), (x_1, y_1) \in B$ vérifiant $y_0 \leq y_1 \leq y_0 + d$
Alors il existe ≤ 8 points $(x_i, y_i) \in B$ tels que $y_0 \leq y_i \leq y_1$

DistanceBande(B, d) : $n = \#B$

1. Trier B par ordonnées croissantes → $\Theta(n \log n)$
2. Pour $i = 0$ à $n - 1$:
3. Pour $j = i + 1$ à $i + 7$:
4. $d_{ij} \leftarrow$ distance$(B_{[i]}, B_{[j]})$ } $\Theta(1)$
5. Si $d_{ij} < d$: $d \leftarrow d_{ij}$
6. Renvoyer d

Complexité

$\Theta(n) + \Theta(n \log n)$

Algorithme : deuxième version

DistanceMinimale(T) : $n = \#T$

0. Si $n \leq 4$: renvoyer DistanceNaïve(T)
1. Trier T par abscisses croissantes
   $(G, D) \leftarrow (T_{[0,\lceil n/2 \rceil[}, T_{[\lceil n/2 \rceil, n[})$
2. $(d_G, d_D) \leftarrow$ (DistanceMinimale(G), DistanceMinimale(D))
   $d \leftarrow \min(d_G, d_D)$
3. $m \leftarrow (T_{[\lceil n/2 \rceil - 1]} + T_{[\lceil n/2 \rceil]})/2$
   $B \leftarrow [(x, y) \in T : m - d \leq x \leq m + d]$
4. $d_B \leftarrow$ DistanceBande(B, d)
5. Renvoyer $\min(d, d_B)$

Complexité

$t(n) \leq 2\,t(\lceil n/2 \rceil) + \Theta(n \log n)$

$a = 2$
$b = 2$
$d = 1$

$\hookrightarrow t(n) = \Theta(n \log^2 n)$

Algorithme : version finale

DistanceRec(X, Y) : $n = \#X\ (= \#Y)$

0. Si $n \leq 4$: renvoyer DistanceNaïve(X)
1. $(X_G, X_D) \leftarrow (X_{[0,\lceil n/2 \rceil[}, X_{[\lceil n/2 \rceil, n[})$
   $(Y_G, Y_D) \leftarrow ([(x,y) \in Y : (x,y) < X_{\lceil n/2 \rceil}], [(x,y) \in Y : (x,y) \geq X_{\lceil n/2 \rceil}])$
2. $(d_G, d_D) \leftarrow (\text{DistanceRec}(X_G, Y_G), \text{DistanceRec}(X_D, Y_D))$
   $d \leftarrow \min(d_G, d_D)$
3. $m \leftarrow (X_{[\lceil n/2 \rceil - 1]} + X_{[\lceil n/2 \rceil]})/2$
   $B \leftarrow [(x,y) \in Y : m - d \leq x \leq m + d]$
4. $d_B \leftarrow$ DistanceBande(B, d)
5. Renvoyer $\min(d, d_B)$

$t_{Rec}(n) = 2\, t_{REC}(\lceil \frac{n}{2} \rceil) + \Theta(n)$

$\hookrightarrow \Theta(n \log n)$

DistanceMinimale(T) : $n = \#T$

0. Si $n \leq 4$: renvoyer DistanceNaïve(T)
1. $X \leftarrow$ tableau T trié par abscisses croissantes
   $Y \leftarrow$ tableau T trié par ordonnées croissantes
2. Renvoyer DistanceRec(X, Y)

$\Theta(n \log n)$
$\Theta(n \log n)$
$\Theta(n \log n)$

Bilan sur la distance minimale

Théorème

DistanceMinimale(T) renvoie la distance minimale entre deux points de T en temps $O(n \log n)$, où $n = \#T$

Remarque

- Modification simple pour aussi renvoyer les points minimisant la distance

Preuve du théorème

Déjà fait

Conclusion sur « diviser-pour-régner »

Conception d'algorithme : trois étapes

1. Diviser
2. Résoudre
3. Combiner

Analyse de complexité

1. Équation de récurrence
2. *Master theorem*

Domaines d'application

- Tableaux, arbres, graphes, …
- Géométrie algorithmique
- Calcul formel et numérique (entiers, polynômes, matrices)