Diviser pour régner

Quentin Fortier

November 7, 2022

Diviser pour régner

Pour résoudre un problème, il est courant de le ramener à des sous-problèmes plus simples.

Diviser pour régner

Pour résoudre un problème, il est courant de le ramener à des sous-problèmes plus simples.

Deux grandes méthodes pour le faire :

1. **Diviser pour régner** (*divide and conquer*) : résoudre les sous-problèmes (récursivement) puis les combiner pour obtenir une solution du problème initial.

Diviser pour régner

Pour résoudre un problème, il est courant de le ramener à des sous-problèmes plus simples.

Deux grandes méthodes pour le faire :

1. **Diviser pour régner** (*divide and conquer*) : résoudre les sous-problèmes (récursivement) puis les combiner pour obtenir une solution du problème initial.
2. **Programmation dynamique / mémoïsation** (*dynamic programming*): similaire, mais en conservant en mémoire tous les sous-problèmes pour éviter de les calculer plusieurs fois.

Diviser pour régner

Méthode diviser pour régner pour résoudre un problème $P(n)$ de taille n :

- Si n est un cas de base, renvoyer directement la solution.
- Sinon :
  1. Séparer (*divide*) $P(n)$ en k sous-problèmes $P(n_1), ..., P(n_k)$.
  2. Résoudre récursivement et de façon indépendante $P(n_1), ..., P(n_k)$.
  3. Combiner (*conquer*) les solutions à $P(n_1), ..., P(n_k)$ pour résoudre $P(n)$.

Diviser pour régner

Méthode diviser pour régner pour résoudre un problème $P(n)$ de taille n :

- Si n est un cas de base, renvoyer directement la solution.
- Sinon :
    1. Séparer (*divide*) $P(n)$ en k sous-problèmes $P(n_1), ..., P(n_k)$.
    2. Résoudre récursivement et de façon indépendante $P(n_1), ..., P(n_k)$.
    3. Combiner (*conquer*) les solutions à $P(n_1), ..., P(n_k)$ pour résoudre $P(n)$.

Si tous les sous-problèmes sont de même taille $\frac{n}{k}$, alors la complexité $C(n)$ de résolution de $P(n)$ vérifie :

$$C(n) = \underbrace{f(n)}_{\text{séparation}} + kC(\frac{n}{k}) + \underbrace{g(n)}_{\text{fusion}}$$

Dichotomie

La méthode par dichotomie est un cas particulier de la méthode diviser pour régner, où on se ramène à un seul sous-problème :

```
let dichotomie t e =
    (* détermine si e appartient au tableau trié t *)
    let rec aux i j =
    (* détermine si e appartient à t.(i), ..., t.(j) *)
        if i > j then false (* aucun élément *)
        else let m = (i + j)/2 in (* milieu *)
             if t.(m) = e then true
             else if t.(m) < e then aux (m + 1) j
             else aux i (m - 1)
    in aux 0 (Array.length t - 1)
```

Complexité :

Dichotomie

La méthode par dichotomie est un cas particulier de la méthode diviser pour régner, où on se ramène à un seul sous-problème :

```
let dichotomie t e =
    (* détermine si e appartient au tableau trié t *)
    let rec aux i j =
    (* détermine si e appartient à t.(i), ..., t.(j) *)
        if i > j then false (* aucun élément *)
        else let m = (i + j)/2 in (* milieu *)
             if t.(m) = e then true
             else if t.(m) < e then aux (m + 1) j
             else aux i (m - 1)
    in aux 0 (Array.length t - 1)
```

Complexité : $\mathrm{O}(\log(n))$ où n est la taille de `t`.

Tri fusion

[5, 1, -4, 2, -8, 7]

Tri fusion

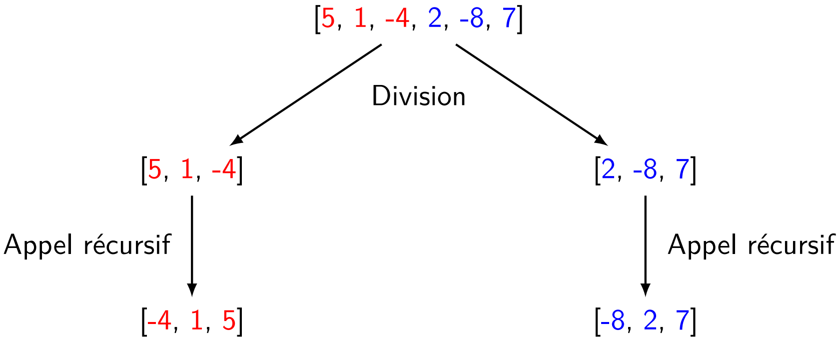
[5, 1, -4, 2, -8, 7]

Division

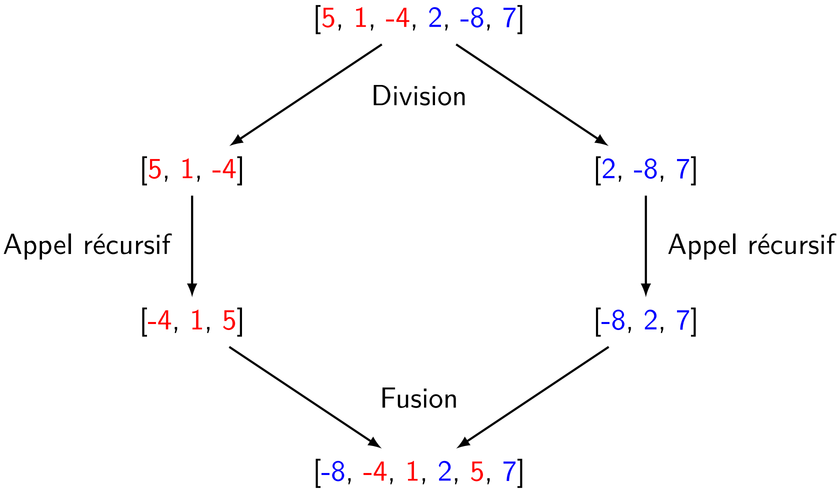
[5, 1, -4]

[2, -8, 7]

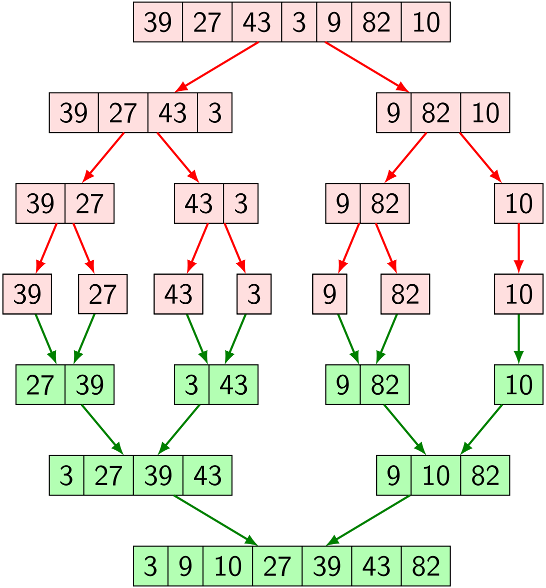
Tri fusion


[5, 1, -4, 2, -8, 7]
Division
[5, 1, -4]
[2, -8, 7]
Appel récursif
Appel récursif
[-4, 1, 5]
[-8, 2, 7]

Tri fusion


[5, 1, -4, 2, -8, 7]
Division
[5, 1, -4]
[2, -8, 7]
Appel récursif
Appel récursif
[-4, 1, 5]
[-8, 2, 7]
Fusion
[-8, -4, 1, 2, 5, 7]

Tri fusion


39 27 43 3 9 82 10
39 27 43 3
9 82 10
39 27
43 3
9 82
10
39
27
43
3
9
82
10
27 39
3 43
9 82
10
3 27 39 43
9 10 82
3 9 10 27 39 43 82

Tri fusion

Diviser une liste en deux :

```
let rec split = function
    | [] -> [], []
    | [e] -> [e], []
    | e1::e2::q -> let q1, q2 = split q in
                   e1::q1, e2::q2
```

<u>Complexité</u> :

Tri fusion

Diviser une liste en deux :

```
let rec split = function
    | [] -> [], []
    | [e] -> [e], []
    | e1::e2::q -> let q1, q2 = split q in
                   e1::q1, e2::q2
```

Complexité : O(n) où n est la taille de la liste

Tri fusion

Fusionner deux listes triées :

```
let rec fusion l1 l2 = match l1, l2 with
    | [], _ -> l2
    | _, [] -> l1
    | e1::q1, e2::q2 when e1 < e2 -> e1::fusion q1 l2
    | e1::q1, e2::q2 -> e2::fusion l1 q2
```

Complexité :

Tri fusion

Fusionner deux listes triées :

```
let rec fusion l1 l2 = match l1, l2 with
    | [], _ -> l2
    | _, [] -> l1
    | e1::q1, e2::q2 when e1 < e2 -> e1::fusion q1 l2
    | e1::q1, e2::q2 -> e2::fusion l1 q2
```

Complexité : O(n) où n est la taille de la plus petite liste

Tri fusion

```
let rec tri = function
    | [] -> []
    | [e] -> [e] (* tri ne termine pas sans ce cas *)
    | l -> let l1, l2 = split l in
           fusion (tri l1) (tri l2);;
```

Complexité :

Tri fusion

```
let rec tri = function
    | [] -> []
    | [e] -> [e] (* tri ne termine pas sans ce cas *)
    | l -> let l1, l2 = split l in
           fusion (tri l1) (tri l2);;
```

Complexité : Soit $C(n)$ la complexité de `tri l` pour `l` de taille n.

Tri fusion

```
let rec tri = function
    | [] -> []
    | [e] -> [e] (* tri ne termine pas sans ce cas *)
    | l -> let l1, l2 = split l in
           fusion (tri l1) (tri l2);;
```

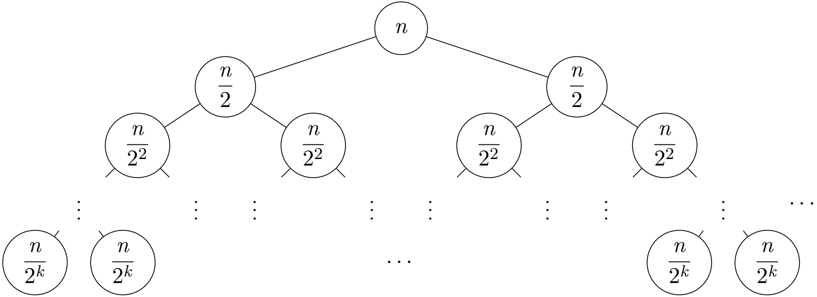
Complexité : Soit $C(n)$ la complexité de `tri l` pour `l` de taille n.

$$
\begin{aligned}
C(n) &= \underbrace{O(n)}_{split} + \underbrace{O(n)}_{fusion} + 2C(n/2) \leq Kn + 2C(n/2) \\
&\leq Kn + 2K\frac{n}{2} + 4C(n/4) = 2Kn + 4C(n/4) \\
&\leq ... \leq pKn + 2^p C(n/2^p) \underset{p=\log_2(n)}{=} \boxed{O(n\log_2(n))}
\end{aligned}
$$

où Kn est un majorant de la complexité de split plus fusion.

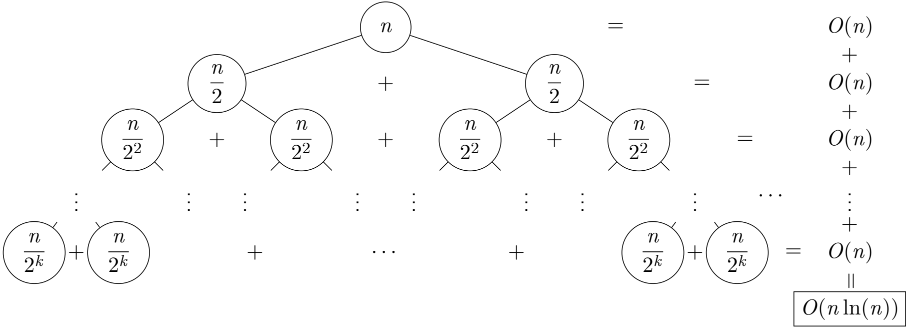
Tri fusion

On peut représenter les appels récursifs du tri fusion sous forme d'un arbre et compter le nombre d'opérations niveau par niveau :



Chaque rond (sommet) correspond à un appel récursif, avec la taille du sous-tableau à l'intérieur.

On peut représenter les appels récursifs du tri fusion sous forme d'un arbre et compter le nombre d'opération niveau par niveau :



Chaque rond (sommet) correspond à un appel récursif, avec la taille du sous-tableau à l'intérieur.

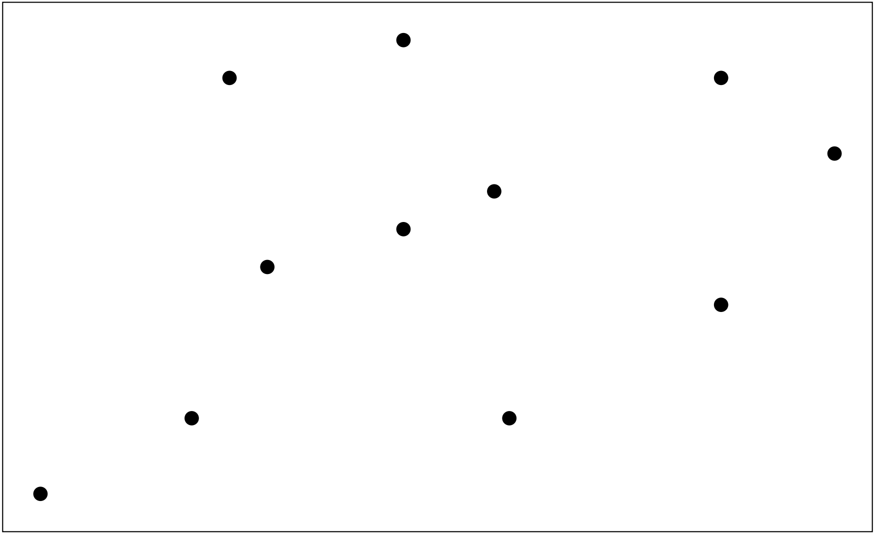
Paire de points les plus proches

Problème

Entrée : n points dans le plan.

Sortie : la plus petite distance entre 2 points.

Paire de points les plus proches



Paire de points les plus proches

<u>1ère solution</u> :

Paire de points les plus proches

1ère solution : calculer toutes les distances en conservant le minimum.

```
let dist p q =
    ((fst p -. fst q)**2. +. (snd p -. snd q)**2.)**0.5

let closest_brute points =
    let n = Array.length points in
    let d = ref max_float in
    for i = 0 to n - 1 do
        for j = i + 1 to n - 1 do
            d := min !d (dist points.(i) points.(j))
        done
    done;
    !d
```

Complexité :

Paire de points les plus proches

1ère solution : calculer toutes les distances en conservant le minimum.

```
let dist p q =
    ((fst p -. fst q)**2. +. (snd p -. snd q)**2.)**0.5

let closest_brute points =
    let n = Array.length points in
    let d = ref max_float in
    for i = 0 to n - 1 do
        for j = i + 1 to n - 1 do
            d := min !d (dist points.(i) points.(j))
        done
    done;
    !d
```

Complexité : O(n^2)

Paire de points les plus proches

<u>2ème solution</u> (diviser pour régner) :

Paire de points les plus proches

2ème solution (diviser pour régner) :

1. Choisir une abscisse x_m séparant les points en 2 sous-ensembles P_1 et P_2 de même taille (à ± 1)

Paire de points les plus proches

2ème solution (diviser pour régner) :

1. Choisir une abscisse x_m séparant les points en 2 sous-ensembles P_1 et P_2 de même taille (à ± 1)
2. Trouver les plus petites distances d_1 et d_2 dans P_1 et P_2

Paire de points les plus proches

2ème solution (diviser pour régner) :

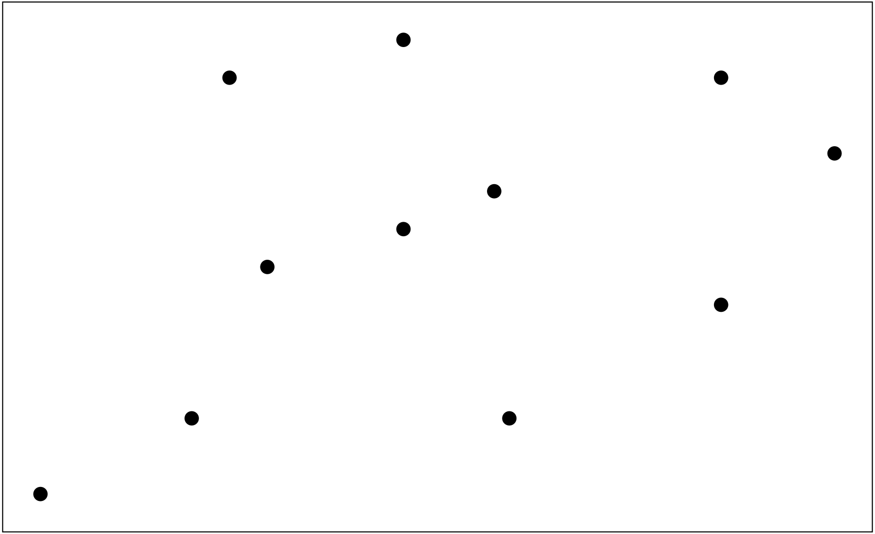
1. Choisir une abscisse x_m séparant les points en 2 sous-ensembles P_1 et P_2 de même taille (à ± 1)
2. Trouver les plus petites distances d_1 et d_2 dans P_1 et P_2
3. Trouver la plus petite distance d_3 parmi les points dans la bande d'abscisse $[x_m - \min(d_1, d_2), x_m + \min(d_1, d_2)]$

Paire de points les plus proches

<u>2ème solution</u> (diviser pour régner) :

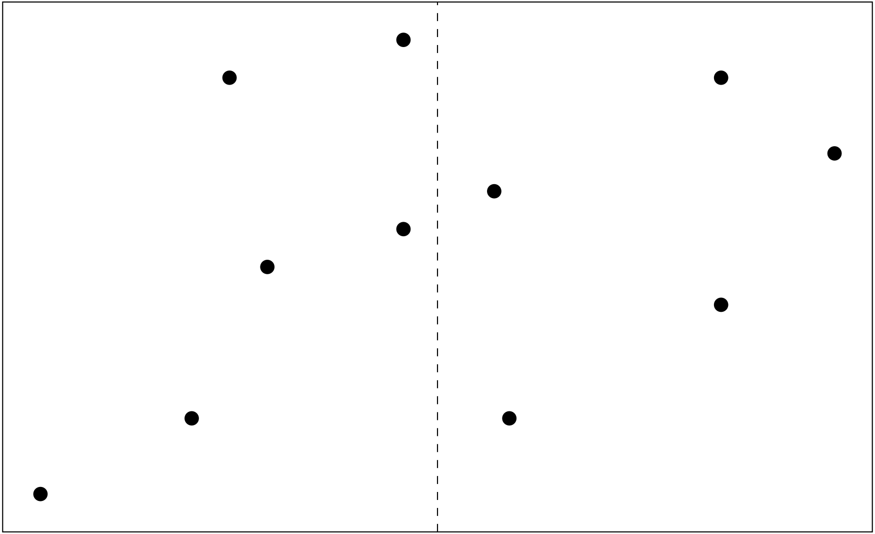
1. Choisir une abscisse x_m séparant les points en 2 sous-ensembles P_1 et P_2 de même taille (à ± 1)
2. Trouver les plus petites distances d_1 et d_2 dans P_1 et P_2
3. Trouver la plus petite distance d_3 parmi les points dans la bande d'abscisse $[x_m - \min(d_1, d_2), x_m + \min(d_1, d_2)]$
4. Renvoyer $\min(d_1, d_2, d_3)$.

Paire de points les plus proches



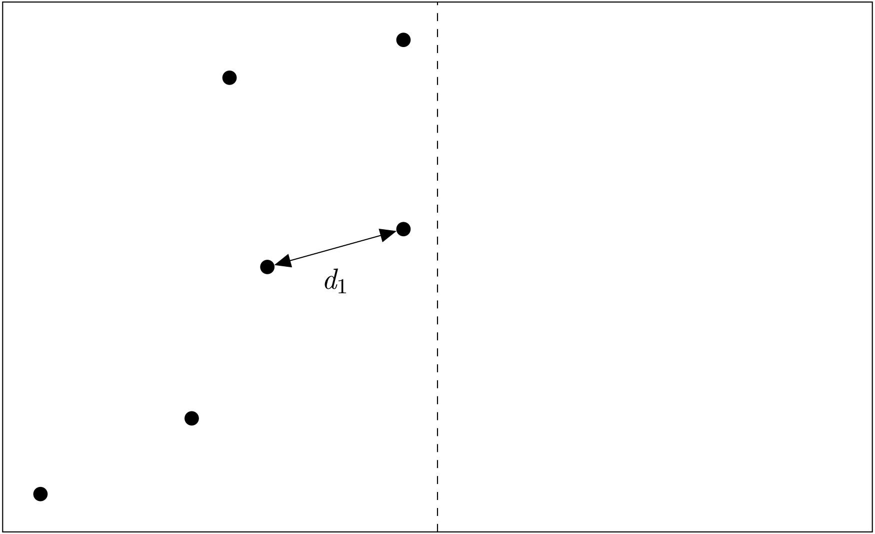
Paire de points les plus proches

Séparer les points en 2 sous-ensembles de même taille (à ± 1) :



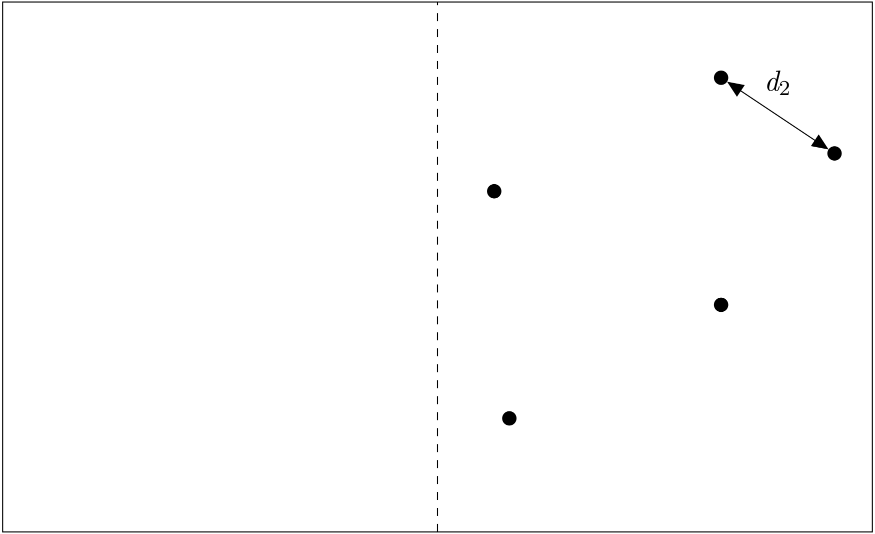
Paire de points les plus proches

Calculer la plus petite distance d_1 dans la 1ère moitié :



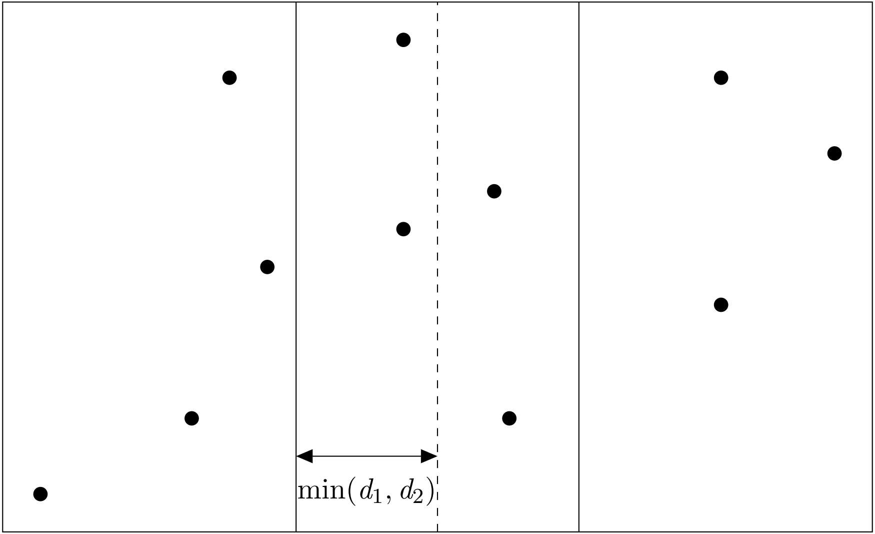
Paire de points les plus proches

Calculer la plus petite distance d_2 dans la 2ème moitié :



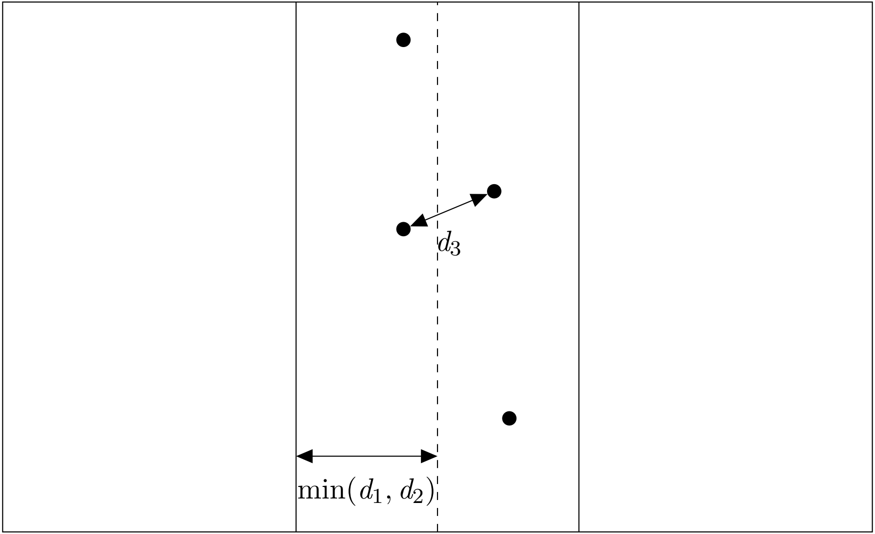
Paire de points les plus proches

Calculer la plus petite distance d_3 dans la bande centrale :



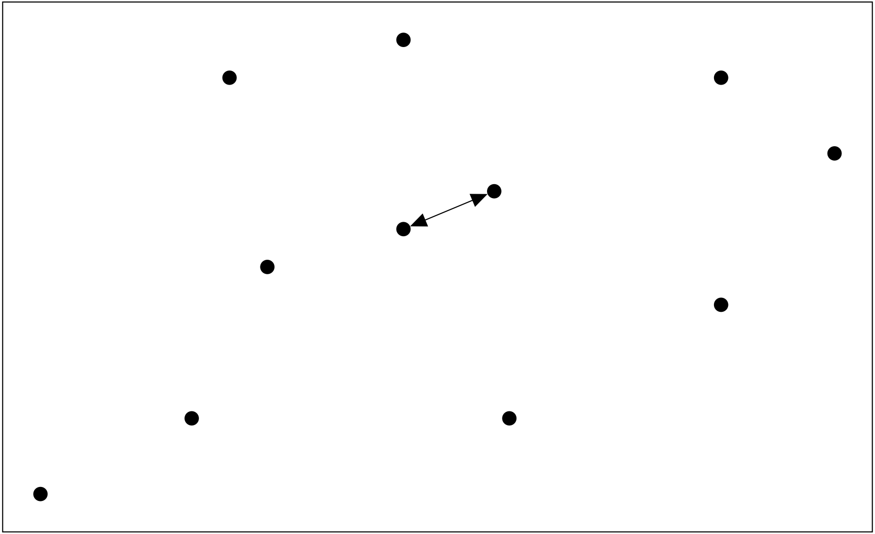
Paire de points les plus proches

Calculer la plus petite distance d_3 dans la bande centrale :



Paire de points les plus proches

$\min(d_1, d_2, d_3)$ est la plus petite distance de l'ensemble des points :



Paire de points les plus proches

On note $d_0 = \min(d_1, d_2)$.

Théorème

Soient $p_1 \in P_1$ et $p_2 = (x_2, y_2) \in P_2$ vérifiant $x_2 > x_m + d_0$. Alors :

$$d(p_1, p_2) > d_0$$

Paire de points les plus proches

On note $d_0 = \min(d_1, d_2)$.

Théorème

Soient $p_1 \in P_1$ et $p_2 = (x_2, y_2) \in P_2$ vérifiant $x_2 > x_m + d_0$. Alors :

$$d(p_1, p_2) > d_0$$

De même si $x_1 < x_m - d_0$.

Corollaire

Pour trouver la plus petite distance entre un point de P_1 et un point de P_2, on peut se ramener à trouver la plus petite distance entre deux points dans la bande centrale.

Paire de points les plus proches

Théorème

On suppose les points triés par ordre croissant d'ordonnée dans un tableau P.
Deux points de la bande centrale situés à une distance $< d_0$ sont séparés par au plus 6 points dans P.

Paire de points les plus proches

Théorème

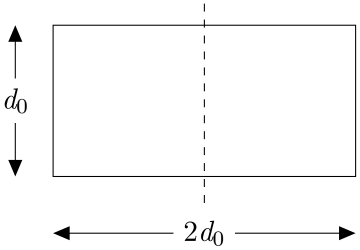
On suppose les points triés par ordre croissant d'ordonnée dans un tableau P.
Deux points de la bande centrale situés à une distance $< d_0$ sont séparés par au plus 6 points dans P.

Preuve : Supposons que $d(P[i], P[j]) < d_0$ avec $i < j$.
Il faut montrer que $j - i < 8$.

Paire de points les plus proches

Preuve : Supposons que $d(P[i], P[j]) < d_0$ avec $i < j$.

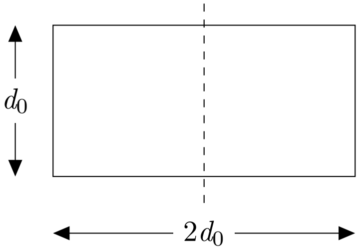
Il faut montrer que $j - i < 8$. Pour cela, notons y_i l'ordonnée de $P[i]$ et considérons le rectangle de sommet inférieur gauche $(x_m - d_0, y_i)$ et de sommet supérieur droit $(x_m + d_0, y_i + d_0)$:



Paire de points les plus proches

<u>Preuve</u> : Supposons que $d(P[i], P[j]) < d_0$ avec $i < j$.

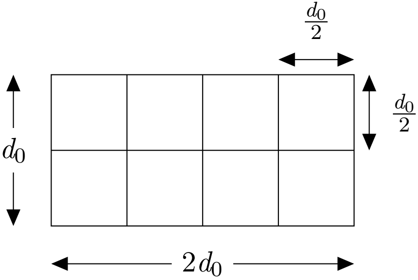
Il faut montrer que $j - i < 8$. Pour cela, notons y_i l'ordonnée de $P[i]$ et considérons le rectangle de sommet inférieur gauche $(x_m - d_0, y_i)$ et de sommet supérieur droit $(x_m + d_0, y_i + d_0)$:



Comme $d(P[i], P[j]) < d_0$, $P[j]$ doit être dans ce rectangle.

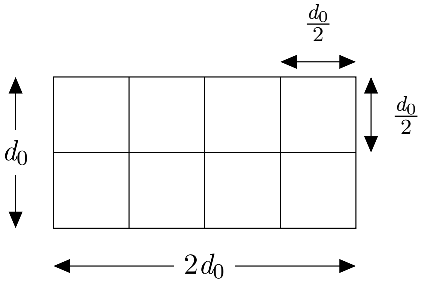
Paire de points les plus proches

Subdivisons ce rectangle en 8 petits rectangles de côté $\frac{d_0}{2}$:



Paire de points les plus proches

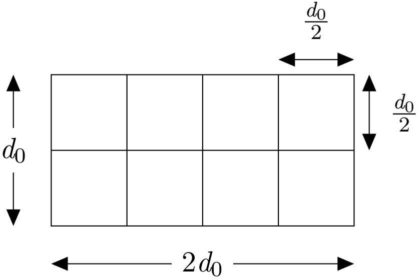
Subdivisons ce rectangle en 8 petits rectangles de côté $\frac{d_0}{2}$:



- La diagonale d'un petit rectangle est $\frac{d_0}{\sqrt{2}} < d_0$

Paire de points les plus proches

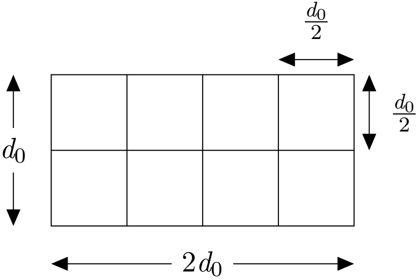
Subdivisons ce rectangle en 8 petits rectangles de côté $\dfrac{d_0}{2}$:



- La diagonale d'un petit rectangle est $\dfrac{d_0}{\sqrt{2}} < d_0$
- 2 points dans le même petit rectangle sont dans le même sous-ensemble (P_1 ou P_2)

Paire de points les plus proches

Subdivisons ce rectangle en 8 petits rectangles de côté $\dfrac{d_0}{2}$:



- La diagonale d'un petit rectangle est $\dfrac{d_0}{\sqrt{2}} < d_0$
- 2 points dans le même petit rectangle sont dans le même sous-ensemble (P_1 ou P_2)
- Il y a donc au plus 1 point par petit rectangle

Paire de points les plus proches

- La diagonale d'un petit rectangle est $\frac{d_0}{\sqrt{2}} < d_0$

Paire de points les plus proches

- La diagonale d'un petit rectangle est $\frac{d_0}{\sqrt{2}} < d_0$
- 2 points dans le même petit rectangle sont dans le même sous-ensemble (P_1 ou P_2)

Paire de points les plus proches

- La diagonale d'un petit rectangle est $\frac{d_0}{\sqrt{2}} < d_0$
- 2 points dans le même petit rectangle sont dans le même sous-ensemble (P_1 ou P_2)
- Il y a donc au plus 1 point par petit rectangle

Paire de points les plus proches

- La diagonale d'un petit rectangle est $\dfrac{d_0}{\sqrt{2}} < d_0$
- 2 points dans le même petit rectangle sont dans le même sous-ensemble (P_1 ou P_2)
- Il y a donc au plus 1 point par petit rectangle
- D'après le principe des tiroirs,
  il y a au plus 8 points dans le grand rectangle

Paire de points les plus proches

- La diagonale d'un petit rectangle est $\dfrac{d_0}{\sqrt{2}} < d_0$
- 2 points dans le même petit rectangle sont dans le même sous-ensemble (P_1 ou P_2)
- Il y a donc au plus 1 point par petit rectangle
- D'après le principe des tiroirs,
  <u>il y a au plus 8 points dans le grand rectangle</u>
- $P[i]$ et $P[j]$ sont dans le grand rectangle, donc ils sont séparés par au plus 6 points dans P

Paire de points les plus proches

```
(* renvoie la plus petite distance dans la bande centrale *)
let closest_strip points =
    let d = ref max_float in
    let n = Array.length points in
    for i = 0 to n - 1 do
        for j = i + 1 to min (n - 1) (i + 7) do
            d := min !d (dist points.(i) points.(j))
        done
    done;
    !d;;
```

Paire de points les plus proches

```
let rec closest points_x points_y =
    let n = Array.length points_x in
    if n <= 3 then closest_brute points_x
    else
        let xm = fst points_x.(n/2) in
        let points_x1, points_x2 = split xm points_x in
        let points_y1, points_y2 = split xm points_y in
        let d1 = closest points_x1 points_y1 in
        let d2 = closest points_x2 points_y2 in
        let d = min d1 d2 in
        min d (closest_strip (select (xm -. d) (xm +. d) points_x)
```

Paire de points les plus proches

Complexité :

Paire de points les plus proches

<u>Complexité</u> : Soit $C(n)$ la complexité de `closest` pour n points.
Supposons que n est une puissance de 2, pour simplifier :

$$\begin{aligned}
C(n) &= 2C(\frac{n}{2}) + \underbrace{\mathsf{O}(n)}_{\text{diviser}} + \underbrace{\mathsf{O}(n)}_{\text{bande centrale}} \\
&= 2C(\frac{n}{2}) + \mathsf{O}(n)
\end{aligned}$$

Paire de points les plus proches

Complexité : Soit $C(n)$ la complexité de `closest` pour n points.
Supposons que n est une puissance de 2, pour simplifier :

$$\begin{aligned} C(n) &= 2C(\frac{n}{2}) + \underbrace{\mathsf{O}(n)}_{\text{diviser}} + \underbrace{\mathsf{O}(n)}_{\text{bande centrale}} \\ &= 2C(\frac{n}{2}) + \mathsf{O}(n) \end{aligned}$$

Même équation que pour le tri fusion ! Donc :

$$\boxed{C(n) = O(n\log(n))}$$