Diviser pour Régner

Alain Camanes

alain.camanes@free.fr

Stanislas

Option Informatique
2021-2022

↪ Idée. Problème sur une structure à n éléments.

1. Résolution du problème sur des structures de *base*.
2. Découpage en problèmes *plus petits* (de taille $n/2$) et *indépendants*.
3. Résoudre ces problèmes *récursivement*.
4. *Reconstruire* la solution du problème de départ.

↪ Notations. Soit x un réel.

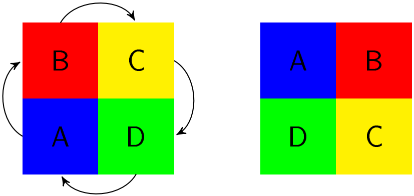
$$\lfloor x \rfloor \leqslant x < \lfloor x \rfloor + 1,$$
$$\lceil x \rceil - 1 < x \leqslant \lceil x \rceil .$$

↪ Propriété.

$$\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil = n.$$

Plan

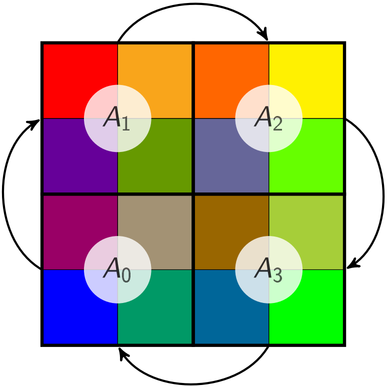
1/4 de tour pour 4 pixels

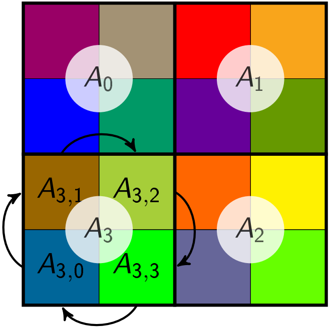


$\hookrightarrow$ Peut être effectué *en place* !

1/4 de tour pour 16 pixels

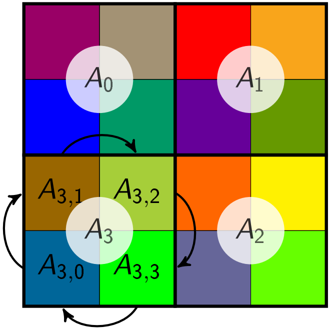
$\hookrightarrow$ Rotation des grands carrés.

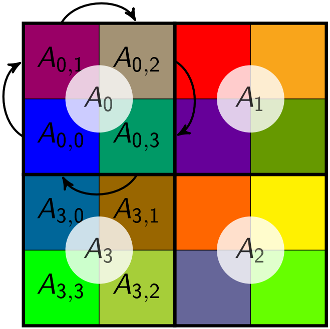




1/4 de tour pour 16 pixels

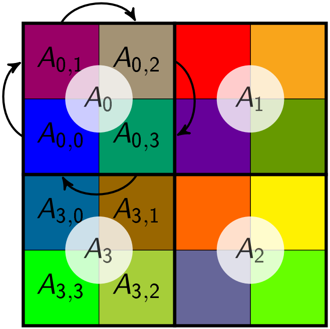
$\hookrightarrow$ Rotation de A_3.

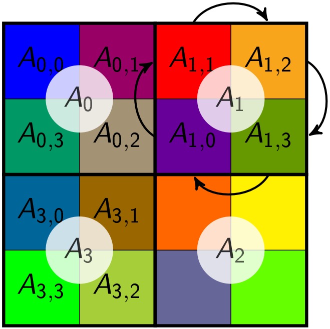




1/4 de tour pour 16 pixels

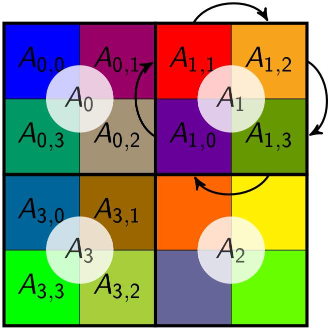
↪ Rotation de A_0.

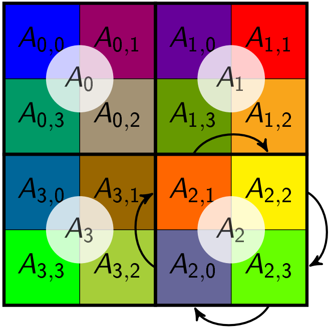




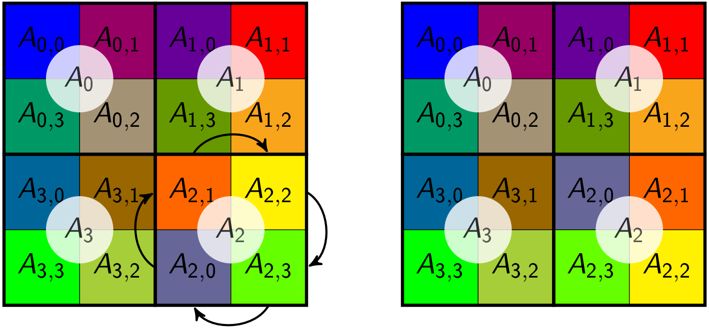
1/4 de tour pour 16 pixels

↪ Rotation de A_1.



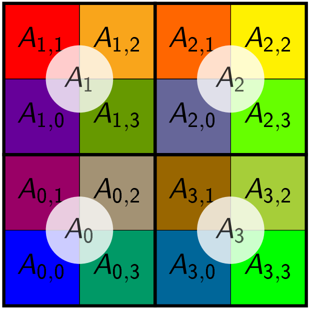


$\hookrightarrow$ Rotation de A_2.

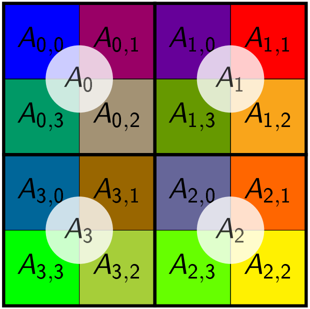


1/4 de tour pour 16 pixels

Avant



Après



1/4 de tour pour 2^{2^n} pixels ?

↪ Algorithme.

1. Si $n = 1$, permutation des 4 pixels.
2. Si $n \geqslant 2$,
    1. permutation des 4 images de taille $2^{2^{n-1}}$,
    2. récursivement : permutation de chacune de ces 4 images.

↪ Algorithme *en place* !

↪ Admirons !

Plan

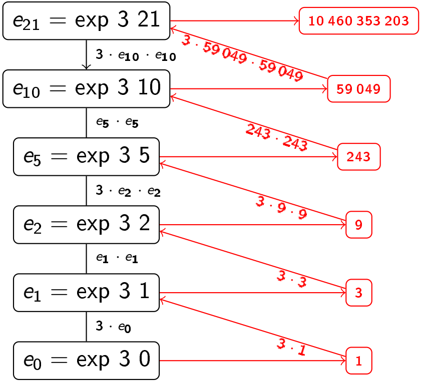
Exponentiation rapide (Exponentiation by squaring)

$$
\begin{aligned}
x^n &= (x \cdot x)^{n/2}, \text{ si } n \text{ est pair,} \\
&= x \cdot (x \cdot x)^{n/2}, \text{ si } n \text{ est impair.}
\end{aligned}
$$

```
let rec expo_rapide x n =
  match n with
  | 0 -> 1
  | _ -> let y = expo_rapide x (n/2) in
         if n mod 2 = 0 then y * y
         else x * y * y
```

Exponentiation rapide - Exemple

```
let rec expo_rapide x n =
  match n with
  | 0 -> 1
  | _ -> let y = expo_rapide x (n/2) in
         if n mod 2 = 0 then y * y else x * y * y
```



Théorème

L'exponentiation rapide nécessite $\Theta(\log_2 n)$ multiplications.

$\hookrightarrow$ Décomposition binaire. $n = \overline{b_{t-1} \cdots b_0}^2$.

$$2^{t-1} \leqslant n \leqslant \sum_{k=0}^{t-1} 2^k = 2^t$$

$\hookrightarrow$ Nombre de multiplications.

Si n est *pair*, alors $b_0 = 0$ et 1 multiplication.

Si n est *impair*, alors $b_0 = 1$ et 2 multiplications.

$$C_n = C_{\overline{b_{t-1} \cdots b_0}^2} = C_{\overline{b_{t-1} \cdots b_1}^2} + (1 + b_0) = \sum_{i=0}^{t-1} (1 + b_i).$$

$$t \leqslant C_n \leqslant 2t.$$

$\hookrightarrow$ *Calcul* de 3^{15}. $15 = \overline{1111}^2$ soit 7 multiplications.
Cependant,

- $x_1 = 3 \cdot 3 = 3^2$.
- $x_2 = 3 \cdot x_1 = 3^3$.
- $x_3 = x_2 \cdot x_2 = 3^6$.
- $x_4 = x_3 \cdot x_3 = 3^{12}$.
- $x_5 = x_4 \cdot x_2 = 3^{15}$.

soit 5 multiplications.

$\hookrightarrow$ Optimalité. $x_0, x_1, \dots, x_k$: valeurs successives calculées.
Outils de calculs : multiplication + valeurs précédentes $\rightarrow x_i = x^{p_i}$.

- $p_0 = 1$.
- $p_1 = 2$.
- $p_k = n$.
- $p_i = p_j + p_\ell$.

Récurrence forte :

$$p_i \leqslant 2^i.$$

Deux autres algorithmes

↪ Récursif II.

```
let rec expo_rec x n =
  match n with
  | 0 -> 1
  | _ -> if n mod 2 = 0 then expo_rec (x*x) (n/2)
         else x * (expo_rec (x*x) (n/2))
```

↪ Itératif. *Invariant* $e \cdot y^m = x^n$.

```
let exp_iter x n = if n = 0 then 1 else
    begin
      let e = ref 1 and y = ref x and m = ref n in
      while !m > 0 do
        if !m mod 2 = 0 then (y:=!y* !y; m:=!m/2)
        else (e:=!e* !y; y:=!y* !y; m:=!m/2)
      done;
      !e
    end
```

↪ Définition.

$$F_0 = 0,\ F_1 = 1,$$
$$F_{n+2} = F_{n+1} + F_n.$$

Théorème

Le calcul du n-ème nombre de Fibonacci nécessite $\Theta(\log_2 n)$ multiplications matricielles.

↪ Propriété.

$$\begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} \cdot \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

Plan

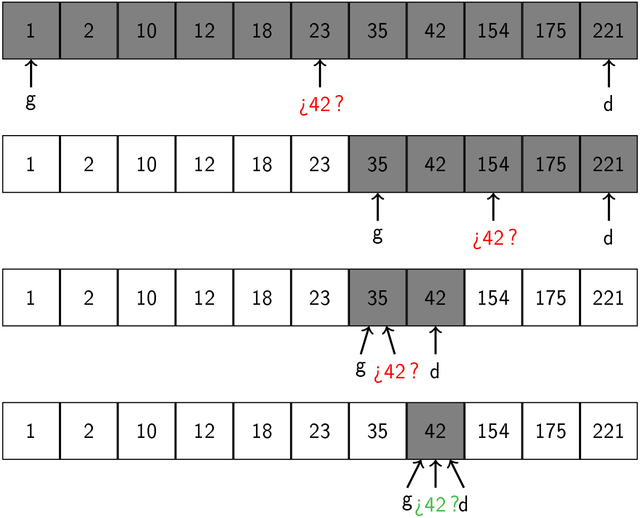
Recherche dichotomique (Binary Search)

t trié croissant ; recherche de l'élément x. On pose $m = \left\lfloor \frac{n}{2} \right\rfloor$.

- Si $x = t_m$, terminé.
- Si $x < t_m$, recherche de x dans $\{t_0, \ldots, t_{m-1}\}$.
- Si $x > t_m$, recherche de x dans $\{t_{m+1}, \ldots, t_{n-1}\}$.

```
let binary_search t el =
  let rec aux g d =
    if d < g then false else
      let c = (g + d) / 2 in
        if t.(c) = el then true
        else if t.(c) < el then aux (c+1) d
        else aux g (c-1)
  in aux 0 (Array.length t -1)
```

Recherche dichotomique - Exemple



Théorème

La recherche dichotomique nécessite au plus $\Theta(\log_2 n)$ comparaisons.

$$C_n = 1 + \max\left\{C_{\lfloor \frac{n}{2} \rfloor},\ C_{\lceil \frac{n}{2} \rceil}\right\}.$$

$\hookrightarrow$ Puissances de 2. $C_{2^k} = 1 + C_{2^{k-1}}$.

$$C_{2^k} = C_1 + k = C_1 + \log_2(n).$$

$\hookrightarrow$ Croissance. (Récurrence forte).

$\hookrightarrow$ Cas général. $2^k \leqslant n < 2^{k+1}$, $k = \lfloor \log_2(n) \rfloor$.

$$\begin{array}{ccccc} 2^k & \leqslant & n & < & 2^{k+1} \\ C_{2^k} & \leqslant & C_n & \leqslant & C_{2^{k+1}} \\ C_1 + \lfloor \log_2(n) \rfloor & \leqslant & C_n & \leqslant & C_1 + 1 + \lfloor \log_2(n) \rfloor \end{array}$$

Plan

Théorème

La multiplication de deux entiers de n bits s'effectue en $\Theta(n^{\log_2 3})$ opérations.

$\hookrightarrow$ Décomposition. En général $m = \left\lfloor \frac{n}{2} \right\rfloor$.

$$x = x_1 b^m + x_0,\ y = y_1 b^m + y_0.$$

$\hookrightarrow$ Naïvement : 4 multiplications !

$$xy = \underbrace{(x_1 \times y_1)}_{z_2} b^{2m} + \underbrace{(x_1 \times y_0 + x_0 \times y_1)}_{z_1} b^m + \underbrace{(x_0 \times y_0)}_{z_0}.$$

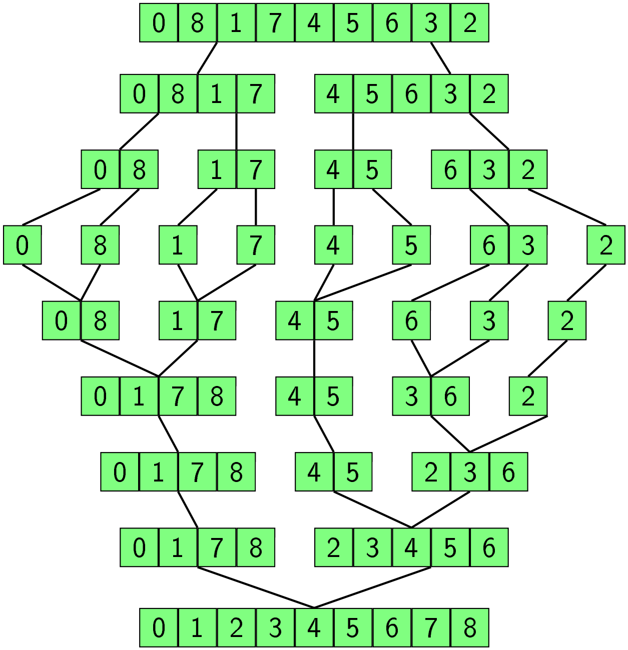
$\hookrightarrow$ Astucieusement : 3 multiplications !

$$z_1 = (x_1 + x_0) \times (y_1 + y_0) - z_2 - z_0.$$

$\hookrightarrow$ Récurrence.

$$T_n = T_{\left\lfloor \frac{n}{2} \right\rfloor} + 2 \cdot T_{\left\lceil \frac{n}{2} \right\rceil} + cn + d.$$

Plan


0 8 1 7 4 5 6 3 2
0 8 1 7
4 5 6 3 2
0 8
1 7
4 5
6 3 2
0
8
1
7
4
5
6 3
2
0 8
1 7
4 5
6
3
2
0 1 7 8
4 5
3 6
2
0 1 7 8
4 5
2 3 6
0 1 7 8
2 3 4 5 6
0 1 2 3 4 5 6 7 8

Copier les éléments de `t.(j..)` dans le tableau `u.(i..)`.

```
blit : 'a array -> int -> 'a array -> int -> int ->
unit
```

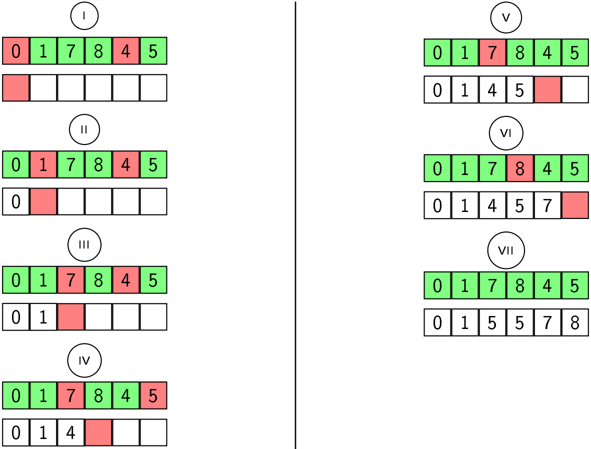
L'appel `Array.blit t1 o1 t2 o2 len`

- copie `len` éléments
- du tableau `t1`, en commençant par celui d'indice `o1`,
- dans le tableau `t2`, en commençant à l'élément d'incide `o2`.

Fusion

Théorème

Deux tableaux triés sont fusionnés avec un nombre de comparaisons linéaire.



Le code

```
let fusion t i1 f1 i2 f2 =
  let n = f2-i1+1 in
  let rec aux t i1 i2 u iu =
    if i1 = f1+1 then
      (Array.blit t i2 u iu (n-iu);
       Array.blit u 0 t  (f2-n+1) n;)
    else
      (if i2 = f2+1 then
         (Array.blit t i1 u iu (n-iu);
          Array.blit u 0 t (f2-n+1) n;)
       else
         (if t.(i1) < t.(i2)
          then (u.(iu) <- t.(i1);
                aux t (i1+1) i2 u (iu+1);)
          else
            (u.(iu) <- t.(i2);
             aux t i1 (i2+1) u (iu+1);)))
    in aux t i1 i2 (Array.make n 0) 0;;
```

Tri fusion (Merge sort)

```
let tri_fusion t =
  let rec aux a b =
    if a < b then let c = (a + b) / 2 in
                  (aux a c ;
                  aux (c+1) b;
                  fusion t a c (c+1) b;)
  in aux 0 (Array.length t - 1);;
```

Théorème

Le tri fusion nécessite $\Theta(n \log_2 n)$ comparaisons. Il nécessite une complexité spatiale linéaire et est stable.

$\hookrightarrow$ Fusion lourde avec tableaux - Découpage lourd avec listes...

$$C_n = C_{\lfloor \frac{n}{2} \rfloor} + C_{\lceil \frac{n}{2} \rceil} + n.$$

$\hookrightarrow$ Puissance de 2. Si $n = 2^p$.

$$\begin{aligned} C_{2^p} &= 2C_{2^{p-1}} + 2^p \\ \frac{C_{2^p}}{2^p} &= \frac{C_{2^{p-1}}}{2^{p-1}} + 1 \\ C_{2^p} &= C_1 2^p + \log_2(2^p) \cdot 2^p. \end{aligned}$$

$\hookrightarrow$ Croissance. (Récurrence forte)

$\hookrightarrow$ Cas général. $n \in [2^p, 2^{p+1}[$, $p = \lfloor \log_2 n \rfloor$.

$$\begin{array}{ccccc} C_{2^p} & \leqslant & C_n & \leqslant & C_{2^{p+1}} \\ C_1 2^p + p2^p & \leqslant & C_n & \leqslant & C_1 2^{p+1} + (p+1)2^{p+1} \end{array}$$

Tris - Résumé

	Meilleur	Pire	Spatiale	Stabilité
Bulle	$\Theta(n)$	$\Theta(n^2)$	$\Theta(1)$	✓
Sélection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	✓
Insertion	$\Theta(n)$	$\Theta(n^2)$	$\Theta(1)$	✓
Fusion	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	✓

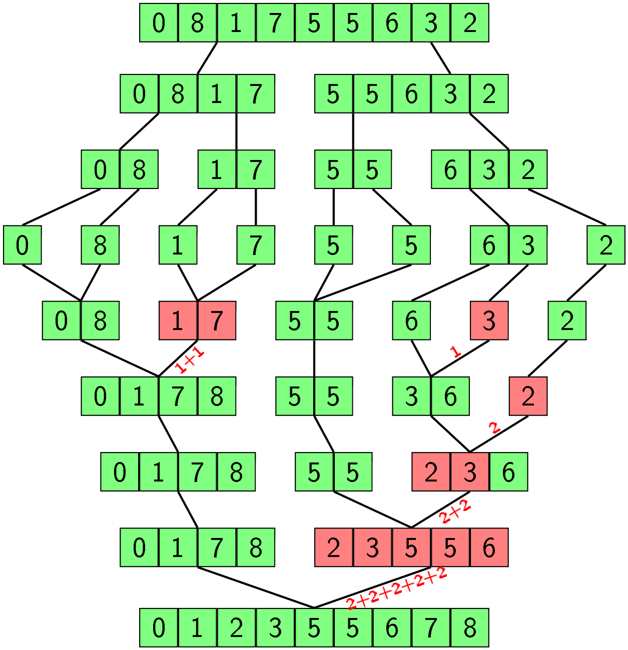
$\hookrightarrow$ Hypothèses. t : tableau de taille n.
t : entrées *distinctes*.
$t \;:\; [\![0, n-1]\!] \rightarrow [\![0, n-1]\!]$ *permutation* (bijection).

$\hookrightarrow$ Nombre d'inversions.

$$I(t) = \sharp \left\{ 0 \leqslant i < j \leqslant n-1 \;;\; t.(i) > t.(j) \right\}.$$

$\hookrightarrow$ Naïf.

$$\Theta(n^2)$$


0 8 1 7 5 5 6 3 2
0 8 1 7
5 5 6 3 2
0 8
1 7
5 5
6 3 2
0
8
1
7
5
5
6 3
2
0 8
1 7
5 5
6
3
2
1+1
1
0 1 7 8
5 5
3 6
2
2
0 1 7 8
5 5
2 3 6
2+2
0 1 7 8
2 3 5 5 6
2+2+2+2+2
0 1 2 3 5 5 6 7 8

Inversions - Complexité

Théorème

L'algorithme du tri fusion permet de déterminer le nombre d'inversions d'une permutation en $\Theta(n \log_2 n)$ comparaisons.

Fusion de deux tableaux triés $t = t1 \cup t2$.

- $t_1[0 \cdots i_0]$ et $t2[0 \cdots j_0]$ *déjà fusionnés*.
- Si $t_2[j_0 + 1] < t_1[i_0 + 1]$, alors $t_1[i_0 + 1 \cdots \ell_1]$ et $t_2[j_0]$ sont *inversés* $\rightarrow \ell_1 - i_0 + 1$ inversions.

Deux compteurs `left` et `inversions` initialisés à 0, puis

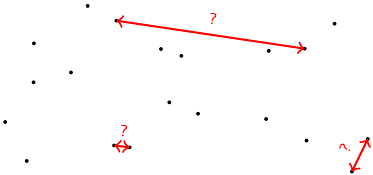
- dès qu'on stocke un élément de la liste de gauche, on *incrémente* `left`.
- dès qu'on stocke un élément de la liste de droite, on *incrémente* `inversions` de `(longueur t1) - left`.

Plan

Points les plus proches (Closest pair)

$\hookrightarrow$ Données. n *points du plan* $((x_0, y_0), \ldots, (x_{n-1}, y_{n-1}))$.

$\hookrightarrow$ Problème. *Paire* de points les plus proches.

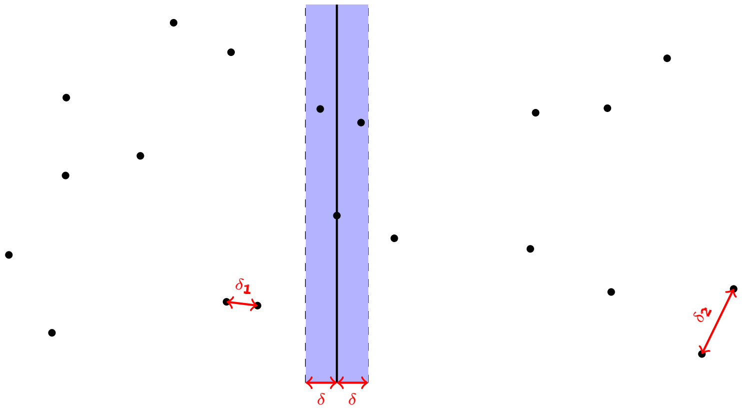


$\hookrightarrow$ Dimension 1. *Tri* puis étude des points *successifs*.

$$\Theta(n \log_2 n).$$

$\hookrightarrow$ Dimension 2 naïf. $\Theta(n^2)$.

Points les plus proches : Diviser

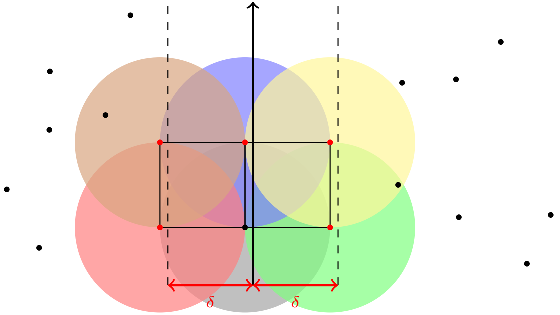


Points les plus proches : Régner !

Combien de points à distance au plus δ et dans l'autre bande ?

- Rectangle de dimension $2\delta \times \delta$.
- Au plus 6 points.

Comparaison aux 5 suivants par ordonnées croissantes.



Algorithme

$\hookrightarrow$ Donnéee. Une liste de points ℓ.

$\hookrightarrow$ Préliminaires. Construire deux listes triées ℓ_x et ℓ_y.

$\hookrightarrow$ Diviser.

- On découpe ℓ_x par rapport à x_{median} en ℓ_{x_1} et ℓ_{x_2}.
- Calcul récursif des distances minimales δ_1, δ_2.

$$\delta = \min\{\delta_1, \delta_2\}.$$

$\hookrightarrow$ Recoller.

- Deux bandes de longueur δ autour de $x_{median} \to \widetilde{\ell}_y$.
- Pour chaque point, distance aux 5 suivants.
- Conclusion.

Algorithme

$\hookrightarrow$ Donnéee. Une liste de points ℓ.

$\hookrightarrow$ Préliminaires. Construire deux listes triées ℓ_x et ℓ_y.

$$\Theta(n \log_2 n)$$

$\hookrightarrow$ Diviser.

- On découpe ℓ_x par rapport à x_{median} en ℓ_{x_1} et ℓ_{x_2}. $\Theta(n)$
- Calcul récursif des distances minimales δ_1, δ_2.

$$\delta = \min\{\delta_1, \delta_2\}.$$

$\hookrightarrow$ Recoller.

- Deux bandes de longueur δ autour de $x_{median} \to \widetilde{\ell}_y$. $\Theta(n)$
- Pour chaque point, distance aux 5 suivants. $\Theta(n)$
- Conclusion.