M1 Info - Graphes et programmation dynamique

Cours 8 - Plus proche paire de points

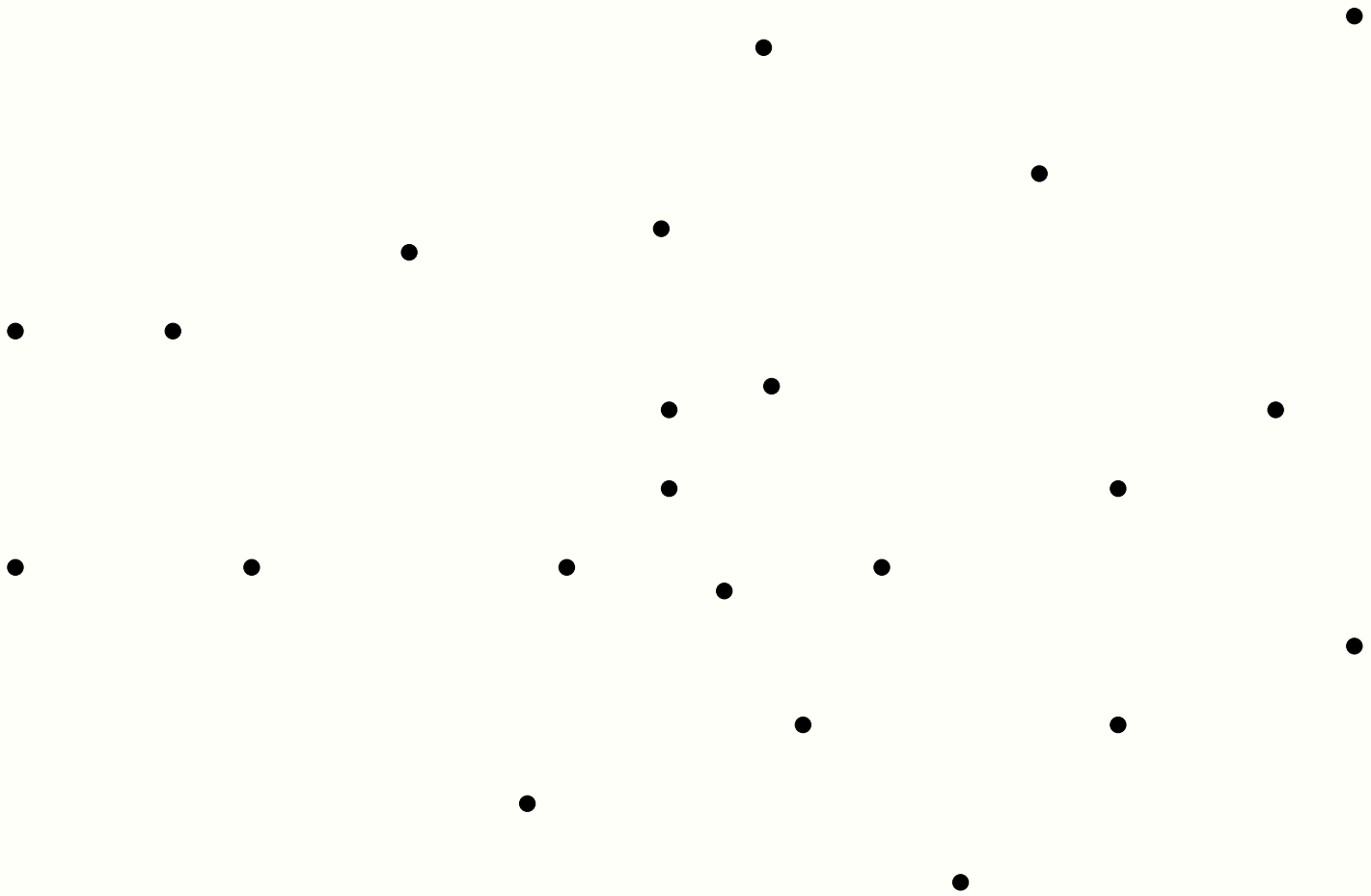
Diviser pour regner

Semestre Automne 2022-2023 - Université Côte D'azur

Christophe Crespelle
christophe.crespelle@univ-cotedazur.fr

Le probleme : plus proche paire de points

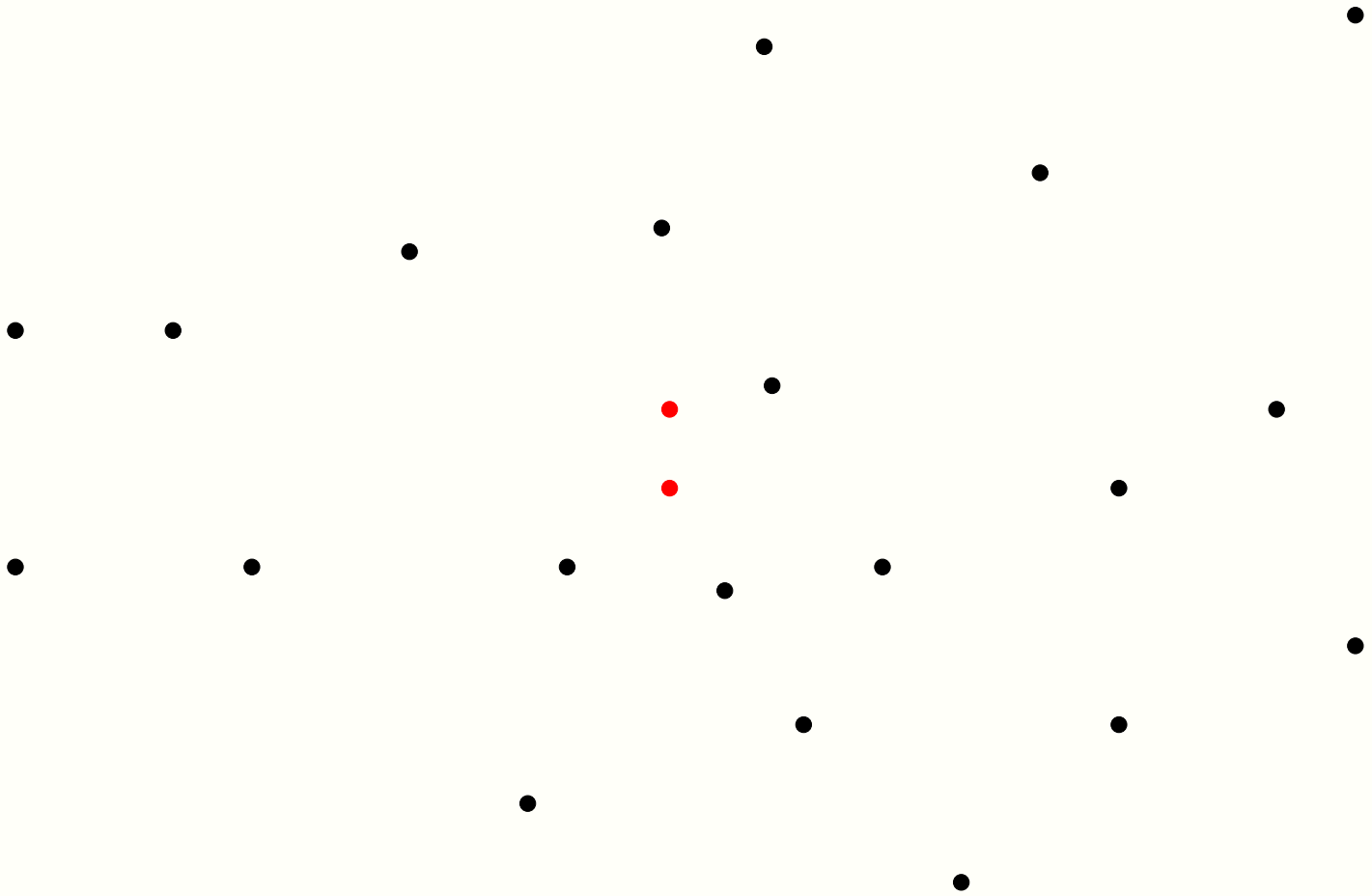
Donnee : Un ensemble de n points dans le plan



Resultat : Une paire de points a distance minimum.

Le probleme : plus proche paire de points

Donnee : Un ensemble de n points dans le plan



Resultat : Une paire de points a distance minimum.

Exemple d'application

- Probleme de base en cartographie numerique
  - Transport maritime
  - Transport aerien

Exemple d'application

- Probleme de base en cartographie numerique
  - Transport maritime
  - Transport aerien
- Donnees classees selon deux criteres
  (ou plus $\Longrightarrow$ se generalise en dimension d)
  - Trouver les deux donnees les plus similaires
  - Mettre en correspondance, detecter des alias, resolution de separation, etc.

Algorithme brute force

- Parcourir toutes les paires de points

Algorithme brute force

- Parcourir toutes les paires de points
- Garder celle de distance minimum

Algorithme brute force

- Parcourir toutes les paires de points
- Garder celle de distance minimum
- Complexite : $O(n^2)$

Algorithme brute force

- Parcourir toutes les paires de points
- Garder celle de distance minimum
- Complexite : $O(n^2)$

Peut on faire mieux ? ? ?

Algorithme brute force

- Parcourir toutes les paires de points
- Garder celle de distance minimum
- Complexite : $O(n^2)$

Peut on faire mieux ? ? ?

$\Longrightarrow$ Algo en $O(n \log n)$ avec "diviser pour regner" !

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)
3. Recoller les solutions sur les morceaux pour obtenir la solution globale

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)
3. Recoller les solutions sur les morceaux pour obtenir la solution globale

Remarques :

- On peut toujours faire 1. et 2.

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)
3. Recoller les solutions sur les morceaux pour obtenir la solution globale

Remarques :

- On peut toujours faire 1. et 2.
- La difficulte c'est recoller

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)
3. Recoller les solutions sur les morceaux pour obtenir la solution globale

Remarques :

- On peut toujours faire 1. et 2.
- La difficulte c'est recoller
  - Est-ce plus facile que resoudre le probleme initial ?

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)
3. Recoller les solutions sur les morceaux pour obtenir la solution globale

Remarques :

- On peut toujours faire 1. et 2.
- La difficulte c'est recoller
  - Est-ce plus facile que resoudre le probleme initial ?
  - la complexite du recollage va donner la complexite totale

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)
3. Recoller les solutions sur les morceaux pour obtenir la solution globale

Remarques :

- On peut toujours faire 1. et 2.
- La difficulte c'est recoller
  - Est-ce plus facile que resoudre le probleme initial ?
  - la complexite du recollage va donner la complexite totale
    - Et on va pas avoir a la calculer ;)

Diviser pour regner

1. Casser l'instance donnee en plusieurs "morceaux"
2. Resoudre le probleme sur chaque morceau (appel recursif)
3. Recoller les solutions sur les morceaux pour obtenir la solution globale

Remarques :

- On peut toujours faire 1. et 2.
- La difficulte c'est recoller
  - Est-ce plus facile que resoudre le probleme initial ?
  - la complexite du recollage va donner la complexite totale
    - Et on va pas avoir a la calculer ;)
    - grace au **Master theorem**

Avant de commencer : cas plus simple en dimension 1

- Les points sont sur une ligne

Avant de commencer : cas plus simple en dimension 1

- Les points sont sur une ligne

- Trier les points par coordonnee croissante

Avant de commencer : cas plus simple en dimension 1

- Les points sont sur une ligne

- Trier les points par coordonnee croissante

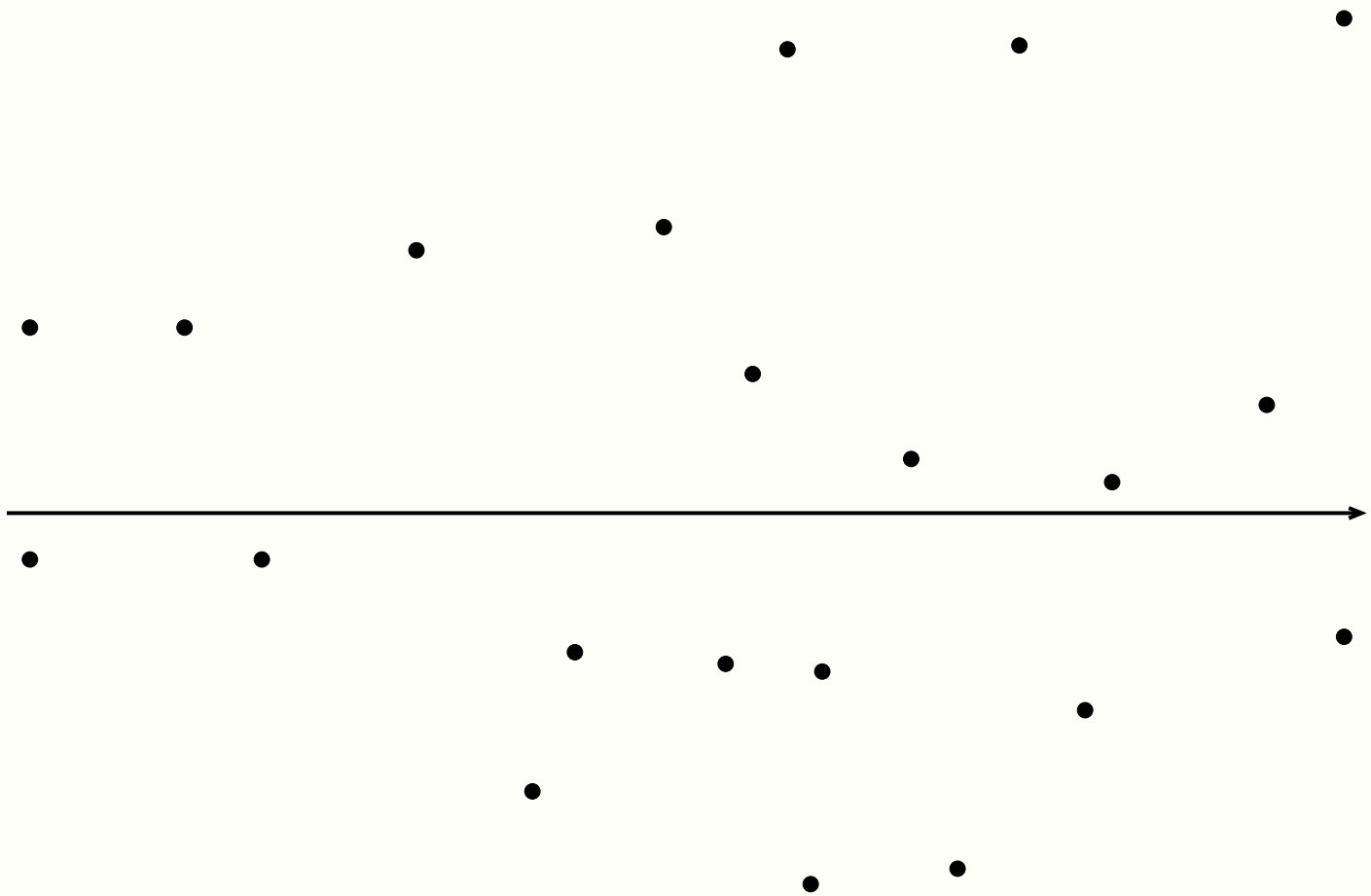
- Les parcourir en verifiant la distance avec le suivant

Avant de commencer : cas plus simple en dimension 1

- Les points sont sur une ligne

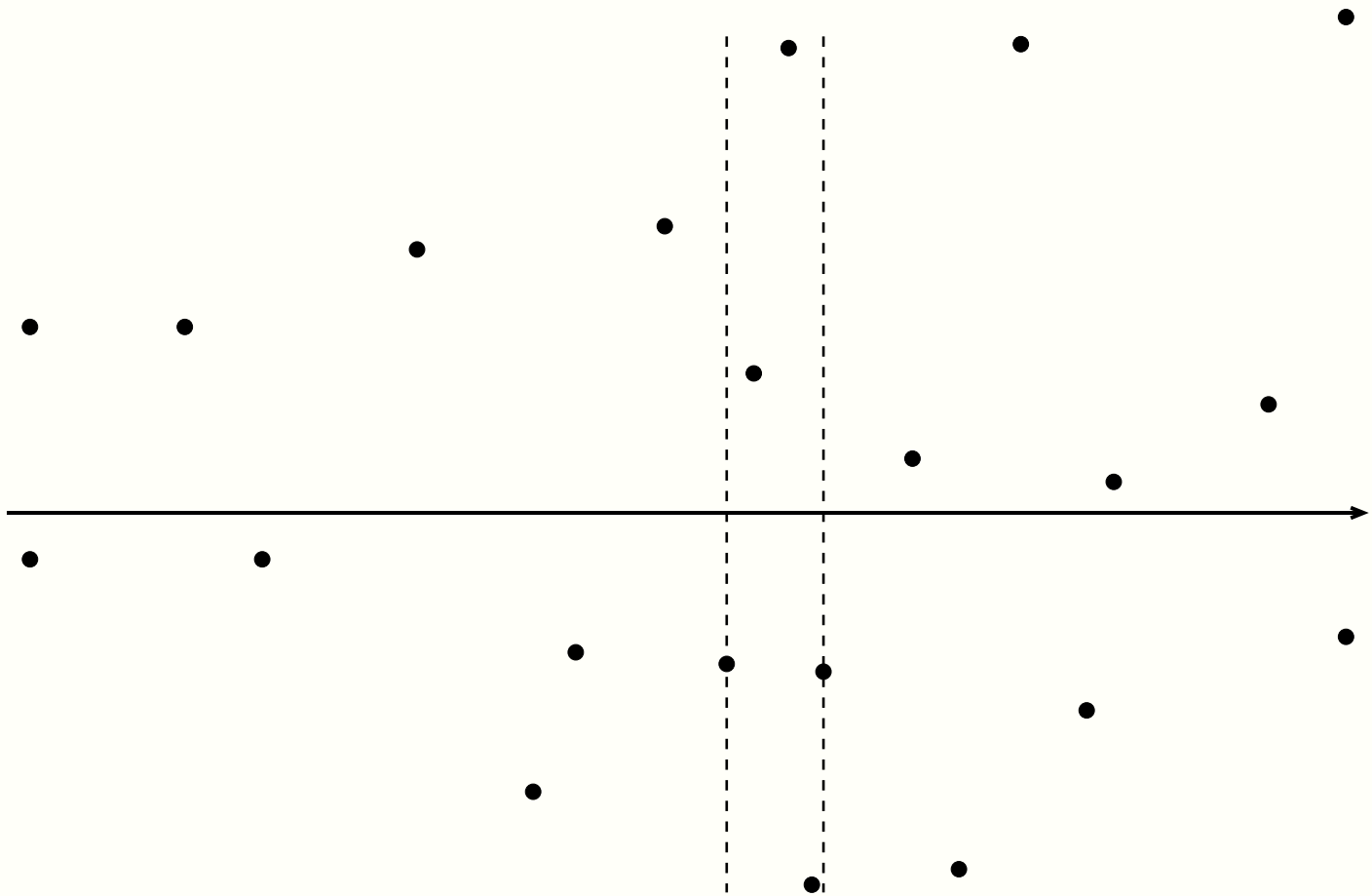
- Trier les points par coordonnee croissante
- Les parcourir en verifiant la distance avec le suivant
- complexite $O(n \log n)$

Pourquoi ca ne marche pas en dimension 2 ?



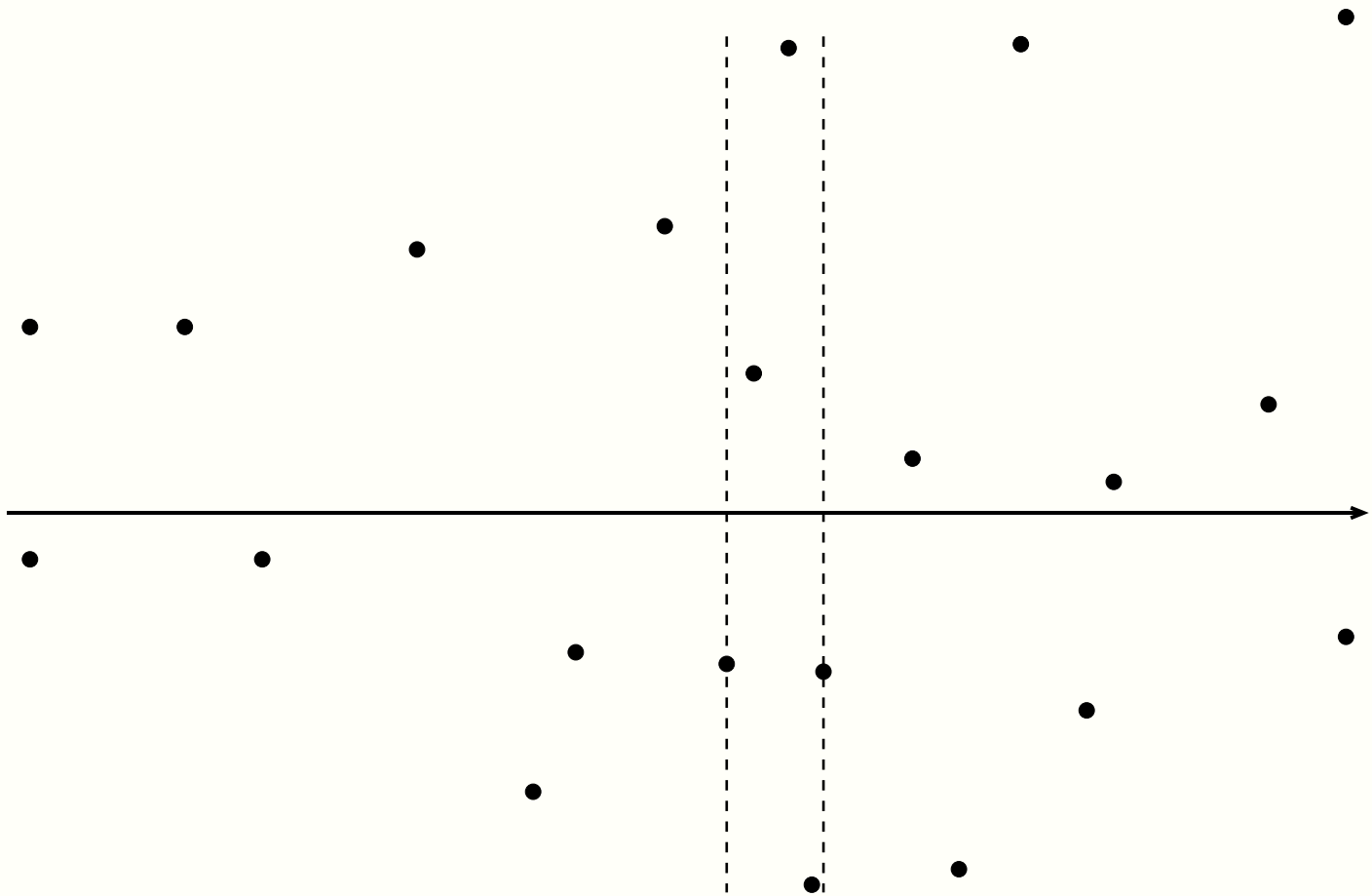
- Scan selon une dimension

Pourquoi ca ne marche pas en dimension 2 ?



- Scan selon une dimension
  - Probleme : des points proches selon une dimension peuvent etre tres eloignes sur la deuxieme...

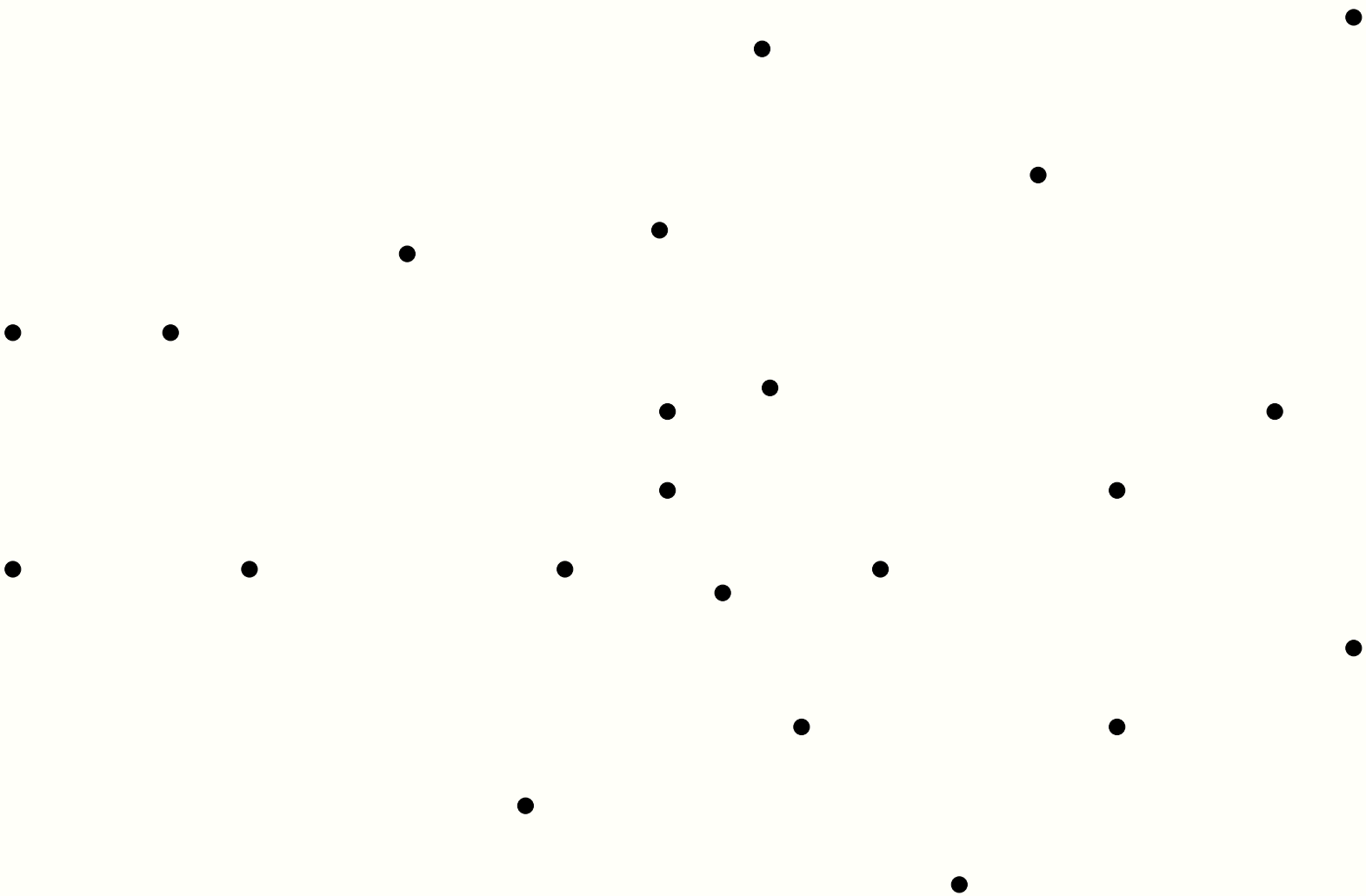
Pourquoi ca ne marche pas en dimension 2 ?



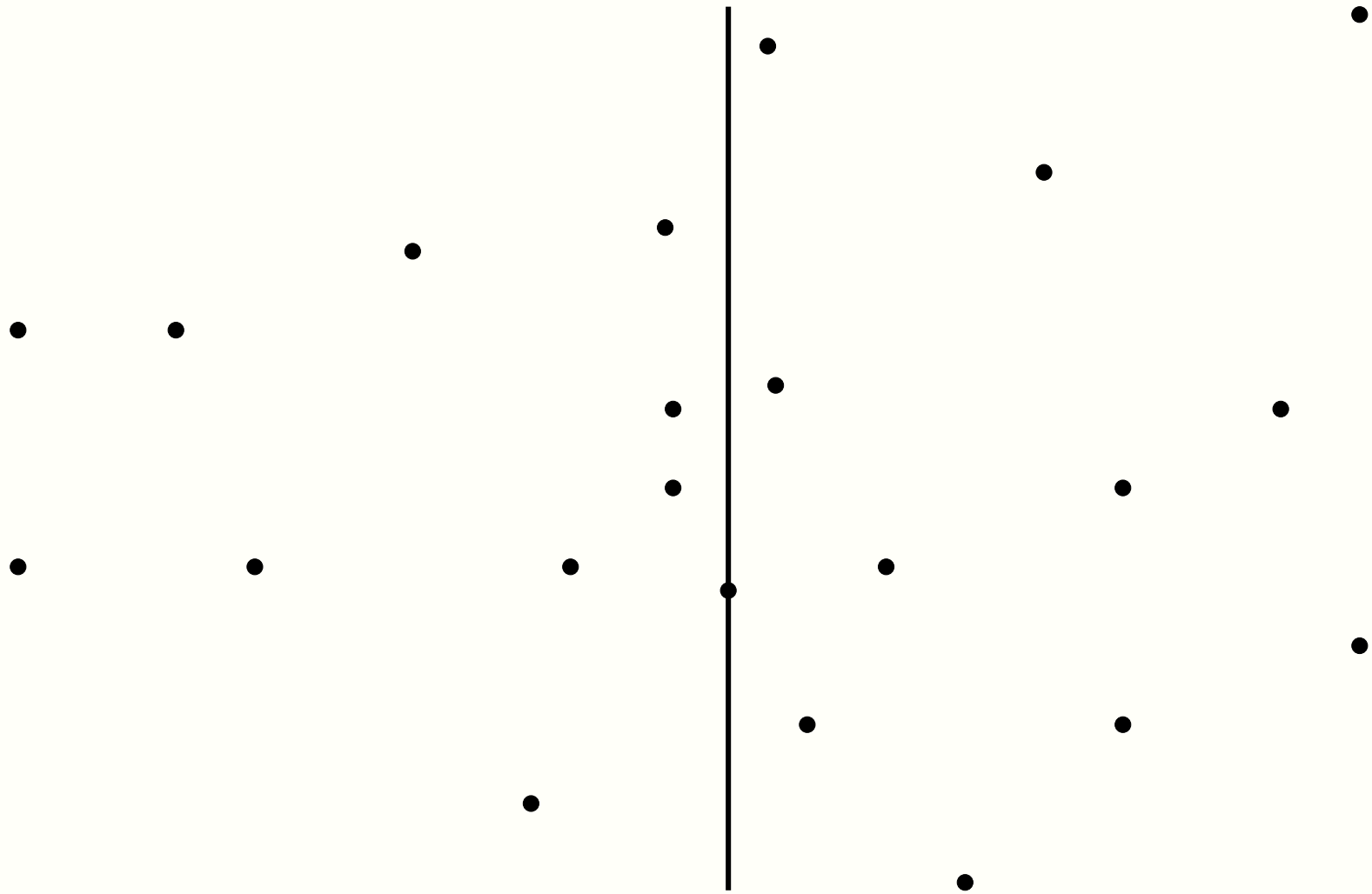
- Scan selon une dimension
  - Probleme : des points proches selon une dimension peuvent etre tres eloignes sur la deuxieme...

$\Longrightarrow$ quand meme $O(n \log n)$ grace a "diviser pour regner" !

Diviser pour regner sur *Plus proche paire*

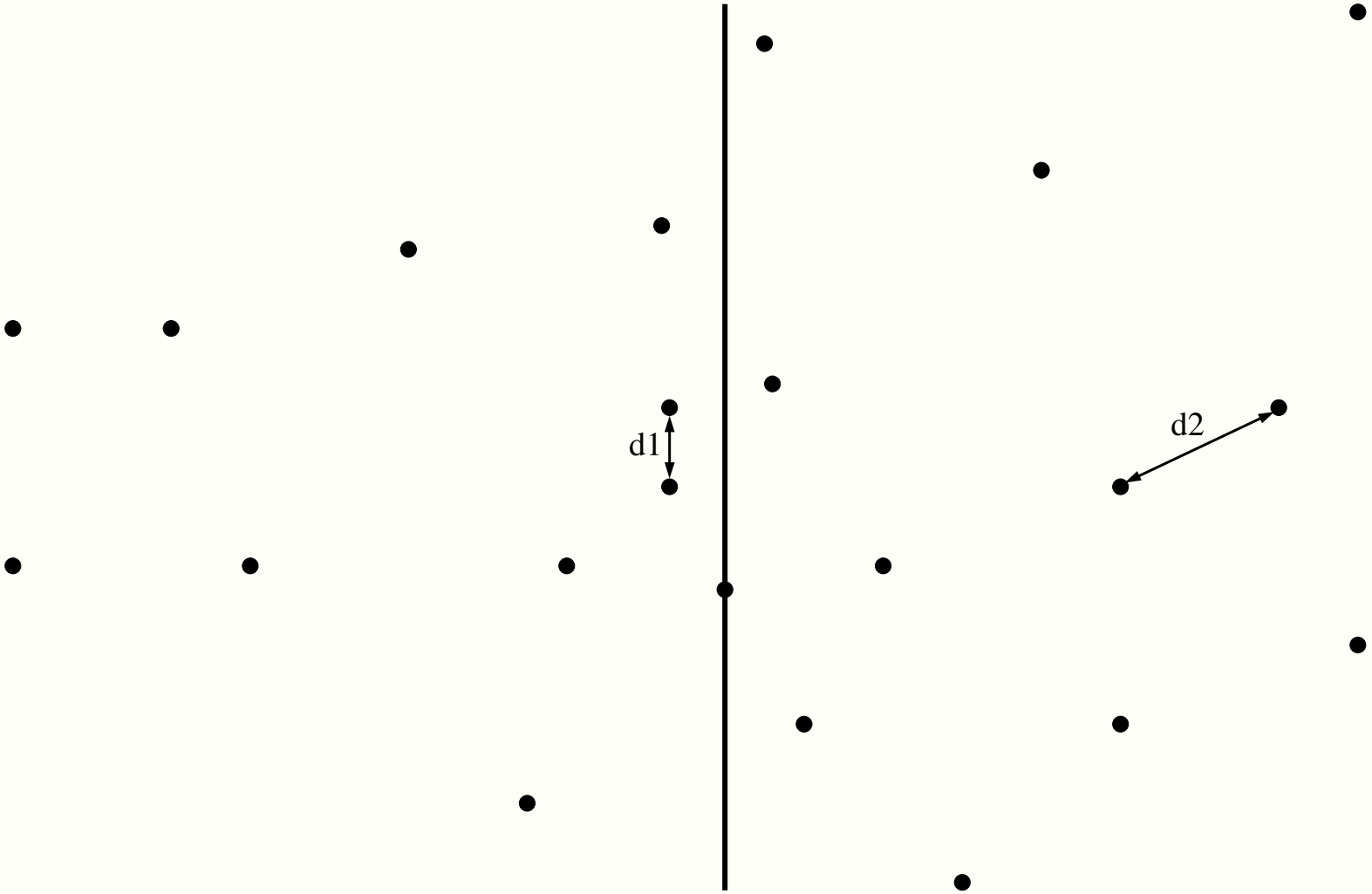


Diviser pour regner sur *Plus proche paire*



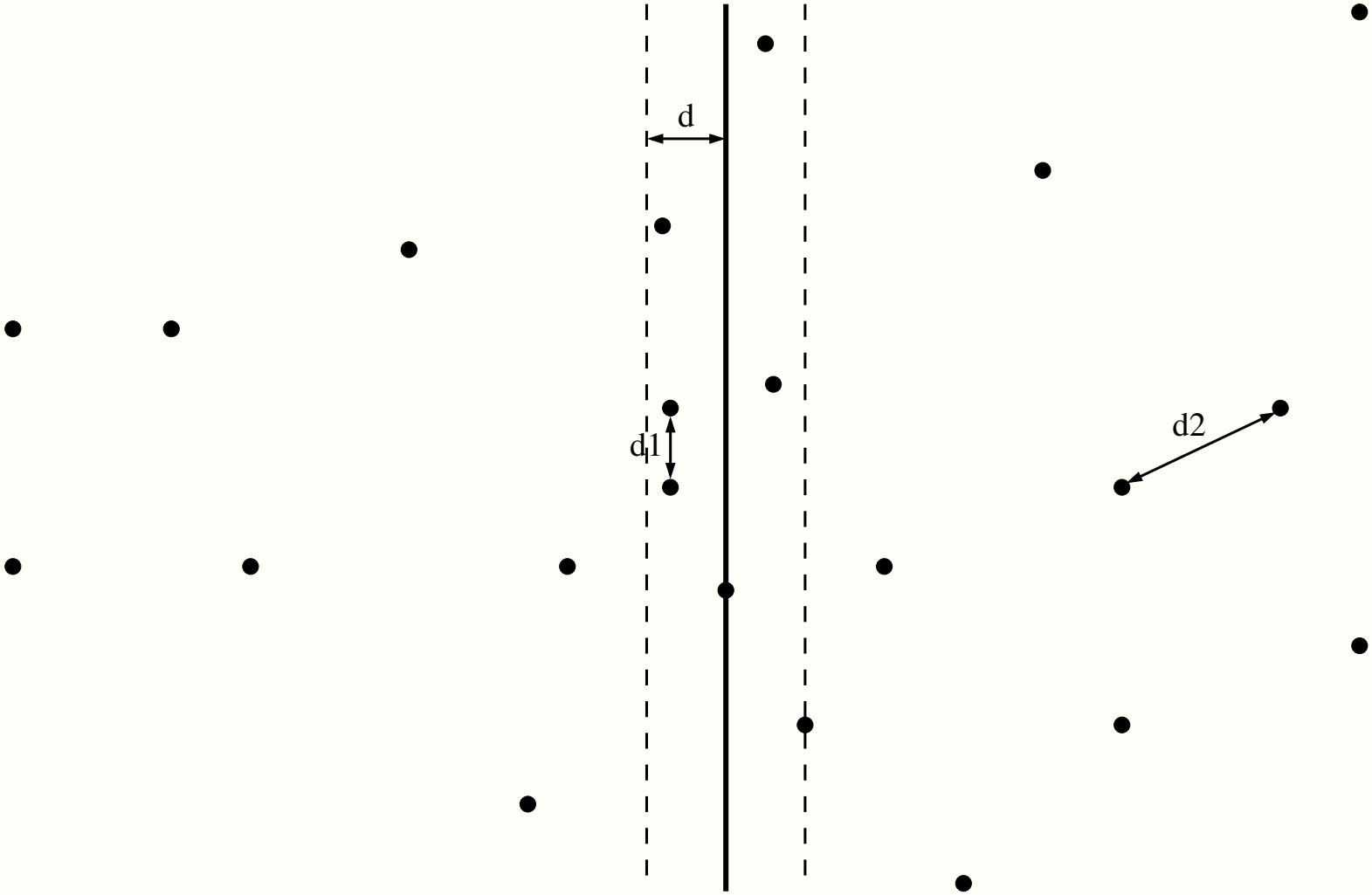
- Diviser en deux selon l'axe des x

Diviser pour regner sur *Plus proche paire*



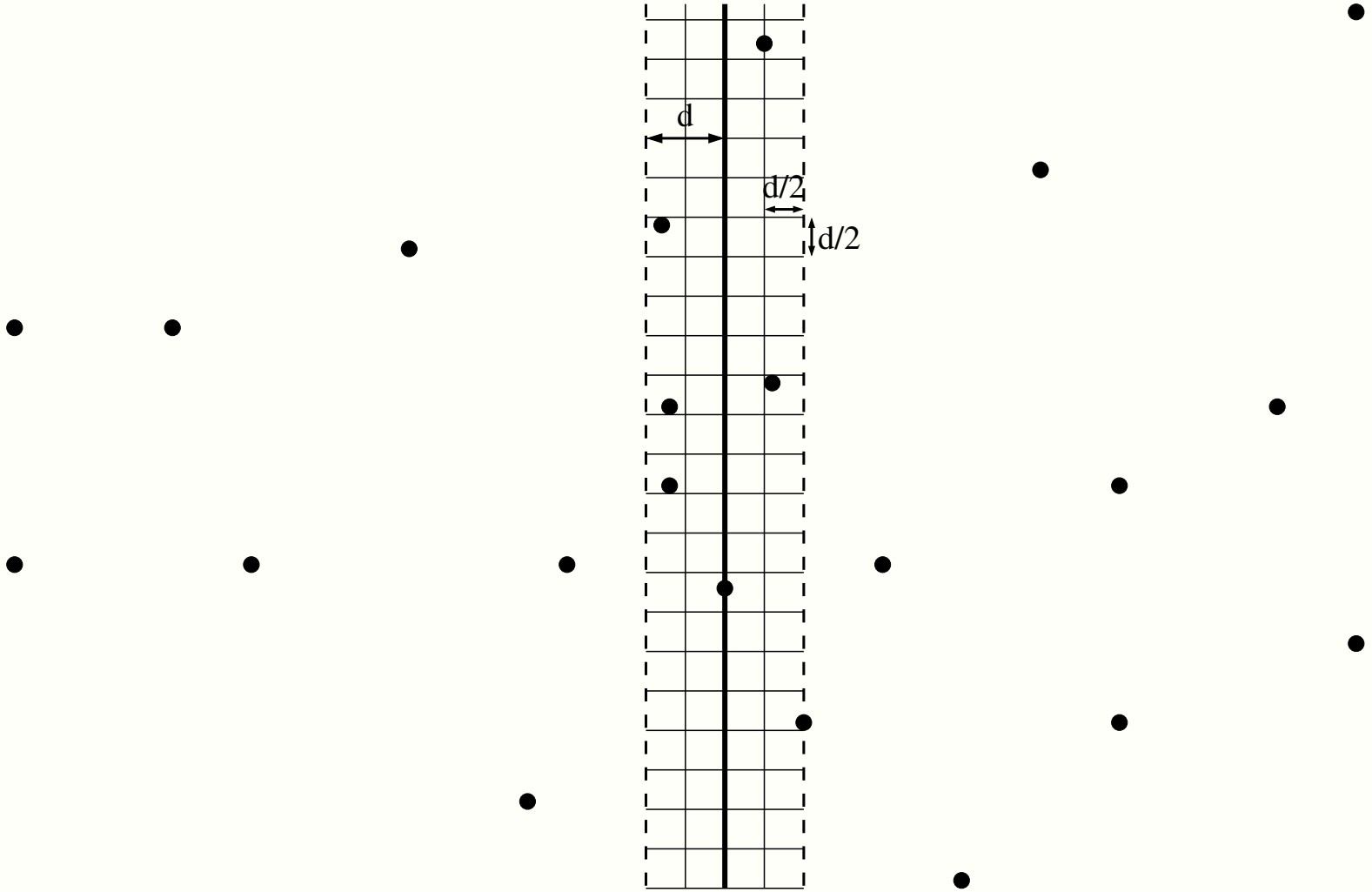
- Determiner une plus proche paire dans chaque moitie

Operation de recollement



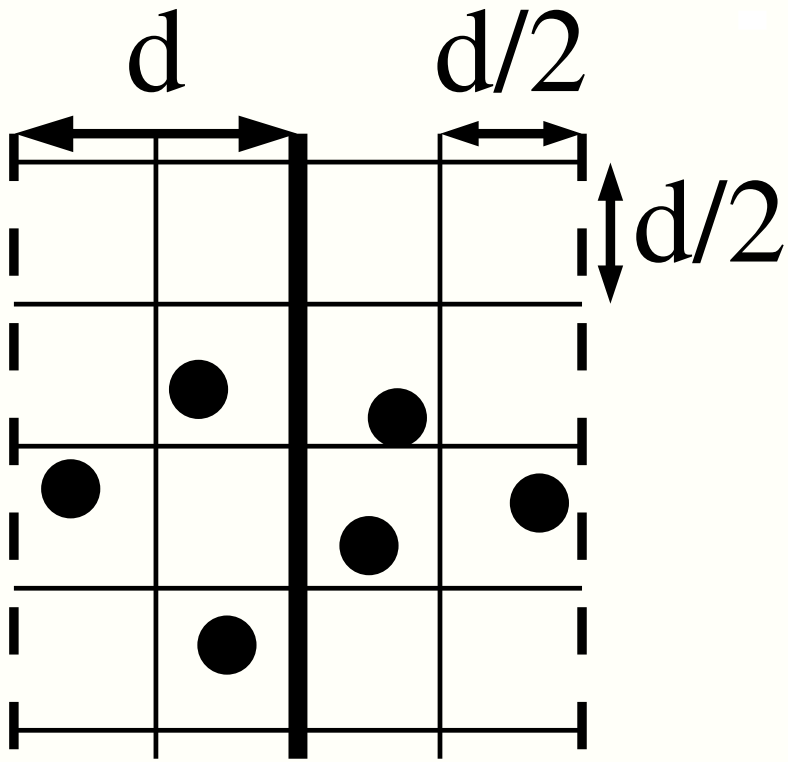
- Recoller : plus proche paire a cheval sur les deux moities

Operation de recollement



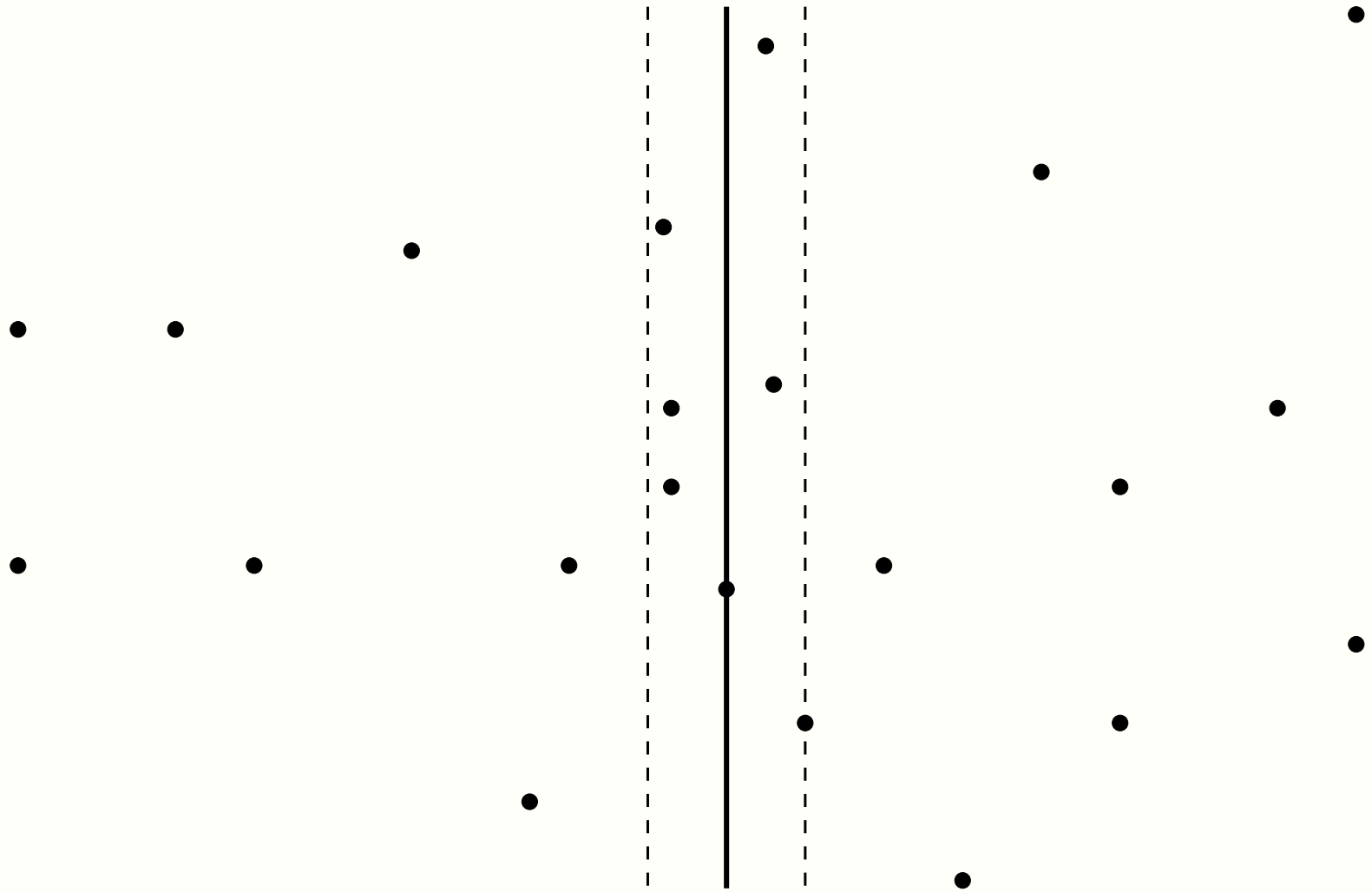
- Obs. clef : quadrillage a $d/2$ de la bande de recollement

Operation de recollement



Lemme : Deux sommets a distance au plus δ sont separes par au plus 10 autres sommets dans S ordonnee selon y.

Operation de recollement



- Recollement : scan vertical de la bande de recollement S

Operation de recollement

Algorithme 1 : vertical-scan(S_y)

```
1  si |S_y| > 1 alors
2      d_min ← +∞ ; s_0 ← ⊥ ; s_1 ← ⊥;
3      pour u ∈ S_y a y croissant faire
4          pour v a au plus 11 positions apres u dans S_y faire
5              si d(u, v) < d_min alors
6                  d_min ← d(u, v) ; s_0 ← u ; s_1 ← v;
7              fin
8          fin
9      fin
10     retourner (s_0, s_1);
11 fin
```

Operation de recollement

Algorithme 1 : vertical-scan(S_y)

```
1  si |S_y| > 1 alors
2      d_min ← +∞ ; s_0 ← ⊥ ; s_1 ← ⊥;
3      pour u ∈ S_y a y croissant faire
4          pour v a au plus 11 positions apres u dans S_y faire
5              si d(u, v) < d_min alors
6                  d_min ← d(u, v) ; s_0 ← u ; s_1 ← v;
7              fin
8          fin
9      fin
10     retourner (s_0, s_1);
11 fin
```

- Complexite ?

Operation de recollement

Algorithme 1 : vertical-scan(S_y)

```
1  si |S_y| > 1 alors
2      d_min ← +∞ ; s_0 ← ⊥ ; s_1 ← ⊥;
3      pour u ∈ S_y a y croissant faire
4          pour v a au plus 11 positions apres u dans S_y faire
5              si d(u, v) < d_min alors
6                  d_min ← d(u, v) ; s_0 ← u ; s_1 ← v;
7              fin
8          fin
9      fin
10     retourner (s_0, s_1);
11 fin
```

- Complexite : $O(|S_y|)$

Diviser pour regner sur *Plus proche paire*

Algorithme 2 : plus-proche-paire(P_x, P_y)

1 **plus-proche-paire**(P_x, P_y)
2 **si** $|P_x| \leq 3$ **alors** resoudre le probleme directement
3 **sinon**
4 $x^* \leftarrow$ abscisse du point median selon x;
5 $L \leftarrow \{v \in P_x \mid x(v) \leq x^*\}$; $R \leftarrow \{v \in P_x \mid x(v) > x^*\}$;
6 construire L_x, L_y, R_x, R_y;
7 $(l_0, l_1) \leftarrow$ **plus-proche-paire**(L_1, L_2);
8 $(r_0, r_1) \leftarrow$ **plus-proche-paire**(R_1, R_2);
9 $\delta \leftarrow \min\{d(l_0, l_1), d(r_0, r_1)\}$;
10 $S \leftarrow \{v \in P_x \mid x(u^*) - \delta \leq x(v) \leq x(u^*) + \delta)\}$;
11 construire S_y;
12 $(s_0, s_1) \leftarrow$ **vertical-scan**(S_y);
13 retourner argmin$\{d(s_0, s_1), d(l_0, l_1), d(r_0, r_1)\}$;
14 **fin**

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$
- 2 appels recursifs sur tailles $n/2$

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$
- Extraire S trie selon y : $O(n)$
- 2 appels recursifs sur tailles $n/2$

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$
- Extraire S trie selon y : $O(n)$
- Appel a vertical-scan(S_y) : $O(n)$
- 2 appels recursifs sur tailles $n/2$

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$
- Extraire S trie selon y : $O(n)$
- Appel a vertical-scan(S_y) : $O(n)$
- 2 appels recursifs sur tailles $n/2$

Reccurence :

- On note $T(n)$ le temps d'execution sur n points

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$
- Extraire S trie selon y : $O(n)$
- Appel a vertical-scan(S_y) : $O(n)$
- 2 appels recursifs sur tailles $n/2$

Reccurence :

- On note $T(n)$ le temps d'execution sur n points
- On a $T(n) \leq 2T(n/2) + O(n)$

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$
- Extraire S trie selon y : $O(n)$
- Appel a vertical-scan(S_y) : $O(n)$
- 2 appels recursifs sur tailles $n/2$

Reccurence :

- On note $T(n)$ le temps d'execution sur n points
- On a $T(n) \leq 2T(n/2) + O(n)$
- $\Longrightarrow T(n) = O(n \log n)$

Analyse de complexite de plus-proche-paire(P_x, P_y)

- On note $n = |P|$
- On dispose au depart de P trie selon x et selon y (P_x et P_y)
- Separer L et R et les trier selon x et y : $O(n)$
- Extraire S trie selon y : $O(n)$
- Appel a vertical-scan(S_y) : $O(n)$
- 2 appels recursifs sur tailles $n/2$

Reccurence :

- On note $T(n)$ le temps d'execution sur n points
- On a $T(n) \leq 2T(n/2) + O(n)$
- $\Longrightarrow T(n) = O(n \log n)$
- ... par le Master theorem

Master theorem

Si la complexite satisfait $T(n) = aT(\frac{n}{b}) + O(n^d)$

Master theorem

Si la complexite satisfait $T(n) = aT(\frac{n}{b}) + O(n^d)$

On discute selon qui est **le plus grand entre** d **et** $\log_b(a)$

Master theorem

Si la complexite satisfait $T(n) = aT(\frac{n}{b}) + O(n^d)$

On discute selon qui est **le plus grand entre** d **et** $\log_b(a)$

- Si $d < \log_b(a)$ alors $T(n) = O(n^{\log_b(a)})$
- Si $d = \log_b(a)$ alors $T(n) = O(n^d \log n)$
- Si $d > \log_b(a)$ alors $T(n) = O(n^d)$

Master theorem

Si la complexite satisfait $T(n) = aT(\frac{n}{b}) + O(n^d \log^k n)$

On discute selon qui est **le plus grand entre** d **et** $\log_b(a)$

- Si $d < \log_b(a)$ alors $T(n) = O(n^{\log_b(a)})$
- Si $d = \log_b(a)$ alors $T(n) = O(n^d \log^{k+1} n)$
- Si $d > \log_b(a)$ alors $T(n) = O(n^d)$

Master theorem

Si la complexite satisfait $T(n) = aT(\frac{n}{b}) + O(n^d \log^k n)$

On discute selon qui est **le plus grand entre** d **et** $\log_b(a)$

- Si $d < \log_b(a)$ alors $T(n) = O(n^{\log_b(a)})$
- Si $d = \log_b(a)$ alors $T(n) = O(n^d \log^{k+1} n)$
- Si $d > \log_b(a)$ alors $T(n) = O(n^d)$

Dans le cas qui nous interesse :

- $d = 1, k = 0, a = 2, b = 2$

Master theorem

Si la complexite satisfait $T(n) = aT(\frac{n}{b}) + O(n^d \log^k n)$

On discute selon qui est **le plus grand entre** d **et** $\log_b(a)$

- Si $d < \log_b(a)$ alors $T(n) = O(n^{\log_b(a)})$
- Si $d = \log_b(a)$ alors $T(n) = O(n^d \log^{k+1} n)$
- Si $d > \log_b(a)$ alors $T(n) = O(n^d)$

Dans le cas qui nous interesse :

- $d = 1, k = 0, a = 2, b = 2$
- d'ou $\log_b(a) = \log_2(2) = 1 = d$, on est dans le 2eme cas

Master theorem

Si la complexite satisfait $T(n) = aT(\frac{n}{b}) + O(n^d \log^k n)$

On discute selon qui est **le plus grand entre** d **et** $\log_b(a)$

- Si $d < \log_b(a)$ alors $T(n) = O(n^{\log_b(a)})$
- Si $d = \log_b(a)$ alors $T(n) = O(n^d \log^{k+1} n)$
- Si $d > \log_b(a)$ alors $T(n) = O(n^d)$

Dans le cas qui nous interesse :

- $d = 1, k = 0, a = 2, b = 2$
- d'ou $\log_b(a) = \log_2(2) = 1 = d$, on est dans le 2eme cas
- on a donc $T(n) = O(n \log n)$

Recap' de l'algo

1. Trier les points selon x dans P_x et selon y dans P_y

Recap' de l'algo

1. Trier les points selon x dans P_x et selon y dans P_y

2. Appeler plus-proche-paire(P_x, P_y)

Recap' de l'algo

1. Trier les points selon x dans P_x et selon y dans P_y

2. Appeler plus-proche-paire(P_x, P_y)

... c'est tout !

Recap' de l'algo

1. Trier les points selon x dans P_x et selon y dans P_y

2. Appeler plus-proche-paire(P_x, P_y)

... c'est tout !

- complexite totale : $O(n \log n)$

Et en dimension $d > 2$?

- Comme on en reve :

Et en dimension $d > 2$?

- Comme on en reve :
  - On parcourt les points selon une dimension

Et en dimension $d > 2$?

- Comme on en reve :
  - On parcourt les points selon une dimension
  - Le recollement se ramene au probleme en dim. $d - 1$

Et en dimension $d > 2$?

- Comme on en reve :
  - On parcourt les points selon une dimension
  - Le recollement se ramene au probleme en dim. $d - 1$

- Idee grossiere de l'algo :
  - Complexite du recollement : complexite du Pb. en dim. $d - 1$

Et en dimension $d > 2$?

- Comme on en reve :
  - On parcourt les points selon une dimension
  - Le recollement se ramene au probleme en dim. $d - 1$

- Idee grossiere de l'algo :
  - Complexite du recollement : complexite du Pb. en dim. $d - 1$
  - Master theorem $\Longrightarrow$ complexite en $O(n \log^{d-1} n)$

Et en dimension $d > 2$?

- Comme on en reve :
  - On parcourt les points selon une dimension
  - Le recollement se ramene au probleme en dim. $d - 1$
- Idee grossiere de l'algo :
  - Complexite du recollement : complexite du Pb. en dim. $d - 1$
  - Master theorem $\Longrightarrow$ complexite en $O(n \log^{d-1} n)$
- Polylogarithmique mais...

Et en dimension $d > 2$?

- Comme on en reve :
  - On parcourt les points selon une dimension
  - Le recollement se ramene au probleme en dim. $d - 1$
- Idee grossiere de l'algo :
  - Complexite du recollement : complexite du Pb. en dim. $d - 1$
  - Master theorem $\Longrightarrow$ complexite en $O(n \log^{d-1} n)$
- Polylogarithmique mais... dependance exponentielle en d