Module EA4 – Éléments d'Algorithmique

Dominique Poulalhon
dominique.poulalhon@liafa.univ-paris-diderot.fr

Université Paris Diderot
L2 Informatique, Math-Info et EIDD
Année universitaire 2013-2014

Rang

l'élément de rang `k` d'un tableau `T` est l'unique `x` de `T` tel que

- `T` contient `k` $-$ 1 éléments plus petits que `x`
- `T` contient `len(T)` $-$ `k` éléments plus grands que `x`

Rang

l'élément de rang `k` d'un tableau `T` est l'unique `x` de `T` tel que

- `T` contient $k - 1$ éléments plus petits que `x`
- `T` contient `len(T)` $-$ `k` éléments plus grands que `x`

Cas particuliers

- *si* `T` est trié : `T[k-1]`

Rang

l'élément de rang `k` d'un tableau `T` est l'unique `x` de `T` tel que

- `T` contient $k - 1$ éléments plus petits que `x`
- `T` contient `len(T)` $-$ `k` éléments plus grands que `x`

Cas particuliers

- *si* `T` est trié : `T[k-1]`
- élément de rang 1 : `minimum(T)`

Rang

l'élément de rang `k` d'un tableau `T` est l'unique `x` de `T` tel que

- `T` contient $k - 1$ éléments plus petits que `x`
- `T` contient $\texttt{len(T)} - \texttt{k}$ éléments plus grands que `x`

Cas particuliers

- *si* `T` est trié : `T[k-1]`
- élément de rang 1 : `minimum(T)`
- élément de rang `len(T)` : `maximum(T)`

Rang

l'élément de rang `k` d'un tableau `T` est l'unique `x` de `T` tel que

- `T` contient $k - 1$ éléments plus petits que `x`
- `T` contient `len(T)` $-$ `k` éléments plus grands que `x`

Cas particuliers

- *si* `T` est trié : `T[k-1]`
- élément de rang 1 : `minimum(T)`
- élément de rang `len(T)` : `maximum(T)`
- élément « du milieu » : `médian(T)` (ou `médiane(T)`)

Rang

l'élément de rang `k` d'un tableau `T` est l'unique `x` de `T` tel que

- `T` contient $k-1$ éléments plus petits que `x`
- `T` contient `len(T)` $-$ `k` éléments plus grands que `x`

Cas particuliers

- *si* `T` est trié : `T[k-1]`
- élément de rang 1 : `minimum(T)`
- élément de rang `len(T)` : `maximum(T)`
- élément « du milieu » : `médian(T)` (ou `médiane(T)`)

Rang

l'élément de rang `k` d'un tableau `T` est l'unique `x` de `T` tel que

- `T` contient $k-1$ éléments plus petits que `x`
- `T` contient `len(T)` $-$ `k` éléments plus grands que `x`

Cas particuliers

- *si* `T` est trié : `T[k-1]`
- élément de rang 1 : `minimum(T)`
- élément de rang `len(T)` : `maximum(T)`
- élément « du milieu » : `médian(T)` (ou `médiane(T)`)
  si $n =$ `len(T)` impair : rang $\frac{1}{2}(n+1)$
  si ℓ pair : rang $\frac{1}{2}n$ ou $\frac{1}{2}n+1$

`selection(T, k)`

étant donné un tableau `T` et un entier `k`, déterminer l'élément de rang `k` de `T`

selection(T, k)

étant donné un tableau T et un entier k, déterminer l'élément de rang k de T

Solution n° 1

- trier T
- retourner T[k-1]

`selection(T, k)`

étant donné un tableau `T` et un entier `k`, déterminer l'élément de rang `k` de `T`

Solution n° 1

- trier `T`
- retourner `T[k-1]`

$\Longrightarrow$ $\Theta(n \log n)$ comparaisons (au pire)

minimum(T)

étant donné un tableau T, déterminer le plus petit élément de T

```
def min(T) :
  tmp = T[0]
  for elt in T :
    if elt < tmp : tmp = elt
  return tmp
```

$\Longrightarrow$ $n - 1$ comparaisons (exactement)

maximum(T)

étant donné un tableau T, déterminer le plus grand élément de T

```
def max(T) :
  tmp = T[0]
  for elt in T :
    if elt > tmp : tmp = elt
  return tmp
```

$\Longrightarrow$ $n-1$ comparaisons (exactement)

Sélection – cas particuliers

`min_et_max_simultanés(T)`

étant donné un tableau `T`, déterminer le plus petit et le plus grand éléments de `T`

min_et_max_simultanés(T)

étant donné un tableau T, déterminer le plus petit et le plus grand éléments de T

```
def min_et_max(T) :       # cas où len(T) est impaire
  min = max = T[0]
  for elt1, elt2 in zip(T[1::2], T[2::2]) : # 2 par 2
    if elt1 < elt2 :
      if elt1 < min : min = elt1
      if elt2 > max : max = elt2
    else : ## échanger le rôle de elt1 et elt2
 return min, max
```

$\Longrightarrow \frac{3}{2}(n-1)$ comparaisons (si n impair)

min_et_max_simultanés(T)

étant donné un tableau T, déterminer le plus petit et le plus grand éléments de T

```
def min_et_max(T) :        # cas où len(T) est paire
  if T[0] < T[1] : min, max = T[0], T[1]
  else : min, max = T[1], T[0]
  for elt1, elt2 in zip(T[2::2], T[3::2]) : # 2 par 2
    if elt1 < elt2 :
      if elt1 < min : min = elt1
      if elt2 > max : max = elt2
    else : ## échanger le rôle de elt1 et elt2
 return min, max
```

$\Longrightarrow \frac{3n}{2} - 2$ comparaisons (si n pair)

Sélection – cas général

```
def selection(T, k) :
  for i in range(k) :
    tmp = i
    for j in range(i, len(T)) :
      if T[j] < T[tmp] : tmp = j
    T[i], T[tmp] = T[tmp], T[i]
  return T[k-1]
```

Sélection – cas général

```
def selection(T, k) :
  for i in range(k) :
    tmp = i
    for j in range(i, len(T)) :
      if T[j] < T[tmp] : tmp = j
    T[i], T[tmp] = T[tmp], T[i]
  return T[k-1]
```

$\Longrightarrow$ kn comparaisons (environ)

si `k` est petit, c'est sensiblement mieux que $\Theta(n \log n)$!

Sélection rapide (*Quickselect*)

```
def selection_rapide(T, k) :
  if len(T) == 1 and k == 1 : return T[0]
  pivot, gauche, droite = partition(T)
  position = len(gauche) + 1
  if position == k : return pivot
  if position > k : return selection_rapide(gauche, k)
  return selection_rapide(droite, k - position)
```

Sélection rapide (*Quickselect*)

```
def selection_rapide(T, k) :
  if len(T) == 1 and k == 1 : return T[0]
  pivot, gauche, droite = partition(T)
  position = len(gauche) + 1
  if position == k : return pivot
  if position > k : return selection_rapide(gauche, k)
  return selection_rapide(droite, k - position)
```

Complexité de `selection_rapide` au pire : $\Theta(n^2)$ comparaisons

Complexité de `selection_rapide` dans le meilleur des cas : $\Theta(n)$ comparaisons

Sélection rapide (*Quickselect*)

Complexité de `selection_rapide` en moyenne (*admis*) :

$\Theta(n)$ comparaisons

Sélection rapide (*Quickselect*)

Complexité de `selection_rapide` en moyenne (*admis*) :

$\Theta(n)$ comparaisons

En choisissant comme pivot la médiane d'un échantillon de 5 éléments, on obtient un algorithme de complexité $\Theta(n)$ dans le pire des cas (admis)

Applications du tri en géométrie : 1. Calcul de l'enveloppe convexe

enveloppe_convexe(L)

étant donné une liste L de points du plan, déterminer l'enveloppe convexe des éléments de L

Applications du tri en géométrie : 1. Calcul de l'enveloppe convexe

enveloppe_convexe(L)

étant donné une liste L de points du plan, déterminer l'enveloppe convexe des éléments de L

enveloppe convexe d'une partie $\mathcal{P}$ du plan : plus petite partie convexe $\mathcal{C}$ contenant $\mathcal{P}$

Applications du tri en géométrie : 1. Calcul de l'enveloppe convexe

enveloppe_convexe(L)

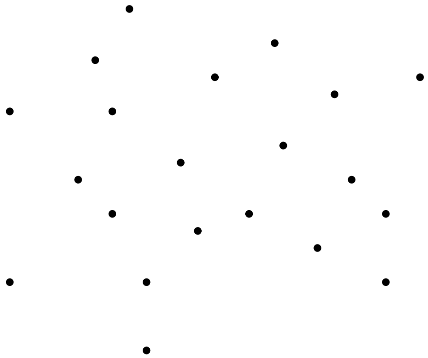
étant donné une liste L de points du plan, déterminer l'enveloppe convexe des éléments de L

enveloppe convexe d'une partie $\mathcal{P}$ du plan : plus petite partie convexe $\mathcal{C}$ contenant $\mathcal{P}$

si $\mathcal{P}$ est un ensemble fini de points, $\mathcal{C}$ est un polygone dont les sommets sont des éléments de $\mathcal{P}$

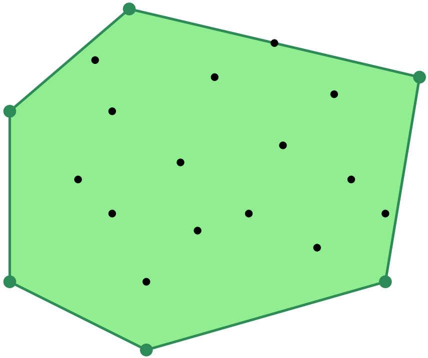
Enveloppe convexe de points

Exemple :



Enveloppe convexe de points

Exemple :

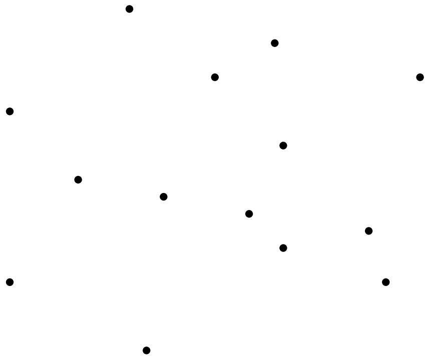


Enveloppe convexe de points

```
def enveloppe_convexe_par_balayage(L) :
  p0 = point_le_plus_bas(L)
  L = trier_selon_angle_polaire(L, p0)
  pile = [L[0], L[1], L[2]]
  for point in L :
    while tourne_a_droite(pile[-2], pile[-1], point) :
      pile.pop()
    pile.append(point)
  return pile
```

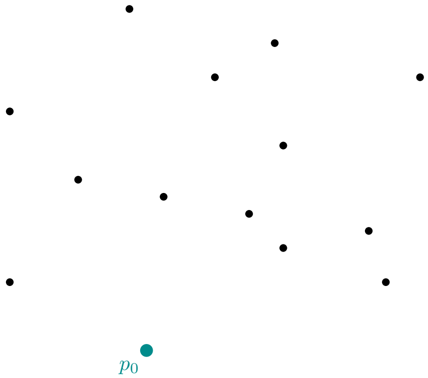
Enveloppe convexe de points

Exemple :



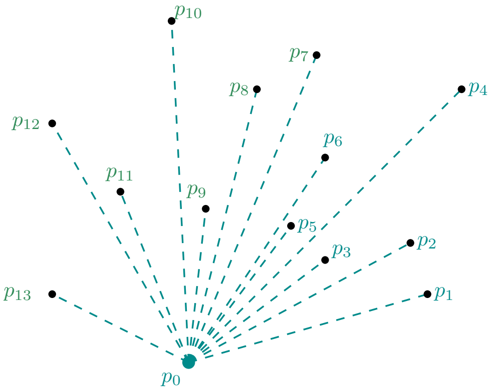
Enveloppe convexe de points

Exemple :



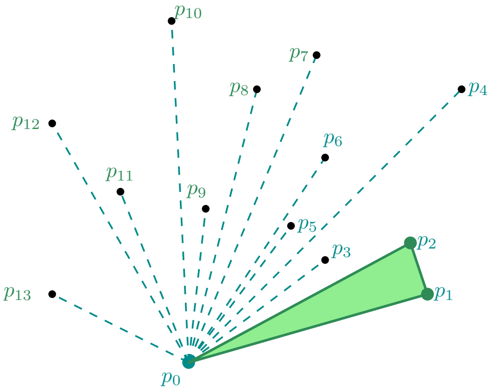
Enveloppe convexe de points

Exemple :



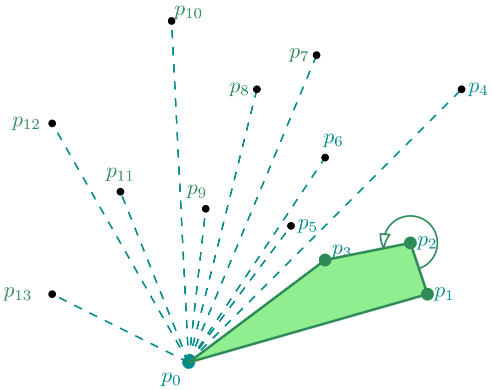
Enveloppe convexe de points

Exemple :



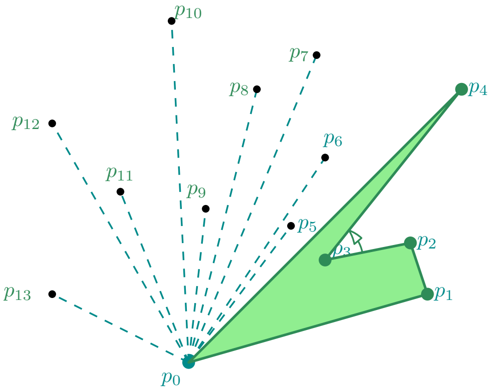
Enveloppe convexe de points

Exemple :



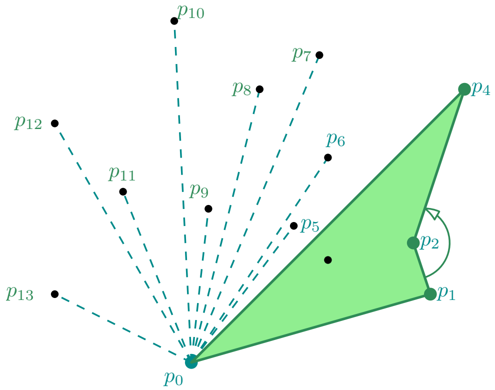
Enveloppe convexe de points

Exemple :



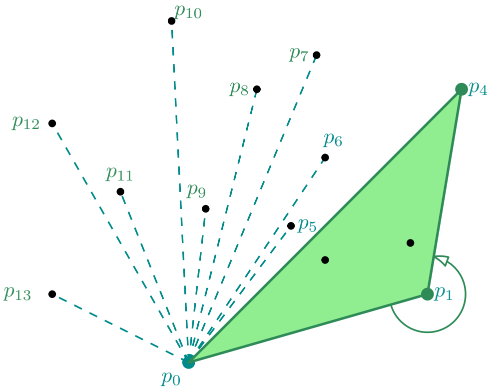
Enveloppe convexe de points

Exemple :

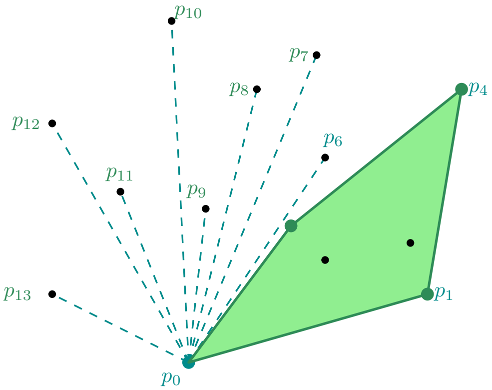


Enveloppe convexe de points

Exemple :

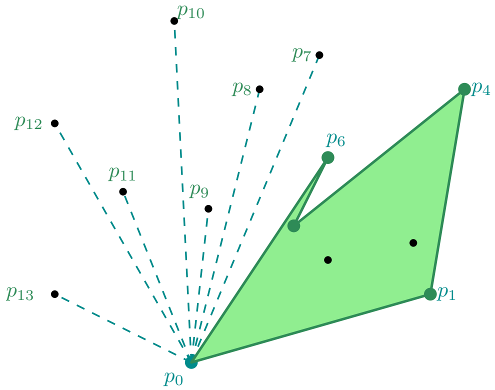


Exemple :



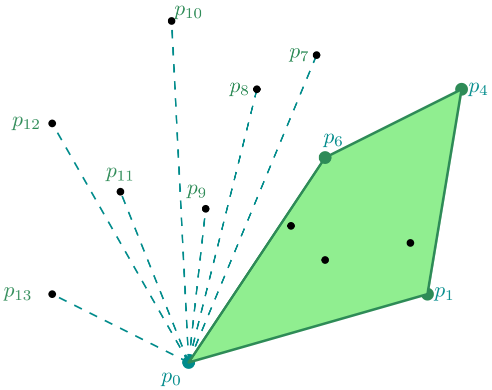
Enveloppe convexe de points

Exemple :



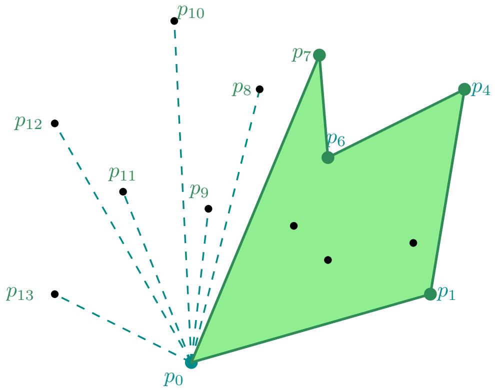
Enveloppe convexe de points

Exemple :

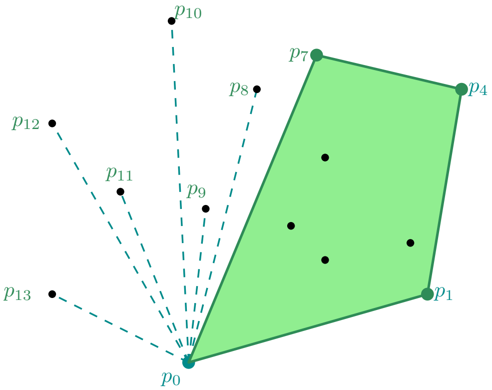


Enveloppe convexe de points

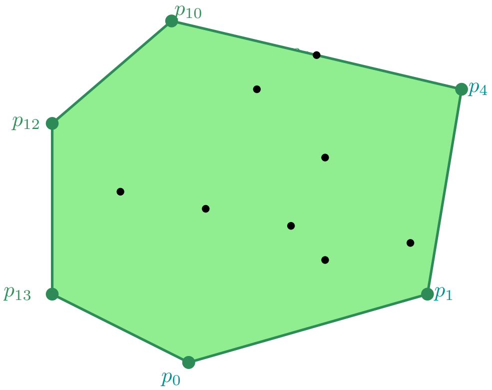
Exemple :



Exemple :



Exemple :



Applications du tri en géométrie : 2. Points les plus proches

distance_minimale(L)

étant donné une liste L de points du plan, déterminer la distance minimale entre deux éléments de L

Applications du tri en géométrie : 2. Points les plus proches

distance_minimale(L)

étant donné une liste L de points du plan, déterminer la distance minimale entre deux éléments de L

Approche diviser pour régner :

distance_minimale(L)

étant donné une liste `L` de points du plan, déterminer la distance minimale entre deux éléments de `L`

Approche diviser pour régner :

- séparer `L` en deux sous-listes `gauche` et `droite`

Applications du tri en géométrie : 2. Points les plus proches

distance_minimale(L)

étant donné une liste `L` de points du plan, déterminer la distance minimale entre deux éléments de `L`

Approche diviser pour régner :

- séparer `L` en deux sous-listes `gauche` et `droite`
- calculer `d1 = distance_minimale(gauche)` et `d2 = distance_minimale(droite)`

distance_minimale(L)

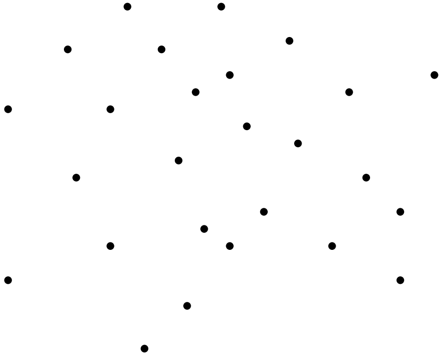
étant donné une liste `L` de points du plan, déterminer la distance minimale entre deux éléments de `L`

Approche diviser pour régner :

- séparer `L` en deux sous-listes `gauche` et `droite`
- calculer `d1 = distance_minimale(gauche)` et `d2 = distance_minimale(droite)`
- chercher s'il existe `p1` dans `gauche` et `p2` dans `droite` plus proches que `min(d1, d2)`

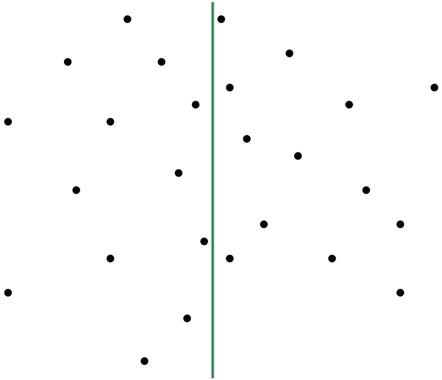
Points les plus proches

Exemple

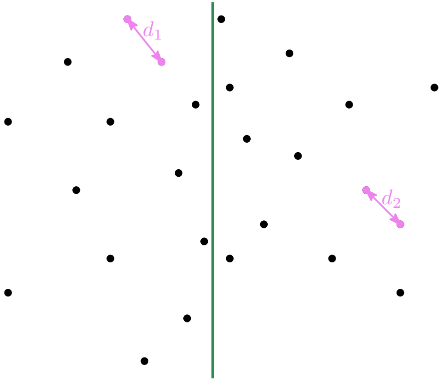


Points les plus proches

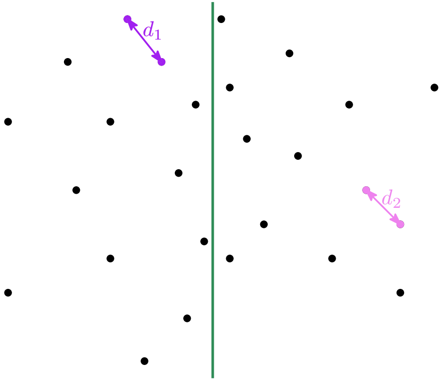
Exemple



Exemple

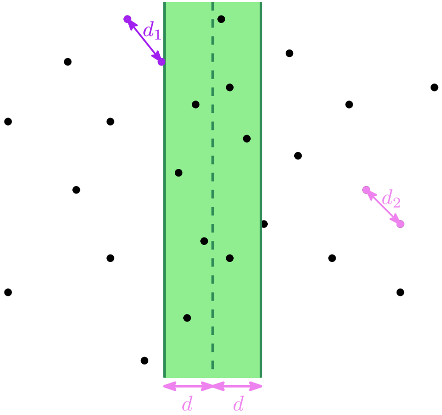


Exemple

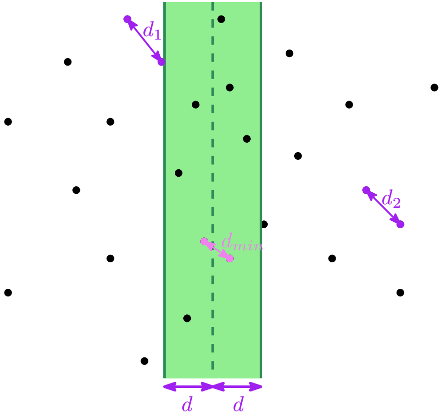


Points les plus proches

Exemple

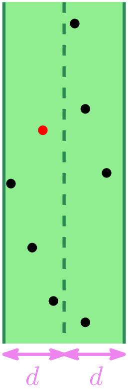


Exemple

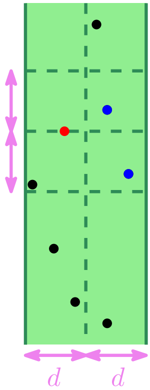


Points les plus proches

Comment trouver (`p1`, `p2`) efficacement ?

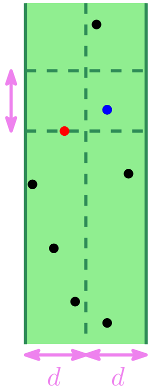


Comment trouver (`p1`, `p2`) efficacement ?

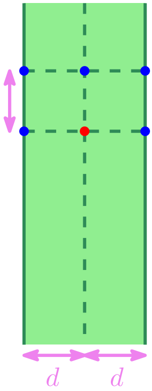


Points les plus proches

Comment trouver (`p1`, `p2`) efficacement ?



Comment trouver (`p1`, `p2`) efficacement ?



Comment optimiser l'algorithme ?

Pour le partitionnement gauche–droite

Trier *une fois pour toutes* la liste des points selon les abscisses
$\Longrightarrow$ étant donné `L_x`, le partitionnement a un coût constant

Comment optimiser l'algorithme ?

Pour le partitionnement gauche–droite

Trier *une fois pour toutes* la liste des points selon les abscisses

$\Longrightarrow$ étant donné `L_x`, le partitionnement a un coût constant

Pour la recherche des couples (p1, p2)

Trier *une fois pour toutes* la liste des points selon les ordonnées

$\Longrightarrow$ étant donné `L_y`, la recherche a un coût linéaire

Comment optimiser l'algorithme ?

Pour le partitionnement gauche–droite

Trier *une fois pour toutes* la liste des points selon les abscisses

$\Longrightarrow$ étant donné `L_x`, le partitionnement a un coût constant

Pour la recherche des couples (p1, p2)

Trier *une fois pour toutes* la liste des points selon les ordonnées

$\Longrightarrow$ étant donné `L_y`, la recherche a un coût linéaire

$$C_{totale}(n) = C_{tris}(n) + C_{rec}(n) = \Theta(n \log n) + C_{rec}(n)$$

$$C_{rec}(n) = 2C_{rec}\left(\frac{n}{2}\right) + O(n)$$

$$\Longrightarrow \quad C_{totale}(n) \in \Theta(n \log n)$$