Tris, 1

Lycée Pierre Corneille – MP

2016-2017

Trier

- Objectif : faciliter les recherches (ex : médiane)

Trier

- Objectif : faciliter les recherches (ex : médiane)
- Moyen : comparaisons à l'aide d'une relation d'ordre total

Trier

- Objectif : faciliter les recherches (ex : médiane)
- Moyen : comparaisons à l'aide d'une relation d'ordre total
- Données numériques : $\leqslant$

Trier

- Objectif : faciliter les recherches (ex : médiane)
- Moyen : comparaisons à l'aide d'une relation d'ordre total
- Données numériques : $\leqslant$
- Données alphabétiques : ordre lexicographique

Trier

Données à trier

```
import random as rd
L = [rd.random() for i in range(10)]
print(L)
```

Trier

Données à trier

```
import random as rd
L = [rd.random() for i in range(10)]
print(L)
```

Tri automatique

```
L.sort()
print(L)
```

Trier

Données à trier

```
import random as rd
L = [rd.random() for i in range(10)]
print(L)
```

Tri automatique

```
L.sort()
print(L)
```

Tri d'un tableau bidimensionnel

```
T.sort()   # tri de chaque ligne
T.sort(0)  # tri de chaque colonne
```

Trier

- Comment ça marche ?

Trier

- Comment ça marche ?
- Rapidité d'exécution (nombre d'opérations)

Trier

- Comment ça marche ?
- Rapidité d'exécution (nombre d'opérations)
- Facilité d'exécution (occupation de la mémoire)

Tri par sélection

On repère la position du plus grand élément d'un tableau T.

```
def pos_max(T):
  p, M = 0, T[0]
  for i in range(1, len(T)):
    if (T[i]>M):
      p, M = i, T[i]
  return p
```

Tri par sélection

On repère la position du plus grand élément d'un tableau T.

```
def pos_max(T):
  p, M = 0, T[0]
  for i in range(1, len(T)):
    if (T[i]>M):
      p, M = i, T[i]
  return p
```

Nombre de comparaisons $\approx$ taille du tableau T

Tri par sélection

On cherche le plus grand élément et on le déplace en queue de liste.
On recommence avec la sous-liste qui reste à trier.

```
def tri_selection(L):
  n = len(L)
  for i in range(n):
    j = pos_max(L[:n-i])
    L[j], L[n-i-1] = L[n-i-1], L[j]
  return L
```

Tri par sélection – complexité

Nombre total de comparaisons

$$\sum_{i=0}^{n-1}(n-i) = \mathcal{O}(n^2)$$

Tri par sélection – complexité

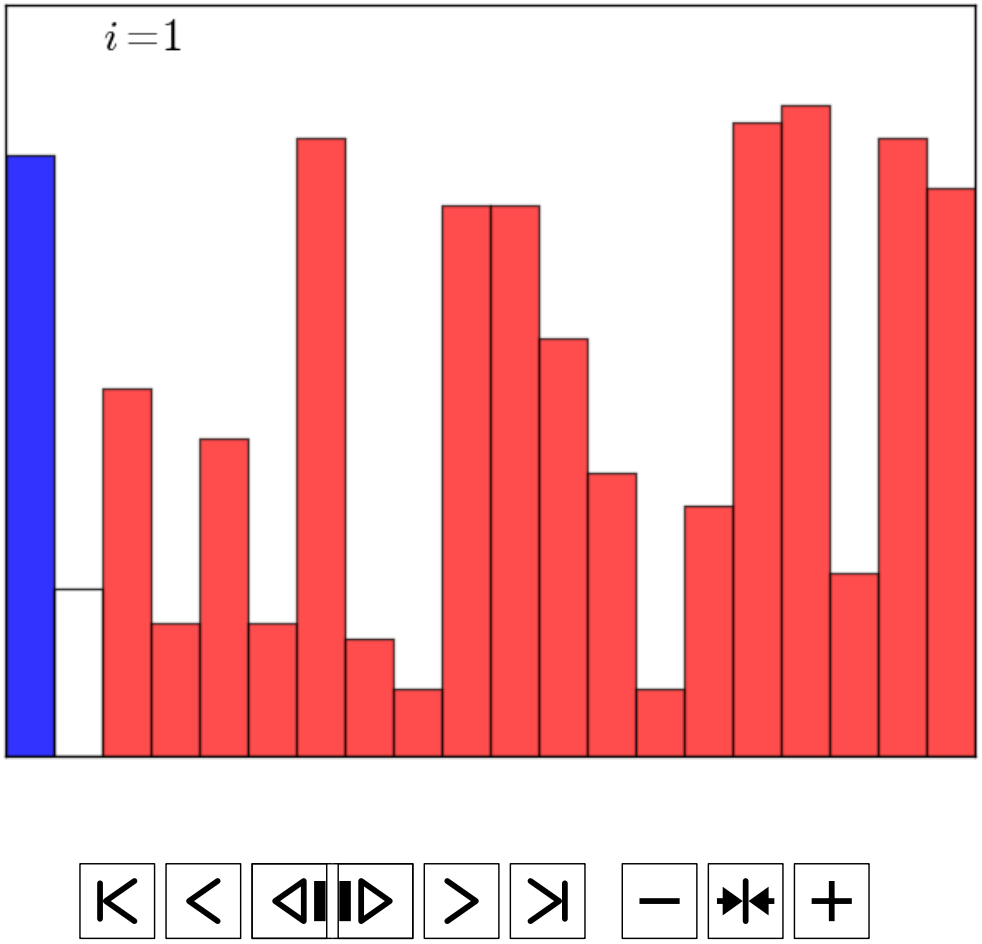
Nombre total de comparaisons

$$\sum_{i=0}^{n-1}(n-i) = \mathcal{O}(n^2)$$

Nombre d'échanges inférieur à la longueur de L (nul si la liste est déjà triée)

Tri par insertion

On suppose que le début de liste est trié et on insère un nouvel élément à la bonne place pour que le début de la liste reste trié.



Tri par insertion

```
def tri_insertion(L):
    n = len(L)
    for i in range(1,n):
        j, x = i, L[i]
                        # Où insérer x dans L[:i] ?
        while (j>0) and (x<L[j-1]):
            L[j], L[j-1] = L[j-1], L[j]
            j -= 1
    return L
```

Tri par insertion – complexité

Pour situer `L[i]` dans la sous-liste triée `L[:i]`

- Meilleur des cas
  Si la liste est triée en croissant, *une seule* comparaison et aucune permutation

Tri par insertion – complexité

Pour situer `L[i]` dans la sous-liste triée `L[:i]`

- Meilleur des cas
  Si la liste est triée en croissant, *une seule* comparaison et aucune permutation
- Pire des cas
  Si la liste est triée en décroissant, i comparaisons et $(i-1)$ permutations

Tri par insertion – complexité

Pour situer `L[i]` dans la sous-liste triée `L[:i]`

- Meilleur des cas
  Si la liste est triée en croissant, *une seule* comparaison et aucune permutation
- Pire des cas
  Si la liste est triée en décroissant, i comparaisons et $(i-1)$ permutations
- Entre n et $\mathcal{O}(n^2)$ comparaisons en tout, autant de permutations

Classer

- Objectif : ordonner selon un critère

Classer

- Objectif : ordonner selon un critère
- Particularité : les valeurs possibles du critère sont connues

Classer

Données : couples (nom, jour de naissance)

```
MP = [('ADOLPHE', 57), ('AMMOR', 131), ('BEN CHABANE', 184),
  ('BENJELLOUN', 91), ('BERNARD', 336),  ('BLOQUET', 23), ...
```

Classer

Données : couples (nom, jour de naissance)

```
MP = [('ADOLPHE', 57), ('AMMOR', 131), ('BEN CHABANE', 184),
  ('BENJELLOUN', 91), ('BERNARD', 336),  ('BLOQUET', 23), ...
```

Critère de classement : jour de naissance

```
jours = [[] for i in range(365)]
```

Classer

Données : couples (nom, jour de naissance)

```
MP = [('ADOLPHE', 57), ('AMMOR', 131), ('BEN CHABANE', 184),
  ('BENJELLOUN', 91), ('BERNARD', 336),  ('BLOQUET', 23), ...
```

Critère de classement : jour de naissance

```
jours = [[] for i in range(365)]
```

Classement : on range chaque élément de la liste à sa place

```
for A in MP:
    jours[A[1]].append(A[0])
```

Classer

Données : couples (nom, jour de naissance)

```
MP = [('ADOLPHE', 57), ('AMMOR', 131), ('BEN CHABANE', 184),
  ('BENJELLOUN', 91), ('BERNARD', 336),  ('BLOQUET', 23), ...
```

Critère de classement : jour de naissance

```
jours = [[] for i in range(365)]
```

Classement : on range chaque élément de la liste à sa place

```
for A in MP:
    jours[A[1]].append(A[0])
```

Ni comparaison, ni permutation : plus rapide que tous les autres algorithmes de tri !

Classer

Données : couples (nom, jour de naissance)

```
MP = [('ADOLPHE', 57), ('AMMOR', 131), ('BEN CHABANE', 184),
  ('BENJELLOUN', 91), ('BERNARD', 336),  ('BLOQUET', 23), ...
```

Critère de classement : jour de naissance

```
jours = [[] for i in range(365)]
```

Classement : on range chaque élément de la liste à sa place

```
for A in MP:
    jours[A[1]].append(A[0])
```

Ni comparaison, ni permutation : plus rapide que tous les autres algorithmes de tri !
Tri stable : l'ordre alphabétique est conservé.