

# cours\_3

January 28, 2020

## 1 Tris et rangs

### 1.1 Définition

On définit le problème du tri comme:

Entrée: une séquence de valeurs  $a_1, a_2, \dots, a_n$

Sortie: une **permutation**  $\sigma$  (un réarrangement) telle que la séquence

$$a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(n)}$$

soit **ordonnée**.

### 1.2 Clé de tri

On parle de **clé de tri** pour désigner l'élément sur lequel l'ordre est mis en place.

- Type de clé usuelles: nombre, chaîne, liste
- Clé définie par l'utilisateur. Ex. pour une chaîne
  - ordre lexicographique (par défaut)
  - nombre de caractères pour une chaîne
  - nombre de consonnes puis nombre de voyelles
  - mois de l'année: 'Janvier' < 'Février'
- Clé simple dans une structure plus complexe.  
Ex.: Tuple (JJ, MM, YYYY)

Le langage de programmation fait donc des choix de clés par défaut.

Il revient au programmeur de les adapter si besoin est, et de définir sa propre de comparaison.

### 1.3 Classification des algorithmes de tri

Définition: Tri en place ou non

Un algorithme de tri est dit **en place** lorsque jamais plus d'un *nombre constant* d'éléments est stocké hors du tableau quelque soit sa taille.

Définition: Tri stable

Un algorithme de tri est dit **stable** lorsque l'ordre des éléments ayant une *clé identique* est maintenu.

On peut rendre stable un algorithme de tri en introduisant une clé secondaire.

Définition: Tri par comparaison ou non (voir TP)

- Un algorithme de tri peut utiliser des comparaisons “<”
- ou non...

Définition: Tri incrémental ou non

- Un algorithme de tri est **incrémental** si il est susceptible de traiter les données une par une sans disposer du tableau en entier.

## 1.4 Tri par sélection

---

### Algorithme 4: Tri par sélection

---

**Données :** T un tableau de  $n$  nombres

**Résultat :** Le tableau T est trié par ordre croissant

```

1  $n \leftarrow \text{longueur}(T)$ ;
2 pour  $i \leftarrow 1$  à  $n - 1$  faire
3    $imin \leftarrow i$ ;
4   pour  $j \leftarrow i + 1$  à  $n$  faire
5     si  $T[j] < T[imin]$  alors  $imin \leftarrow j$ ;
6   fin
7   Échanger  $T[i]$  et  $T[imin]$ ;
8 fin
9 return T;
```

---

Tri par sélection

### 1.4.1 Caractéristiques: Tri par sélection

Il est:

- en place
- stable
- par comparaison
- non incrémental

### 1.4.2 Analyse du Tri par sélection

*Terminaison:* OK

*Correction de l'algorithme* par invariant:

*Avant:*

- $T[1..i-1]$  est trié
- Pour tout  $j=i..n$ ,  $T[j] \geq T[i-1]$

*Après:*

- $T[1..i]$  est trié
- Pour tout  $j=i+1..n$ ,  $T[j] \geq T[i]$

*A la fin:*

- $T[1..n-1]$  est trié
- $T[n] \geq T[n-1]$

=> T est donc trié !

### Effizienz de l'algorithme

Nombre d'échanges ?

Pour i fixé, 1 échange.

Nombre total d'échanges:  $n-1$

Nombre de comparaisons ?

Pour i fixé,  $n-i$  comparaisons.

- $i = 1 \rightarrow n - 1$  comparaisons
- $i = 2 \rightarrow n - 2$  comparaisons
- ...
- $i = n - 2 \rightarrow 2$  comparaisons
- $i = n - 1 \rightarrow 1$  comparaison

Nombre total de **comparaisons**:

$$1 + 2 + 3 + \dots + (n-2) + (n-1) = \frac{n * (n-1)}{2} \sim n^2$$

### 1.4.3 Implémentation en Python

```
[1]: def tri_selection(T):
    n = len(T)
    for i in range(0, n-1):
        imin = i
        for j in range(i+1, n):
            if T[j] < T[imin]:
                imin = j
        T[i], T[imin] = T[imin], T[i]
    return T
```

```
[2]: import random

def estTrie(L, deb=0, fin=None):
    if fin is None:
        fin = len(L)
    for i in range(1, fin):
        if L[i-1] > L[i] :
            return False
    return True

def tableau(n, min=0, max=100):
    return [random.randint(min, max) for _ in range(n)]
```

```
[3]: # Vérification rapide qui n'est pas une preuve de correction
T = [4, 6, -4, 2, 5, 3]
T2 = tri_selection(T)
```

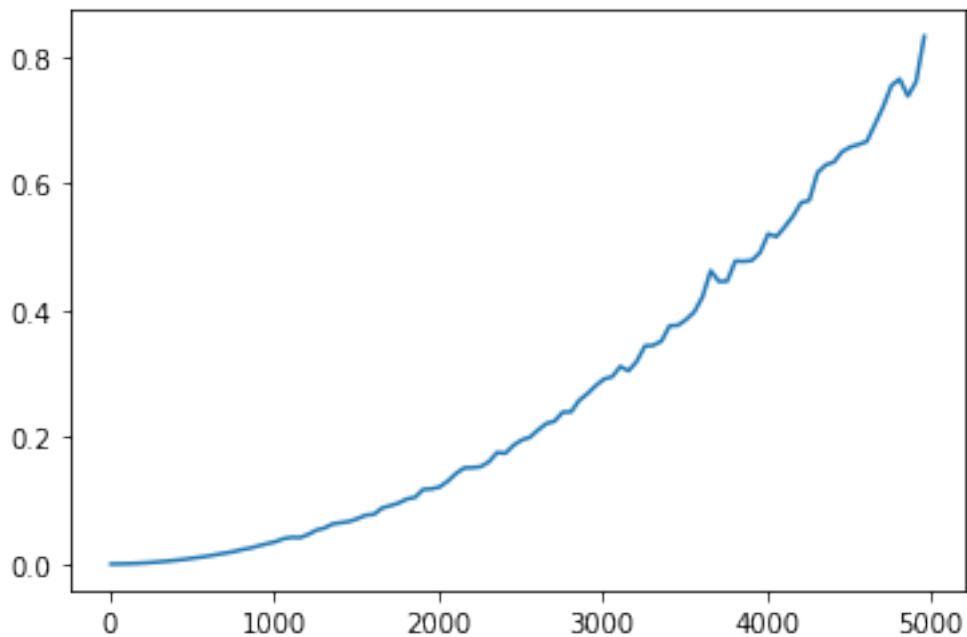
```
print(f'{T} est trié : {estTrie(T)}')
```

[-4, 2, 3, 4, 5, 6] est trié : True

```
[4]: import matplotlib.pyplot as plt
      from time import time

      tps_select = []
      N = list(range(10, 5000, 50))
      for n in N:
          T = tableau(n)
          debut = time()
          _ = tri_selection(T)
          tps_select.append(time() - debut)
```

```
[5]: plt.plot(N, tps_select)
      plt.show()
```



## 1.5 Tri par insertion

---

**Algorithme 5:** Tri par insertion

---

**Données :** T un tableau de  $n$  entiers

**Résultat :** Le tableau T est trié par ordre croissant

```
1 pour  $i \leftarrow 2$  à  $n$  faire
2    $clé \leftarrow T[i];$ 
3    $j \leftarrow i - 1;$ 
4   tant que  $j > 0$  et  $clé < T[j]$  faire
5      $T[j + 1] \leftarrow T[j];$ 
6      $j \leftarrow j - 1;$ 
7   fin
8    $T[j + 1] \leftarrow clé;$ 
9 fin
10 return T;
```

---

Tri par insertion

### 1.5.1 Tri par insertion (démonstration)

source: [Tri par insertion \(Wikipédia\)](#)

### 1.5.2 Caractéristiques: Tri par insertion

Il est:

- en place
- stable (car inf. strict)
- par comparaison
- incrémental

*Terminaison de l'algorithme:* OK

*Correction de l'algorithme* par invariant:

- *Avant:*  $T[1..i-1]$  est trié
- *Après:*  $T[1..i]$  est trié
- *A la fin* ( $i = n$ ), T est donc trié !

*Efficiency de l'algorithme*

Le nombre de comparaisons est proportionnel au nombre d'échanges.

- Pour  $i$  fixé, on distingue plusieurs cas:
  - 2 comparaisons si  $T[i] \geq T[i-1]$
  - $2 \cdot i$  comparaisons si  $T[i] = clé < T[j]$  pour tout  $j=1..i-1$ .
  - $i = 2 \cdot i / 2$  comparaisons en moyenne

Nombre total de **comparaisons**:

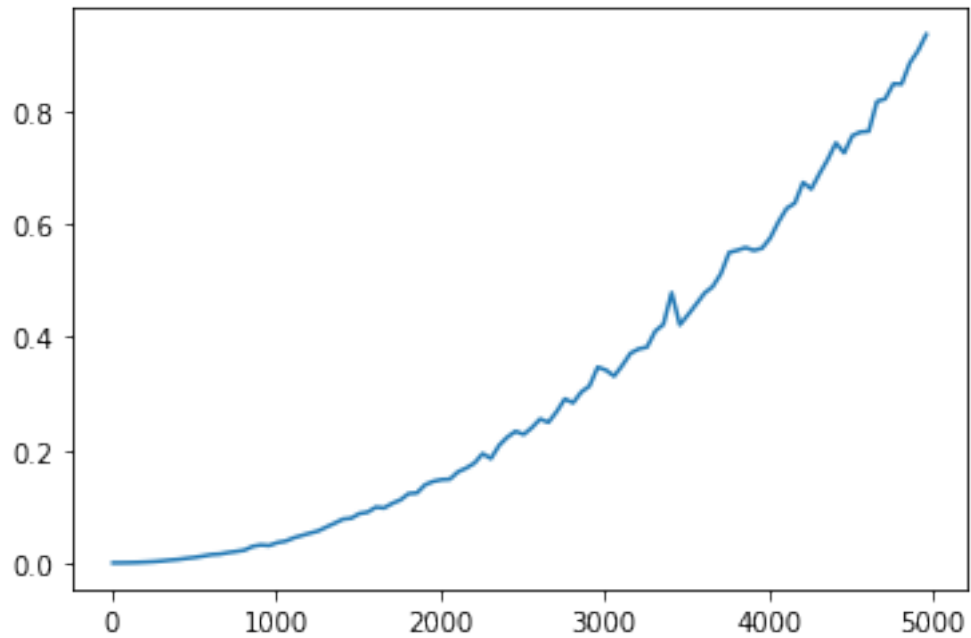
$$2n \leq nbComp(n) \leq 2n^2$$

### 1.5.3 Implémentation en Python

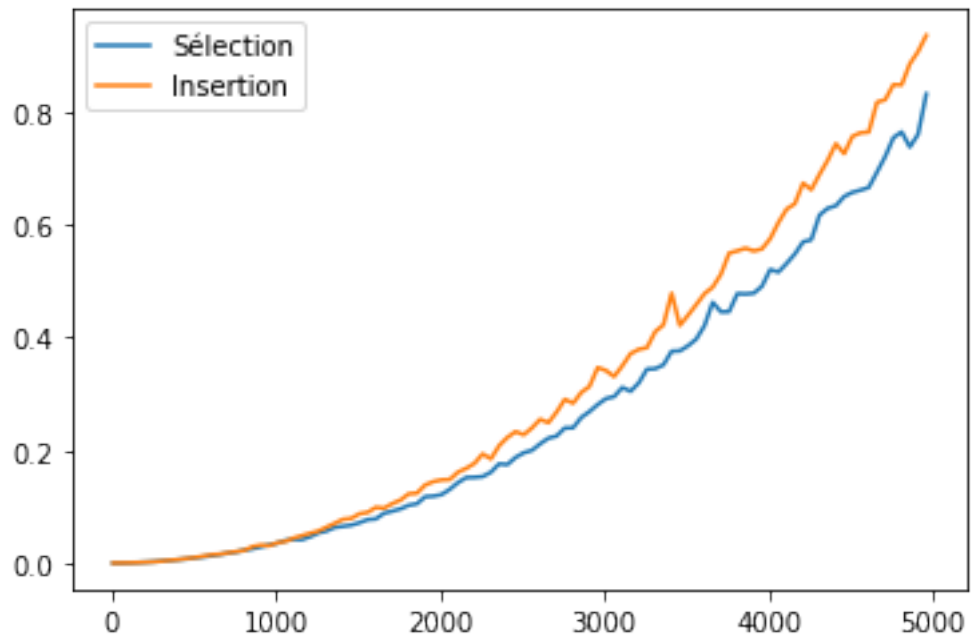
```
[6]: def tri_insertion(T):  
    n = len(T)  
    for i in range(1, n):  
        cle = T[i]  
        j = i - 1  
        while j >= 0 and T[j] > cle:  
            T[j+1] = T[j]  
            j = j - 1  
        T[j+1] = cle  
    return T  
  
[7]: # Vérification rapide qui n'est pas une preuve de correction  
T = [4, 6, -4, 2, 5, 3]  
T2 = tri_insertion(T)  
print(f'{T} est trié : {estTrie(T)}')
```

[-4, 2, 3, 4, 5, 6] est trié : True

```
[8]: import matplotlib.pyplot as plt  
from time import time  
  
tps_insertion = []  
for n in N:  
    T = tableau(n)  
    debut = time()  
    _ = tri_insertion(T)  
    tps_insertion.append(time() - debut)  
  
[9]: plt.plot(N, tps_insertion)  
plt.show()
```



```
[10]: plt.plot(N, tps_select, label="Sélection")  
plt.plot(N, tps_insertion, label="Insertion")  
plt.legend()  
plt.show()
```



```
[11]: T = tableau(5000)
      T2 = T[:]
      T_trie = tri_selection(T)
      T_trie_inv = T_trie[::-1]      # T_trie_inv = reversed(T_trie)
      deb = time(); _ = tri_insertion(T_trie); print(f"Meilleur des cas: {time() - deb}")
      deb = time(); _ = tri_insertion(T2); print(f"Un cas: {time() - deb}")
      deb = time(); _ = tri_insertion(T_trie_inv); print(f"Pire des cas: {time() - deb}")
```

Meilleur des cas: 0.0008358955383300781  
 Un cas: 0.9388899803161621  
 Pire des cas: 1.8763608932495117

```
[12]: T = tableau(5000)
      T2 = T[:]
      T_trie = tri_selection(T)
      T_trie_inv = T_trie[::-1]      # T_trie_inv = reversed(T_trie)
      deb = time(); _ = tri_selection(T_trie); print(f"Cas 1: {time() - deb}")
      deb = time(); _ = tri_selection(T2); print(f"Cas 2: {time() - deb}")
      deb = time(); _ = tri_selection(T_trie_inv); print(f"Cas 3: {time() - deb}")
```

Cas 1: 0.8015439510345459  
 Cas 2: 0.7993617057800293  
 Cas 3: 0.7835958003997803

## 1.6 Dictionnaire en compréhension

```
[13]: import string

      ch = string.ascii_lowercase
      L = [i for i in range(len(ch))]

      D = {k:v+1 for k, v in zip(ch, L)}
      print(D)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20, 'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26}
```