

Trier

INFORMATIQUE COMMUNE - TP n° 1.4 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ✍ coder un algorithme de tri simple et explicite
- ✍ évaluer le temps d'exécution d'un algorithme avec la bibliothèque `time`
- ✍ générer un graphique légendé avec la bibliothèque `matplotlib`

A Trier un tableau

- A1. On souhaite trier des listes Python, considérées ici comme des tableaux, avec des algorithmes différents (cf. algorithmes 1, 2 et 3). Chaque algorithme de tri est implémenté par une fonction Python. Le prototype de ces fonctions est `my_sort(t)`, où `t` est un paramètre formel qui représente le tableau à trier.

```
def my_sort(t):  
    # tri du tableau  
    # for i in range(len(t))  
    #     t[i] = ...
```

Cette fonction, une fois réalisée, trie le tableau `t` passé en paramètre mais ne renvoie rien (i.e. pas de `return`). Expliquer pourquoi.

- A2. Coder les algorithmes de tri par sélection, par insertion et par comptage en respectant le prototype défini à la question précédente¹.
- A3. Tester ces algorithmes sur une **même** liste Python de longueur 20 et contenant de types `int` choisis aléatoirement entre 0 et 100.
- A4. Peut-t-on trier des listes de chaînes de caractères avec ces mêmes codes? Tester cette possibilité à l'aide de la liste `["Zorglub", "Spirou", "Fantasio", "Marsupilami", "Marsu", "Samovar", "Zantafio"]`. Analyser les résultats. Pourquoi est-ce possible? Pourquoi n'est-ce pas possible?

La bibliothèque `matplotlib` permet de générer des graphiques à partir de données de type `list` qui constituent les abscisses et les ordonnées associées. La démarche à suivre est de :

- importer la bibliothèque `from matplotlib import pyplot as plt`
- créer une figure `plt.figure()`
- tracer une courbe `plt.plot(x,y)` si `x` et `y` sont les listes des abscisses et des ordonnées associées. La bibliothèque trace les points $(x[i], y[i])$ sur le graphique.
- ajouter les éléments de légende et de titre,
- montrer la figure ainsi réalisée `plt.show()`.

1. On a le droit de collaborer, de se répartir les algorithmes et de s'échanger les codes s'ils sont corrects!

Algorithme 1 Tri par sélection

```

1: Fonction TRIER_SELECTION( $t$ )
2:    $n \leftarrow \text{taille}(t)$ 
3:   pour  $i$  de 0 à  $n - 1$  répéter
4:      $\text{min\_index} \leftarrow i$                                 ▷ indice du prochain plus petit
5:     pour  $j$  de  $i + 1$  à  $n - 1$  répéter                    ▷ pour tous les éléments non triés
6:       si  $t[j] < t[\text{min\_index}]$  alors
7:          $\text{min\_index} \leftarrow j$                             ▷ c'est l'indice du plus petit non trié!
8:      $\text{échanger}(t[i], t[\text{min\_index}])$                     ▷ c'est le plus grand des triés!

```

Algorithme 2 Tri par insertion

```

1: Fonction TRIER_INSERTION( $t$ )
2:    $n \leftarrow \text{taille}(t)$ 
3:   pour  $i$  de 1 à  $n - 1$  répéter
4:      $\text{à\_insérer} \leftarrow t[i]$ 
5:      $j \leftarrow i$ 
6:     tant que  $t[j - 1] > \text{à\_insérer}$  et  $j > 0$  répéter
7:        $t[j] \leftarrow t[j - 1]$                                 ▷ faire monter les éléments
8:        $j \leftarrow j - 1$ 
9:      $t[j] \leftarrow \text{à\_insérer}$                                 ▷ insertion de l'élément

```

Algorithme 3 Tri par comptage

```

1: Fonction TRIER_COMPTAGE( $t, v_{\max}$ )                                ▷  $v_{\max}$  est le plus grand entier à trier
2:    $n \leftarrow \text{taille}(t)$ 
3:    $c \leftarrow$  un tableau de taille  $v_{\max} + 1$  initialisé avec des zéros
4:   pour  $i$  de 0 à  $n - 1$  répéter
5:      $c[t[i]] \leftarrow c[t[i]] + 1$                                 ▷ compter les occurrences de chaque élément du tableau.
6:    $\text{résultat} \leftarrow$  un tableau de taille  $n$ 
7:    $i \leftarrow 0$ 
8:   pour  $v$  de 0 à  $v_{\max}$  répéter                                ▷ On prend chaque valeur possible dans l'ordre
9:     si  $c[v] > 0$  alors                                          ▷ Si l'élément  $v$  est présent dans le tableau
10:      pour  $j$  de 0 à  $c[v] - 1$  répéter
11:         $\text{résultat}[i] \leftarrow v$                                 ▷ alors écrire autant de  $v$  que d'occurrences de  $v$ 
12:         $i \leftarrow i + 1$                                         ▷ à la bonne place, la ième!
13:   renvoyer  $\text{résultat}$ 

```

La bibliothèque `time` permet notamment de mesurer le temps d'exécution d'un code. Un exemple de code utilisant ces deux bibliothèques est donné ci-dessous. Le graphique qui en résulte est montré sur la figure 1.

Code 1 – Exemple d'utilisation des bibliothèques `time` et `matplotlib`

```
import time
from matplotlib import pyplot as plt

def to_measure(d):
    time.sleep(d) # Do nothing, wait for d seconds

# Simple use
tic = time.perf_counter()
to_measure(0.1)
toc = time.perf_counter()

print(f"Execution time : {toc - tic} seconds")

# Plotting results
timing = []
delay = [d / 1000 for d in range(1, 100, 5)]
for d in delay:
    tic = time.perf_counter()
    to_measure(d)
    toc = time.perf_counter()
    timing.append(toc - tic)

plt.figure()
plt.plot(delay, timing, color='cyan', label='fonction to_measure')
plt.xlabel('Delay', fontsize=18)
plt.ylabel("Execution time", fontsize=16)
plt.legend()
plt.show()
```

-
- A5. À l'aide de la bibliothèque `matplotlib`, tracer les temps d'exécution nécessaires au tri d'un même tableau d'entiers par les algorithmes implémentés. On pourra également les comparer à la fonction `sorted` de Python. Analyser les résultats. Essayer de qualifier les coûts des algorithmes en fonction de la taille du tableau d'entrée.



FIGURE 1 – Figure obtenue à partir des bibliothèques matplotlib et time et du code [1](#)

