1 Introduction

Le but de cet exercice est de coder des algorithmes de tris différents et de comparer leur complexité en temps.

2 Tri par sélection

2.1 Implémentation du tri par sélection

Le tri par sélection (ou tri par extraction) est un algorithme de tri par comparaison.
Sur un tableau de n éléments (numérotés de 0 à $n-1$), le principe du tri par sélection est le suivant :

1. rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
2. rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
3. continuer de cette façon (rechercher le i-ème élément et le placer en position $i-1$) jusqu'à ce que le tableau soit entièrement trié.

Question : Implémenter ce tri en Python. On vous conseille de d'écrire au moins deux fonctions :
— une fonction `minimum_element_index(array, start)` qui renvoie l'index dans `array` de l'élément minimum entre la position `start` et la fin du tableau.
— une fonction `sort(array)` qui trie le tableau par sélection.
Question : D'après vous, combien cet algorithme effectue de comparaisons ?

2.2 Tests en Python

Il est essentiel de tester le code que l'on produit pour être sûr qu'il en contient pas d'erreurs. En Python, on peut utiliser pytest : pour cela, les tests doivent être écrits dans un fichier dont le nom commence par `test_` (qui peut cependant importer d'autres fichiers Python...) et les fonctions contenant les tests à effectuer doivent également avoir un nom commençant par `test_`.

Par exemple, supposons qu'on souhaite tester la fonction suivante :

```
def double(entier):
  return 2 * entier
```

Pour tester ce code, on imagine que si les deux conditions suivantes sont remplies : `double(0)` vaut `0` et `double(21)` vaut `42`, la fonction est sans doute correcte.

On utilise le mot clé `assert` pour vérifier une propriété booléenne : si la propriété n'est pas vérifié, l'exécution s'arrête en signalant où se trouve le problème. Il suffit donc d'ajouter la méthode ci-après au fichier :

```
def test_double():
  assert double(0) == 0
  assert double(21) == 42
```

On peut alors soit exécuter à la main la fonction `test_double()`, soit simplement exécuter `pytest` dans la console de Pycharm (ou directement dans un terminal en se positionnant dans le répertoire du fichier).

Question : Comment peut-on se convaincre que l'algorithme de tri est bien implémenté et trie bien les tableaux ?

Question : Quelles autres parties du code pourrait-on tester ?

2.3 Mesurer la complexité du tri par sélection

On va maintenant mesurer la complexité de ce tri. Pour cela on va tracer une courbe avec sur l'axe des x les tailles des tableaux et sur l'axe des y le temps pour trier un tableau (généré aléatoirement) de cette taille.

2.3.1 Mesurer le temps de calcul

Pour mesurer le temps de calcul, on peut utiliser la fonction `process_time()` du module `time`. Cette fonction retourne un flottant. La différence des valeurs retournées entre deux appels de la fonction est égale au temps de calcul utilisé par Python. Le code suivant permet donc de récupérer le temps de calcul d'un code exécuté :

```
import time

def time_to_do_something():
    start = time.process_time()
    # Code à exécuter
    end = time.process_time()
    return end - start
```

Afin de tracer une courbe, votre premier but sera de générer une liste d'entier `lengths` allant de 10 à 100 allant par pas de 10 (vous pouvez faire varier ces valeurs par la suite). Cette liste représentera la liste des coordonnées sur l'axe x des points de la courbe que vous souhaitez tracer. Ensuite, vous devez calculer, pour chaque élément `length` de `lengths`, le temps pris pour trier un tableau aléatoire de taille `length`. Cette deuxième liste que l'on nommera `times` représentera la liste des coordonnées sur l'axe y des points de la courbe que vous souhaitez tracer. *Attention, le temps retenu ne doit contenir que le temps du tri, pas celui de la génération d'un tableau aléatoire, ni le temps des éventuels tests qu'on a écrit préalablement dans ce TP...*

Question : Écrivez le code générant ces deux listes.

Question : Que-peut-on dire sur les valeurs de la liste `times` ?

2.3.2 Générer des tableaux aléatoires

Dans le module `random` de Python, il existe une fonction `shuffle` qui mélange aléatoirement une liste sur place. Par exemple, le code ci-dessous affiche `[1, 3, 9]`, `[1, 9, 3]`, `[9, 3, 1]`, `[9, 1, 3]`, `[3, 1, 9]` ou `[3, 9, 1]` avec la même probabilité.

```
import random

array = [1, 3, 9]
random.shuffle(array)
print(array)
```

Question : Comment utiliser `shuffle` pour générer un tableau aléatoire de taille n que l'on devra trier ?

2.3.3 Tracer des courbes en Python

Pour tracer des courbes en Python, on va utiliser le module `matplotlib.pyplot`. Pour cela, il suffit de l'importer. Afin d'éviter de réécrire le nom du module en entier à chaque foix, on peut l'importer avec la ligne suivante :

```
import matplotlib.pyplot as plt
```

Les fonction utile pour ce TP du module `matplotlib.pyplot` sont les suivantes :

— `plt.plot(x, y, label=name)` quand `x` et `y` sont des listes de la même taille, génère dans le futur plot la courbe affine par morceaux qui passe par les point $(x[i], y[i])$. C'est la fonction usuelle pour tracer des courbes. Cette courbe aura pour étiquette `name`.
— `plt.xlabel(name)` : donne à l'axe x la légende `name`
— `plt.ylabel(name)` : donne à l'axe y la légende `name`
— `plt.legend()` : met les légendes sur les axes (à appeler après les deux fonction fixant les légendes des axes)
— `plt.show()` : affiche le plot (à appeler après tous les autres appels)

La documentation officielle de cette bibliothèque est au lien suivant : `https://matplotlib.org/index.html`

Question : Tracer la complexité de l'algorithme de sélection à partir des deux listes `lengths` et `times` générées dans la partie précédente.

Question : Est-ce que la courbe obtenue correspond à ce que vous attendiez ?

3 Tri fusion

3.1 Implémentation du tri fusion

Le tri fusion est un algorithme de tri par comparaison.

À partir de deux listes triées, on peut facilement construire une liste triée comportant les éléments issus de ces deux listes (leur *fusion*). Le principe de l'algorithme de tri fusion repose sur cette observation : le plus petit élément de la liste à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste. Ainsi, on peut construire la liste élément par élément en retirant tantôt le premier élément de la première liste, tantôt le premier élément de la deuxième liste.

L'algorithme est naturellement décrit de façon récursive.

1. Si le tableau n'a qu'un élément, il est déjà trié.
2. Sinon, séparer le tableau en deux parties à peu près égales.
3. Trier récursivement les deux parties avec l'algorithme du tri fusion.
4. Fusionner les deux tableaux triés en un seul tableau trié.

Question : Implémenter ce tri en Python. On vous conseille de d'écrire au moins les trois fonctions suivantes :
— Une fonction `merge_sub_arrays` qui fusionne les deux parties du tableau.
— Une fonction `recursive_sort(array, start, stop, temp_array)` qui trie la partie du tableau entre `start` (inclus) et `stop` (exclu).
— Une fonction `sort(array)` qui trie le tableau par le tri fusion.

Question : Peut-on effectuer ce tri sur place, c'est-à-dire sans utiliser un ou plusieurs autres tableaux ?

Question : D'après vous, combien cet algorithme effectue de comparaisons ?

3.2 Comparer la complexité des deux tris

Question : Tracer dans le même graphique la complexité du tris par sélection et celle du tri fusion. Pour cela, vous pouvez simplement dessiner deux courbes dans le même `plot` avec deux étiquettes (`label`) différentes.

Question : Comment se comporte les deux courbes l'une par rapport à l'autre ?

4 Tri rapide

4.1 Implémentation du tri rapide

Le tri rapide est un algorithme de tri par comparaison.

La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. On choisit le pivot au hasard parmi toutes les positions possibles.

Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié, c'est-à-dire que la taille des sous-tableaux est un et qu'ils sont donc automatiquement triés.

Concrètement, pour partitionner un sous-tableau :

1. le pivot est placé à la fin (arbitrairement), en l'échangeant avec le dernier élément du sous-tableau ;
2. tous les éléments inférieurs au pivot sont placés en début du sous-tableau ;
3. le pivot est déplacé à la fin des éléments déplacés.

4.2 Comparer la complexité des trois tris

Question : Tracer dans le même graphique les complexités des trois tris.
Question : Comment se comportent les trois courbes ?