TRAVAUX PRATIQUES n° 12
Algorithmes de tris

Exercice 1 (*Tri par sélection*) :

1) Écrire une fonction `tri_par_selection(L)` programmant l'algorithme du tri par sélection pour trier une liste de nombres dans l'ordre croissant en s'aidant du pseudo-code suivant :

```
"""
def tri_par_selection(L) :
    on note n = longueur de L
    pour tout i allant de 0 à n-1 (bornes incluses) :
        - trouver la position j de l'élément le plus petit de L[i:n] dans L
        - échanger L[i] et L[j]
    retourner L
"""
```

2) Écrire une fonction `tri_par_selection_decroissant(L)` adaptant l'algorithme précédent pour trier une liste de nombres dans l'ordre décroissant.

Exercice 2 (*Tri par insertion*) :

1) Écrire une fonction `insertion(L, i)` qui prend en arguments une liste `L` et un indice `i` tel que $0 \leqslant$ `i` $<$ `len(L)` et qui insère l'élément `L[i]` à sa place parmi les `i` premiers éléments. Cette fonction ne doit rien renvoyer mais doit modifier la liste `L`.

   Par exemple, si `L = [1, 2, 4, 6, 3, 5]`, alors

```
>>> insertion(L, 4)
>>> print(L)
[1, 2, 3, 4, 6, 5]
>>> insertion(L, 5)
>>> print(L)
[1, 2, 3, 4, 5, 6]
```

2) Écrire une fonction `tri_par_insertion(L)` programmant l'algorithme de tri par insertion en utilisant la fonction `insertion` précédente.

Exercice 3 (*Tri par comptage*) :

1) Écrire une fonction `effectifs(L, n)` qui prend en arguments une liste `L` et un nombre entier positif `n` et qui renvoie la liste des effectifs des valeurs entre `0` et `n` dans la liste `L`.

   Par exemple, si `L = [3, 1, 2, 5, 3, 1, 5, 4, 4, 4, 5]`, alors

```
>>> effectifs(L, 6)
[0, 2, 1, 2, 3, 3, 0]
```

2) Écrire une fonction `tri_par_comptage(L, n)` programmant l'algorithme de tri par comptage en utilisant la fonction `effectifs` précédente.

Exercice 4 (*Recherche par dichotomie*) :

1) Écrire une fonction `recherche(L, e)` qui prend en argument une liste `L` et un nombre `e` et qui renvoie `True` si `e` est un élément de `L` et `False` sinon (sans utiliser la syntaxe `if e in L`).

2) Lorsqu'une liste est triée dans l'ordre croissant, on peut rendre la fonction précédente beaucoup plus efficace en faisant une recherche par dichotomie. Écrire une fonction `recherche_par_dichotomie(L, e)` qui prend en argument une liste triée `L` et un nombre `e` et qui renvoie `True` si `e` est un élément de `L` et `False` sinon. On utilisera le pseudo-code suivant :

```
"""
def recherche_par_dichotomie(L, e) :
    debut = 0
    fin = len(L)-1
    tant que debut <= fin :
        milieu = (debut+fin)//2
        si L[milieu] est égal à e
            retourner vrai
        sinon si L[milieu] < e
            debut = milieu+1
        sinon
            fin = milieu-1
    retourner faux
"""
```

3) Pour comparer les deux algorithmes, taper dans le shell les commandes suivantes :

```
>>> import numpy as np
>>> L = np.random.randint(0, 10**9, 10**8)  # Crée une liste aléatoire de 100
   millions d'éléments
>>> recherche(L, 1)                         # Cette instruction devrait prendre
   quelques secondes
>>> L.sort()                                # Cette instruction devrait aussi
   prendre quelques secondes
>>> recherche_par_dichotomie(L, 1)          # Cette instruction devrait être
   instantanée
```

Exercice 5 (*Tri par ordre alphabétique*) :

1) Écrire une fonction `plus_petit(chaine1, chaine2)` prenant en arguments deux chaines de caractères et qui renvoie `True` si `chaine1` est plus petit que `chaine2` pour l'ordre alphabétique, et `False` sinon.

   On pourra utiliser l'instruction `alphabet = [chr(k) for k in range(97, 123)]` *pour créer une liste contenant toutes les lettres de l'alphabet dans l'ordre sans avoir à les écrire.*

2) Écrire une fonction `tri_mot(L)` prenant en argument une liste de chaines de caractères et qui renvoie la liste triée dans l'ordre alphabétique.

   On utilisera le tri de son choix.

3) Écrire une fonction `plus_grand_anagramme(mot)` prenant en argument une chaîne de caractères mot et qui renvoie le plus grand anagramme de mot pour l'ordre alphabétique.

Exercice 6 (*Tri rapide pour ceux qui vont très vite !*) :

Écrire une fonction `tri_rapide(L)` programmant l'algorithme de tri rapide à l'aide du pseudo-code suivant :

```
"""
def tri_rapide(L) :
    on enlève le premier élément de L et on le note pivot
    petits = []
    grands = []
    pour tout élément e de L :
    si e < pivot
        on ajoute e à la liste petits
    sinon
        on ajoute e à la liste grands
    on trie les listes petits et grands à l'aide de la fonction tri_rapide
    on reconstitue la liste L dans l'ordre croissant que l'on renvoie
"""
```