DM2 : Recherche séquentielle et tris

Cette série d'exercices porte sur des révisions de recherche séquentielle et des algorithmes de tri.

Exercice 1

Ecrire une **fonction rechercheLineaire** qui prend en paramètre une liste de chaines de caractères *L* et une chaine de caractères *mot*.
La fonction renvoie l'indice de la dernière occurrence de *mot* dans *L*.
Si *mot* n'est pas trouvé dans *L*, la fonction retourne -1.

Entrée[10]:

```
def rechercheLineaire( L, mot) :
    """
        Entrées : L, liste de chaînes de caractères,
                : mot, chaîne de caractère que l'on cherche dans L
        Sortie : entier qui correspond à l'indice de la derniere
             occurence de mot dans L si mot est present dans l,
                   Si le mot n est pas trouve la fonction renvoie -1
    """
    # A compléter



# test de la fonction
L=["Bonjour","Bonjour", "tout", "le", "monde", "Bonjour"]
print(L)
print( rechercheLineaire(L, "Bonjour") ) # affiche 5
print( rechercheLineaire(L, "Bonsoir") ) # affiche -1
print( rechercheLineaire(L, "le") )      # affiche 3
```

```
['Bonjour', 'Bonjour', 'tout', 'le', 'monde', 'Bonjour']
5
-1
3
```

Exercice 2. Calcul de la somme maximale

Définir 2 **fonctions sommeMaximale et sommeMaximaleLineaire** renvoyant la plus grande somme de 2 éléments contenus dans une liste d'entiers L passée en paramètre.
sommeMaximale correspondra à un algorithme naïf de complexité temporelle quadratique O(n2).
sommeMaximaleLineaire proposera une solution de complexité temporelle linéaire O(n).

Entrée[4]:

```
import numpy as np
def sommeMaximale( L: list )-> int :
    """ sommeMaximale( L: list )-> int :
           de complexité quadratique
        Entrée : L, une liste d'entiers
        Sortie : maxi, max(e1 + e2, (e1,e2) dans L), m vaut 0 si L est vide
    """
    # A compléter

L=[0,5,2,3,4,2]
print( sommeMaximale(L) )

def sommeMaximaleLineaire( L : list ) -> int :
    """
               de complexité lineaire
        Entrée : L, liste d'entiers
        Sortie : max(e1 + e2, (e1,e2) dans L), 0 si L est vide.
        On calcule le maximum et le second maximum de la liste, puis on les som
    """
    # A compléter

L=[0,5,2,3,4,2]
print( sommeMaximaleLineaire( L ) )
```

```
9
9
```

Exercice 3: Fusion de listes

Définir **une fonction fusionListe** qui prend en parametre une liste de listes d'entiers L et renvoie la concaténation de l'ensemble des sous-listes.
exemple.

Entrée[28]:

```
def fusionListe(L) :
    """ Entrée : L, liste de listes d'entiers
        Sortie : Lres, liste d'entiers correspondant à la concaténation de l'en
        des sous-listes de L
    """
    # A compléter

LC=[ [1,2], [7,8,9], [1,2,3,4] ]
print( fusion(LC)) # affiche [1, 2, 7, 8, 9, 1, 2, 3, 4]
```

```
[1, 2, 7, 8, 9, 1, 2, 3, 4]
```

Exercice 4: Fusion de dictionnaires

Définir une **fonction fusionDico** qui prend en paramètre une liste L contenant des dictionnaires dont les clés sont des chaînes de caractère et les valeurs des listes d'entiers.
La fonction retourne un dictionnaire tel que l'ensemble de clés est exactement l'ensemble des clés de tous les dictionnaires de la liste d'entrée et la valeur associée à une clé est la concaténation des valeurs associées à cette clé dans chaque dictionnaire où la clé est définie.

Entrée[7]:

```
def fusionDico( L : list)-> dict :
    """
        Entrées : L, Une liste de dictionnaires (cle : chaine , valeur : liste
        Sortie : dicoRes obtenu en aplatissant la liste...
    """
    # A compléter

# test de la fonction
L = [
        {'math' : [ 8,12 ], 'physique' : [ 14 ] },
        {'math' : [ 6,9,8,9 ], 'LV1' : [ 11 ] },
        {'LV1' : [ 7,6 ]}
]
print( fusionDico(L))
# affiche {'math': [8, 12, 6, 9, 8, 9], 'physique': [14], 'LV1': [11, 7, 6]}
```

```
{'math': [8, 12, 6, 9, 8, 9], 'physique': [14], 'LV1': [11, 7, 6]}
```

Dichotomie

Exercice 5 : Recherche Dichotomique

Implémenter une **fonction rechercheDichotomique** renvoyant l'indice de la dernière occurrence d'un mot dans une liste de mots triés par ordre alphabétique, avec une complexité logarithmique.
Justifier la complexité.

Entrée[1]:

```
def rechercheDichotomique(l,x) :
    """
        Entrées :  l, liste de chaînes de caractères triée par ordre alphabétiq
                   x, chaîne de caractères
        Sortie : l'indice de x dans l si x est dans l, -1 sinon
    """
    # A compléter

#test de la fonction
L= ["abecedaire", "alias", "Alsace", "amarre","arc","arrivee" ]
print( rechercheDichotomique(L,"arc") )    # affiche 4
print( rechercheDichotomique(L,"alinea") ) # affiche -1
```

```
4
-1
```

Prouver la correction de votre programme et justifier sa complexité

Tris

L'objectif de cette partie est de réviser les algorithmes de tri rencontrés en 1ère année. Ces algorithmes tombent très souvent lors des concours : il faut soit les reconnaitre, soit les implémenter intégralement, soit compléter du code déjà fourni.
Il ne sert à rien de recopier tel quel le code produit l'an dernier. Vous perdriez votre temps et le mien. L'idée ici est d'essayer de les recoder, en regardant les principes si vous ne vous en souvenez plus.
En vue des concours, je vous conseille très fortement de vous faire un tableau de synthèse, si vous ne l'avez pas encore fait, que vous pourrez relire avant les épreuves. Ce tableau constituerait une synthèse des tris rencontrés l'an dernier avec le principe, le code, la complexité dans le meilleur et dans le pire des cas, caractéristiques (tris en place ou pas? stable? ...).

Exercice 6: Tri par comptage

Ecrire une **fonction triComptage** qui prend en paramètre une liste d'entiers et une borne supérieure de l'ensemble des valeurs de la liste et renvoie une copie de la liste triée en temps

Entrée[10]:
```
def triComptage(tab, borneSuperieure):
    # A compléter

#test de la fonction
L = [1, 15, 3, 1, 3]
print(triComptage([1, 15, 3, 1, 3], max(L))) #affiche [1, 1, 3, 3, 15]
```

```
[1, 1, 3, 3, 15]
```

Exercice 7 : Tri fusion

Définir la **fonction triFusion** qui paramètre en entrée une liste d'entier et renvoie une liste triée selon l'algorithme du tri partition-fusion.

Entrée[4]:
```
def triFusion( L ):
    """ triFusion( L : list )
        entrees : L, liste
        sortie : listre triee par tri fusion
    """
    # A compléter

#test de la fonction
print( triFusion( [4, 7, 3, 9, 1, 2, 5]))#affiche [1, 2, 3, 4, 5, 7, 9]
```

```
[1, 2, 5, 9]
[1, 2, 3, 4, 5, 7, 9]
```

Exercice 8 : Tri rapide

Définir la fonction triRapide qui prend en entrée un liste d'entier et renvoie une liste triée selon l'algorithme du tri rapide, en utilisant un pivot aléatoire.
Pour la génération aléatoire, vous pouvez utiliser la fonction randint du module random.
randint(start, stop) génère un entier entre start et stop inclus.

Entrée[3]:
```
def triRapide ( L ):
    """ triRapide( L : list )
        entree : L, liste
        sortie : liste triee avec le tri rapide
    """
    # A compléter

# test de la fonction
print ( triRapide([5, 2, 1, 5, 89, 3]) )
```

```
[1, 2, 3, 5, 5, 89]
```