Dans ce TP, nous verrons quelques méthodes de tris de listes ainsi que certaines applications.

I Algorithmes de tris

On cherche à créer un algorithme pour trier une liste de nombres, c'est-à-dire une façon d'ordonner les valeurs d'une liste L dans l'ordre croissant. L'enjeu en général est de le faire avec un temps d'exécution qui soit le plus court possible, c'est-à-dire en faisant le moins d'opérations possibles. On commence par une fonction auxiliaire utilisé dans l'une des méthodes de tris.

Exercice 1: Ecrire une fonction ***maximum*(L)** qui recherche l'élément maximum (*maxi*) d'une liste L ainsi que sa place (*maxind*) dans la liste.
Cette fonction renverra *maxi* et *maxind*.
Comparer avec la fonction ***max*** de Python.

I.1 Tri par sélection :

trier une liste L, c'est échanger le plus grand élément de L avec l'élément à la dernière place et recommencer sur la sous-liste formée par L modifiée auquel on a retiré le dernier élément.

C'est la méthode de tri courante pour le tri manuel d'un petit nombre d'éléments, employée par exemple pour trier un paquet de n copies : on cherche le plus grand élément et on le place en queue de paquet en l'échangeant avec le dernier élément. Il ne reste plus qu'à recommencer avec le paquet restant des $n-1$ premières copies.

Description de l'algorithme :
$k \leftarrow n-1$
Tant que $k > 0$
    Déterminer l'indice, *maxind*, du plus grand élément parmi $L[0]$ à $L[k]$
    Permuter $L[maxind]$ et $L[k]$
    $k \leftarrow k-1$
FinTantque

Pour déterminer *maxind*, il vous suffira d'exploiter la fonction maximum écrite ci-dessus.

Exemple : Si L=[4, 5, 1, 2], l'algorithme va agir ainsi :
•$k = 1$, $maxind = 1$. On échange $L[maxind]$ avec $L[3]$, L devient $[4, 2, 1, 5]$
•$k = 2$, $maxind = 0$. On échange $L[maxind]$ avec $L[2]$, L devient $[1, 2, 4, 5]$
•$k = 3$, $maxind = 1$. On échange rien du tout, L était déjà trié.

Comprendre l'algorithme, puis écrire la fonction ***triselection*(L)** qui renvoie une liste triée.
Tester son efficacité en l'utilisant dans le **shell**.
Installer temporairement dans la fonction suffisamment d'affichages intermédiaires pour visualiser le processus de tri exploité.

I.2 Tri par insertion :

trier une liste L c'est insérer un à un les éléments en bonne place dans le tri des autres.

Le tri par insertion est la méthode naturellement employée lorsque le tri se fait au fur et à mesure de la saisie des données, par exemple le tri de cartes à jouer qui sont distribuées une à une . Si les $k-1$ premières cartes en main sont déjà triées, il s'agit de placer la k-ième. Il faut alors d'abord déterminer sa place parmi les k cartes puis faire les décalages nécessaires à son placement.

Pour des données stockées dans une liste L à n éléments il s'agit, en commençant par le deuxième élément, de considérer chaque élément comme la valeur que l'on souhaite insérer. On regarde ensuite dans les éléments qui le précède celui qui est plus petit et on décale les autres à partir de cet élément.
On obtient ainsi l'algorithme suivant :

T=$[0, \cdots, 0]$
T[0] $\leftarrow$ L[0]
Pour $k = 1$ à $n-1$
    Element$\leftarrow L[k]$
    $p \leftarrow k-1$
    Tant que $p >= 0$ et $T[p] > Element$
        Décaler T[p] vers la droite
        $p \leftarrow p-1$
    FinTantque
    Placer Element au bon endroit FinPour

Exemple : Si L=[4, 5, 1, 3], l'algorithme va agir ainsi :
•$k = 1$, alors Element=5 et comme L[1] >L[0], on laisse Element en place 1, T=[4, 5, 0, 0]

•$k = 2$, alors Element=1, $p = 1$ et l'on va entrer deux fois dans la boucle tant que, si bien que T devient T=[1, 4, 5, 0]
•$k = 2$, alors Element=3, $p = 2$ et l'on va entrer deux fois dans la boucle tant que ($3 < 5$, $3 < 4$, mais $3 > 1$), si bien que T devient T=[1, 3, 4, 5] et le tableau est trié.

Comprendre l'algorithme, puis écrire la fonction ***triinsertion*(L)** qui renvoie une liste triée.
Tester son efficacité en l'utilisant dans le **shell**.
Installer temporairement dans la fonction suffisamment d'affichages intermédiaires pour visualiser le processus de tri exploité.

I.3 Tri par comptage :

on veut trier une liste L qui contient les nombres de 0 à p
L'idée est de compter le nombre d'occurrences de chaque nombre.
•On crée un tableau C de $p + 1$ valeurs, initialisé avec des 0
•On traverse L et on compte le nombre de fois où chaque élément est présent en incrémentant le tableau C à la place de l'élément
•Ensuite on balaie le tableau C et on copie autant de fois une valeur qu'elle apparait dans C

Exemple : si $L = [1, 5, 6, 5, 5, 6, 3]$, on peut compter :

0	1	2	3	4	5	6
0	1	0	1	0	3	2

On a donc : $C = [0, 1, 0, 1, 0, 3, 2]$ et on obtient le tableau trié : $T = [1, 3, 5, 5, 5, 6, 6]$

I.4 Tri par bulle :

Tant qu'il existe un couple d'éléments consécutifs qui n'est pas dans le bon ordre, parcourir la liste en échangeant les éléments consécutifs qui ne sont pas dans le bon ordre.

Exemple :

3	4	**8**	1	7	2	**9**	5	liste de départ
3	**4**	1	**7**	2	**8**	5	9	8 et 9 se déplacent vers la droite
3	1	4	2	7	5	8	9	4, 7 et 8 remontent
1	3	2	**4**	5	**7**	8	9	3, 4 et 7 remontent
1	2	3	4	5	7	8	9	liste triée

On commence par parcourir la liste L en comparant deux par deux les éléments successifs de L, si L[k] >L[$k + 1$] alors on permute ces deux éléments. Ainsi lorsque k aura parcouru les valeurs de 1 à $n - 1$, L[n] contiendra le plus grand élément de L !
Il reste à réitérer le procédé sur L pour k allant de 1 à $n - 2$, puis de 1 à $n - 3$, etc...

Description de l'algorithme :
T← L # On duplique L
Pour $j = 0$ à $n - 2$
    Pour $k = 0$ à $n - 2 - j$
        si T[k] > T[$k + 1$] alors
        permuter T[k] et T[$k + 1$]
    FinPour
FinPour

Comprendre l'algorithme, puis écrire la fonction ***tribulle*(L)** qui renvoie une liste triée.
Tester son efficacité en l'utilisant dans le **shell**.
Installer temporairement dans la fonction suffisamment d'affichages intermédiaires pour visualiser le processus de tri exploité.

Amélioration : Si L est déjà triée la boucle sur n s'exécute néanmoins ! Or il est possible de détecter que la liste est finalement triée, en effet lors du parcours sur L il n'y aura alors aucune permutation !

Installer dans ***tribulle*** une variable Stop, dite " drapeau ", de valeur 1 en début de boucle sur n, et prenant la valeur 0 dès qu'une permutation est faite.
Modifier alors la boucle Pour sur j en boucle Tant que et rajouter une condition sur le drapeau pour éviter de parcourir la boucle lorsque le tableau est trié.
Tester l'efficacité de cette modification.

On peut mesurer le temps d'exécution d'un programme avec les instructions suivantes :

from time import perf_counter
from random import shuffle Dans le **shell** :
☞ L=list(range(1,10000))
☞ shuffle(L) ☞ t1=perf_counter() ;T=***tribulle***(L) ;t2= perf_counter() ;print(t2-t1)

Comparer avec la fonction ***sorted*** de numpy.

II Applications

II.1 Recherche dichotomique

Soit une liste L, contenant plusieurs éléments. On désire savoir à quel endroit se trouve un élément particulier x dans L.
Lorsque l'on recherche un élément dans un tableau, si celui-ci est dans un ordre aléatoire,

il faut examiner chaque élément.
Par contre, si le tableau est trié, on peut en tirer profit pour aller beaucoup plus vite : c'est l'algorithme de recherche dichotomique qui fait l'objet de cet exercice.

Au démarrage, on cherche x dans toute la liste. Si on note n = len(L), alors l'intervalle de recherche commence à l'indice debut = 0 et finit à l'indice fin = n-1.
L'idée est de diviser par deux l'intervalle de recherche, en regardant la valeur prise par l'élément en milieu d'intervalle (d'indice milieu = (debut+fin)//2).
Pour trouver l'algorithme, posez-vous la question de ce qu'il faut faire pour définir le nouvel intervalle de recherche si :
•x = L[milieu] ?
•x < L[milieu] ?
•x > L[milieu]

Exercice 2: Ecrire la fonction ***recherche_tri*(L,x)** qui, étant donnée une liste L (triée) et un élément x (nombre en général, mais on peut imaginer une lettre) renverra un indice correspondant à la place de x s'il est effectivement dans la liste et **False** sinon.
Tester cette fonction avec des listes **triées**, par exemple $w = list(range(1, 1000, 7))$.
On pourra à ce propos écrire une fonction qui teste si une liste donnée L de réels est bien ordonnée.

II.2 Calcul de la médiane

La médiane d'une liste est la valeur qui se trouve précisément au milieu de la liste, une fois celle-ci triée (dans le cas où la liste est de longueur paire, la médiane est la moyenne entre les deux valeurs du milieu de la liste). Par exemple :
•si L=[4, 5, 88, 74, 15, 2, 61], la médiane est 15.
•si L=[4, 5, 88, 74, 15, 2, 61, 23], la médiane est $\frac{15+23}{2} = 19$.

Exercice 3: Ecrire une fonction ***mediane*(L)** qui étant donnée une liste de nombres **L** renvoie la médiane de cette liste.

Exercice 4: Ecrire une fonction ***medianeffectif*(L,E)** qui étant donnée une liste de nombres **L** triée et une liste **E** correspondants aux effectifs de chaque modalité de **L** renvoie la médiane de cette série statistique.
Comment faire si **L** n'est pas triée ?