

TP6 Python

TRIS DE LISTES

1 Tris naïfs

Dans toute la suite, L désigne une liste de nombres réels que l'on souhaite trier (par ordre croissant).

Comme on souhaite garder en mémoire la liste initiale, on **commencera toujours** par dupliquer la liste L dans une nouvelle variable.

1.1 Le tri comptage

Le tri par comptage est une méthode plutôt simple à mettre en œuvre.

Pour une liste sans répétition. Si on note C une copie de la liste L à trier, cette méthode consiste à parcourir L et à compter pour chaque élément $L[i]$ le nombre d'éléments n_i de L qui sont strictement plus petits.

On place alors dans C la valeur $L[i]$ en position n_i : $C[n_i] = L[i]$.

Exemple. On considère la liste $L = [12, -3, 17, 9, 1]$.

On commence par créer une liste C de la même taille que L qui contiendra à terme les éléments de L triés :

```
1 C = [0 for i in range(len(L))]
```

— On place $L[0]=12$: $n_0 = 3$ donc $C[3]=L[0]$:

```
1 C = [0, 0, 0, 12, 0]
```

— On place $L[1]=-3$: $n_1 = 0$ donc $C[0]=L[1]$:

```
1 C = [-3, 0, 0, 12, 0]
```

— On place $L[2]=17$: $n_2 = 4$ donc $C[4]=L[2]$:

```
1 C = [-3, 0, 0, 12, 17]
```

— ...

Travail demandé

1. Écrire une fonction `Tri_comptage1(L)` qui prend en entrée une liste L sans répétition et renvoie la liste obtenue en triant L .
2. Adapter la tri comptage pour les listes avec répétition et écrire une fonction `Tri_comptage2(L)` qui prend en entrée une liste L avec répétitions et renvoie la liste obtenue en triant L .
3. Tester avec la liste $[3.5, -2.1, 4, -2.1, 0, 1, 1, 9, 8, 4, -2]$.

1.2 Tri par insertion

Le tri par insertion est généralement le tri que l'on utilise instinctivement pour classer des documents *etc.*

On note C une liste vide dans laquelle on mettra les éléments de L que l'on a trier.

1. On commence par regarder le premier élément de L qu'on place temporairement en position 0 dans C .
2. Ensuite, on considère le deuxième élément $L[1]$ de L . S'il est inférieur à $C[0]$, on l'insère juste avant. S'il est supérieur, on l'insère juste après.
3. On considère ensuite le troisième élément de L et on cherche à le placer en parcourant à nouveau les deux éléments déjà triés. S'il est inférieur à $C[0]$, on l'insère juste avant ; sinon s'il est inférieur à $C[1]$, on l'insère juste avant ; sinon on l'insère juste après.
4. ...

Exemple. On considère la liste $L=[12, -3, 17, 9, 1]$.

On commence par créer une liste C de la même taille que L qui contiendra à terme les éléments de L triés :

```
1 C = []

— On place temporairement C[0]=12 à sa place :
1 C = [12]

— On compare L[1]=-3 à L[0]. Comme L[1]<L[0], on l'insère avant :
1 C = [-3, 12]

— On place L[2]=17 en le comparant à L[0] et L[1]. Comme L[2]>L[1], on l'insère
juste après : :
1 C = [-3, 12, 17]

— ...
```

Travail demandé

1. Écrire une fonction `Tri_insertion(L)` qui prend en entrée une liste L sans répétition et renvoie la liste obtenue en triant L avec le tri par insertion.
Indication : on pourra utiliser la commande `L.insert(i, x)` qui insère la valeur x à la position i dans la liste L .
2. Tester avec la liste L de l'exemple et la liste $[3.5, -2.1, 4, -2.1, 0, 1, 1, 9, 8, 4, -2]$.

1.3 Le tri par sélection

Le tri par sélection est une méthode passe cette fois-ci par la recherche d'un élément maximum. Elle se fait directement par échanges à l'intérieur de la liste L (ou d'une copie C).

En notant n la longueur de L .

1. On commence par chercher le plus grand élément de la liste C qu'on échange avec le dernier élément.

2. On cherche le maximum des $n-1$ premiers éléments de C et on l'échange avec l'avant dernier élément de C .
3. ...

Exemple. On considère la liste $L=[12,-3,17,9,1]$.

On commence par créer une copie C de L :

- On cherche le plus grand élément de C et on l'échange avec le dernier élément :

```
1 C = [12, -3, 17, 9, 1]
```

- On cherche le maximum des $n-1$ premiers termes et on l'échange avec l'avant dernier :

```
1 C = [9, -3, 17, 12, 1]
```

- On cherche le maximum des $n-2$ premiers termes et on l'échange avec le terme d'indice $n-3$:

```
1 C = [1, -3, 9, 12, 17]
```

- ...

Travail demandé

1. Écrire une fonction `ind_max(L,n)` qui renvoie l'indice du plus grand élément parmi les n premiers éléments de la liste L .
2. En déduire une fonction `Tri_selection(L)`.

2 Tri rapide

Le tri rapide ou tri fusion est une méthode de tri plus rapide que les précédentes.

L'idée est de prendre un élément de la liste (pour faire simple : le premier) que l'on appelle pivot et de le mettre à sa place définitive en plaçant à gauche tous les éléments qui sont plus petits et à droite tous ceux qui sont plus grands.

On recommence ensuite le tri de la même façon sur les deux sous-listes obtenues.

Exemple. On considère la liste $L=[10,1,5,19,3,3,2,17]$.

1. On choisit 10 comme premier pivot.
2. On construit la liste des éléments plus petits (strictement) que 10 et de ceux plus grands que 10, ici : $[1,5,3,3,2]$ et $[19,17]$.
3. On recommence par récursivité le classement sur les sous-listes $[1,5,3,3,2]$ et $[19,17]$, ce qui donne $[1,2,3,3,5]$ et $[17,19]$.
4. On remet les listes classées dans le bon ordre : $[1,2,3,3,5, 10, 17,19]$.

Travail demandé

1. Écrire une fonction qui prend en entrée une liste L et qui construit les parties gauche et droite de L avec $L[0]$ comme pivot.
2. En déduire une fonction triant une liste par la méthode de tri rapide. On utilisera le principe de récursivité.

3 Comparaison des vitesses

Avec la fonction `time()` du module `time`, on peut déterminer le temps d'exécution d'un programme :

```
1 from time import time
2
3 def vitesse(f,L):
4     debut = time()
5     f(L)
6     fin = time()
7     return fin-debut
```

Travail demandé

1. Écrire une fonction liste `liste_alea(n)` qui construit une liste aléatoire de `n` nombres entiers compris dans $\llbracket 0, 5n \rrbracket$.
2. Écrire une fonction `Temps_moyen(tri,n)` qui prend en entrée l'une des fonctions de tri définies précédemment et un entier `n` et qui renvoie le temps moyen mis pour trier 10 listes de `n` nombres.
3. Pour chacune des méthodes de tri, créer une liste contenant les temps moyens pour trier une liste de n nombres pour $n \in \{10, 50, 100, 500, 1000, 2000\}$.
4. Pour chacune des méthodes de tri, tracer le graphique représentant le temps d'exécution en fonction de la taille de la liste.
Quelles sont les méthodes qui semblent le plus efficace ?