Rappels et compléments sur les listes

Étant donnée une liste `L`, `L.append(3)` rajoute l'élément `3` à la fin de la liste `L`. De plus, `L.pop()` enlève le dernier élément de la liste, la commande `a=L.pop(5)` enlève l'élément d'indice 5 de la liste `L` et la variable `a` prend pour valeur l'élément retiré.

Exercice 1. Essayez de prédire la liste `L` après les commandes suivantes puis tester sur Python.

```
L=[8,9,8,7,2,3,10]
L.append(15)
L.pop(2)
L.pop()
```

Étant données deux listes `L` et `M`, la commande `L+M` réalise la concaténation des deux listes. Essayez `N=[1,2,3]+[2,8,9]`, ou bien `P=[1,2,3]*4`.
Une chose surprenante en Python, si `L` est une liste, et si la commande `M=L` est rentrée, alors une modification de `M` entraîne une modification de `L` (et vice-et-versa). Si ce n'est pas le comportement désirée, et que vous voulez faire évoluer `L` et `M` de façon indépendante, la commande `M=L.copy()` pare le problème [1].

Exercice 2. Prédire les deux `print` suivants **avant** de taper le code.

```
L=[1,2,3]
M=L
M[1]=50
print(L,M)

L=[1,2,3]
M=L.copy()
M[1]=50
print(L,M)
```

Rappelons enfin qu'une fonction peut modifier une liste même sans `return` c'est ce qu'on appelle un **effet de bord**.

Exercice 3. Prédire l'issue du code suivant, puis tester-le.

```
def Fonction(L):
    L.pop(0)
    M=L*3
    return M


L=[1,2,3]
N=Fonction(L)
print(L,N)
```

Généralités sur les tris de listes

- Trier une liste consiste à renvoyer une liste qui contient les mêmes éléments (avec même occurrence) de la liste initiale mais rangés de façon croissante.
- On dit qu'un algorithme de tri est **en place** s'il modifie directement les éléments de la liste qu'il est en train de trier. Cela peut être important si on ne dispose pas de beaucoup de mémoire.
- On dit qu'un tri est dit **stable** s'il ne permute pas deux éléments égaux.

Un algorithme de tri : le tri fusion

Exercice 4 (𝄞⋆⋆). 1. Écrire une fonction `Fusion(L,M)`, où `L` et `M` sont deux listes triées par ordre croissant. Cette fonction renvoie la liste contenant les éléments de `L` et `M` de façon triée. Ainsi, `Fusion([1,3,8],[2,3,4,6,7,11,13])` vaudra `[1,2,3,3,4,6,7,8,11,13]`. À partir d'une liste `N` initialement vide, on ajoutera à `N` soit le premier élément de `L` soit le premier élément de `M` (suivant qui est le plus petit) et on retirera cet élément de la liste dont il est issu. On répétera cette opération tant qu'il y a des éléments dans les deux listes.

2. Tester avec `L=[1,3,8]`, `M=[2,4,6,7,11,13]`, `N=Fusion(L,M)` et `print(L,M,N)`

1. Les commandes `M=L[0:len(L)]` ou bien `M=L[:]` fonctionnent aussi.

Exercice 5 (𝄞⋆⋆)**.** Créer une fonction **récursive** `TriFusion(L)` qui va trier la liste `L`. Pour cela, si la liste a un élément ou moins, on a rien à faire pour la trier. Sinon, on coupe la liste au milieu, on trie chacune de ces deux sous-listes de façon récursive. Puis on les fusionne les deux sous-listes triées grâce à la fonction `Fusion` créée à l'exercice précédent.

Amélioration

Exercice 6 (𝄞⋆⋆)**.** Écrire une fonction `Fusion2`, similaire à la fonction `Fusion`, n'utilisant pas `pop` mais compare seulement `L[i]` et `M[j]` où `i` et `j` sont des indices qui indiquent l'avancée de la fusion dans la liste `L` et dans la liste `M`, au début `i` et `j` vaut 0.

Comparaison temporelle

Exercice 7 (⋆)**.** Reprendre l'exercice sur le tri à bulles (TP4). Générer une liste au hasard et la trier avec le tri fusion utilisant la fonction `Fusion` avec le tri fusion utilisant la fonction `Fusion2` et avec le tri à bulles pour comparer les temps. Créer un graphe avec trois courbes représentant les temps de calculs en fonction la taille de la liste. Quel algorithme vous paraît le plus performant ?

Remarque 1. La moralité c'est que le tri à bulle n'est pas un bon tri. Le tri fusion fonctionne mieux et il fonctionne d'autant mieux dans la version où on utilise pas `pop(0)` (gourmand en complexité).