Retour sur le tri des listes

Exercice 1 (𝄞⋆⋆)**.** On se propose de voir un nouvel algorithme de tri que l'on appelle tri rapide.

1. Créer une fonction `Concatene(L,M,N)` qui a trois liste `L`, `M` et `N` renvoie la liste concaténée des éléments de `L`, `M` et `N`. Ainsi, `Concatene([1,2,8],[2],[10,-11])` doit renvoyer `[1,2,8,2,10,-11]`).
2. Créer une fonction `Pivot(L)`, où `L` est une liste, qui renvoie un triplet **trois listes** `(G,[L[0]],D)` où `G` est est une liste contenant les éléments de `L` strictement plus petit que `L[0]` et `D` est une liste contenant les éléments de `L` qui sont supérieurs ou égales à `L[0]` (mais sans y ajouter `L[0]`) de sorte que `len(G)+1+len(D)=len(L)`.
   Par exemple si `L=[3,8,3,2,6,1]`, alors la fonction doit renvoyer `([2,1],[3],[3,6,8])`.
3. Enfin créer la fonction `TriRapide(L)` qui va trier `L` de façon récursive, pour cela si `L` a un élément ou moins, quelle liste faut-il renvoyer ? Sinon, diviser la liste `L` grâce à la fonction `Pivot`, trier les liste `G` et `D` de façon récursive, et combiner le tout, grâce à la fonction `Concatene`.
4. Écrire un jeu de tests de cette fonction.
5. Tester ensuite cet algorithme sur une liste créée au hasard, utiliser les commandes suivantes :

```
import numpy as np# à mettre en début de fichier
L=[np.random.randint(1,50) for i in range(20)]
#liste crée par compréhension
```

6. Ce tri est-il en place ? Stable ? Revoir la définition de ces termes au besoin au TP9.
7. Créer une deuxième variante de la fonction `Concatene` une doit faire deux lignes l'autre doit fonctionner par `append` successifs.

Retour sur les algorithmes glouton

Exercice 2 (𝄞⋆⋆)**.** Une salle de conférence reçoit des demandes d'utilisation de cette salle sous la forme `[deb,fin,nomconf]`, où `deb` est l'heure de début, `fin` est l'heure de fin et `nomconf` est le nom de la conférence. Son but est de permettre à un maximum de conférences d'utiliser cette salle. Et donc de trouver parmi la liste des demandes une sous-liste de conférences ordonnées par heure croissante de début, de sorte qu'une conférence se finisse avant le début de la suivante. Par exemple si la liste des demandes est : `[[0,1,'C8'],[1,2,'C4'],[0,2,'C5'],[1,3,'C7'],[3,4,'C9'],[0,2,'C6']]` Alors on ne pourra pas accepter simultanément la conférence `C6` et la `C7` par exemple. Pour cela, on va utiliser un algorithme glouton. On va commencer par trier la liste par heure croissante de fin. On commencera par la conférence qui finit le plus tôt, puis en raisonnant par étape, à chaque étape, rajouter la conférence qui finit le plus tôt parmi celles qui commencent après la dernière conférence rajoutée. Détaillons sur l'exemple ci-dessus, une fois triée par ordre croissante de fin la liste est : `[[0,1,'C8'],[1,2,'C4'],[0,2,'C6'],[0,2,'C5'],[1,3,'C7'],[3,4,'C9']]`

- Ainsi, on commence par `planning=[[0,1,'C8']]`,
- On voit que l'on peut mettre la conférence `C4` à la suite, ainsi `planning=[[0,1,'C8'],[1,2,'C4']]`
- Malheureusement, il n'est pas possible de rajouter la conférence `C6`, ni `C5`, ni `C7`. En revanche, on peut mettre la conférence `C9`, ainsi `planning=[[0,1,'C8'],[1,2,'C4'],[3,4,'C9']]`

1. Parmi les lignes :

```
L=sorted(L,key=lambda a:a[0])
L=sorted(L,key=lambda a:a[1])
L=sorted(L,key=lambda a:a[2])
```

   une seule fait trie la liste `L` par heure croissante de fin, trouvez laquelle et supprimer les deux autres.
2. Écrire la fonction `FairePlanning` qui va appliquer l'algorithme et tester sur la liste `L`.
3. Écrire une fonction `ListeDemandes(N)` qui génère une liste de N demandes de conférences non triée de la forme de `L` dont les horaires de début et de fin sont aléatoires entre 8h et 17h avec une durée entière maximum de 3h. Utiliser la commande `np.random.randint(a,b)` pour générer un nombre entier dans $[\![a\,;b-1]\!]$.
4. Pour $N = 50$, quel est le ratio moyen de conférences acceptées ?
5. Afficher sur un graphe le ratio moyen de conférences acceptées en fonction de N.
6. Écrire une fonction de tri rapide (sans utilisation de fonctions intermédiaires comme `pivot` ou `Concatene`) pour pouvoir l'utiliser à la place de la commande `sorted`.

On pourrait imaginer aussi que l'on a plusieurs salles (ou plusieurs chambres d'hôtel) et adapter ce principe. Dès qu'on a remplit une salle de conférence, on remplit les autres suivant le même principe.