Tris de listes

Le tri est une opération courante, et donc importante, en informatique. Trier une liste c'est ordonner ses éléments selon un ordre donné (numérique, alphabétique, lexicographique, etc.).
Précisons ici ce que nous entendons par liste : une liste est un ensemble d'éléments accessibles via un indice (allant de 0 à $n-1$ s'il y a n éléments) dont on peut changer les éléments (et en particulier échanger deux éléments de celle-ci). Les éléments de cette liste sont tous de même type, et on dispose d'une relation d'ordre qui permet de comparer deux éléments entre deux sous la forme « a $<$ b ». Il existe de très nombreux algorithmes de tri, plus ou moins efficace, et surtout plus ou moins efficaces selon les propriétés initiales de la liste à trier (mais nous ne traiterons pas de ce deuxième point).

Exercice 1
Cet exercice est à faire en groupe.

1. Prendre les cartes numérotées (de 1 à 10) et d'une même couleur d'un jeu de carte, les mélanger et n'en conserver que cinq. Placer ces cinq cartes faces cachées dans une file, c'est à dire les unes à côté des autres.

   Le but est de changer l'ordre des cartes de manière à ce qu'elles soient triées par ordre croissant de numéro, par déplacements ou échanges successifs.

2. Mettre en place une stratégie de tri, puis l'appliquer (ne pas manipuler tout de suite, mais se mettre d'abord d'accord dans le groupe sur la manière de procéder).

   Attention à respecter la contrainte suivante : il n'est possible de voir (et comparer) que deux cartes en même temps. Autrement dit deux cartes doivent être replacées faces cachées une fois utilisées pour être comparées.

   Il est possible de retenir des valeurs, tant que ce n'est pas l'ensemble des valeurs vues.

3. Mettre à l'écrit la stratégie de tri utilisée, que nous appellerons maintenant algorithme de tri, le plus précisément possible, sachant qu'un autre groupe devra comprendre et appliquer cet algorithme à partir de ces explications.

   On précise à nouveau l'élément suivant pour homogénéiser les productions :

   On identifiera les cartes dans la file par des indices allant de 0 à 9.

4. Donner votre algorithme de tri avec un autre groupe, prendre un algorithme d'un autre groupe. Appliquer alors cet algorithme, le plus « bêtement » possible, c'est à dire en cherchant d'éventuelles failles qui feraient échouer le tri.

5. Dans le cas où l'algorithme n'est pas assez précis ou n'est pas correct, indiquer à l'autre groupe l'erreur ou les erreurs détectées.

   Récupérer l'algorithme de votre groupe et, si nécessaire, l'améliorer puis revenir à la question 4., éventuellement en changeant de groupe avec lequel communiquer.

6. Une fois l'algorithme de tri fonctionnel, le traduire dans le langage Python et le tester sur quelques exemples de listes (y compris la liste vide).

   Utiliser le code ci-dessous comme point de départ, et définir la fonction `tri` :

```
from random import shuffle,randint

L=[i for i in range(1,11)]
```

```
shuffle(L) # mélange la liste

def tri(liste):
    return liste

print(tri(L))
```

7. Automatiser le test de l'algorithme. Pour cela :
   (a) Définir une fonction `bien_triee(l)` qui prend comme argument une liste `l` et qui retourne un booléen. La fonction retourne `True` si `l` est bien triée, `False` sinon.
   (b) Définir une fonction `test(f_tri)` qui prend comme argument une fonction de tri `f_tri` et qui répète 1000 fois les actions suivantes :
   - Définir une liste `L` de 100 valeurs entières prises aléatoirement (utiliser la fonction `randint`) ;
   - Exécuter la fonction de tri sur une copie `L` (autrement dit, donner en argument à la fonction la liste `L[:]` et pas seulement `L`, pour être certain que la liste `L` ne soit pas modifiée) et nommer `Lt` la liste retournée par la fonction.
   - Si la liste `Lt` est mal triée, afficher `L` et `Lt` pour permettre de détecter les erreurs de code éventuelles.

   À la fin, la fonction doit afficher un texte indiquant que le test est terminé.
   (c) Exécuter alors la fonction de test sur la fonction de tri définie précédemment.
8. Deux algorithmes sont à connaître en première NSI, le tri par sélection et le tri par insertion, qui sont des tris assez « naturels ». Ceux-ci sont donnés dans le cours, et seront étudiés en détail suite à cette activité.

   Traduire ces deux algorithmes de tri sous forme de fonctions Python et exécuter la fonction de test sur ces fonctions.
9. Pour aller plus loin, on peut comparer le temps d'exécution de l'algorithme de tri avec celui des tris par insertion et par sélection, puis ceux des deux algorithmes de tri de Python (la méthode `l.sort()` et la fonction `sorted(l)`) sur de très grandes listes.

   Pour cela, le mieux est de tester toutes ces fonctions sur les mêmes jeux de listes, pour que la comparaison ait du sens.
10. On peut même créer la courbe du temps d'exécution (moyen) en fonction de la taille de la liste pour chacune des fonctions de tri, en utilisant le module matplotlib.

    Pour utiliser matplotlib, si la librairie n'est pas installée (ni facilement installable) sur votre ordinateur, on peut utiliser le site codabrainy.

Note : En terminale NSI on étudie un tri plus efficace que les deux algorithmes vus en première, à savoir le tri fusion.