TP 4 : tri de listes

L'objectif de ce TP est d'écrire des algorithmes permettant de trier par ordre croissant les coefficients d'une liste de nombres. On pourra regarder (avant le TP) les vidéos de la chaîne Youtube "AlgoRythmics"

1 Le tri sélection

1.1 Principe

Pour trier une liste, cet algorithme repose sur le principe suivant :

* rechercher le plus petit coefficient de la liste.
* échanger ce coefficient avec le premier coefficient de la liste : ce coefficient est trié.
* répéter ces opérations sur les coefficients de la liste restant à trier.

Ainsi, pour qu'une liste de longueur n soit triée, on va effectuer $n-1$ fois ces opérations.

Appliquer à la main cet algorithme sur la liste [4 | 3 | 1 | 5 | 2 | 3].

[4 | 3 | 1 | 5 | 2 | 3]

[| | | | | |]

[| | | | | |]

[| | | | | |]

[| | | | | |]

1.2 Algorithme

1. Écrire une fonction `indice_min` qui prend en argument une liste `l` et un indice `i` et qui renvoie l'indice `imin` du plus petit coefficient de la liste dont l'indice est supérieur ou égal à `i`.
2. Écrire une fonction `echange` qui prend en argument une liste `l` et deux indices `i` et `j`, et qui renvoie la liste obtenue en échangeant les éléments d'indices `i` et `j`.
3. En combinant les deux fonctions précédentes, écrire une fonction `tri_selection` qui prend en argument une liste `l` et qui renvoie la liste triée correspondante.
4. Tester votre programme sur plusieurs listes.
5. Reprendre la fonction `tri_selection` et écrire une fonction `tri_selection_2` sans utiliser de fonctions auxiliaires.

2 Le tri insertion

2.1 Principe

Pour trier une liste, cet algorithme repose sur le principe suivant :

* on commence par regarder le premier coefficient uniquement, qui forme donc à lui seul une liste triée de nombres.
* on regarde ensuite le second coefficient que l'on place alors par rapport au premier coefficient ; les deux premiers coefficients sont alors triés.
* on regarde ensuite le troisième coefficient que l'on place alors par rapport aux deux premiers ; les trois premiers coefficients sont alors triés.
* ainsi de suite jusqu'à utilisation de tous les coefficients.

Ainsi, pour qu'une liste de longueur n soit triée, on va effectuer $n-1$ fois ces opérations.
Cette méthode est généralement utilisée pour trier des cartes.
Appliquer à la main cet algorithme sur la liste [4 | 3 | 1 | 5 | 2 | 3].

[4 || 3 | 1 | 5 | 2 | 3]

[| || | | | |]

[| | || | | |]

[| | | || | |]

[| | | | || |]

[| | | | | |]

2.2 Déplacement dans une liste

Écrire une fonction `insere` qui prend en argument un indice `i` et une liste `l` dont les `i` premiers éléments sont triés, c'est-à-dire les éléments dont les indices sont compris entre `0` et `i-1`, et qui renvoie la liste dont les `i+1` premiers éléments sont triés, c'est-à-dire les éléments dont les indices sont compris entre `0` et `i`.
Par exemple si `l`=[**1**,**4**,**6**,**7**,3,5,7] et `i`=4, on obtient la liste [**1**,**3**,**4**,**6**,**7**,5,7].

Méthode :

1. On stocke dans une variable `S` le terme en position `i` et que l'on souhaite insérer .

   i
   [**1**,**4**,**6**,**7**,3,5,7]
   ↓
   S

2. On parcourt de droite à gauche les termes déjà triés jusqu'à trouver la position `j` où placer l'élément stocké dans `S`.

   j
   [**1**,**4**,**6**,**7**,3,5,7]
   ↑

3. On décale d'un cran la sous-liste constituée par les éléments d'indice `j` inclus à `i` exclu.

   j j
   [1,**4**,**6**,**7**,3,5,7] → [**1**,**4**,**4**,**6**,**7**,5,7]

4. On replace le terme stocké dans `S` en position `j`.

   j
   [**1**,**4**,**4**,**6**,**7**,5,7] → [**1**,**3**,**4**,**6**,**7**,5,7]
   ↑
   S

2.3 L'algorithme de tri complet

1. En utilisant la fonction précédente, écrire une fonction `tri_insertion` qui prend en argument une liste `l` et qui renvoie la liste triée correspondante.
2. Tester votre programme sur plusieurs listes.
3. Reprendre la fonction `tri_insertion` et écrire une fonction `tri_insertion_2` sans utiliser de fonctions auxiliaires.

3 Le tri à bulles

Pour trier une liste, cet algorithme repose sur le principe suivant :

- parcourir de gauche à droite une liste.
- comparer deux à deux les coefficients voisins successifs et les échanger s'ils ne sont pas bien rangés.
- lors du premier parcours, le plus grand coefficient (au moins) se retrouve bien placé.
- lors de chaque parcours, un coefficient supplémentaire est bien placé.

Ainsi, pour qu'une liste de longueur n soit triée, on va effectuer $n-1$ fois ces opérations.
Dans cet algorithme, les éléments les plus grands vont progressivement « remonter » vers la droite de la liste, comme des bulles remontent à la surface.
Appliquer à la main cet algorithme sur la liste [4 | 3 | 1 | 5 | 2 | 3].

[4 | 3 | 1 | 5 | 2 | 3]

[| | | | | |]

[| | | | | |]

[| | | | | |]

[| | | | | |]

Écrire une fonction `tri_bulle` qui prend en argument une liste `l` et qui renvoie la liste triée correspondante.
Variante
On constate que lors de certains parcours, plus d'un coefficient supplémentaire est bien placé. Il est donc parfois inutile de répéter $n-1$ fois ces opérations.
On va ainsi répéter ces opérations tant que la liste n'est pas triée (*ie* tant que des opérations sont effectuées).
Pour ce faire, on introduira une variable booléenné `operation` initialisée à `True` et qui, à l'issue d'un parcours de la liste, contiendra `True` si des opérations ont été effectuées et `False` sinon.