TP11 - Tris

→ *Q.1) Récupérez le fichier* `TP11.py` *sur cahier de prépa dans lequel vous pourrez sauvegarder votre travail.*

Le but de ce TP est d'étudier différents algorithmes de tris de listes : on dispose d'une liste de valeurs qu'on souhaite ordonner par ordre croissant.

→ *Q.2) Écrire une fonction* `triee(L)` *qui renvoie* `True` *si la liste* `L` *est triée par ordre croissant et* `False` *sinon.*

On dispose de trois listes pour tester les tris. Attention, ces trois listes ne doivent pas être modifiées! On pourra pour ce faire effectuer des copies.

A) Tri par sélection

Le principe du tri par sélection est le suivant :

- on cherche le plus petit élément de la liste et on l'échange avec l'élément en première position ;
- on cherche le plus petit élément de la liste à partir de la deuxième position et on l'échange avec celui en deuxième position ;
- on poursuit jusqu'à arriver au dernier élément de la liste.

→ *Q.3) Effectuer le tri par sélection sur la liste* `['C', 'A', 'T', 'G', 'CA', 'AT', 'TC', 'CG', 'CAT']`.

→ *Q.4) Écrire une fonction* `echange(L, i, j)` *qui échange les éléments d'indices* `i` *et* `j` *de la liste* `L`.

→ *Q.5) Écrire une fonction* `indice_min(L, i)` *qui renvoie le plus petit indice* `j` *tel que* `L[j]` *est l'élément le plus petit dans* `L[i:]`.

→ *Q.6) Écrire une fonction itérative* `tri_selection_iter(L)` *qui trie la liste* `L` *en utilisant le tri par sélection.*

→ *Q.7) Évaluer la complexité de ce tri.*

→ *Q.8) Écrire une fonction récursive* `tri_selection_rec(L, i)` *qui trie la liste* `L[i:]` *en utilisant le tri par sélection.*

Le tri par sélection est un tri :

- **comparatif** : il effectue des comparaisons entre les éléments de la liste à trier ;
- **stable** : deux éléments égaux seront dans le même ordre dans la liste triée que dans la liste initiale ;
- **en place** : il modifie la liste initiale.

B) Tri rapide en place

On a vu en cours comment coder le tri rapide, mais notre implémentation n'est pas en place : lorsqu'on sépare la liste en deux à l'aide du pivot, on créé deux nouvelles listes, et lorsqu'on regroupe les deux listes aussi.
On va tenter d'implémenter le tri rapide en place.
On dispose d'une liste `L` et on souhaite répartir les éléments de la liste « autour » de son premier élément : on définit une fonction `partition(L, g, d)` qui réorganise les éléments de `L[g:d]` en mettant en premier les éléments inférieurs ou égaux à `L[g]` (le pivot), le pivot, puis ceux strictement supérieurs et renvoie la position du pivot. On procède de la façon suivante :

- on utilise une variable `m` qui indique où se termine la sous-liste des éléments inférieurs ou égaux au pivot ;
- on parcourt la liste entre les indices `g+1` et `d-1` ;
- à chaque itération, on teste si l'élément de `L` est inférieur ou égal au pivot, auquel cas, on l'échange avec l'élément `L[m+1]` et on incrémente `m` ;
- à la fin, on replace le pivot de sorte qu'il sépare les éléments qui lui sont inférieurs et supérieurs.

→ *Q.9) Faire la répartition sur la liste* `[8, 2, 4, 23, 15, 17, 0, 1]`.

→ *Q.10) Écrire la fonction* `partition(L, g, d)`.

→ *Q.11) Vérifier que la propriété « les éléments de* `L[g:m+1]` *sont inférieurs ou égaux à L[g] et ceux de* `L[m+1:i+1]` *sont supérieurs à* `L[g]` *» est un invariant de boucle et en déduire que* `partition` *fait bien ce qui est annoncé.*

→ *Q.12) À l'aide de la fonction précédente, écrire une fonction récursive* `tri_rapide_place_annexe(L, g, d)` *qui effectue le tri rapide de* `L[g:d]` *mais sans utiliser de copie de* `L`.

→ *Q.13) Écrire une fonction* `tri_rapide_place(L)` *qui effectue un tri rapide en place.*

C) Tri par comptage

Le tri par comptage permet de trier une liste dont les éléments proviennent d'un ensemble fini (assez petit). Pour nous, on dispose d'une liste `L` d'entiers compris entre `m` et `M`. Pour chaque entier entre `m` et `M`, on compte le nombre d'occurences de cet entier dans `L`. On stocke ces valeurs dans une liste `Nb` de taille `M-m+1`. On remplit alors une autre liste triée à l'aide de `Nb` qui a les mêmes éléments que `L`.

→ *Q.14) Écrire une fonction* `comptage(L)` *qui prend en argument une liste d'entiers et renvoie deux entiers* `m` *et* `M` *et une liste* `Nb` *tels que tous les éléments de* `L` *sont entre* `m` *et* `M` *et* `Nb[i]` *est le nombre d'occurences de m+i dans* `L`.

→ *Q.15) Écrire une fonction* `tri_comptage(L)` *qui prend en argument une liste d'entiers et renvoie une nouvelle liste triée ayant les mêmes éléments que* `L`.

→ *Q.16) Est-ce un tri comparatif? stable? en place?*

→ *Q.17) Quelle est sa complexité?*

D) Exercices

Exercice 1 - Complexités.

1. Pour le tri fusion : notons $C(n)$ la complexité pour le tri fusion sur une liste de taille n.
   (a) Déterminer une relation entre $C(n)$ et $C(n/2)$.
   (b) On suppose que $n = 2^k$ pour un certain $k \in \mathbb{N}$. Exprimer $C(n)$ en fonction de k.
   (c) En déduire que $C(n) = O(n\log_2(n))$.
2. Pour le tri rapide.
   (a) Justifier que si la liste est déjà triée, la complexité du tri rapide est quadratique en la taille de la liste.
   (b) Quel est le meilleur des cas pour le tri rapide?
   (c) Justifier que dans ce cas, la complexité du tri rapide est quasi-linéaire.

Exercice 2 - Preuve du tri sélection.
Montrer que la propriété « `L[:i+1]` est triée et tous les éléments de `L[i+1:]` sont supérieurs ou égaux à ceux de `L[:i+1]` » est un invariant de boucle pour le tri sélection. En déduire la correction du tri.

E) Bilan

- ▷ Différents types de tris :
  — stable ;
  — en place ;
  — comparatif.
- ▷ Les tris par insertion et par sélection sont simples et en place, mais de complexité quadratique.
- ▷ Le tri fusion n'est pas en place mais de complexité temporelle quasi-linéaire.
- ▷ Le tri rapide peut-être en place, de complexité quadratique dans le pire des cas et quasi-linéaire dans le meilleur des cas.
- ▷ Lorsqu'on a des informations a priori sur la liste à trier, on peut améliorer les complexité : par exemple avec le tri par comptage.