Algorithmes de tri

Le but d'un algorithme de tri est de classer des données[1] dans l'ordre croissant ou décroissant.
Ici nous présenterons ces données sous formes de listes.
Il existe plusieurs algorithmes différents. Nous en présentons ici deux.
Dans tout le texte, on fixe une liste `L` d'entiers à trier, et $n \in \mathbb{N}^*$ sa longueur.

1 Tri par insertion

Principe : On construit une nouvelle liste `LTriée` en parcourant les éléments de `L` et en les insérant un à un à la bonne position dans `LTriée`.

Exemple 1

Si on a à trier `L = [4, 2, 9, 4]` :
On commence avec `LTriée = [4]`.
Comme $2 \leqslant 4$, on insère 2 au début de `LTriée` ; on obtient `LTriée = [2, 4]`.
Comme $9 \geqslant 4$, on insère 9 à la fin de `LTriée` ; on obtient `LTriée = [2, 4, 9]`.
Comme 4 est entre 2 et 4, on insère le dernier 4 à cette position dans `LTriée`, et on obtient : `LTriée = [2, 4, 4, 9]`.

```
1 def triParInsertion(L):
```

Complexité (représente le nombre d'opérations effectuées *dans le pire des cas* pour trier une liste de taille n) :

1. Souvent des nombres, mais cela peut aussi être des mots, comme dans un dictionnaire, *etc.*

Ainsi pour une liste de taille 100, le nombre d'opérations est de l'ordre de
Sur un processeur de 10 GHz (10 milliards d'opérations à la seconde), cela prend un temps de l'ordre de la microseconde.

Pour une liste de taille 100000, le nombre d'opérations est de l'ordre de , donc le temps de calcul de l'ordre de la seconde.

Pour une liste de taille 100 millions, le nombre d'opérations est de l'ordre de , donc le temps de calcul de l'ordre du million de secondes, soit environ 11 jours.

2 Tri par sélection

Principe : On construit une nouvelle liste `LTriée` en supprimant, à chaque étape, le plus petit élément de `L` pour le rajouter à la fin de `LTriée`.

On aura besoin, pour supprimer le minimum de `L`, d'une fonction annexe renvoyant l'indice du minimum.