Bonus

Complexité des tris

Les tris par sélection, par insertion et à bulles vus en classe de première sont tous trois des tris en complexité $O(n^2)$. Nous verrons en classe de terminal de nouveaux tris moins intuitifs et plus complexes mais ayant une complexité en $O(n\log(n))$, ce qui améliore énormément le temps de calcul sur des listes contenant beaucoup d'éléments (très souvent $\log(n)$ est en réalité $\log_2(n)$).

Par exemple pour $n=10000$, on a $n^2=10^8$ alors que $n\log(n)=10^4\times\log_2(10^4)\approx10^4\times14\approx1{,}4\times10^5$.
$n\log(n)$ est dans ce cas là environ 1000 fois plus petit que n^2.

Python offre déjà la possibilité de trier les listes de deux façons, `sort()` et `sorted()`, tous deux implémentant des algorithmes de tri en $O(n\log(n))$.

```
# la méthode sort() modifie la liste initiale pour la trier
l1 = [1, 5, 3, 4, 2]
l1.sort()
print(l1) # [1, 2, 3, 4, 5]

# la fonction sorted(), qui ne modifie pas la liste initiale
# et retourne une nouvelle liste triée
l2 = [1, 5, 3, 4, 2]
l3 = sorted(l2)
print(l3) # [1, 2, 3, 4, 5]
```

Exercice 1

Utiliser et modifier le code suivant pour comparer les performances de vos fonctions de tri par rapport la fonction `sorted()` de python.

```
import time
import random

taille_liste = 10**4

# Création d'une liste en compréhension avec des valeurs aléatoires
debut = time.time()
l = [random.randint(0, taille_liste) for _ in range(taille_liste)]
fin = time.time()
print("La création a pris", fin - debut, "secondes pour s'exécuter.")

# Sorted(), un tri en O(n*log(n))
debut = time.time()
l2 = sorted(l)
fin = time.time()
print("Sorted() a pris", fin - debut, "secondes pour s'exécuter.")

# Vos fonctions
# ...
```

Exercice 2

Le tri par comptage (counting sort en anglais) est un cas particulier de tri de complexité $O(n+k)$.
Il offre de meilleurs performances que les tris classiques car on dispose d'une information supplémentaire sur la liste à trier : les valeurs sont entières et positive. Nous disposons ainsi d'une liste de n valeurs entières positives et ces valeurs sont comprises entre 0 et un maximum k.

Exemple

Les 1000 élèves du lycée ont passé un QCM de 5 questions et les notes sont des entiers compris entre 0 et 5 inclus. Il n'y a donc que 6 notes différentes possibles et la liste ne va contenir que des 0, 1, 2, 3, 4 ou 5.

Pseudo-code :

```
tri_par_comptage(liste l, entier k):

    comptage ← liste de k + 1 zéros
    résultat ← liste de même taille que l

    Pour i de 0 à taille(l) - 1 faire
        comptage[l[i]] += 1
    Fin Pour

    Pour i de 1 à k faire
        comptage[i] = comptage[i] + comptage[i - 1]
    Fin Pour

    Pour i de taille(l) - 1 décroissant jusqu'à 0 faire
        comptage[l[i]] += -1
        résultat[comptage[l[i]]] = l[i]
    Fin Pour

    retourner résultat
```

En vous aidant du pseudo-code, implémentez le tri par comptage en python et testez le.

```
l = [1, 2, 0, 0, 2, 1, 4, 4, 5, 3, 3, 3, 3]
assert tri_par_comptage(l, 5) == [0, 0, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4, 5]
```