Tris

Fiche d'exercices

Exercice 1. Tri par comptage

On se place ici dans le cas particulier où on sait que notre liste ne contient que des nombres entiers entre 0 et une valeur maximale `v_max` connue.

La liste ne contient pas nécessairement tous les entiers entre 0 et `v_max`.
Un entier présent peut l'être plusieurs fois.

Par exemple `L_exple = [1, 4, 0, 3, 5, 4, 2, 2, 4, 0]` avec `v_max = 5`.

1. Écrire une fonction `comptage(L:list, v_max:int)-> list` qui retourne une liste `count` de taille `v_max+1` telle que `count[k]` correspond au nombre d'apparitions de la valeur `k` dans `L`

   Par exemple `count(L_exple, 5)` retournerait `[2, 1, 2, 1, 3, 1]`.

   (on retrouve entre autre qu'il y a 3 fois le chiffre 4)
2. En utilisant la fonction précédente, écrire une fonction `tri_comptage(L, v_max)` qui retourne une copie triée de `L` en procédant ainsi :
   — utilisation de la fonction `comptage` sur `L`
   — création d'une liste image vide
   — complétion de cette liste image en insérant `count[0]` fois le chifre 0 puis `count[1]` fois le chifre 1, *etc.*

Exercice 2. Étude du tri par insertion

1. Commencer par relire le principe du tri par insertion et le code correspondant.
2. Implémenter en python le tri par insertion (en utilisant le moins possible votre cours)
3. On veut maintenant déterminer la complexité du tri par insertion. On comptabilisera les **comparaisons** effectuées.
   a. Lors de la phase d'insertion (boucle tant que), quel est le nombre de comparaisons effectuées dans le meilleur et le pire des cas ?
   b. En déduire la complexité de la fonction dans le meilleur et le pire des cas. On donnera un type de liste correspondant à chacun de ces cas.

Exercice 3. Adaptation du tri par insertion

Le code incomplet de la fonction `triInsertion` est donné ci-dessous.
La fonction `triInsertion` admet trois arguments : une liste de données `liste` et deux indices `g` et `d`.
Elle trie dans l'ordre croissant la partie de la liste comprise entre les indices `g` et `d` inclus.

Compléter cette fonction (avec le nombre de lignes de votre choix).

```
def triInsertion(liste,g,d):
    for i in range(g+1,d+1):
        j = i-1
        tmp = liste[i]
        while



        liste[j+1] = tmp
```

Exercice 4. Médiane d'une liste

1. Rappeler la définition de la médiane d'une liste de nombres
2. En utilisant le tri par insertion précédemment codé, écrire une fonction `mediane(L)` qui retourne la médiane de la liste L.

   Remarque. Bien évidemment, le choix du tri par insertion est purement arbitraire. Tout autre tri fonctionnerait aussi.

Exercice 5. Mise en place du tri fusion

1. Commencer par relire le principe du tri fusion et le code correspondant.
2. Implémenter en python le tri fusion (en utilisant le moins possible votre cours)

Exercice 6. Adaptation du tri fusion D'après Mines-Ponts 2020

Dans un contexte d'étude d'objets géométriques tri-dimensionnels (3D), on met en place les modélisations suivantes :

— Un **sommet** sera une liste de trois flottants, les coordonnées du point.

```
S = [2. , 3.2, 1.5]
```

— Une facette (triangle de 3 points) sera représentée par une liste de 3 sommets.

```
F = [[2. , 3.2, 1.5], [7. , -1.2, 6.5], [-2.6 , 4., 1.7]]
```

— On suppose aussi connue une fonction `aire(F)` qui retourne l'aire d'une facette.

On propose le code (incomplet) du tri-fusion ci-dessous.

```
def fusion(L1, L2):
    # A completer (sur une ou plusieurs lignes)

def trier_facettes(L):
    # A completer (sur une ou plusieurs lignes)
```

1. Compléter la fonction `fusion`, prenant comme argument deux listes de facettes `L1` et `L2` (supposées chacune triée par *aire décroissante*) et renvoyant une nouvelle liste composées des facettes de `L1` et `L2` triées par *aire décroissante*.
2. Compléter la fonction récursive `trier_facettes`, prenant comme argument une liste de facettes `L` et renvoyant une nouvelle liste de facettes triées dans l'ordre des aires décroissantes, par la méthode du tri-fusion.

N.B. L'algorithme de tri-fusion n'était pas du tout donné dans le sujet ! Il fallait le retrouver et l'adapter par soi-même...

Exercice 7. Étude du tri rapide

1. Commencer par relire le principe du tri rapide et le code correspondant.
2. Implémenter en python le tri rapide (en utilisant le moins possible votre cours)
3. Quelle est la complexité du tri rapide pour une liste déjà triée (pire cas) ?

   On admet que la complexité du tri rapide est en moyenne $O(n\ln(n))$