Révisions listes et tableaux

Thèmes : listes, tableaux, parcours, tri récursif, codage binaire d'un entier positif

1 Trois plus grandes valeurs

On souhaite déterminer les 3 plus grandes valeurs présentes dans une liste. On testera les fonctions de cet exercice à l'aide d'une liste de valeurs aléatoires qu'on peut obtenir avec la commande :

```
import random as rd
l = [rd.randint(1, p) for i in range (n)]
```

Question 1

On commence par s'attaquer à un problème plus simple. Écrire une fonction `valmax(l)` prenant en entrée une liste non vide et retournant la plus grande valeur de ce tableau.

Question 2

Écrire une fonction `val3max(l)` prenant en entrée une liste contenant des entiers positifs et retournant un triplet (a, b, c) contenant les 3 valeurs les plus grandes apparaissant dans le tableau avec $a < b < c$. Par exemple sur la liste $[1, 5, 7, 4, 2, 3, 1, 7]$ la réponse sera $(4, 5, 7)$ (même si la valeur 7 apparaît deux fois). La complexité de votre solution devra être $O(n)$ où n est la longueur de l.
Remarque : Le triplet résultat pourra contenir des valeurs -1 s'il n'y a moins de 3 valeurs distinctes dans la liste. Par exemple sur la liste $[4, 4, 4, 4, 4]$ le résultat sera $(-1, -1, 4)$.

INDICATIONS

Se baser sur ce qui a été fait pour `valmax`. Lors du parcours de la liste :

- mémoriser dans 3 variables a, b, c les trois plus grandes valeurs trouvées jusqu'à présent
- pour chaque valeur rencontrée lors du parcours, tester s'il convient de modifier a, b et/ou c
- a, b, c pourront être initialisées avec la valeur -1.

2 Tri par partition-fusion

On souhaite trier une liste selon le principe *diviser pour régner* en utilisant l'algorithme du *tri par partition-fusion*. Pour trier une liste, par exemple $l = [1; 7; 3; 4; 2; 10; 5]$, on procède en 3 étapes :

1. on partitionne l en deux listes $l_1 = [1; 3; 2; 5]$ (contenant les cases de l d'indice pair) et $l_2 = [7; 4; 10]$ (contenant les cases de l d'indice impair);
2. on trie *récursivement* les listes l_1 et l_2, on obtient $u_1 = [1; 2; 3; 5]$ et $u_2 = [4; 7; 10]$;
3. on *fusionne* les listes u_1 et u_2, on obtient $u = [1; 2; 3; 4; 5; 7; 10]$ qui correspond bien au tri de l par ordre croissant.

Question 3

Écrire une fonction `partition(l)` prenant en entrée une liste d'entiers et retournant le couple (l_1, l_2) défini ci-dessus.
(*Bonus*) Il est également possible de procéder en coupant la liste l en son milieu. Écrire une fonction `partition_bis(l)` utilisant cette méthode.

INDICATIONS

Débuter initialement avec $l_1 = l_2 = []$, puis parcourir l. Lors du parcours ajouter les valeurs rencontrées dans l_1 ou dans l_2 selon la parité de l'indice.

Question 4

Écrire une fonction `fusion(u1, u2)` prenant en arguments deux listes u_1 et u_2 **déjà triées par ordre croissant** et retournant une liste u triée par ordre croissant contenant les valeurs de u_1 et u_2. On utilisera l'algorithme suivant :

$u \leftarrow [\,], i_1 \leftarrow 0, i_2 \leftarrow 0$
Tant que i_1 et i_2 sont des indices valides **Faire**
    Si $u_1[i_1\,] \leq u_2[i_2\,]$ **Alors**
        ajouter $u_1[i_1\,]$ dans u
        incrémenter i_1
    Sinon
        ajouter $u_2[i_2\,]$ dans u
        incrémenter i_2
    FinSi
Fin Tant que
$u \leftarrow u + u_1[i_1 :] + u_2[i_2 :]$ (concaténation)

Question 5

1. Écrire une fonction de test vérifiant le bon fonctionnement de la fonction `fusion` à l'aide d'assertions.
2. Justifier théoriquement que la fonction `fusion` termine.
3. Écrire une fonction `est_triee(l)` retournant `True` si et seulement si la liste `l` est triée par ordre croissant.
4. Ajouter une ou des assertions pour tester la validité des entrées dans la fonction `fusion`.

Question 6

Écrire une fonction *récursive* `tri_fusion(l)` prenant en entrée une liste et retournant une nouvelle liste u correspondant au tri par ordre croissant de l.

INDICATIONS

Utiliser l'algorithme proposé et les fonctions précédentes. Bien identifier les cas de base de la récursivité.

Tester votre fonction de tri. On rappelle que la complexité temporelle de cet algorithme est $O(n\log(n))$ et qu'il n'existe pas de meilleure complexité pour un tri par comparaisons.

3 Entiers conjugués

Un entier $0 \leq n \leq 255$ peut-être représentée de façon unique à l'aide d'un tableau de 8 bits (il s'agit donc d'un octet) où chaque bit représente une puissance de 2 :

1	2	4	8	16	32	64	128

Ainsi, l'entier 41 sera représenté sous forme de tableau de bits par :

1	0	0	1	0	1	0	0

car $41 = 32 + 8 + 1$.

Question 7

1. Écrire une fonction `decodage(t)` prenant en entrée une liste `t` de 8 bits et retournant l'entier n qu'il représente.
2. Inversement, écrire une fonction `codage(n)` prenant en entrée un entier $0 \leq n \leq 255$ et retournant la liste de son codage sur un octet.

INDICATIONS

Pour le codage, la case d'indice i sera obtenue en étudiant la parité de `n // (2**i)`.

Question 8

On appelle rotation droite l'opération consistant à transformer un tableau

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

en le tableau

a_7	a_0	a_1	a_2	a_3	a_4	a_5	a_6

Écrire une fonction `rotation(t)` réalisant la rotation droite d'un tableau `t` donné : cette fonction modifiera le tableau passé en argument et ne retournera aucun résultat.

Question 9

Deux tableaux seront dits *conjugués* si on peut obtenir l'un à partir d'un nombre quelconque de rotations droite de l'autre. Deux entiers seront dits conjugués si leurs représentations sous forme d'octet sont 2 tableaux conjugués.

1. Écrire un programme demandant à l'utilisateur d'entrer deux entiers n_1 et n_2 testant s'ils sont conjugués.
2. Écrire un programme demandant à l'utilisateur d'entrer un entier $0 \leq n \leq 255$ et affichant tous les entiers $0 \leq n' \leq 255$ tels que n et n' sont conjugués.