ITC - PC

TD - manipulation de listes 2

Exercice 1 (*Rotation d'une liste*)

On appelle rotation d'une liste `L` le fait de décaler tous les éléments d'une place vers la droite, à l'exception du dernier qui est placé en première place. Par exemple, la rotation de la liste `[1,2,3,4]` est la liste `[4,1,2,3]`.

1. Rédiger une fonction `rotation(L)` qui renvoie une *nouvelle* liste égale à la rotation de la liste initiale.
2. Rédiger une fonction `rotation2(L)` qui *modifie* la liste `L` pour la remplacer par sa rotation.
3. Rédiger une fonction `rotation3(k, L)` qui renvoie une nouvelle liste égale à `k` rotations de la liste `L`.

Exercice 2

À partir d'une liste `L` d'entiers naturels, rédiger une fonction `entierManquant(L)` qui renvoie le plus petit entier naturel absent de la liste.
Par exemple, pour `L = [1,3,7,6,4,1,2,0]` cette fonction devra renvoyer 5.

Exercice 3

Une liste `L` de n termes représente une permutation lorsque chaque entier de $[\![1,n]\!]$ est présent une et une seule fois dans la liste.
Par exemple, `[2, 4, 3, 1]` représente une permutation, mais pas `[4, 3, 2, 5]`.
Rédiger une fonction `permutation(L)` qui renvoie `True` lorsque `L` représente une permutation, et `False` dans le cas contraire.

Exercice 4

Une liste non vide `L` de n entiers relatifs étant donné, on cherche la valeur maximale de la quantité $\Delta_k = (L[0]+...+L[k])-(L[k+1]+...+L[n-1])$ lorsqu'on fait varier k dans $[\![0,n-2]\!]$.

1. Rédiger une fonction `delta(k, L)` qui calcule la quantité Δ_k et en déduire une fonction `equilibre(L)` qui résout le problème posé.
   Évaluer la complexité temporelle de cette dernière fonction.
2. Trouver une fonction `equilibre2(L)` qui résout ce problème en temps linéaire

Exercice 5 (*Frog river one - Codility*)

A small frog wants to get to the other side of a river. The frog is initially located on one bank of the river (position 0) and wants to get to the opposite bank (position X+1). Leaves fall from a tree onto the surface of the river.

You are given an array A consisting of N integers representing the falling leaves. A[K] represents the position where one leaf falls at time K, measured in seconds.

The goal is to find the earliest time when the frog can jump to the other side of the river. The frog can cross only when leaves appear at every position across the river from 1 to X (that is, we want to find the earliest moment when all the positions from 1 to X are covered by leaves). You may assume that the speed of the current in the river is negligibly small, i.e. the leaves do not change their positions once they fall in the river.

For example, you are given integer X = 5 and array A such that :
`A= [1, 3, 1, 4, 2, 3, 5, 4]`
In second 6, a leaf falls into position 5. This is the earliest time when leaves appear in every position across the river.

Write a function : `def Solution(X, A)` that, given a non-empty array A consisting of N integers and integer X, returns the earliest time when the frog can jump to the other side of the river.
If the frog is never able to jump to the other side of the river, the function should return -1.

Exercice 6

Un tableau contient un nombre impair d'entiers positifs. Chacun de ces entiers est présent un nombre pair de fois, à l'exception d'un seul.
Rédiger une fonction `impair(t)` qui prend pour argument un tel tableau et renvoie cet unique entier présent un nombre impair de fois. Analysez la complexité de votre algorithme.

Exercice 7

On dispose d'un stock illimité de pièces de $c_0, c_1, ..., c_{p-1}$ euros qui sera représenté en Python par une liste `c` de longueur p, et on souhaite dénombrer le nombre de manières possibles d'obtenir n euros avec ces pièces.
L'objectif de cet exercice est de définir une fonction `décompose(n, c)` qui répond à ce problème.
Par exemple, pour connaître le nombre de décompositions de 50€ à l'aide de pièces et de billets de 1€, 2€, 5€, 10€ et 20€ on écrira :

```
>>> decompose(50, [1, 2, 5, 10, 20]
450
```

Combien y-a-t'il de décompositions de n qui utilisent la pièce c_0 ? et de décompositions de n qui ne l'utilisent pas ? En déduire une fonction récursive `decompose` qui répond au problème.