Plus longue suite croissante

Une version naïve

Q1. ***Validation*** *: écrire une fonction qui prend une liste* `L` *d'entiers en entrée et qui renvoie* `True` *si les termes de la liste forment une suite strictement croissante et* `False` *sinon.*

```
def test_croissance_stricte(L):
    n = len(L)
    if n<=1:
        return True
    for i in range(n-1):
        if L[i+1]<=L[i]:
            return False
    return True
```

Q2. *Quelle est la complexité de cette fonction (par rapport à la taille de la liste* `L`*)?*
Une seule boucle avec un nombre d'opérations en temps constant à l'intérieur : la complexité est en $O(n)$ (où n est la taille de la liste).

Q3. ***Création des sous-listes*** *: on veut une fonction qui, à partir d'une liste d'entiers* `L`*, renvoie la liste de toutes les sous-listes de* `L`*.*

```
def sous_listes(L):
    if len(L)==0:
        return [[]]
    n = len(L)
    last = L[-1]
    sL = L[:n-1]
    A = sous_listes(sL)
    listes = []
    for a in A:
        listes.append(a+[last])
    listes = A+listes
    return listes
```

Q4. *Combien de sous-listes renvoie cette fonction?*
On a exactement 2^n sous-listes créées (elles sont en bijection avec $\{0,1\}^n$).

Q5. ***Détermination d'une plus longue suite strictement croissante :***

```
def PLSSC_1(L):
    t = 0 # plus grande longueur
    suite = []  # plus longue suite rencontrée - l'initialisation n'est pas utile
    n = len(L)
    S = sous_listes(L)
    for s in S:
        if len(s)>t and test_croissance_stricte(s):
            t = len(s)
            suite = s
    return t,suite
```

Q6. *Quels sont les défauts de cette méthode?*

- création et stockage de toutes les sous-listes (notamment celles qui ne sont pas croissantes)
- une complexité exponentielle

Une version par programmation dynamique

Q7. *On se donne une liste* `L` *strictement croissante d'entiers et un entier* k*. Comment savoir si la liste* `L+[k]` *est strictement croissante?*
Il suffit de comparer le dernier élément de `L` avec `k` (et regarder si `k` est strictement supérieur).

Q8. *Écrire une fonction* `PLSSC2(L)` *qui prend une liste d'entiers* `L` *en argument et qui renvoie la longueur de la plus longue suite strictement croissante dans cette liste*

```
def PLSSC_2(L):
    n = len(L)
    if n<=1:
        return n
    l_max = 1
    liste = [[L[0]]]  # pour conserver toutes les suites strictement croissantes
    for k in range(1,n):
        terme = L[k]
        new_listes = [[terme]]
        for l in liste:
            if l[-1]<terme:
                new_listes.append(l+[terme])
                if len(l)+1>l_max:
                    l_max= len(l)+1
        liste = liste + new_listes
    return l_max
```

Q9. *Comment modifier cette fonction pour qu'elle retourne à la fois la longueur de la plus longue suite strictement croissante ainsi que toutes les suites strictement croissantes de cette longueur ?*
Il suffit de rajouter, avant le return :

```
    listes = []
    for l in liste:
        if len(l)==l_max:
            listes.append(l)
    return l_max,listes
```

Amélioration de la version précédente

Q10. *Écrire une fonction* `PLSSC3(L)` *qui détermine la longueur de la plus longue sous suite strictement croissante, ainsi que l'une de ses suites, en suivant l'algorithme précédent*

```
def PLSSC_3(L):
    n = len(L)
    if n<=1:
        return L
    plus_longue_fin = [[L[0]]]   # pour les plssc se terminant en position p
    l_max = 1
    best_of_bests = [L[0]]
    for k in range(1,n):
        terme = L[k]
        meilleure = [terme]
        for l in plus_longue_fin:
            if l[-1]<terme:
                if len(l)+1>len(meilleure):
                    meilleure = l+[terme]
        if len(meilleure)>l_max:
            l_max = len(meilleure)
            best_of_bests = meilleure
        plus_longue_fin.append(meilleure)
    return l_max,best_of_bests
```

Q11. *Quelle est la complexité de cette fonction (en fonction du nombre n d'éléments dans* `L`*) ?*
on a deux boucles imbriquées. La première comporte $n-1$ passages. À l'intérieur, il y a plusieurs opérations en temps constants et une boucle qui porte sur une liste (`plus_longue_fin`) de taille $k \leqslant n$. Les calculs à l'intérieur de cette seconde boucle sont en temps constant. On a donc une complexité en $O(n^2)$.