Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

Exercice 5 - Programmation, Niveau 1re

1. a. Si tous les éléments du tableau sont positifs, en prenant comme sous-séquence le tableau entier, on obtiendra la valeur la plus grande.

1. b. Si tous les éléments du tableau sont négatifs, il faudra prendre la sous-séquence constituée d'une seule valeur, celle qui est la plus proche de zéro.

2. a.

Solution 1 :

```
def somme_sous_sequence(lst, i, j):
    somme = 0
    for i in range(i, j+1):
        somme = somme + lst[i]
    return somme
```

Solution 2 :

```
def somme_sous_sequence2(lst, i, j):
    return sum(lst[i:j+1])
```

2. b. La bonne réponse est 55. En effet on constate qu'il y a deux boucles imbriquées parcourant la liste.

```
for i in range(n):
      for j in range(i, n):
```

La complexité de l'algorithme est quadratique $O(n^2)$.
Pour un tableau de 10 cases, on peut former, 10 blocs d'une case, 9 blocs de 2 cases, 8 blocs de 3 cases, etc … 1 bloc de 10 cases.
$1 + 2 + 3 + \cdots + n = n(n+1)/2$ soit pour $n = 10$, $10 \times (10+1)/2 = 55$ comparaisons.

2. c.

```
def pgsp(lst):
    n = len(lst)
    somme_max = lst[0]
    for i in range(n):
        for j in range(i, n):
            s = somme_sous_sequence(lst, i, j)
            if s > somme_max :
                somme_max = s
                deb, fin = i, j
    return somme_max, deb, fin
```

3. a.

i	0	1	2	3	4	5	6	7
lst[i]	-8	-4	6	8	-6	10	-4	-4
S(i)	-8	-4	6	14	8	18	14	10

Correction

NSI - 2021 Métropole Libre Jour 2 (21-NSIJ2ME2)

3. b.

```
def pgsp2(lst):
    sommes_max = [lst[0]]
    for i in range(1, len(lst)):
        if sommes_max[i-1] <= 0:  # sommes_max[-1] revient au même
            sommes_max.append(lst[i])
        else:
            sommes_max.append(lst[i] + sommes_max[i-1])  # ou avec sommes_max[-1]
    return max(sommes_max)
```

3. c. La solution obtenue par cette approche est plus avantageuse que celle de la question 2. b. car elle permet d'éviter de refaire plusieurs calculs identiques. Sa complexité est inférieure car on ne parcourt qu'une fois le tableau. C'est une complexité linéaire $O(n)$ ce qui est plus efficace que la complexité quadratique précédente $O(n^2)$. L'exécution de la fonction sera plus rapide.