PROBLEME DE LA SOMME MAXIMALE

0	1	2	3	4	5	6	7	8	9
4	-5	2	-1	3	-1	1	2	-7	5

- On considère un tableau T[0,n-1] de n entiers (tableau 0 based index)
- On recherche un sous tableau **contigu** dont la somme est maximale
- Le problème présente un intérêt uniquement si le tableau contient des valeurs négatives
- On considère que $n \geq 1$, et que le sous tableau a une taille minimale de 1.

Questions

1/ Trouver la solution sur l'exemple ci dessus

2/Déterminer un algorithme naïf qui résout le problème , indiquer sa complexité.

3/Notons S(j) la somme maximale pouvant être atteinte pour toutes les fenêtres se terminant en position j. Déterminer un moyen de calculer S(j) en fonction de S(j-1) et T[j].

4/Calculer S(j) sur l’exemple ci dessous pour j compris entre 0 et 7.

5/Déterminer un algorithme de programmation dynamique qui retourne la somme maximale du tableau. Déterminer sa complexité temporelle et spatiale.

6/Améliorer cet algorithme pour diminuer sa consommation mémoire

7/Modifier l'algorithme trouvé en 4 pour qu'il retourne les indexs du sous tableau contenant la somme maximale.

CORRECTION

Question 1 : La somme maximale est 6

QUESTION 2 – Solution naïve

On calcule les sommes des sous tableaux T[i,j], pour tout couple (i,j) tel que

$$0 \leq i \leq j < n$$

et on extrait ensuite le maximum

```
max ← moins l'infini
pour j de 0 à n-1
pour i de 0 à j
    s ← somme(i,j)
    si s> max alors max ← s
fin pour
fin pour
afficher « la somme maximale est : « + max

somme(i,j)
res ← 0
pour k de i à j
    res ← res +T[k]
fin pour
retourner res
```

- Complexité : $O\left(n^3\right)$

QUESTION 3 – Réfléchir différemment

- Considérons tous les sous tableaux qui **terminent** en position j
- Dénotons par S(j) la somme maximale pouvant être atteinte pour toutes les fenêtres **terminant** en position j
- Deux cas sont possibles :
  - On prend une fenêtre maximale **terminant** en position j-1 puis on l'étend à la position j
  - On commence une nouvelle fenêtre à la position j
- On en déduit

S(j) = max(S(j-1)+T[j] , T[j])

QUESTION 4

0	1	2	3	4	5	6	7	8	9
4	-1	2	1	4	3	4	6	-1	5

QUESTION 5 – PROGRAMMATION DYNAMIQUE

```
// On calcule S(j) pour j de 0 à n-1
S ← tableau de taille n
S[0] ← T[0]
pour j de 1 à n-1
    S[j] ← max (S[j-1]+T[j],T[j])
fin pour

// Recherche du max dans S
max ← S[0]
pour j de 1 à n-1
    si S[j]> max alors max ← S[j]
fin pour
afficher « la somme maximale est : « + max
```

Complexité en temps de calcul : O(n)
Complexité spatiale : O(n)

QUESTION 6 – OPTIMISATION

- Dans certains cas, on peut réduire l'espace utilisé
- Par exemple, ici , la seule valeur dont a besoin est S[j-1]

```
s ← T[0]
max ← s
pour j de 1 à n-1
    s ← max (s+T[j],T[j])
    si s>max alors max ← s
fin pour
afficher « la somme maximale est : « + max
```

Complexité en temps de calcul : O(n)
Complexité spatiale : O(1)

QUESTION 7 – Recherche d'une solution

Deux solutions sont possibles :

- on récupère la solution après avoir construit le tableau dynamiquement

- on conserve l'information pendant que le tableau est construit

Conservation de l'information

```
// On calcule S(j) pour j de 0 à n-1 en conservant l'information
// du sous tableau utilisé dans P
// P[x] contient l'index de départ du sous tableau correspondant
// à S[x]
S ← tableau de taille n
P ← tableau de taille n
S[0] ← T[0]
pour j de 1 à n-1
    si (S[j-1]+T[j]>T[j]) alors
        S[j] ← S[j-1]+T[j]
        P[j] ← P[j-1]
    sinon
        S[j] ← T[j]
        P[j] ← j
    fin si
fin pour

// Recherche du chemin utilisé
m ← 0
pour j de 1 à n-1
    si S[j]> S[m] alors m ← j
fin pour

afficher « le sous tableau P[m] , m a une somme maximale
```