


Institut Supérieur d'Informatique et de Mathématiques  	Année Universitaire : 2022-2023
	<p style="text-align: center;"><b>DS</b></p> Matière : <b>Conception et analyse d'algorithmes</b> Filière : <b>ING 1 INFO</b> Enseignant : <b>Taoufik Sakka Rouis</b>

### Exercice 1: (9 Points)

Soit  $T$  un tableau d'entiers relatifs de taille  $N$ , un sous-tableau  $T_{ij}$  de  $T$  est défini comme la suite des éléments  $T[k]$  tels que  $i \leq k \leq j$ ;  $i$  et  $j$  sont respectivement la borne inférieure et la borne supérieure du sous-tableau  $T_{ij}$  avec  $0 \leq i < N$  et  $i \leq j < N$ . On appelle somme d'un sous-tableau la somme de ses éléments.

Le but de cet exercice est d'implémenter une fonction qui permet de calculer et d'afficher le sous-tableau maximum (Le sous-tableau dont la somme de ses éléments est la plus grande parmi tous les sous-tableaux d'un tableau  $T$ ). Pour simplifier les choses, on supposera que le maximum est atteint pour un unique sous-tableau (c'est-à-dire qu'il n'y a pas deux sous-tableaux différents qui peuvent avoir la même somme).

1. Pour résoudre ce problème, donner la (les) cause (s) de penser à la programmation dynamique? Quelle technique de programmation dynamique est la plus adéquate pour implémenter cette fonction (Justifier votre réponse). **(2 p)**
2. En utilisant la programmation dynamique, écrire une fonction de prototype «void SommeSeqMax( int \*t, int N) » qui permet de calculer et d'afficher la somme des éléments du sous-tableau maximum. **(6 p)**
3. Évaluer la complexité de votre solution. **(1 p)**

**Exemple :** pour le tableau  $T$  suivant :

31	-41	59	26	-53	58	97	-93	-23	84	35	-98	-80	-72	-85
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

**Cette fonction affiche le message suivant:**

Le sous-tableau maximum est le sous-tableau compris entre les bornes 2 et 10.  
Sa somme maximale est = 190.

**Exercice 2: (5 Points)**

Imaginer une structure de données dotée S des opérations suivantes :

- Insérer (S, x) : permettant d'insérer x dans l'ensemble S.
- SupprimeElem(S, i) : permettant de supprimer l'élément de la position i
- Card (S) : permettant de déterminer le nombre d'éléments de S
- Vide (S) : permettant de vérifier si la SD est vide ou non
- SupprimeMoitie (S): permettant de supprimer  $n/2$  éléments de S. Avec **n** est le nombre d'éléments dans la SD à ce moment de l'exécution.

Supposer maintenant que la structure de donnée est initialement vide et que les coûts réels de ces opérations sont les suivants (voir le tableau ci-dessous),

1. En utilisant la méthode des potentiels, déterminer le **bon coût amorti** de chacune de ces opérations :

Opération	Coût réel	Coût amorti
Inserer (S, x)	1	....
SupprimeElem (S, i)	Max (n-i, 0)	....
Card (S)	n	....
Vide (S)	n	....
SupprimeMoitie (S)	$n/2$	....

**Exercice 3: (6 Points : 3+3)**

1. Sachant que ces deux implémentations ont le même rôle, en utilisant le paradigme diviser pour régner, proposer une troisième implémentation pour cette fonction.

```
int Fonction1 ( int * t, int n) {
    if (n==1)
        return *t;
    else {
        if (Fonction1 (t, n-1) > *(t+n-1) )
            return Fonction1 (t, n-1);
        else
            return *(t+n-1);
    }
}
```

```
int Fonction2 ( int * t, int n) {
    int x;
    if (n==1)
        return *t ;
    else {
        x= Fonction2 (t, n-1);
        if (x > *(t+n-1) )
            return x ;
        else
            return *(t+n-1);
    }
}
```

2. Calculer la complexité asymptotique de chacune de ces trois implémentations. Quelle est la fonction la plus performante (du point de vue complexité de calcul).