Nom, Prenom:_____ **Groupe:**_____

TP2 : Complexité [4 heures]

Préparation

Nous allons analyser quatre algorithmes de résolution d'un problème de recherche de sous-séquence maximale défini comme suit :

Soit x une séquence (un tableau) de n nombres. Trouver la sous-séquence dont la somme d'éléments est maximale.

Remarque : Si tous les nombres sont négatifs on considère que la sous-séquence de somme maximale est vide et la somme maximale est 0.

Exemple : x = {31, -41, 59, 26, -53, 58, 97, -93, -23, 84}

La sous-séquence de somme maximale est {59, 26, -53, 58, 97} et cette somme est $x_{3..7} = 187$.

a. Déterminer les sous-séquences continues de somme maximale pour les séquences suivantes :

x={-4, 2, 7, 8, -12, 5}

x={2, 6, 20, 2,15, 5}

x={-3, 6, 8, 6, -4, 2, 8, 10, -4}

x={-3, 2, 5, 1, -9, 9, -7}

b. Proposer un algorithme permettant de trouver la sous-séquence maximale d'un tableau x[] contenant NbElements.

c. Pour une séquence de 10 valeurs, déterminer le nombre de sommes effectuées, justifier votre réponse.

Nom, Prenom:_____ **Groupe:**_____

Travail pratique

Nous allons mesurer les temps d'exécution de différents algorithmes. Pour cela, la classe uTime.cpp est fournie. Elle permet de mesurer le temps écoulé en µs. Voici les principales méthodes:

```
class uTime {
public:

      // Constructor, initialize the timer
      uTime();

      // Return the elapsed time in us since last start/restart
      double          usSinceStart(void);

      // Restart the timer
      double          Restart(void);
...
```

Le programme s'éxécutant sur un système multi-tâches, vous veillerez à limiter le nombre de fenêtres ouvertes pour ne pas fausser les résultats. Voici un exemple permettant de mesurer le temps d'exécution de deux fonctions:

```
uTime Timer_us;

Timer_us.Restart();
Function_1 ();
printf ("f1 : %.0f us\n",Timer_us.usSinceStart());

Timer_us.Restart();
Function_2 ();
printf ("f2 : %.0f us\n",Timer_us.usSinceStart());
```

Exercice 1

Nous allons comparer les quatres algorithmes suivants résolvant le problème de la sous-séquence de somme maximale :

Algorithme 1
Une approche directe consiste à parcourir toutes les sous-séquences. On calcule la somme des éléments de chaque sous-séquence et on compare avec la somme maximale trouvée jusqu'à maintenant. Autrement dit, pour chaque $1 \leq l \leq u \leq n$ on calcule la somme :

$$x_{l...u} = x_l + ... + x_u.$$

Algorithme 2
Une méthode simple pour accélérer l'algorithme précédent est basée sur le fait que :

$$x_{l...u} = x_{l...u-1} + x_u.$$

On peut donc utiliser la somme $x_{l...u-1}$ calculée précédemment pour calculer la somme $x_{l...u}$.

Nom, Prenom:_____ Groupe:_____

Algorithme 3
C'est un algorithme basé sur le principe « diviser pour régner ». On divise la séquence en deux. On calcule la somme maximale dans chaque moitié et on trouvela sous-séquence de somme maximale qui contient les deux éléments adjacents au milieu.. La solution est la maximale parmi ces trois sommes.

Algorithme 4
Supposons que le problème soit résolu pour x_1 ... x_{i-1}. La solution pour x_1 ... x_i est soit la solution précédente, soit la séquence de somme maximale qui se termine avec l'élément x_i.

Les algorithmes sont implémentés dans la classe MaxSubSum.cpp dont voici les principales méthodes :

```
class MaxSubSum
{
public:
    MaxSubSum();         // Constructeur
    MaxSubSum(int N);    // Constructeur avec n éléments
    ~MaxSubSum();        // Destructeur
    void Print();        // Affiche la séquence
    int Algo1();         // Algorithme 1
    int Algo2();         // Algorithme 2
    int Algo3();         // Algorithme 3
    int Algo4();         // Algorithme 4
...
```

Notez que la méthode MaxSubSum::Printf() ne peut être utilisée que pour des séquences de petite taille. Voici un exemple de programme :

```
    // Crée une séquence de 20 éléments
    MaxSubSum Sub= MaxSubSum(20);

    // Affiche les éléments de la séquence
    Sub.Print();

    // Calcule et affiche la meilleure somme
    printf ("Algorithme 1 : %d \t",Sub.Algo1());
```

a. Proposer de façon intuitive la complexité de chaque algorithme.

Algorithme 1	Algorithme 2	Algorithme 3	Algorithme 4

b. Pour chaque algorithme, mesurer les temps d'exécution et compléter les fichiers tableurs OpenOffice (TP2_Ex1_Algox.ods)

Nom, Prenom:_____ **Groupe:**_____

c. [4 points] En déduire des résultats expérimentaux la complexité de chaque algorithme.

Algorithme 1	Algorithme 2	Algorithme 3	Algorithme 4

d. [0,5 point] Les temps de calcul sont approximés en utilisant un facteur proportionnel K: O(K), O(K.n) ... En vous servant de ce coefficient, estimez le temps nécessaire pour les algorithmes 1 et 2 pour analyser une séquence de 1 000 000 éléments.

e. [0,5 point] Comparer ces résultats avec les valeurs relevées pour les algorithmes 3 et 4.

Exercice 2

Dans cet exercice, nous allons nous intéresser au problème des tours de Hanoï (cf. TP1 ex4).

a. En procédant de façon similaire à l'exercice 1, compléter le fichier TP2_Ex2_Time.ods

b. [2 points] En déduire l'ordre de grandeur de la complexité en temps de l'algorithme

c. Compléter le fichier TP2_Ex2_Move.ods avec le nombre de pièces déplacées pour résoudre chaque problème.

d. [2 points] En déduire l'ordre de grandeur de la complexité en nombre de mouvements de l'algorithme

e. [1 point] Concluez.

Nom, Prenom:_____ **Groupe:**_____

Exercice 3

Dans cet exercice, nous allons nous intéresser au calcul de la suite de Fibonacci (cf. TP1 ex3). Nous allons comparer les formes itératives et récursives.

a. En procédant de façon similaire à l'exercice 1, compléter le fichier TP2_Ex3_Recursif.ods

b. [2 points] En déduire l'ordre de grandeur de la complexité de l'algorithme récursif.

c. En procédant de façon similaire à l'exercice 1, compléter le fichier TP2_Ex3_Itératif.ods

d. [2 points] En déduire l'ordre de grandeur de la complexité de l'algorithme iteratif.

e. [1 point] Concluez

Exercice 4

Dans cet exercice, nous allons nous intéresser au calcul de puissance (cf. TP1 ex2). Nous allons comparer les deux algorithmes proposés sur la base de la complexité en terme de multiplications.

a. Relever le nombre de multiplications nécessaires au calcul de 2^n avec la fonction Puissance1() et compléter le fichier TP2_Ex4_Puissance1.ods.

b. [2 points] En déduire l'ordre de grandeur de la complexité de la fonction Puissance1().

c. Relever le nombre de multiplications nécessaires au calcul de 2^n avec la fonction Puissance2() et compléter le fichier TP2_Ex4_Puissance2.ods.

d. [2 points] En déduire l'ordre de grandeur de la complexité de la fonction Puissance2().

e. [1 point] Avec les deux fonctions, comment évolue la complexité lorsque l'on double la taille des données ?