# EXAMEN

Semestre : 1 ☐ 2 ■

Session : Principale ☐ Rattrapage ■

Unité d'enseignement : Equipe complexité
Module (s) : Complexité appliquée à la RO
Classe(s) : 4ème année

Nombre d'exercices : 3 Nombre de pages : 3

Date : 21/06/2023 Heure 15h Durée :1h30

**Exercice 1 : (6points)**

Déterminer la complexité de chaque séquence ci-dessous **en justifiant votre réponse.**

| | | |
|---|---|---|
| **Seq1 :**<br>Pour i=1 à N faire<br>S=0<br>M=N-i<br>Pour j=1 à M faire<br>S=S+j<br>Fin pour<br>Fin pour | **Seq2 :**<br>S=0<br>Pour i=1 à N faire<br>Si (i mod 2=0)<br>N=N+1<br>Fin Si<br>S=S+i<br>Fin pour | **Seq3 :**<br>i=1<br>S=0<br>Tant que (i<N) faire<br>S=S+i<br>N=N-1<br>i=i+1<br>Fin Tant que |
| **$O(n^2)$** | **$O(n)$** | **$O(n)$** |
| **Seq4 :**<br>S=0<br>Pour j=1 à N faire<br>i=1<br>Tant que (i<N) faire<br>i=i*2<br>S=S+i<br>Fin Tant que<br>Fin pour | **Seq5 :**<br>S=0<br>Pour i=1 à N faire<br>Pour j=i à N faire<br>S=S+i<br>Fin pour<br>Fin pour | **Seq6 :**<br>S=0<br>i=1<br>Si (N mod 3==0)<br>Pour i=1 à N faire<br>S=S+i<br>Fin pour<br>Sinon<br>Tant que (i<N) faire<br>i=i /3<br>S=S+i<br>Fin Tant que<br>Fin Si |
| **$O(n\log_2 n)$** | **$O(n^2)$** | **$O(n)$** |

**Exercice 2**

Soit le code ci-dessous écrit en langage C où la fonction **Recherche** permet de vérifier en premier lieu si le tableau donné en paramètre est trié à l'aide de la fonction **Est_trie.** S'il est trié, on effectue une recherche d'une valeur **x** dans le tableau en utilisant la méthode dichotomique (**Recherche_dichotomique),** sinon on effectue une recherche en utilisant la méthode séquentielle (**Recherche_sequentielle).**

1. Pour **n=4**, donner une proposition des valeurs des paramètres **tab** et **x** afin de maximiser le temps d'exécution du programme et donner d'autres valeurs qui le minimise.
   (1pts) exemple de valeurs qui maximisent le nombre d'opérations : [2,3,4,1] et x=1
   (1pts) exemple de valeurs qui minimisent le nombre d'opérations : [4,3,2,1] et x=4

2. Calculer la complexité du programme dans le pire et au meilleur des cas à O près (en terme de comparaison) 3pts

**Est_trie a une complexité : au pire O(logn), au meilleur O(1)**
**Recherche_dichotomique : au pire O(logn), au meilleur O(1)**
**Recherche_sequentielle : au pire O(n), au meilleur O(1)**
**→ la complexité totale au pire O(n)+O(n)=O(n), au meilleur O(1)+O(1)=O(1)**

3. Déduire une optimisation du programme.

```
Void Recherche( int tab[], int n, int x)
{
int pos ;
if (Est_trie(tab,n)==1)
    pos=Recherche_dichotomique(tab,n,x) ;
else
    pos=Recherche_sequentielle(tab,n,x) ;
if(pos==-1)
    printf(''la valeur %d non trouvable dans le tableau'', x) ;
else
    printf(''la valeur %d existe dans la position %d  du tableau'',x,pos) ;
}
```

```
int Est_trie (int tab[],int n)
{  int i=0, trie=1 ;
   while (trie==1 && i<n-2)
    {   if (tab[i] <= tab[i+1])
           i++ ;
        else
           trie=0 ;
    }
     return trie ;
}
```

**L'algorithme optimisé :**

```
Void recherche(int tab[],int n, int x)
{
  If (Recherche_sequentielle(tab,n,x)==-1) printf ('trouvé') ;
  Else printf('non trouvé') ;
}
```

**au pire O(n), au meilleur O(1)**

**Exercice 3 :**

On se propose de calculer le Produit Matriciel des deux matrices carrées **A** et **B** de taille ***n***. Soit ***C*** la matrice carrée de taille ***n*** résultantes. Pour ce faire, ils existent plusieurs algorithmes.

1. Un premier algorithme itératif appelé **AlgoPM_naïf (ci-dessous)** Calculer la complexité de **AlgoPM_naïf**

```
AlgoPM_naïf (A, B,C)
   For i = 1 to n do
        For j=1 to n do
            C(i,j)=0
            For k=1 to n do
                C(i,j)= C(i,j) +A(i,k)*B(k,j)
            End For
         End For
     End For
End_ AlgoPM_naïf
```

**1pt : $O(n^3)$**

2. Un deuxième algorithme récursif se basant sur le principe « Diviser pour Régner » appelé **AlgoPM_Rec**. Cet algorithme décompose les matrices ***A***, ***B*** et ***C*** en sous-matrices de taille *n*/2**n*/2 comme suit :

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}; B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}; C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

L'équation $C = A*B$ peut alors se récrire :

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix};$$

avec $c_{11}=a_{11}*b_{11}+a_{12}*b_{21}$
$c_{12}=a_{11}*b_{12}+a_{12}*b_{22}$
$c_{21}=a_{21}*b_{11}+a_{22}*b_{21}$
$c_{22}=a_{21}*b_{12}+a_{22}*b_{22}$

Calculer le nombre d'opérations de multiplications de matrices carrées de taille n/2 et le nombre d'addition de telles matrices. Donner alors l'équation de récurrence de la complexité de **AlgoPM_Rec** puis la calculer.

2pts : $T(n)=8T(n/2)+O(n^2) \rightarrow T(n)= \Theta(n^{\log_2 8})= \Theta(n^3)$

3. Un troisième algorithme récursif semblable à **AlgoPM_Rec** appelé **AlgoPM_Strassen.** Cet algorithme applique la méthode de Strassen utilisant **7** nouvelles matrices $\mathbf{M_i}$ qui servent à exprimer les $\mathbf{c_{ij}}$ avec uniquement 7 multiplications au lieu de 8.

$M_1=(a_{11}+a_{22})*(b_{11}+b_{22})$
$M_2=(a_{21}+a_{22})*b_{11}$
$M_3=a_{11}*(b_{12}-b_{22})$
$M_4=a_{22}*(b_{21}-b_{11})$
$M_5=(a_{11}+a_{12})*b_{22}$
$M_6=(a_{21}-a_{11})*(b_{11}+b_{12})$
$M_7=(a_{12}-a_{22})(b_{21}+b_{22})$

Les $c_{ij}$ sont alors exprimées comme suit :
$c_{11}=M_1+M_5-M_5+M_7$
$c_{12}=M_3+M_5$
$c_{21}=M_2+M_4$
$c_{22}=M_1-M_2+M_3+M_6$

Donner la formule de récurrence de la complexité de **AlgoPM_DeStrassen** puis la calculer.

3pts : $T(n)=7T(n/2)+O(n^2)= \Theta(n^{\log_2 7})= \Theta(n^{2,8})$