EXAMEN

Module: Algorithmique et complexité	**Niveau:** 3 A
Documents: Non autorisés	**Durée:** 1h30 min
Enseignants : I. Denden, R. Guetari, S. Mesfar et O. Mourali	**Date :** 7 Novembre 2012

Exercice-1 : (2 points)
Donner le nombre de « plop » affichés par le programme Tata présenté dans ***Encadré-1*** et justifier votre réponse.

Encadré-1
```
void Tata (int A[] , int N,int M)
{   int i,j ;
    for(i=0;i<M;i++)
    {
        for(j=1;j<N;j=j*3)
        {
            Printf("plop %d\n",A[i]);
        }
    }
}
```

Exercice-2: (6 points)
Considérer le sous-programme présenté dans ***Encadré-2***

1. Que retournent et qu'affichent les appels suivants ?
   a. dosomething (2, "L=")
   b. dosomething (3, "")
2. Prouver que, pour tout entier naturel n l'appel de cette procédure engendre l'affichage d'exactement 2^n chaînes de caractères.
3. Expliciter les différentes étapes pour la dérécursivation de cet algorithme.
4. Donner l'algorithme correspondant à chaque étape

Indication : Pour dérécursiver un algorithme ayant une récursivité multiple, il suffit de procéder à l'élimination de la récursivité des appels récursifs (terminaux et non terminaux) l'un à la suite de l'autre.

Encadré-2
```
void dosomething (int n, string s)
{
   if n <= 0 then
   {  Printf (s)
        }
   else
   {   dosomething (n - 1, s + "a");
       dosomething (n - 1, s + "b");
   }
}
```

Exercice-3: (5 points)
Soit U la suite définie par :

$$U_n = \begin{cases} 3 & si \quad n = 0 \\ \dfrac{3*U_{n-1}}{n^2} & si \quad n > 0 \end{cases}$$

5. Ecrire une fonction récursive « int Suite(int n) » pour calculer le terme U_n de la suite.
6. Donner le type de récursivité de cet algorithme (terminale ou non). Justifier
7. Calculer la complexité de la fonction Suite en nombre d'opérations à un O près (avec explication).
8. Ecrire une fonction itérative int IterSuite(int n) issue de la dérécursivation de la fonction Suite.

Exercice-4 : (7 points)
Etant donné un arbre binaire de recherche A. Chaque nœud contient : **info** : une valeur réelle, **FG** : un pointeur sur le fils gauche et **FD** : un pointeur sur le fils droit. La structure « arbre » contient un seul pointeur « racine » pointant sur le premier nœud de l'arbre (voir ***Encadré-3***).

a- Ecrire une fonction float Moyenne(struct arbre A) permettant de retourner la valeur moyenne des nœuds d'un arbre. On rappelle que la valeur moyenne d'un ensemble d'éléments est égale à la somme de leurs valeurs divisée par leur nombre.

Encadré-3
```
struct node
{   float  info ;
    struct node *FG ;
    struct node *FD;
};
struct arbre
{   struct node *racine ;} ;
```

b- Déterminer la relation de récurrence permettant de décrire la complexité de l'algorithme proposé
c- En déduire cette complexité à un *O* près.
d- On suppose que l'arbre binaire est un arbre binaire de recherche. Donner une fonction float Min(struct arbre A) permettant de donner la valeur minimale de l'arbre.
e- Estimer sa complexité à un *O* près (avec explication).

Rappel :

<u>Résolution des équations de récurrence :</u>

1- *Equation linéaire d'ordre 1 :* $u_n = a.u_{n-1} + f(n)$ ***alors*** $u_n = a^n(u_0 + \sum_{i=1}^{n} \frac{f(i)}{a^i})$

2- *Récurrence de type « diviser pour régner »*

Si on a: $T(n) = a.T(n/b) + c.n^k$ *alors T(n) peut alors être approximée comme suit :*

	Si on a :			*Alors le coût est :*
1	$a > b^k$	*Ou*	$f(n) \equiv O(n^{\log_b(a)-\varepsilon})$ avec $\varepsilon > 0$	$T(n) = \Theta(n^{\log_b a})$
2	$a = b^k$	*Ou*	$f(n) \equiv \Theta(n^{\log_b(a)})$	$T(n) = \Theta(n^k \log n)$
3	$a < b^k$	*Ou*	$f(n) \equiv \Omega(n^{\log_b(a)+\varepsilon})$ avec $\varepsilon > 0$, et pour un $d < 1, af(\frac{n}{b}) \leq df(n)$	$T(n) = \Theta(f(n)) = \Theta(n^k)$

<u>Somme des termes des suites usuelles :</u>

La somme des termes d'une suite arithmétique U de 0 à n est : $\sum_{0 \leq p \leq n} U_p = \frac{n+1}{2}(U_0 + U_n)$.

La somme des termes d'une suite géométrique U de raison *r* en partant de 0 à n est: $U_0 \frac{r^{n+1} - 1}{r - 1}$