Université des Sciences et de la Technologie Houari Boumediène
Faculté d'Electronique et d'Informatique
Département d'Informatique
LMD Master 1ère Année IL 2011/12
Module ''Algorithmique Avancée et Complexité''
Date : 02/01/2012

Corrigé de l'interrogation
<Corrigé préparé par le responsable du module, Mr ISLI>

Exercice 1 <4 points=2+2>:
Soit f(n)=n*logn le nombre d'opérations élémentaires du pire cas d'un algorithme A.

1. Montrez que f(n)=O(n^2).
2. A-t-on f(n)=Θ(n^2) ? Expliquez.

Solution :

1) Pour montrer que f(n)=O(n^2), il suffit de trouver c>0 et n_0≥0 (ou de montrer leur existence) tels que ∀n≥n_0 f(n) ≤c*n^2 :
   n*logn≤c*n^2 //on divise les deux membres par n^2
   $\frac{logn}{n}$≤c //$\lim_{n\to+\infty}\frac{logn}{n}$=0 donc le c et le n_0 demandés existent !
   Prendre c=1 et n_0=1
   Conclusion : on a bien f(n)=O(n^2)
2) Pour avoir f(n)=Θ(n^2), vu qu'on a déjà f(n)=O(n^2), il faut avoir en plus n^2=O(f(n)). Tentons donc de montrer n^2=O(n*logn) en cherchant c>0 et n_0≥0 tels que ∀n≥n_0 n^2≤c*nlogn :
   n^2≤c*nlogn //on divise les deux membres par c*n^2
   $\frac{1}{c}$≤$\frac{logn}{n}$
   $\lim_{n\to+\infty}\frac{logn}{n}$=0 donc le c et le n_0 demandés n'existent pas : on ne peut pas trouver de c>0 tel que $\frac{1}{c}$≤ε, ∀ε>0 !
   Conclusion : f(n)≠Θ(n^2)

Exercice 2 <4 points >:
Soient A1, A2 et A3 trois algorithmes conçus pour la même tâche T, et dont les nombres d'opérations élémentaires du pire cas sont, respectivement, $f_1(n)=n^{20}$logn, $f_2(n)$=n! et $f_3(n)=8^n$. Comparez les complexités des trois algorithmes. Expliquez.

Solution :
$f_1(n)=n^{20}$logn=Θ(n^{20}logn)
$f_2(n)$=n!= Θ(n!)
$f_3(n)=8^n$= Θ(8^n)
$\lim_{n\to+\infty}\frac{n^{20}logn}{8^n}=\lim_{n\to+\infty}\frac{8^n}{n!}$=0 : Pour les grandes valeurs de n, n^{20}logn est négligeable devant 8^n, qui à son tour est négligeable devant n!

Donc l'ordre décroissant de préférence des trois complexités est Θ(n^{20}logn), Θ(8^n), Θ(n!) : la complexité de l'algorithme A1 est meilleure que celle de l'algorithme A3, qui à son tour est meilleure que celle de l'algorithme A2

Exercice 3 <7 points=1+(0,5+1+0,5+1)+3>:

1. Illustrez à l'aide d'un exemple la notion d'instance d'un problème.
2. Définissez les notions suivantes de la théorie de la NP-complétude :
    a. Certificat
    b. Algorithme de validation
    c. La classe NP
    d. Problème NP-complet
3. Donnez un algorithme polynômial de validation pour le problème **Subset_Sum** (somme de sous-ensemble) suivant :
    - **Description :** un ensemble $S=\{a_1,...,a_n\}$ de n entiers ; et un entier m
    - **Question :** S admet-il un sous-ensemble dont la somme des éléments est m ?

    Justifiez la polynômialité de l'algorithme.

Solution :

Pour les questions 1. et 2. voir cours

Question 3 : nous donnons ci-après, sous forme d'une fonction booléenne, un algorithme de validation validation_subs pour le problème **Subset_Sum** : l'algorithme aura comme arguments un certificat c et une instance I de **Subset_Sum** : l'instance I est une paire (S,m) où S est un ensemble de n entiers donné sous forme d'un tableau, et m est un entier ; le certificat c est un tableau de booléens (0 et 1) de taille n, représentant un sous-ensemble de S : c[i]=1 signifie que l'élément numéro i de S appartient au sous-ensemble représenté par c

```
Booléen validation_subs(c,S,m){
    somme=0
    Pour i=1 à n faire
        Si c[i]=1 alors somme=somme+S[i] finsi
    Fait
    Si somme=m alors retourner VRAI sinon retourner FAUX finsi
    }
```

Le nombre f(n) d'opérations élémentaires du pire cas de l'algorithme de validation est clairement un polynôme de degré 1 en n. Donc f(n)=O(n) : l'algorithme de validation est bien polynômial, et le problème **Subset_Sum** appartient donc à la classe NP.

Exercice 4 <5 points=3+2>: Donnez un algorithme linéaire d'insertion d'un nouvel élément dans une liste chaînée triée, de telle sorte que la liste résultante reste triée. Justifiez la linéarité de l'algorithme.

Solution :

On suppose que les éléments de la liste sont des enregistrements à deux champs : un champ clef (valeur) de type entier, et un champ suiv de type pointeur sur un élément de la liste. On suppose aussi que la liste est triée par ordre croissant. La liste L est complètement donnée par un pointeur tete pointant sur son premier élément. L'algorithme suivant, Insertion(a,L), insère un nouvel élément a dans la liste triée L

```
Insertion(a,L){
Créer(N)    //création dynamique d'un enregistrement de même type que les éléments de L…
            //le nouvel enregistrement est pointé par N…
N.clef=a    //la valeur à insérer dans L est la clef de l'enregistrement nouvellement créé…
si tete=NIL alors                   //la liste est vide…
    tete=N
    tete.suiv=NIL
sinon
    P=tete
    Pred=tete
    tant que P≠NIL et a>P.clef faire
            Pred=P
            P=P.suivant
    fait
    si P=NIL alors          //insertion en fin de liste…
            Pred.suiv=N
            N.suiv=NIL
    sinon
            si P=tete alors         //insertion en début de liste…
                    N.suiv=tete
                    tete=N
            sinon                   //insertion en milieu de liste
                    Pred.suiv=N
                    N.suiv=P
            finsi
    finsi
finsi
}
```

Le nombre f(n) d'opérations élémentaires du pire cas de l'algorithme est clairement un polynôme de degré 1 en n, la taille de la liste : donc f(n)=O(n), et l'algorithme est bien linéaire.