COMPLEXITE

DEUXIEME PARTIE : RECURSIVITE

EXERCICE 20

Résoudre l'équation de récurrence $u_n=3u_{n-1}-2u_{n-2}+n$, $u_0=0$, $u_1=0$

Commençons par rechercher les solutions de l'équation homogène correspondante : $u_n=3u_{n-1}-2u_{n-2}$

Pour cela, considérons le polynôme caractéristique associé : $x^2-3x-2=0$.

Il a deux racines simples 1 et 2.

Donc l'ensemble des solutions de l'équation homogène est de la forme $h^n=a2^n+b1^n$

On sait qu'il existe une solution particulière de l'équation non homogène de la forme $s_n= P(n)1^n$, où P(n) est un polynôme en n de degré 2. On cherche donc c,d,et e tels que

$s_n =cn^2+dn+e$ soit solution de l'équation $u_n=3u_{n-1}-2u_{n-2}+n$

On doit donc avoir $cn^2+dn+e= 3(c(n-1)^2+d(n-1)+e)-2(c(n-2)^2+d(n-2)+e)+n$ pour tout n

Et donc c=3c-2c [coefficient du terme en n carré]

d=3(-2c+d)-2(-4c+d)+1 [coefficient du terme en n]

e=3(c-d+e)-2(4c-2d+e) [terme constant]

D'où 2c+1=0 et 5c-d=0 et finalement on obtient comme solution particulière

$s_n =-(1/2)n^2 -(5/2)n+e$

Toutes les solutions de l'équation $u_n=3u_{n-1}-2u_{n-2}+n$, sont donc de la forme

$u_n= a2^n-1/2\ n^2 -5/2n+k$

Puisque les deux premiers termes de la suite sont nuls, on a 0=a+k et 0=2a-3+k donc a=3 et k=-3

Et finalement $u_n= 3.2^n-(1/2)\ n^2 -(5/2)n-3$

EXERCICE 21

Evaluer la complexité en nombre de multiplications de

```
public int sommeFactoriel(int n) {
        int factorieln ;
                if (n<=1) {
```

```
                    return n+1;
                }
                else {
                    factorieln = n * (sommeFactoriel (n-1) – sommeFactoriel (n-2)
        ) ;
        return
sommeFactoriel (n-1) + factorieln ;
                }
  }
```

```
public int sommeFactoriel(int n){
        int factorieln , somme ;
  if (n<=1) {
                return n+1;
        }
  else {
                somme = sommeFactoriel (n-1) ;
        factorieln = n * (somme– sommeFactoriel (n-2)) ;
        return somme + factorieln ;
   }
}
```

Notons c_n le nombre de multiplications induites par le calcul de sommeFactoriel, cette suite vérifie alors les relations de récurrence suivantes

a) $c_n = 2c_{n-1} + c_{n-2} + 1, \quad c_0 = c_1 = 0$

b) $c_n = c_{n-1} + c_{n-2} + 1, \quad c_0 = c_1 = 0$

c) $c_n = c_{n-1} + 1, \quad c_0 = c_1 = 0$

Et donc on obtient selon les cas, une complexité

a) $c_n = \alpha(1+\sqrt{2})^n + \beta(1-\sqrt{2})^n + \gamma = \Theta((1+\sqrt{2})^n)$

b) $c_n = \alpha(\frac{1+\sqrt{5}}{2})^n + \beta(\frac{1-\sqrt{5}}{2})^n + \gamma = \Theta((\frac{1+\sqrt{5}}{2})^n)$

c) $c_n = n$ -1

EXERCICE 22

On se propose de multiplier entre eux des « grands nombres ».

a) Si l'on utilise la méthode naïve, combien de multiplications élémentaires sont effectuées ?
Soient U et V deux nombres de 2n chiffres en base B.
On peut donc écrire $U=U_1B^n+U_2$ et $V=V_1B^n+V_2$ où U_1, U_2, V_1, V_2 sont des nombres à n chiffres en base B.

b) On utilise l'égalité $(U_1B^n+U_2)(V_1B^n+V_2)= U_1V_1B^{2n} +(U_1V_2 + U_2V_1)B^n + U_2V_2$ pour calculer récursivement la multiplication. C'est à dire que l'on ramène le problème d'une multiplication de deux nombres de 2n chiffres à celui de 4 multiplications de deux nombres de n chiffres, 4 décalages et trois additions. On suppose qu'additions et décalages s'effectuent en

$\Theta(n)$. Etablir une relation de récurrence permettant d'évaluer la complexité de cet algorithme récursif de multiplications et la résoudre.
c) On utilise maintenant l'égalité
$(U_1B^n+U_2)(V_1B^n+V_2)= U_1V_1B^{2n} +((U_1-U_2)(V_2-V_1) + U_2V_2 +U_1V_1)B^n + U_2V_2$ pour calculer récursivement la multiplication. C'est à dire que l'on ramène le problème d'une multiplication de deux nombres de 2n chiffres à celui de 3 multiplications de deux nombres de n chiffres, 5 décalages et 6 additions. On suppose qu'additions et décalages s'effectuent en $\Theta(n)$. Etablir une relation de récurrence permettant d'évaluer la complexité de cet algorithme récursif de multiplications et la résoudre.

a) La méthode naïve effectue n^2 multiplications.
b)
Notons m_n, la complexité de la multiplication de deux nombres de n chiffres, on a
On a $m_n=4m_{n/2}+\alpha n+\beta$

Cette équation est de la forme $m_n=am_{n/b}+P(n)$ où P est un polynôme de degré un.
Je compare P(n) et $n^{\log_2 4} =n^2$ .Comme P(n) est négligeable devant n^2, j'en déduit que la complexité
est en $\Theta(n^2)$

c) Cette fois-ci l'on obtient $m_n = 3m_{k-1} + Q(n)$ et Où Q est un polynôme de degré un.
Cette fois encore, Q(n) est négligeable devant $n^{\log_2 3}$
La complexité est donc en $\Theta(n^{\log_2 3})$ et l'on a gagné par rapport à la méthode naïve.

EXERCICE 23

On se propose dans cet exercice de calculer la complexité de plusieurs algorithmes dont le but est de fusionner les p listes triées de longueur n contenues dans un tableau de listes en une seule liste triée de longueur np.

On suppose définie une classe Liste contenant entre autre une méthode permettant de fusionner une liste l1 triée de longueur n1 et un liste triée l2 de longueur n2 dont la signature est

public static Liste fusion (Liste l1, Liste l2)

et la complexité est en $\Theta(n1+n2)$.

Question 1

Une première solution consiste en l'utilisation de la méthode suivante

```
public static Liste fusionMultiple(Liste[] mesListes) {
        Liste L=mesListes[1];
        for (int i=2; i < mesListes.length; i++){
```

```
                L= Liste.fusion(L,mesListes[i]);
                }
        return L;
}
```

Déterminer la complexité de cette méthode en fonction de n et de p.

Question 2

On suppose maintenant que p est une puissance de 2 et l'on propose maintenant d'utiliser l'algorithme de multifusion récursif suivant :

Pour multifusionner p listes de taille n
Si p=2 utiliser fusion
Sinon Multifusionner (récursivement) les p/2 première listes
Multifusionner (récursivement) les p/2 dernières listes
Utiliser fusion pour fusionner le résultat des deux premières étapes.

Soit c(n,p) = la complexité de la fusion de p listes de taille n par cette méthode.

a) Déterminez la relation de récurrence suivie par cette suite, ainsi que c(n,2).

Posez d(n,p)=c(n,p)/n.

b) Déterminez la relation de récurrence suivie par cette suite.

c) Montrez que d(n,p) ne dépend pas de n. On pourra montrer par induction sur p que pour tout p >=2, d(n,p)=d(1,p) pour tout n >0.

d) Posez d(1,p)=f(p), et déterminez l'équation de récurrence suivie par f(p). Résoudre cette équation. En déduire c(n,p).

Corrigé

Question 1
Il est facile de montrer les invariants suivants :

```
public static Liste fusionMultiple(Liste[] mesListes) {
        Liste L=mesListes[1];
        for (int i=2; i < mesListes.length; i++){// L est une liste de longueur (i-1).n
                L= Liste.fusion(L,mesListes[i]); // Cette ligne a une complexité en Θ(in)
                }
        return L;
}
```

On en déduit que la complexité de fusion multiple est en $\Theta(p^2n)$

Question 2

a) On a c(n,p)=2c(n,p/2)+pn
et c(n,2)=n
b) d(n,p)=2d(n,p/2)+p et d(n,2)=1
c) On montre par récurrence sur k que $d(n,2^k)=g(k)$ (ne dépend que de k)

Pour k=1 c'est vrai et g(1)=1
Supposons le résultat vrai pour l'entier k
$d(n, 2^{k+1})=2g(k)+2^k$, et est donc bien indépendant de n.

d) On a donc f(p)=2f(p/2)+p, on a donc f(p)=plogp, et par suite c(n,p)=nplogp.
L'algorithme récursif de la question 2 a donc une complexité meilleure que l'algorithme de la question un.

EXERCICE 24 : ANALYSE DE COMPLEXITE ET PREUVE D'ALGORITHME RECURSIF.

On considère le programme java récursif suivant où b est une constante entière
On suppose défini un objet *table* à partir d'une classe Table dérivée de la classe Vector en y ajoutant la méthode table.echanger (int i, int j) qui échange table.elementAt(i) et table.elementAt(j).

```
public void T(int debut, int fin){ // opère sur la Table table dans la tranche table[debut..fin]
        int n=fin-debut+1 ; //la dimension de la tranche
        if (n>1) { if (n=2) {// tri par ordre croissant des deux éléments de la tranche
                        if (table.elementAt(debut)>table.elementAt(fin)) {table.echanger(debut,
                fin) ;}}
                  else {  T( debut, debut+n/b-1) ;
                          T( fin-n/b+1, fin) ;
                T(début, début+n/b-1) ;
                        }
                  }
        }
```

a) Etablir la relation de récurrence vérifiée par la complexité de cet algorithme
b) Si b=3/2, (dans ce cas bien sûr l'algorithme utilise la partie entière de n/b) quelle en est la complexité ?
c) Question bonus : démontrer que si b=3/2, T est un tri

Corrigé

a) C(n)=3C(n/b)+1 et C(1)=C(2)=1
b) On est ici dans le cadre d'une méthode de type diviser pour regner ou l'on remplace un problème de taille n , par a=3 problèmes de taille n/b=2n/3, les phases de division et de recombinaison des solutions prenant un temps constant.
On a donc $\log_b a=\log_{3/2} 3$, Comme la complexité des phases de division et recombinaison (temps constant est négligeable devant $n^{\log_b a}$, la complexité de la méthode est en $\Theta(n^{\log_b a})$.
c) La preuve se fait par induction, je suppose que T fait bien un tri sur les tables de taille 2n/3.

Soit maintenant une table de taille n. Je partitionne fictivement T en trois sous table contiguës T1,T2 et T3 de taille n/3.
Par hypothèse de récurrence après
T(début, début+n/b-1) qui opère sur T1 union T2
T1 et T2 sont triées et tous les éléments de T1 sont inférieurs ou égaux à ceux de T2

Après T(fin-n/b+1, fin) qui opère sur T2 union T3
T2 et T3 sont triées et tous les éléments de T2 sont inférieurs ou égaux à ceux de T3
De plus les éléments de T3 sont tous supérieurs ou égaux à ceux de T1, car dans T2 union T3 il y a au moins n/3 éléments supérieurs ou égaux à tous ceux de T1.
T3 est donc d'hors et déjà composé des n/3 plus grands éléments de T, tous déjà triés.

Après T(début, début+n/b-1) qui opère a nouveau sur T1 union T2 toute la table est donc triée.