Complexité : corrigé de l'examen

Exercice 1.- a) $f(n) = 10n^3 + 15n^4 + 3n^2 2^n = \Theta(n^2 2^n)$ car

$$3n^2 2^n \leq f(n) \leq 5n^2 2^n \quad \forall n \geq 10$$

b) $10n^2 + 5 = O(n)$; **Faux** car :
Supposons que : $\exists c, n_0 > 0$ tels que $\forall n \geq n_0 \quad 10n^2 + 5 < c * n$.
Ceci implique que : $\quad 10n^2 - c * n + 5 < 0 \quad (1)$.
En calculant les racines de cette équation, on obtient : $n_1 = \frac{c-\sqrt{c^2-200)}}{20}$; $\quad n_2 \frac{=c+\sqrt{c^2-200}}{20}$.
Et l'équation (1) n'est vérifiée que pour des petites valeurs de $n \in]n-1, n-2[$, donc contradiction.

c) $n^3 * 2^n + 6n^2 * 3^n = O(n^3 * 2^n)$; **Faux** car :
Supposons que $n^3 * 2^n + 6n^2 * 3^n = O(n^3 * 2^n)$. Ceci implique que :
$\exists c, n_0 > 0$ tels que $\forall n \geq n_0, \quad n^3 * 2^n + 6n^2 * 3^n < c * n^3 * 2^n$. Ceci implique que $c > 1 + \frac{6*3^n}{n*2^n}$ qui est une valeur croissante qui dépend de n et non constante, d'où la contradiction.

d) $2n^2 * 2^n + n^2 logn = \Theta(n^2 * 2^n)$ **Vrai** car :
$c_1 * n^2 * 2^n \leq 2n^2 * 2^n + n^2 logn \leq c_2 * 2n^2 * 2^n$
D'une part $c_1 * n^2 * 2^n \leq 2n^2 * 2^n + n^2 logn$ évident, et $c_1 = 2$. D'autre part $n^2 * lon < n^2 * 2^n$ pour $n \geq 2$.
Donc $2n^2 * 2^n + n^2 logn \leq 3 * 2n^2 * 2^n$; donc $c_1 = 2, c_2 = 3, n_0 = 2$.

e) Résoudre :

$$T(n) = \begin{cases} 1 & \text{si} \quad n = 1 \\ 7 * T(n/2) & \text{si} \quad n > 1 \end{cases}$$

$$\begin{aligned} T(n) = \quad & 7 * T(n/2) \\ & 7 * 7 * T(\tfrac{n}{2^2}) \\ & 7 * 7 * 7 * T(\tfrac{n}{2^3}) \\ & \cdots \\ & 7^k * T(\tfrac{n}{2^k}) \end{aligned}$$

On pose $n = 2^k, \Longrightarrow logn = k * log2$, donc $k = log_2 n$. D'où $T(n) = 7^k * T(\frac{2^k}{2^k}) = 7^k$
$T(n) = \Theta(7^{log_2 n}) = \Theta(n^{log_2 7})$. Comme $log_2 7 = 2.81$ donc $T(n) > n^2$.

Exercice 2.- Soit $S[1..n]$ un tableau d'entiers et x un entier donné. Écrire un algorithme qui permet de vérifier s'il existe ou non dans S deux entiers x_1, x_2 tels que $x = x_1 + x_2$.

a) **Algorithme naïf**

```
fonction verifier (S[1..n]:tableau entier; n,x: entier):booleen

  pour i=1 à n-1 faire
     pour j=i+1 à n faire
        si (S[i] + S[j] == x) alors retourner 1;
retourner 0;

finFonction;
```

La complexité : On compte $nbComp$ le nombre de comparaisons :

Pour i=1 on fait n-1 comparaisons ; pour i=2 on fait n-2 comparaisons ... pour i=n-1 on fait 1 comparaison

Donc : $nbComp = (1 + 2 + ... + n - 1) = n(n-1)/2$. Donc cette solution est en $O(n^2)$.

b) **Un algorithme en** $\Theta(n * logn)$

Le tableau étant trié, on fixe une valeur de $x_1 = S[i]$ et on cherche $x_2 = x - x_1$ par dichotomie dans le tableau. Donc x_1 vaut successivement $S[1], S[2], \ldots, S[n]$. Pour un tableau de taille n, cette recherche est en $\Theta(log_2 n)$. On la répète dans la boucle n fois. Donc la complexité est en $\Theta(n * logn)$.

```
 pour i = 1 à n faire
x2 = x - S[i];
si (recherDicho (S, x2) == 1) alors retourner 1;
fsi;
finPour;

retourner 0;
```

Exercice 3.- Procédure qui calcule l'élément de fréquence maximale :

a) Algo :

```
Procédure FrequenceMaximal (A[1..n]: tableau entier, n:entier)

        /* freqMax: est la fréquence maximale recherchée
        M= A[i] est l'élement qui a la fréquence maximal
        freq: est la fréquence de l'élement courant */


M= A[1]; freq = 1; freqMax = 1;

Pour i =2 à n faire
   si A[i] == A[i-1] alors freq ++;
                           si freq > freqMax alors freqMax = freq;
                                                     M= A[i];
                                             sinon freq =1;
                            fsi
fsi;

finPour;
finProcédure;
```

b) Invariant :

"à la fin de la k-ème itération il existe $1 <= j <= k$ tel que $M_k = A[j]$ et $freqMax_k = max(freq_j)$"

c) Complexité : Complexité : $\Theta(n)$ car le tableau est parcouru entièrement...

d) Validité : La terminaison est évidente (l'algorithme consiste en une boucle finie qui termine quand $i = n$).

   La consistence : par récurrence sur k. (cf. validité d'un algorithme de recherche du maximum dans un tableau).

e) Procédure récursive

```
Procedure MaxFreqRec (A, i, n)

Si i<n  Alors
            Si A[i] = A[i-1] Alors freq ++
                    Si freq > freqMax Alors freqMax = freq
                                        Sinon freq =1;
                    fsi
                    MaxFreqRec (A, i+1, n)
                fsi
fsi
finProcedure
```

   Appel initial avec freq =1, freqMax =1, M=A[1], i=2

f) Invariant et complexité : similaire au cas itératif