CSI 3505, Automne 2006 – Devoir 2

Les devoirs doivent être déposé dans la boite appropriée située au premier étage de l'édifice SITE avant midi, lundi le 23 octobre 2006. Les devoirs remis en retard ne seront pas acceptés.

1. (5 points)

Soit un algorithme de type « Diviser pour régner » qui divise une instance de taille n en 9 sous-instances de taille n/3. De plus, n opérations sont nécessaires pour diviser le problème et recombiner les solutions. Supposez que n est toujours une puissance de 3 et que le problème de taille 1 ne nécessite aucune opération.

a) Déterminez la relation de récurrence T(n) décrivant le temps d'exécution de cet algorithme.

$T(1) = 0$
$T(n) = 9*T(n/3) + n$

b) Résolvez la relation de récurrence pour obtenir une formule non-recursive pour T(n) en utilisant la technique de substitution.

[Rappel : $k^0 + k^1 + k^2 + \ldots + k^n = (k^{n+1}-1)/(k-1)$]

$T(n) = 9*T(n/3) + n$
$= 81*T(n/9) + 9*(n/3) + n$
$= 729*T(n/27) + 81*(n/9) + 9*(n/3) + n$
$= ...$
$= \sum_{i=0}^{\log_3 n-1} 9^i \frac{n}{3^i}$
$= n \sum_{i=0}^{\log_3 n-1} 3^i$
$= n \cdot (3^{\log_3 n} - 1) / 2$
$= n(n-1)/2$

2. (5 points)

Soit la relation de récurrence suivante :

$T(0) = 0$
$T(n) = T(n-1) + n^2$ pour $n>0$

Démontrez, par induction, que $T(n) = n^3/3 + n^2/2 + n/6$.

Cas de base : $T(0) = 0^3/3 + 0^2/2 + 0/6 = 0$

Induction : supposons que $T(n) = n^3/3 + n^2/2 + n/6$.

Alors, $T(n+1) = T(n) + (n+1)^2$
$= n^3/3 + n^2/2 + n/6 + n^2 + 2n + 1$
$= n^3/3 + 3n^2/2 + 13n/6 + 1$
$= (n^3 + 3n^2 + 3n + 1)/3 + (n^2 + 2n + 1)/2 + (n+1)/6$
$= (n+1)^3/3 + (n+1)^2/2 + (n+1)/6$
Donc si c'est vrai pour n, c'est aussi vrai pour n+1

3. (5 points)

Soit $f(n) = n^{1/2} * \log_2 n$ et $g(n) = n$.

Est-ce que $f(n) \in \Theta(g(n))$? Sinon, est-ce que $f(n) \in O(g(n))$ ou $\Omega(g(n))$? Démontrez votre réponse. [Suggestion : calculez la limite pour n tendant vers l'infini de f(n)/g(n) en utilisant la règle de l'hospital.]

Rappel : $\frac{dx^k}{dx} = kx^{k-1}, \quad \frac{d\log_e x}{dx} = \frac{1}{x}, \quad \log_a b = \frac{\log_c a}{\log_c b}$

$$\lim_{n\to\infty} \frac{n^{1/2}\log_2 n}{n} = \lim_{n\to\infty} \frac{\log_2 n}{n^{1/2}} = \lim_{n\to\infty} \frac{\ln n / \ln 2}{n^{1/2}} = \lim_{n\to\infty} \frac{1/n\ln 2}{1/2n^{1/2}}$$ (règle de l'hospital)

$$= \lim_{n\to\infty} \frac{2}{n^{1/2}\ln 2} = 0$$

donc $f(n) \in O(g(n))$ mais n'est pas $\Theta(g(n))$

4. (5 points)

Soit $f(n) = 2^{n+\log n}$ et $g(n) = 3^n$, où le logarithme est en base 2.

En utilisant la définition formelle du « grand-O », démontrez que $f(n) \in O(g(n))$.

$f(n) = 2^{n+\log n} = 2^n * 2^{\log n} = n * 2^n$

mais, $n < (1.5)^n$ pour tout $n >= 1$

alors, $f(n) < (1.5)^n * 2^n = (1.5*2)^n = 3^n = g(n)$

donc pour tout n>1, et pour c = 1, $f(n) <= c*g(n)$

5. (5 points)

Décrivez, en pseudo code, un algorithme d'ordre $\Theta(\log(n))$ qui utilise l'approche « diviser pour régner » afin de calculer la $n^{ième}$ puissance d'une valeur quelconque x (i.e. x^n). Vous pouvez supposer que n est une puissance de 2 (i.e. $n = 2^k$). Démontrez clairement que votre algorithme est $\Theta(\log(n))$.

```
puissance(float x, int n) {

        if (n == 1)
                return x;
        else
                return puissance(x*x, n/2);
}
```

pour cet algorithme :

$T(1) = 0$
$T(n) = T(n/2) + 1$

Substitution :

$T(n) = T(n/2) + 1 = T(n/4) + 1 + 1 = T(n/8) + 1 + 1 + 1 = \log_2 n * 1 = \log_2 n \in \Theta(\log(n))$