Université Mohamed Khider Biskra **Date : 28/05/2023**
FSE&SNV
Département d'informatique **Nom &prénom : ……..**
Module : CO

Niveau : Master 1 **Examen** **Note: ……..**

Questions. (4 points)

Q1. Quelle est la différence entre la complexité des algorithmes itératifs et les algorithmes récursifs ? Comment sont-elles exprimées?

……

……..……

……

……..……

……

……..

Q2. Comment optimiser la complexité d'un algorithme ?

……

……..……

……

……..……

……

……..

……

Exercice 1. (08pts)

A. Étant donné que $T(n) = 27T([n/3]) + 3n^3 + \log n$

Trouvez le comportement asymptotique de T(n). Justifiez votre réponse.

……

……..

……

B. Considérons maintenant un algorithme dont le nombre d'opérations sur une entrée de taille n est donné par la relation de récurrence :

1. Donner la complexité de T(n)? $T(1)=0$ et $T(n)=T(\frac{n}{3})+ T(\frac{2n}{3})+n$

……..

……

……..

2. Cette récurrence peut-elle etre résolue en utilisant le théorème maitre ?

……..

C. Pour chacune des affirmations suivantes, dire si elle est vraie ou fausse. Rappeler les définitions utilisées et justifier vos réponses.

1. Est-ce que $\sum_{i=1}^{n} i \quad = \Theta(n^2)$?

……..

……

……..

2. Est-ce que $n^2 = \Omega(2^n)$?

……..

……

……..

……..

……

……..

……..

3. Est-ce que n =Ω(n logn)?

……..

……..

……

4. Soit un algorithme A. Est-il correct de dire du temps d'exécution de A que : si le pire cas est en O(f(n)) et le meilleur cas en Ω(f(n)) alors en moyenne A est en Θ(f(n))?

……..

……..

……

<u>Exercice</u> 3. (08pts)

Soient deux entiers x et y codés sur n bits (on supposera que n est une puissance de 2). On souhaite multiplier ces deux entiers entre eux, en travaillant au niveau des bits. La multiplication de deux entiers de n bits se fait de manière triviale en $O(n^2)$, la multiplication d'un entier par une puissance de 2, et l'addition de 2 entiers se font en temps linéaire O(n).

1. La méthode diviser pour régner n'est pas toujours meilleure qu'un algorithme naïf. Pour illustrer cela, donnez un algorithme diviser pour régner ayant une complexité en $O(n^2)$ pour multiplier deux entiers x et y de n bits chacun.

2. Proposez un algorithme diviser pour régner ayant une complexité inférieure à $O(n^2)$. Donnez la formule de récurrence pour votre algorithme et sa complexité finale (en O). On supposera pour simplifier que l'addition/multiplication de deux nombres de taille n est un nombre de taille n.

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$
$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R$$

Fig. Codage de deux entiers x et y

Bon courage

Corrigé type

Questions. (04pts)

voir le cours

Correction 1.

A. On utilise le formulaire des solutions d'équations de récurrence (cours):

$T(n) = c.T(n/d) + O(n^k)$ → si $c = d^k$, alors $T(n) = O(n^k.\log n)$

→ si $c < d^k$, alors $T(n) = O(n^k)$

→ si $c > d^k$, alors $T(n) = O(n^{\log \text{ base d de c}})$

On applique le Master theorem avec a = 27, b = 3, $f(n) = 3n^3 + \log n$.

L'exposant est $k = \log_b a = \log_3 27 = 3$, et puisque $f(n) = \Theta(n^k)$ on a la perturbation moyenne. On conclut par Master theorem que : $T(n) = \Theta(n^k \log n) = \Theta(n^3 \log n)$.

B.

1. On déduit une complexité O(n log n).
2. Le théorème maitre ne permet pas de résoudre cette récurrence.

C.

1. **Oui**. on a $\sum_{i=1}^{n} i$ =(n-1+1) (n+1)/2=n(n+1)/2

$\exists c > 0, \quad \exists n0 \in N, \quad \forall n \geq n0, \frac{n(n+1)}{2} \geq cn^2$

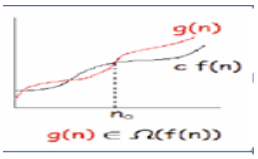
$n \geq 0, \frac{n}{2} \geq 0$ donc $\frac{1}{2}n^2 \leq \frac{n^2+n}{2} = \frac{n(n+1)}{2}$ $c = \frac{1}{2}$ $n0 = 0$

2. **Non**. selon la définition vue dans le cours

Notation Ω

- Quand la complexité g(n) de l'algorithme est **minorée** par f(n), on dit qu'il est en **Ω(f(n))**.

$\Omega(f(n)) = \{ g, \exists n0, \exists c>0, \forall n>n0, g(n) \geq c.f(n) \}$



$n^2 = \Omega(2^n)$ donc $\exists$ *c et un entier* $n0$:

$\forall n \geq n0, \quad n^2 \geq c.2^n$

$\forall n \geq n0, \quad \frac{2^n}{n^2} \leq \frac{1}{c}$

mais

$\lim_{n\to\infty} \left(\frac{f(x)}{g(x)}\right) = \lim_{n\to\infty} \left(\frac{2^n}{n^2}\right) = \infty$, donc on ne peut pas borner par une valeur.

3. **Non**,

par définition $n = \Omega(n \log n)$ ssi. (meme définition precedente de minoré de cours)

$\exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 < cn \log n \leq n.$

2. Réponse non. Par définition, on a $n = \Omega(n \log n)$ si et seulement si :

$$\exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 0 < cn \log n \leq n.$$

Supposons l'existence d'un tel c et d'un tel n_0. Alors pour n'importe quel $n \geq n_0$ on doit avoir $\log n \geq \frac{1}{c}$. Ceci est en contradiction avec le fait que $\lim_{+\infty} \log n = +\infty$.

---

4. **Oui, voir le cours**

4. Réponse Oui. Le meilleur cas minore la moyenne donc un minorant asymptotique du meilleur cas est également un minorant asymptotique de la moyenne. De même le pire cas majore la moyenne donc un majorant asymptotique de la moyenne est également un majorant asymptotique de la moyenne. Ainsi la moyenne est en $\Omega(f(n))$ et en $\Gamma(f(n))$ c'est à dire bien en $\Theta(f(n))$.

Correction 2.

1. On peut écrire x et y de la façon suivante :

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$

$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R$$

On peut alors multiplier x et y de la façon suivante :

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

On a donc besoin de 4 appels récursifs pour multiplier des entiers de taille n/2 pour x_L y_L , x_L y_R , x_R y_L et x_R y_R . Il faut ensuite faire les multiplications par $2^{n/2}$ et les additions, cela peut être fait en O(n). On a donc la formule de récurrence suivante :

$$T(n)=4T(n/2)+O(n)$$

Ce qui nous donne une complexité totale en $O(n^2)$ par le master theorem.

2. En réorganisant un peu l'expression précédente, on peut améliorer l'algorithme pour avoir une complexité en $O(n^{log_2^3})$. Seulement trois multiplications sont suffisantes : x_L y_L , x_R y_R et $(x_L + x_R)(y_L + y_R)$, puisque:

$$(x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R = x_L y_R + x_R y_L.$$

On a donc l'algorithme suivant :

Algorithm 1: multiplier(x,y)

début
  Data: 2 entiers x et y de n bits
  si $n = 1$ **alors**
    retourner xy
  $p_1 = multiplier(x_L, y_L)$;
  $p_2 = multiplier(x_R, y_R)$;
  $p_3 = multiplier(x_L + y_R, y_L + y_R)$;
  retourner $p_1 \times 2^n + (p_3 - p_1 - p2) \times 2^{n/2} + p_2$

On a donc la formule de récurrence suivante : $T(n)=3T(n/2)+O(n)$

Ce qui nous donne $O(n^{log_2^3}) \approx O(n^{1.59})$.