POLYTECHNIQUE
MONTRÉAL

UNIVERSITÉ
D'INGÉNIERIE

LOG1810

STRUCTURES DISCRÈTES

TD 6 : ALGORITHMES ET ANALYSE DE COMPLEXITÉ

H2025

SOLUTIONNAIRE

Exercice 1 :

Partie A

Dites si les affirmations suivantes sont vraies ou fausses. Justifiez vos réponses.

a) $2n^2 + 3n + 2 \in \Theta(n^2)$

Solution : **Vrai**
Effectivement, il est possible de trouver des constantes C_1, C_2 et k tel que $C_1 n^2 \leq 2n^2 + 3n + 2 \leq C_2 n^2$ pour $n > k$.

b) $nlog(n) + 4$ est $O(n^2)$

Solution : **Vrai**
Effectivement il est possible de trouver des constantes C et k tel que $nlog(n) + 4 < Cn^2$ pour $n > k$.

c) $g(n) \in \Theta(n) \rightarrow g(n) \in O(n^2)$

Solution : **Vrai**
Si $g(n) \in \Theta(n)$ alors on sait que n est une borne supérieure de $g(n)$, alors forcément n^2 est aussi une borne supérieure de $g(n)$

d) $f(n) \in \Theta(n^2) \rightarrow f(n) \in o(n^2)$

Solution : **Faux**
n^2 est une borne supérieure et **inférieure** de $f(n)$, $f(n)$ n'est donc pas $o(n^2)$.

e) $h(n) \in O(n!) \rightarrow h(n) \in O(2^n)$

Solution : **Faux**
n! croit plus vite que 2^n, nous n'avons donc pas la garantie que si $h(n)$ est $O(n!)$, $h(n)$ sera $O(2^n)$. Contre-exemple : $h(n) = n!$.

f) $z(n) \in \Omega(n^n) \rightarrow z(n) \in \Omega(n!)$

Solution : **Vrai**
n^n croit plus vite que $n!$, ainsi, si n^n est une borne inférieur, alors $n!$ st aussi une borne inférieur.

Partie B

Donnez une évaluation du comportement asymptotique de chacune des fonctions suivantes, en utilisant le grand-O. Utilisez les propriétés de combinaison de fonctions grand-O. Justifiez vos réponses.

a) $[n^3 + 2n + \log(\mathrm{n})]$

Solution :
$[n^3 + 2n]$ est $O(n^3)$
$[log(n)]$ est $O(log(n))$
$[n^3 + 2n + \log(\mathrm{n})]$ est donc $O\left(max\left(n^3, log(n)\right)\right) = O(n^3)$

b) $\left[log(n) + \sqrt{n}\right]\left[\sqrt{n} + n^{1/3}\right]$

Solution :
$\left[log(n) + \sqrt{n}\right]$ est $O\left(max(log(n), \sqrt{n})\right) = O\left(\sqrt{n}\right)$
$\left[\sqrt{n} + n^{1/3}\right]$ est $O\left(max\left(\sqrt{n}, n^{1/3}\right)\right) = O\left(\sqrt{n}\right)$
$\left[log(n) + \sqrt{n}\right]\left[\sqrt{n} + n^{1/3}\right]$ est donc $O\left(\sqrt{n}\sqrt{n}\right) = O(n)$

c) $[12log(n^n) + n + 3][15\mathrm{n}! + 13^{\mathrm{n}}]$

Solution :
$log(n^n) = nlog(n)$
$[12log(n^n) + n + 3]$ est $O(max(nlog(n), \mathrm{n}) = O\left(\mathrm{nlog(n)}\right)$
$[15\mathrm{n}! + 13^{\mathrm{n}}]$ est $O\left(max(\mathrm{n}!, 13^n)\right) = O(\mathrm{n}!)$
$[12log(n^n) + n + 3][15\mathrm{n}! + 13^{\mathrm{n}}]$ est donc $O\left(n!\, nlog(n)\right)$

d) $[n + 5]^{99}[log^2(n) + nlog(n)]$

Solution :
$[n + 5]$ est $O(\mathrm{n})$
Or, comme l'exponentiation est une répétition de multiplication :
$[n + 5]^{99}$ est $O((n)^{99}) \Rightarrow O(n^{99})$
$[log^2(n) + nlog(n)]$ est $O\left(nlog(n)\right)$
Ainsi, $[n + 5]^{99}[log^2(n) + nlog(n)]$ est $O\left(n^{100}log(n)\right)$

Exercice 2 :

a) Montrez que $3x^2 + 5\log(x) + 7$ est $\Theta(x^2)$.

Solution :

Nous avons les deux fonctions suivantes :

- $f(x) = 3x^2 + 5\log(x) + 7$
- $g(x) = x^2$

Montrons dans un premier temps que $3x^2 + 5\log(x) + 7$ est $O(x^2)$, i.e. trouvons deux constantes tel que :

$$|f(x)| \leq C_1|g(x)| x > k_1$$

Nous savons d'abord que $log(x) \leq x^2 \; pour \; x > 2$.

Ainsi, nous avons :

$$3x^2 + 5\log(x) + 7 \leq 3x^2 + 5x^2 + 7x^2 \quad \text{pour x>2}$$
$$3x^2 + 5\log(x) + 7 \leq 15x^2 \quad \text{pour x>2}$$
$$|3x^2 + 5\log(x) + 7| \leq 15|x^2| \quad \text{pour x>2}$$

Ainsi, nous trouvons $C_1 = 15$ et $k_1 = 2$. Nous avons donc montré que $3x^2 + 5\log(x) + 7$ est $O(x^2)$.

Montrons ensuite que $3x^2 + 5\log(x) + 7$ est $\Omega(x^2)$, i.e. trouvons deux constantes tel que :

$$C_2|g(x)| \leq |f(x)| \qquad x > k_2$$

Ainsi, nous avons :

$$x^2 \leq 3x^2 \quad \text{pour x>0}$$
$$x^2 \leq 3x^2 + 5\log(x) + 7 \quad \text{car } 5log(x) + 7 > 0 \text{ pour x>0}$$
$$|x^2| \leq |3x^2 + 5\log(x) + 7| \quad \text{pour } x > 0$$

Ainsi, nous trouvons $C_2 = 1$ et $k_2 = 0$. Nous avons donc montré que$3x^2 + 5\log(x) + 7$ est $\Omega(x^2)$. Ainsi, comme nous avons montré que $3x^2 + 5\log(x) + 7$ est $O(x^2)$ et que $3x^2 + 5\log(x) + 7$ est $\Omega(x^2)$, alors $3x^2 + 5\log(x) + 7$ est $\Theta(x^2)$

CQFD

b) Montrez que $\frac{6n^3+n^2+\sqrt{n}}{5n}$ n'est pas $O(n)$.

Solution :

Raisonnons par l'absurde.
Supposons que $\frac{6n^3+n^2+\sqrt{n}}{5n}$ est $O(n)$ et montrons que nous arrivons à une contradiction.

Par définition, pour que $\frac{6n^3+n^2+\sqrt{n}}{5n}$ soit $O(n)$, il existe deux constantes C et k telles que pour $n > k$, $\frac{6n^3+n^2+\sqrt{n}}{5n} \leq Cn$. Nous avons donc :

$$\frac{6n^3 + n^2 + \sqrt{n}}{5n} \leq Cn$$

$$\Rightarrow 6n^3 + n^2 + \sqrt{n} \leq 5Cn^2$$

$$\Rightarrow 6n + 1 + \frac{1}{\sqrt{n}} \leq 5C$$

Or, peu importe la valeur de C, il n'est pas possible $6n + 1 + \frac{1}{\sqrt{n}} \leq 5C$ pour tout n > k, car n peut être arbitrairement grand. Ainsi, nous arrivons a une contradiction.

Par conséquent, notre supposition comme quoi $\frac{6n^3+n^2+\sqrt{n}}{5n}$ est $O(n)$ est fausse. $\frac{6n^3+n^2+\sqrt{n}}{5n}$ n'est donc pas $O(n)$.

CQFD

Exercice 3 :

Vous êtes à votre premier stage et votre superviseur vous confie la tâche suivante : analyser la performance de deux algorithmes pour déterminer lequel est le plus pertinent à utiliser dans un projet.

Algorithme 1 :

```
1.    for i:=1 to n
2.       for j:=1 to n
3.          call fcn1()
```

La fonction fcn1() effectue 1 opération élémentaire.

a) Donnez le nombre d'opérations effectués par cet algorithme et en déduire sa complexité temporelle. Justifiez vos réponses.

Solution :

Ligne 1 : 2 opérations (comparaison et incrément de i) *n* fois. Donc *2n* opérations.

Ligne 2 : 2 opérations (comparaison et incrément de j) $n\ x\ n$ fois. Donc $2n^2$ opérations.

Ligne 3 : 1 opération $n\ x\ n$ fois. Donc n^2 opérations.

Ainsi, en additionnant le tout, nous arrivons à $3n^2 + 2n$ opérations. L'algorithme a donc une complexité temporelle en $O(n^2)$.

Algorithme 2 :

```
1.  for i:= 1 to n
2.      call fcn2()
3.
4.  for i:=1 to n, i:=i×2
5.      for j:=1j  to n
6.          call fcn3()
```

L'algorithme prétraite d'abord les données avec la fonction fcn2() et effectue le travail principal dans la fonction fcn3(). La fonction fcn2() effectue 100 opérations élémentaires et fcn3() effectue 2 opérations élémentaires.

b) Donnez le nombre d'opérations effectués par cet algorithme et en déduire sa complexité temporelle. Justifiez vos réponses.

Solution :

Prétraitement :

Ligne 1 : 2 opérations (comparaison et incrément de i) *n* fois. Donc *2n* opérations.

Ligne 2 : 100 opérations n fois. Donc $100n$ opérations.

Ainsi, en additionnant le tout, nous arrivons à $102n$ opérations pour le prétraitement.

Bloc principal :

Analysons le comportement de la boucle à la ligne 4. i prendra les valeurs suivantes à chaque itérations :

- Itération 1 : 1
- Itération 2 : 2
- Itération 3 : 4
- …
- Itération k : 2^k

La boucle s'arrête lorsque $i > n$ soit $2^k > n \Rightarrow log_2(2^k) = log_2(n) \Rightarrow k > log_2(n)$ Ainsi, la boucle va s'exécuter :

$$k = \lfloor log_2(n) \rfloor + 1$$

Ligne 4 : 2 opérations (comparaison et incrément de i) *k* fois. Donc *2k* opérations.
Ligne 5 : 2 opérations (comparaison et incrément de j) $n\ x\ k$ fois. Donc $2nk$ opérations.
Ligne 6 : 2 opérations $n\ x\ k$ fois. Donc 2nk opérations.

Ainsi, en additionnant le tout, nous arrivons à 2k+4nk opérations.

Algorithme complet

Ainsi, nous avons au total :

$$2k + 4nk + 102n = 4n\lfloor log_2(n)\rfloor + 106n + 2\lfloor log_2(n)\rfloor + 2$$

On en déduit que l'algorithme a une complexité temporelle en $O\big(nlog(n)\big)$.

c) D'après votre analyse, quel algorithme semble le plus performant ? Justifiez votre réponse en comparant les complexités temporelles de chaque algorithme.

<u>Solution</u>

L'algorithme 2 est plus performant car sa complexité $O\big(nlog(n)\big)$ croît moins vite que celle de l'algorithme 1, qui est en $O(n^2)$. Pour de grandes valeurs de n, l'algorithme 2 effectuera moins d'opérations que l'algorithme 1, ce qui le rend plus efficace.

d) Après analyse, votre superviseur vous informe que l'algorithme sera exécuté sur des ensembles de seulement 30 éléments. Sachant cela, votre réponse change-t-elle ? Pourquoi ?

<u>Solution</u>

Comparons le nombre d'opérations effectués par chaque algorithme en sachant que $n = 30$

Algorithme 1 :

$$3(30)^2 + 2(30) = 2760 \text{ opérations}$$

Algorithme 2 :

$$4(30)\lfloor log_2(30)\rfloor + 106(30) + 2\lfloor log_2(30)\rfloor + 2 = 3670 \text{ opérations}$$

On remarque que, bien que la complexité temporelle du deuxième algorithme soit meilleure que celle du premier, le prétraitement introduit un coût initial important. Pour de petites valeurs de n, ce surcoût n'est pas compensé par la plus faible complexité de la boucle principale. Ainsi, dans le cas de $n = 30$, l'algorithme 1 est plus performant.

Cependant, plus n grandit, plus l'algorithme 2 devient avantageux, car sa croissance en $O\big(nlog(n)\big)$ reste inférieure à celle de $O(n^2)$ à mesure que n grandit. Par exemple, à $n = 50$ nous avons :

Algorithme 1 :

$$3(50)^2 + 2(50) = 7600 \text{ opérations}$$

Algorithme 2 :

$$4(50)\lfloor log_2(50)\rfloor + 106(50) + 2\lfloor log_2(50)\rfloor + 2 = 6312 \text{ opérations}$$

Exercice 4

Dans cette question, nous allons analyser un algorithme de tri appelé tri-fusion (Merge Sort), qui repose sur le paradigme diviser pour régner. Sans entrer dans les détails de son implémentation, l'algorithme fonctionne de la façon suivante :

1. Diviser :
   - On coupe le tableau en deux parties égales.
   - Ce processus est répété récursivement jusqu'à obtenir des sous-tableaux de taille 1 (qui sont par définition triés).

2. Fusionner :
   - On fusionne les sous-tableaux triés de manière ordonnée, en comparant les éléments un par un.
   - On répète cette fusion jusqu'à reconstituer le tableau d'origine, mais cette fois totalement trié.

a) Établissez la relation de récurrence du temps d'exécution de l'algorithme tri-fusion. Notez $T(n)$ le temps d'exécution de tri-fusion sur un tableau de taille n. Indice :

   - On divise le tableau en deux sous-tableau de taille $n/2$ sur lesquels on exécutera récursivement l'algorithme tri-fusion.
   - On ajoute la phase de fusion qui parcours tous les éléments du tableau, donc qui est de complexité $O(n)$.

   Solution

   - On subdivise le tableau en deux sous tableau de taille $n/2$: $2T(n/2)$.

   - La fusion est $O(n)$.

   Nous avons donc :

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

b) Développez l'équation de récurrence après k itérations.

   Solution

   - Itération 0 : $T(n) = 2T(n/2) + O(n)$
   - Itération 1 : $T(n) = 2[2T(n/4) + O(n/2)] + O(n) = 2(2T(n/4)) + O(n) + O(n)$
   - Itération 2 : $T(n) = 2\left(2\left(2T(n/8) + O(n/4)\right)\right) + O(n) + O(n) = 2\left(2\left(2T(n/8)\right)\right) + O(n) + O(n) + O(n)$
   - ...
   - Itération k : $T(n) = 2^k T(n/2^k) + kO(n)$

c) À partir de l'équation de récurrence développée, déterminez la complexité temporelle de l'algorithme tri-fusion. *Indice : T(1) est une constante.*

Solution

Nous savons que la récursion s'arrête lorsque la taille des sous-tableaux arrive à une taille de 1, donc quand $n/2^k = 1$ Nous pouvons donc déterminer le nombre d'itérations en fonction de la taille du tableau initial n :

$$n/2^k = 1$$
$$\Rightarrow k = log_2(n)$$

De plus, avec $T(1) = O(1)$, nous avons :

$$T(n) = 2^{log_2(n)}T(1) + log_2(n)O(n)$$
$$\Rightarrow T(n) = nO(1) + log_2(n)O(n)$$
$$\Rightarrow T(n) \in O\big(nlog(n)\big)$$

Nous avons ainsi déterminé la complexité temporelle de l'algorithme tri-fusion.