Série 1 : Outils mathématiques, comparaisons, ordre de complexité

Exercice 1 : *Exercices sur les preuves par récurrences*

a) Montrer que

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Solution1 :

Preuve par récurrence :

n=1 $\sum_{i=1}^{1} i = 1$ *et* $\frac{n(n+1)}{2} = \frac{1(1+1)}{2} = 1$ *donc vrai pour n=1*

On suppose qu'elle est vraie pour n et on montre qu'elle est vraie pour n+1

On montre que : $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$

$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n(n+1)}{2} + \frac{2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$ Vraie c.q.f.d

Solution2 :

$$S = \sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n$$

1xS=1+ 2 + 3 + … +(n-2)+(n-1)+n
1xS=n+(n-1)+(n-2)+ … + 3 + 2 + 1

$$2\text{x}S = \underbrace{(n+1)+(n+1)+(n+1)+\ldots+(n+1)+(n+1)+(n+1)}_{n \text{ fois } (n+1)}$$

$\Rightarrow$ S= $\frac{n(n+1)}{2}$

b) Montrer que

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Solution :

$$S = \sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + (n-1)^2 + n^2$$

Démonstration par récurrence :

i=1 → S= $1^2 = \frac{1(1+1)(2x1+1)}{6} = \frac{6}{6} = 1$ est vraie

on suppose que $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ est vraie

On démontre que

$$\sum_{i=1}^{n+1} i^2 = \frac{(n+1)((n+1)+1)(2(n+1)+1))}{6} = \frac{(n+1)(n+2)(2n+3)}{6}$$

$$S = \sum_{i=1}^{n+1} i^2 = 1^2 + 2^2 + \cdots + (n-1)^2 + n^2 + (n+1)^2$$

$$= \frac{n(n+1)(2n+1)}{6} + (n+1)^2$$

$$= \frac{n(n+1)(2n+1)+6(n+1)^2}{6} = \frac{(n+1)(2n^2+7n+6)}{6}$$

$$= \frac{(n+1)(n(2n+1)+6(n+1))}{6}$$

$= \frac{(n+1)(n+2)(2n+3)}{6}$ Donc par récurrence la propriété est vraie $\forall n$

c) Montrer par récurrence (et non en recherchant des expressions séparées pour les deux sommes) que :

$$\underbrace{\sum_{i=1}^{n} i^3}_{=S1} = \underbrace{\left(\sum_{i=1}^{n} i\right)^2}_{=S2}$$

Solution :

Démonstration par récurrence :

i=1 → S1(1)=S2(1)=1 est vraie

On suppose que S1(n)=S2(n), et on démontre que S1(n+1)=S2(n+1)

On montre que : $\sum_{i=1}^{n+1} i^3 = \left(\sum_{i=1}^{n+1} i\right)^2 = \frac{n^2+(n+1)^2}{4}$

$$\sum_{i=1}^{n+1} i^3 = \sum_{i=1}^{n} i^3 + (n+1)^3$$

$$\left(\sum_{i=1}^{n+1} i\right)^2 = (\sum_{i=1}^{n} i)^2 + (n+1)^2 + 2(n+1)\sum_{i=1}^{n} i$$

$$= (\sum_{i=1}^{n} i)^2 + (n+1)^2 + 2(n+1)\frac{n(n+1)}{2}$$

$$= (\sum_{i=1}^{n} i)^2 + (n+1)^2 + n(n+1)^2$$

$$= (\sum_{i=1}^{n} i)^2 + (n+1)^3$$

$\sum_{i=1}^{n} i^3 + (n+1)^3 = \sum_{i=1}^{n} i^2 + (n+1)^3$ Donc vrai par récurrence.

<u>Exercice 2 :</u> Comparaison de croissance

Soit un ordinateur pour lequel toute instruction possède une durée de 10^{-6} secondes. On exécute un algorithme qui utilise, pour une donnée de taille, *n*, *f(n)* instructions, *f(n)* étant *n*, n^2, n^3, *log n*, *n log n et* 2^n.

a) Remplir un tableau qui donne, en fonction de la taille *n=10, 20, 30* et *60*, et de la fonction f(n), la durée d'exécution de l'algorithme.
b) Reclassez le tableau en fonction de l'ordre croissant des *f(n).*

n / f(n)	10	20	30	60
log n	$\log 10*10^{-6}$ $=2,3*10^{-6}$	$\log 20*10^{-6}$ $=3,00*10^{-6}$	$\log 30*10^{-6}$ $=3,40*10^{-6}$	$\log 60*10^{-6}$ $=4,09*10^{-6}$
N	$10*10^{-6}$ $=10^{-5}$	$20*10^{-6}$ $=2*10^{-5}$	$30*10^{-6}$ $=3*10^{-5}$	$60*10^{-6}$ $=6*10^{-5}$
n log n	$10\log 10*10^{-6}$ $=2,3*10^{-5}$	$20\log 20*10^{-6}$ $=5,99*10^{-5}$	$30\log 30*10^{-6}$ $=1,02*10^{-4}$	$60\log 60*10^{-6}$ $=2,46*10^{-4}$
n^2	10^2*10^{-6} $=10^{-4}$	20^2*10^{-6} $=4*10^{-4}$	30^2*10^{-6} $=9*10^{-4}$	60^2*10^{-6} $=3,6*10^{-3}$
n^3	10^3*10^{-6} $=10^{-3}$	20^3*10^{-6} $=8*10^{-3}$	30^3*10^{-6} $=2,7*10^{-2}$	60^3*10^{-6} $=2,16*10^{-1}$
2^n	$2^{10}*10^{-6}$ $=1,02*10^{-3}$	$2^{20}*10^{-6}$ $=1,05$	$2^{10}*10^{-6}$ $=1,07*10^{3}$	$2^{10}*10^{-6}$ $=1,15*10^{12}$

Remarque :

- Le tableau ci-dessus est calculé en utilisant un logarithme népérien.
- Il est intéressant de remarquer que $1,15*10^{12}$ s ≈ 36 558 ans.

Exercice 3 : Classer dans l'ordre croissant les complexités suivantes :

$O(n^2)$, $O(3^n)$, $O(2^n)$, $O(n^2 \log n)$, $O(1)$, $O(n \log n)$, $O(n^3)$, $O(n!)$, $O(\log n)$, $O(n)$.

Solution :

Il est évident que $1 \leq n \leq n^2 \leq n^3 \quad \forall n \geq 1$

On a également $1 \leq \log n \quad \forall n \geq 3$ *(log1=0, log2=0,69 et log3=1,09)*

$$\log n \leq n \quad \forall n \geq 1$$

$$n \leq n\log n \leq n^2 \log n$$

donc : $1 \leq \log n \leq n \leq n\log n \leq n^2 \log n \leq n^2 \leq n^3$

on a : $n^3 \leq 2^n \Leftrightarrow \log n^3 \leq \log 2^n \Leftrightarrow 3\log n \leq n\log 2 \quad \forall n \geq 3$

donc : $n^3 \leq 2^n \quad \forall n \geq 10 \quad (\log 10 = 2,30)$

Il est évident que : $2^n \leq 3^n \; \forall n \geq 1$

Enfin $3^n \leq n! \; \forall n \geq 7$

Classement :

$O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n \log n) \subseteq O(n^2) \subseteq O(n^2 \log n) \subseteq O(n^3) \subseteq O(2^n) \subseteq O(3^n) \subseteq O(n!)$.

Exercice 4 :

Quelles sont les complexités de :

$T_1(n)= 3n\log n + \log n$

$T_2(n)= 2^n + n^3+25$

$T_3(n,k)=k+n$ où $k\leq n$

Classer les dans l'ordre croissant.

Solution :

$T_1(n)= O(nlogn)$

Selon la définition O on a :

$$0 \leq 3nlogn + logn \leq c.nlogn$$

$$0 \leq \frac{3nlogn + logn}{nlogn} \leq \frac{c.nlogn}{nlogn}$$

$$0 \leq 3 + \frac{1}{n} \leq c$$

$$\frac{1}{n} \leq 1\ \forall n \geq 1 \quad \Rightarrow c = 4\ et\ n_0 = 1$$

$T_2(n)= O(2^n)$

Selon la définition O on a :

$$0 \leq 2^n + n^3 + 25 \leq c.2^n$$

$$0 \leq \frac{2^n + n^3 + 25}{2^n} \leq \frac{c.2^n}{2^n}$$

$$0 \leq 1 + \frac{n^3}{2^n} + \frac{25}{2^n} \leq c$$

$$\frac{n^3}{2^n} \leq 1\ \forall n \geq 7\ et\ \frac{25}{2^n} \leq 1\ \forall n \geq 5 \quad \Rightarrow c = 3\ et\ n_0 = 7$$

$T_3(n,k)=O(n)$ selon la règle de la somme on a :

O(k)+O(n)=max(O(k),O(n))= O(n) puisque k≤n

<u>*Classement :*</u> T_3 , T_1 , T_2,

O(n) O(n log n) $O(2^n)$

Exercice 5 :

Supposons qu'on ait écrit une procédure pour additionner m matrices carrées de nxn. Si l'addition de deux matrices carrées nécessite un temps d'exécution de $O(n^2)$ quelle sera la complexité de cette procédure en fonction de m et n ?

Solution :

On a $a_1, a_2, \dots, a_m$ matrices , donc m-1 additions

$\Longrightarrow$ complexité $O(m-1) * O(n^2) \equiv O((m-1)*n^2) \equiv O(m.n^2)$

Exercice 6 :

Supposons que deux algorithmes résolvent le même problème l'un s'exécute en $T_1(n)=400\ n$ et l'autre en $T_2(n)=n^2$. Quelles sont les complexités de ces deux algorithmes ? Pour quelles valeurs de n doit-on préférer l'algorithme de complexité plus élevée ?

Solution :

$T_1(n) = O(n)$

$T_2(n) = O(n^2)$

Pour n=0 $T_1(n)=T_2(n)=O$

Pour n=1 $T_1(n)=400$, $T_2(n)=1$

Pour n=2 $T_1(n)=800$, $T_2(n)=4$

…

n=400 $T_1(n)=400^2$, $T_2(n)=400^2$

Pour n>400 $T_1(n) < T_2(n)$

Donc l'algorithme de complexité la plus élevée (T_2) n'est préférable que pour $0 \leq n < 400$

Remarque : Cet algorithme montre que quand la constante est trop importante elle peut influer. Donc pour tout n<400 L'algorithme T_2 est préférable.

Exercice 7 :

Soit *f(n)* et *g(n)* deux fonctions positives asymptotique. En s'aidant de la définition de base de la notation Θ, prouver que :

$$max(f(n),g(n))=\Theta(f(n)+g(n))$$

Solution

Tout d'abord, clarifions ce qu'est la fonction max (f (n), g (n)). Définissons la fonction h (n) = max (f (n), g (n)).

Alors

$$h(n) = \begin{cases} f(n) \text{ si } f(n) \geq g(n) \text{ ,} \\ g(n) \text{ si } f(n) < g(n) \text{ .} \end{cases}$$

Comme *f(n)* et *g(n)* sont asymptotiquement non négatif, il existe n_0 tel que $f(n) \geq 0$ *et* $g(n) \geq 0$ pour tout $n \geq n_0$.

Ainsi, pour $n \geq n_0$, $f(n) + g(n) \geq f(n) \geq 0$ et $f(n) + g(n) \geq g(n) \geq 0$.

Étant donné que pour tout n particulier, *h(n)* est soit *f(n)* ou *g(n)*, nous avons $f(n) + g(n) \geq h(n) \geq 0$, ce qui montre que $h(n) = max\ (f(n), g(n)) \leq c2\ (f(n) + g(n))$ pour tout $n \geq n_0$ (avec c2 = 1 dans la définition de Θ).

De même, puisque pour tout *n* particulier, *h (n)* est la plus grande de *f(n) et g(n)*, nous avons pour tout $n \geq n_0$, $0 \leq f\ (n) \leq h\ (n)$ et $0 \leq g\ (n) \leq h\ (n)$. L'ajout de ces deux inégalités donne $0 \leq f\ (n) + g\ (n) \leq 2h\ (n)$, ou de manière équivalente $0 \leq (f(n)+g(n))/2 \leq h(n)$, ce qui montre que $h\ (n) = max\ (f(n), g(n)) \geq c1\ (f(n) + g(n))$ pour tout $n \geq n_0$ (avec c1 = 1/2 dans la définition de Θ).

Exercice 8 :

Peut-on écrire : $2^{n+1}=O(2^n)$?

$2^{2n}=O(2^n)$?

Solution:

$2^{n+1} = O(2^n)$, mais $2^{2n} \neq O(2^n)$.

Pour montrer $2^{n+1} = O(2^n)$, nous devons trouver les constantes $c, n0 > 0$ telles que : $0 \le 2^{n+1} \le c \cdot 2^n$ *pour tout* $n \ge n_0$.

Sachant que $2^{n+1} = 2 \cdot 2^n$ *pour tout* n, la définition est satisfaite avec $c = 2$ et $n_0 = 1$.

Pour montrer $2^{2n} \neq O(2^n)$, On suppose qu'il existe les constantes $c, n_0 > 0$ telles que :

$0 \le 2^{2n} \le c \cdot 2^n$ *pour tout* $n \ge n_0$.

Alors $2^{2n} = 2^n \cdot 2^n \le c \cdot 2^n \Rightarrow 2^n \le c$. Mais il n'existe pas de constante supérieure à 2^n, l'hypothèse conduit donc à une contradiction

Exercice 9 :

Montrer que les affirmations suivantes sont correctes :

(a) $5n^2-6n=\Theta(n^2)$
(b) $n!=O(n^n)$
(c) $2n^2+n \log n=\Theta(n^2)$
(d) $\sum_{i=0}^{n} i^2 =\Theta(n^3)$
(e) $\sum_{i=0}^{n} i^3 =\Theta(n^4)$
(f) $n^{2^n} +6*2^n=\Theta(n^{2^n})$

Solution :

(a) $5n^2-6n=\Theta(n^2)$

$n^2 \leq 5n^2-6n \leq 5n^2 \quad \forall n \geq 2$

(b) $n\,!=O(n^n)$

$n.(n-1).\ \ldots\ .1 \leq n.\ n.\ \ldots\ .n \quad \forall n \geq 1$

(c) $2n^2+n \log n=\Theta(n^2)$

$C_1n^2 \leq 2n^2+n \log n \leq C_2n^2$

$$C_1 \leq 2 + \frac{n \log n}{n^2} \leq C_2$$

$\forall n \geq 1 \quad 0 \leq \frac{n \log n}{n^2} \leq 1$ avec $C_1=2$, $C_2=3$ et $n_0=1$

(d) $\sum_{i=0}^{n} i^2 = \Theta(n^3)$

$$\frac{n^3}{3} \leq \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \leq n^3 \quad \forall n \geq 1$$

(e) $\sum_{i=0}^{n} i^3 = \Theta(n^4)$

$$\frac{n^4}{4} \leq \sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 \leq n^4 \quad \forall n \geq 1$$

(f) $n^{2^n} + 6*2^n = \Theta(n^{2^n})$

$C_1n^{2^n} \leq n^{2^n} + 6*2^n \leq C_2n^{2^n}$

$$C_1 \leq 1 + \frac{6*2^n}{n^{2^n}} \leq C_2$$

$\forall n \geq 3 \quad 0 \leq \frac{6*2^n}{n^{2^n}} \leq 1$ avec $C_1=1$, $C_2=2$ et $n_0=3$