IFT2015 hiver 2010 — Notes de cours

Miklós Csűrös

13 janvier 2010

2 Notation asymptotique

2.1 Mesures de performance

Pour comprendre le comportement d'une solution informatique à un problème, on mesure la performance :

- ⋆ usage de mémoire («espace»)
- ⋆ vitesse d'exécution («temps»)
- ⋆ d'autres mesures possibles, selon l'application (p.e., quantité de données transmises sur un réseau, ou nombre de *cache miss*)

Le but peut être

- ⋆ comparer l'efficacité de différentes solutions au même problème
- ⋆ donner un pronostic des performances (comment la performance dépend de l'entrée et de l'environnement)
- ⋆ initialiser les valeurs des paramètres de la structure ou de l'algorithme (comment la performance dépend de la valeur des paramètres)

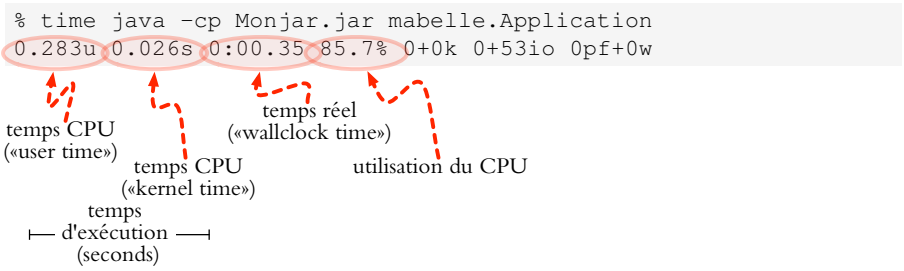
L'usage maximal de mémoire (*memory footprint*) peut être mesuré et/ou analysé mathématiquement.

Le temps d'exécution peut être mesuré :

- ⋆ par le programme (Java) :

```
...
long T0 = System.currentTimeMillis(); // temps de début
...
long dT = System.currentTimeMillis()-T0; // temps (ms) dépassé
```

- ⋆ par le shell (Linux/Unix)



2.2 Temps

Pour l'analyse mathématique, on doit définir la notion de «temps». Pour cela, on doit définir un **ensemble d'instructions** et les coûts associés à leur exécution. On parle des instructions élémentaires dont le coût est l'unité de temps.

Langage assembly. Instructions executées par le CPU directement (code machine).

```
[Java]
int Y;
int X = (Y+4)*3;

[Assembly]
mov eax, Y     ; affectation registre EAX
add eax, 4     ; EAX = EAX + 4
mov ebx, 3     ; EBX = 3
imul ebx       ; EAX = EAX*EBX
mov X, eax;
```

Chaque instruction prend un cycle (p.e., avec 3GHz, le code prend $5 \cdot \frac{1}{3 \cdot 10^9}\ s \approx 1.7\,\mu s$). L'ensemble d'instructions dépend du CPU...

CPU modèle. On peut définir un ensemble d'instructions de niveau basse sur un CPU hypothétique. Exemple : langages MIX et MMIX utilisé par Donald Knuth dans «The Art of Computer Programming» (instructions : transfer de données entre le mémoire et les registres du CPU, opérations arithmétiques, instructions de contrôle).

Modèle mathématique de calcul. Abstraction ultime : machine de Turing.

RAM modèle. Machine à accés aléatoire — «pseudocode»

- ⋆ programme de «lignes»
- ⋆ mémoire infini

⋆ nombre fini de variables
⋆ instructions : affectations, opérations arithmétiques, contrôle (`si-alors-sinon`, `tandis que`)

2.3 Croissance de fonctions

On s'intéresse au temps de calcul (ou autre mesure de performance) en fonction de l'entrée. Par exemple : temps d'exéuction sur une liste de taille n : $T(n) = 3n + 2$.

On peut choisir entre deux solutions avec $f(n)$ et $g(n)$ comme temps d'exécution : si $f(n) < g(n)$, alors utiliser la solution avec $f(n)$.

⋆ $f(n) = 5n + 2\,;\, g(n) = 3n + 14$
⋆ $f(n) = 3n^2 + 2n + 1\,;\, g(n) = 182n + 28$

Et si on exécute les deux solutions sur des machines différentes (une est c fois plus rapide que l'autre) : est-ce qu'il existe un $N_0 < \infty$ tel que $f(n) > c \cdot g(n)$ pour tout $n \geq N_0$?

S'il existe un tel N_0 pour tout c, g est décisivement meilleure asymptotiquement : on écrit $g(n) = o\big(f(n)\big)$ (voir définition plus tard).

2.4 Hiérarchie de croissances

Hiérarchie de quelques notions fréquentes :
– **constant**
– **Ackermann inverse** $\alpha(m, n)$
– **logarithme itéré** $\log^*$
– **logarithmique** $\log$
– **linéaire** N
– $N \log N$
– **quadratique** N^2
– **cubique** N^3
– **[polynomial]** N^d
– **exponentiel** x^N
– **superexponentiel** (p.e, factorielle $N!$)
– **non-calculable** (p.e., fonction de Radó)

2.5 Quelques fonctions notables

Logarithmes.

$$\lg x = \log_2 x \qquad \text{et} \qquad \ln x = \log_e x \quad \{x > 0\}$$
$$\log(xy) = \log x + \log y; \log(x^a) = a \log x$$
$$2^{\lg n} = n; n^n = 2^{n \lg n}; \log_a n = \frac{\lg n}{\lg a}; a^{\lg n} = n^{\lg a}$$

Factorielle. $n! = 1 \cdot 2 \cdots n$. Approximation de Stirling :

$$n! \approx \sqrt{2\pi n}\Big(\frac{n}{e}\Big)^n$$

On voit aussi que $\ln(n!) \approx n \ln n - n + \frac{1}{2} \ln n + \ln \sqrt{2\pi}$, essentiellement $n \ln n$

Nombre harmonique.

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n} \approx \ln n.$$

2.6 Notation aymptotioque – définitions

On travaille avec les définitions suivantes.

Définition 1. *Soit* $f\colon \mathbb{N} \mapsto \mathbb{R}^+$ *et* $g\colon \mathbb{N} \mapsto \mathbb{R}^+$ *deux functions positives*[1]. *On écrit*

$$f \in \Theta(g) \quad \textit{si } \exists c_1, c_2 > 0,\ \exists N \in \mathbb{N}\, \forall n \geq N \quad c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n); \tag{1a}$$

$$f \in O(g) \qquad \textit{si } \exists c > 0,\ \exists N \in \mathbb{N}\, \forall n \geq N \quad f(n) \leq c \cdot g(n); \tag{1b}$$

$$f \in \Omega(g) \qquad \textit{si } \exists c > 0,\ \exists N \in \mathbb{N}\, \forall n \geq N \quad f(n) \geq c \cdot g(n); \tag{1c}$$

$$f \in o(g) \qquad \textit{si } \forall c > 0,\ \exists N \in \mathbb{N}\, \forall n \geq N \quad f(n) \leq c \cdot g(n). \tag{1d}$$

REMARQUE. Le seuil N n'est pas nécessaire dans les définitions (mais peut simplifier les preuves) si $g(n) > 0$ pour tout $n \geq 0$. On peut imposer $N = 0$ toujours : par exemple si $f(n) \leq c \cdot g(n)$ pour tout $n \geq N$, alors $f(n) \leq c' \cdot g(n)$ pour tout $n \geq 0$ avec

$$c' = \max\Big\{c, \frac{f(0)}{g(0)}, \frac{f(1)}{g(1)}, \frac{f(2)}{g(2)}, \ldots, \frac{f(N-1)}{g(N-1)}\Big\}.$$

[1] En fait, on peut permettre que $f(n) \leq 0$ ou que $f(n)$ n'existe pas pour certains n si le nombre d'exceptions est fini. Même chose pour g. On travaillera avec des expressions comme $n^2 - 5n \lg n$ ou $n/\lg n$ qui ne sont pas toujours positives ou n'existent pas pour tout $n = 0, 1, 2, \ldots$. En permettant un nombre fini d'exceptions, on écrira sans problème que, disons, $T(n) \in O(\log \log n)$. Une définition toute précise parlerait de fonctions $f, g\colon \{m, m+1, m+2, \ldots\} \mapsto \mathbb{R}^+$ avec $m \geq 0$ et d'un seuil $N \geq m$.

REMARQUE. On peut aussi inventer des définitions équivalentes qui sont plus générales. P.e., $f \in O(g)$ si et seulement s'il existe $c > 0, a \in \mathbb{R}, N \geq 0$ tels que $f(n) \leq cg(n) + a$ pour tout $n \geq N$. Parfois il est plus facile de démontrer $f \in O(g)$ avec une telle définition.

Exemple 1. On montre que $12n^2 + 3n + 7 \in \Theta(n^2)$. Pour cela, il faut trouver c_1, c_2, N qui satisfont (1a). $c_1 = 12$ est un choix trivial. Pour c_2, on veut que

$$12n^2 + 3n + 7 \leq c_2 n^2.$$

Donc, $c_2 > 12$, et n'importe quel choix est possible. Disons, $c_2 = 14$. Il faut trouver N tel que $12n^2 + 3n + 7 \leq 14n^2$ pour tout $n \geq N$. Donc on a l'inégalité $0 \leq 2n^2 - 3n - 7$ qu'on peut résoudre explicitement : $n \geq \frac{1}{4}\left(3 + \sqrt{9 + 56}\right) \approx 2.8$. Alors, (1a) est vrai avec $c_1 = 12$, $c_2 = 14$, $N = 3$. □

Exemple 2. On montre que $3n^4 + 100n \lg n - 2\sqrt{n} \in O(n^4)$. On veut trouver c tel que

$$3n^4 + 100n \lg n - 2\sqrt{n} \leq cn^4.$$

On voit que $c > 3$ est requis, et en fait, tous les choix sont possibles. Par exemple, avec $c = 10$, on veut que $3n^4 + 100n \lg n - \sqrt{n} \leq 10n^4$. Cela mène à l'inégalité $0 \leq 7n^4 - 100n \lg n + \sqrt{n}$. Au lieu d'essayer de résoudre cette inégalité explicitement, on choisit un seuil N «très grand» et on vérifie si l'inégalité est vrai. Avec $n = 2^{10} = 1024$, on a $7n^4 - 100n \lg n + \sqrt{n} = 7 \cdot 1024^4 - 100 \cdot 1024 \cdot 10 + 32 > 0$ facilement. Alors, (1b) est vrai avec $c = 10$, $N = 1024$. □

Théorème 1. *Logarithme est une fonction de croissance beaucoup plus lente que n'importe quelle fonction polynomiale. Plus précisement, pour tout $\epsilon > 0$,*

$$\lg n \in o\big(n^\epsilon\big).$$

Démonstration. On veut démontrer que pour tout $c > 0$, il existe $N_c \geq 0$ tel que

$$\lg n \leq cn^\epsilon \tag{2}$$

pour tout $n \geq N_c$. On a que

$$\begin{aligned} n^\epsilon &= 2^{\epsilon \lg n} \\ &= (1 + x)^{\lg n} && \text{avec } x = 2^\epsilon - 1 > 0 \\ &\geq 1 + xa + x^2 \frac{a(a-1)}{2} && \text{si } a = \lg n \geq 2. \end{aligned} \tag{3}$$

On voit que

$$a \le c(1 + xa + \frac{a(a-1)}{2}x^2)$$
$$0 \le a^2\frac{cx^2}{2} - a\left(\frac{cx^2}{2} - cx + 1\right) + c$$

est valide pour tout

$$a \ge a_0(c, x)$$

où $a_0(c, x)$ est la solution la plus grande de l'équation $\alpha a^2 + \beta a + c = 0$ avec $\alpha = \frac{cx^2}{2}$ et $\beta = -\frac{cx^2}{2} + cx - 1$, ou $a_0(c, x) = 0$ si une telle solution n'existe pas. Selon (3), Equation (2) est valide si $n \ge N_c = \lceil 2^t \rceil$ avec $t = \max\{2, a_0(c, 2^\epsilon - 1)\}$. ■

2.7 Arithmétique avec grand-O

Proprement dit, O, Θ, o, etc. dénotent des ensembles de fonctions. Quand même, on les utilise souvent dans des expressions comme $f(n) = 2n + o(n)$. La règle d'interprétation dans un tel cas est la suivante.

- ⋆ À la droite d'une égalité, on veut dire qu'«il existe une fonction appartenant à l'ensemble avec laquelle l'équation est correcte».
- ⋆ À la gauche d'une égalité, on veut dire que «pour toute fonction appartenant à l'ensemble, l'équation est correcte».

Par exemple, si on dit que «le temps de calcul est $n + \Theta(\sqrt{n})$», il y a une égalité implicite ($T(n) = n + \Theta(\sqrt{n})$) : c'est une notation simple du fait que le temps de calcul est $T(n) = n + g(n)$ où $g(n) \in \Theta(\sqrt{n})$.

Exemple 3. L'égalité $n^{O(1)} = 2^{O(\log n)}$ veut dire que pour toute fonction $f(n) = n^{g(n)}$ avec $g(n) \in O(1)$, il existe $h(n) \in O(\log n)$ tel que $f(n) = 2^{h(n)}$. (Exercice : démontrer cette égalité.) □

Attention : selon les règles, la signe d'égalité n'est pas symmétrique" $n+O(\log n) = O(n^2)$ mais $O(n^2) \ne n + O(\log n)$.

Il n'est pas difficile de voir les règles d'arithmétique suivantes.

- ⋆ On a que $O(f_1) + O(f_2) = O(f_1 + f_2) = O\big(\max\{f_1, f_2\}\big)$. Dans d'autres mots, si $g_1 \in O(f_1)$ et $g_2 \in O(f_2)$, on a $g_1(n)+g_2(n) \in O\big(f_1(n)+f_2(n)\big)$, ou, d'une façon équivalente, $g_1(n) + g_2(n) \in O\big(\max\{f_1(n), f_2(n)\}\big)$.
- ⋆ On a que $O(f_1) \cdot O(f_2) = O(f_1 \cdot f_2)$. Dans d'autres mots, si $g_1 \in O(f_1)$ and $g_2 \in O(f_2)$, alors $g_1(n) \cdot g_2(n) \in O\big(f(n)\big)$ avec $f(n) = f_1(n) \cdot f_2(n)$.

2.8 Récurrences

Lorsqu'on a une fonction $f(n)$ qui est définie par récurrences, et on doit démontrer $f \in O(g)$ avec une fonction g quelconque, on utilise souvent des preuves par induction.

Théorème 2. *On considère les nombres Fibonacci*

$$F(n) = \begin{cases} 1 & \text{si } n = 0, 1, \\ F(n-1) + F(n-2) & \text{si } n > 1. \end{cases}$$

On a que $F(n) \in O(2^n)$.

Démonstration. On a besoin de c et de N qui satisfont (1b). Donc on montre que pour c et N *spécifiés plus tard*,

$$\forall n \geq N \colon F(n) \leq c2^n. \tag{4}$$

Cas de base. On vérifie que

$$\begin{aligned} F(0) &\leq c2^0 && \text{si } c \geq 1, \\ F(1) &\leq c2^1 && \text{si } c \geq \frac{1}{2}. \end{aligned}$$

Hypothèse d'induction. Supposons que pour tout $0 \leq k < n$,

$$F(k) \leq c2^k \tag{H}$$

Cas inductif. On a

$$\begin{aligned} F(n) &= F(n-1) + F(n-2) && \text{par définition} \\ &\leq c2^{n-1} + c2^{n-2} && \text{par (H)} \\ &= c2^n\Big(\frac{1}{2} + \frac{1}{4}\Big) \\ &< c2^n && \text{pour tout } c > 0. \end{aligned}$$

Donc, Eq. (4) est vrai avec $N = 0$ et $c = 1$. ■

Exemple 4. On considère la récurrence

$$T(n) = T(n-1) + O(1)$$

(recherche de maximum dans un tableau de taille n). On sait donc qu'il existe $c_T > 0$ tel que

$$T(n) \leq T(n-1) + c_T \qquad \text{si } n = 1, 2, \ldots. \tag{5}$$

Par substitutions itérées, on développe l'expansion

$$\begin{aligned} T(n) &\leq T(n-1) + c_T \\ &\quad \big(T(n-2) + c_T\big) + c_T \\ &\leq \ldots \\ &\leq T(n-k) + kc_T \\ &\leq \ldots \\ &\leq T(0) + n \cdot c_T. \end{aligned}$$

On va donc montrer que $T(n) = O(n)$. □

Théorème 3. *Si $T\colon \mathbb{N} \mapsto \mathbb{R}^+$ satisfait la récurrence*

$$T(n) = T(n-1) + O(1),$$

alors $T(n) \in O(n)$.

Démonstration. On montre par induction en n que pour $c > 0$ et $N \geq 0$ spécifiés plus tard,

$$\forall n \geq N\colon T(n) \leq cn. \tag{6}$$

Cas de base. Pour $n = 1$, on a $T(1) \leq c$ (première contrainte sur c).

Hypothèse d'induction. Supposons que pour un $n \geq 1$, on a

$$T(k) \leq ck \qquad \text{pour tout } 0 < k < n. \tag{H}$$

.

Cas inductif. Donc

$$\begin{aligned} T(n) &\leq T(n-1) + c_T && \text{par (5)} \\ &\leq c(n-1) + c_T && \text{par (H)} \\ &\leq cn && \text{si } c \geq c_T. \end{aligned}$$

On voit qu'avec $c = \max\{c_T, T(1)\}$ et $N = 1$, (6) est vrai. ■

2.9 Substitution de variables et fonctions lisses

Exemple 5. On considère la récurrence $T(n) = T(\lceil n/2 \rceil) + O(1)$, qui vaut pour le temps de calcul de recherche binaire. Si $n = 2^k$ avec $k = 0, 1, \ldots$, on a $T(2^k) = T(2^{k-1}) + O(1)$ pour tout $k \geq 1$. On définit la fonction $U(k) = T(2^k)$. Par Théorème 3, $U(k) = O(k)$ et donc $T(n) \leq c \lg n$ est valide avec un $c > 0$ quelconque pour $n = 1, 2, 4, 8, \ldots$. Est-ce qu'on peut conclure que $T(n) \in O(\log n)$? Oui, car $\log n$ est une fonction *lisse.* □

Définition 2. *Une fonction $f\colon \mathbb{N} \mapsto \mathbb{R}^+$ est* lisse *(«*smooth*»), si*

$$f(bn) \in O\big(f(n)\big)$$

por un entier $b \geq 2$.

REMARQUE. Les fonctions polynomiales et logarithmiques sont lisses. Par exemple, pour $f(n) = n^3$, on a $(2n)^3 = 8 \cdot n^3$ et donc $f(2n) \in O\big(f(n)\big)$. Par contre, les fonctions exponentielles ne sont pas lisses : si $f(n) = 3^n$, alors $f(2n) = 9^n$, ce qui ne peut être borné par $c \cdot 3^n$ avec aucune constante $c > 0$.

Théorème 4. *Soit $f, g\colon \mathbb{N} \mapsto \mathbb{R}^+$ tels que $f(b^k) \leq cg(b^k)$ pour tout $k = K, K+1, K+2, \ldots$ avec une constante $c > 0$, un seuil $K \geq 0$ et un entier $b \geq 2$. Si f, g sont des fonctions non-décroissantes et g est lisse, alors $f \in O(g)$.*

REMARQUE. Théorème 4 est utile dans le cas d'une récurrence avec des termes $T(\lceil n/b \rceil)$ ou $T(\lfloor n/b \rfloor)$. Si on est chanceux, on peut résoudre la récurrence pour $n = b^k$, soit directement, soit par substitution de variables. En arrivant au résultat que $T(b^k) \in O\big(f(2^k)\big)$, il faut vérifier si f est lisse, et conclure que $T(n) \in O\big(f(n)\big)$. Sans Théorème 4, la preuve peut devenir assez compliquée, comme celle du Théorème 5 ci-dessus.

Pour les tris efficaces comme Quicksort, le temps de calcul satisfait la récurrence

$$T(n) = T\big(\lceil n/2 \rceil\big) + T\big(\lfloor n/2 \rfloor\big) + O(n). \tag{7}$$

On peut essayer d'obtenir la solution rapidement avec «lissage» en examinant $n = 2^k$:

$$T'(k) = 2T'(k-1) + O(2^k).$$

La solution est $T'(k) = O(k2^k)$, d'où $T(n) = O(n \log n)$ (la fonction $g(n) = n \lg n$ est lisse).

Théorème 5. *Si $T(n)\colon \mathbb{N} \mapsto \mathbb{R}^+$ satisfait la récurrence* (7) *alors*

$$T(n) = O(n \log n).$$

Démonstration. Par (7), il existe $c_0 \in \mathbb{R}^+$ et $N_0 \in \mathbb{N}^+$ tels que

$$T(n) \leq T\big(\lceil n/2 \rceil\big) + T\big(\lfloor n/2 \rfloor\big) + c_0 n \tag{8}$$

pout tout $n \geq N_0$. On va démontrer que

$$T(n) \leq cn\lfloor \lg n \rfloor \tag{$\star$}$$

pour tout $n \geq N$, avec c et N à spécifier plus tard. En fait, on va démontrer par induction en k que

$$T(n) \leq cnk \tag{$\star\star$}$$

pour tout $2^k \leq n < 2^{k+1}$ quand $2^k \geq N$.

Cas de base. Soit $k_0 = \lfloor \lg N_0 \rfloor$. On a

$$T(n) \leq cnk_0$$

pour tout $2^{k_0} \leq n < 2^{k_0+1}$ si

$$c \geq \max\left\{ \frac{T(2^{k_0})}{k_0 2^{k_0}}, \frac{T(2^{k_0}+1)}{k_0(2^{k_0}+1)}, \frac{T(2^{k_0}+2)}{k_0(2^{k_0}+2)}, \ldots, \frac{T(2^{k_0+1}-1)}{k_0(2^{k_0+1}-1)} \right\}.$$

Hypothèse d'induction. Supposons que

$$T(n) \leq cnk \tag{H}$$

pour tout $2^k \leq n < 2^{k+1}$ avec un $k \geq k_0$.

Cas inductif. Soit n tel que $2^{k+1} \leq n < 2^{k+2}$. Alors,

$$\begin{aligned}
T(n) &\leq T\big(\lceil n/2 \rceil\big) + T\big(\lfloor n/2 \rfloor\big) + c_0 n && \text{par } (\star\star) \text{ si } 2^{k+1} \geq N_0\,; \\
&\leq ck\lceil n/2 \rceil + ck\lfloor n/2 \rfloor + c_0 n && \text{par (H)}\,; \\
&= ckn + c_0 n && \text{car } \lceil n/2 \rceil + \lfloor n/2 \rfloor = n\,; \\
&\leq c(k+1)n && \text{si } c \geq c_0.
\end{aligned}$$

En choisissant $c = \max\left\{ c_0, \max_{2^{k_0} \leq n < 2^{k_0+1}} \left\{ \frac{T(n)}{nk_0} \right\} \right\}$ et $N = 2^{k_0}$ avec $k_0 = \lfloor \lg N_0 \rfloor$, toutes les conditions de la preuve sont satisfaites et donc ($\star$) est vrai pour tout $n \geq N$. ■

2.10 Croissance asymptotique et lim

On peut évaluer la relation entre deux fonctions f, g rapidement en calculant $\lim_{n\to\infty} \frac{f(n)}{g(n)}$. Si la limite existe, alors $f = o(g)$ ou $f = \Theta(g)$, comme Théorème 6 le montre ici.

Théorème 6. *Soit $f, g\colon \mathbb{N} \mapsto \mathbb{R}^+$ deux fonctions.*

1. *$f \in o(g)$ si et seulement si la limite $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ existe et*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

2. *Si*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c,$$

avec une constante quelconque $0 < c < \infty$, alors $f \in \Theta(g)$.

REMARQUE. Notez que pour $o(\cdot)$, la limite donne une réponse définitive («si et selement si») mais on peut avoir $f = \Theta(g)$ sans que la limite existe. Par exemple,

$$f(n) = \begin{cases} n & \text{si } n \text{ est pair,} \\ 2n & \text{si } n \text{ est impair} \end{cases}$$

est clairement $\Theta(n)$ mais $\lim_{n\to\infty} f(n)/n$ n'existe pas.

Exemple 6. Une autre preuve pour Théorème 1 utilise la limite. On a

$$\begin{aligned} \lim_{n\to\infty} \frac{\lg n}{n^\epsilon} &= \lim_{n\to\infty} \frac{n^{-1}/\ln 2}{\epsilon n^{\epsilon-1}} && \text{par règle de L'Hospital} \\ &= \lim_{n\to\infty} \frac{1}{\epsilon(\ln 2) n^\epsilon} \\ &= 0 && \text{car } \epsilon > 0. \end{aligned}$$

□