

# Complexité des algorithmes

---

Semestre 3



# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Complexité . . . . .	4
1.2	Complexité asymptotique . . . . .	5
1.3	Exemple de complexités d’algorithmes . . . . .	7
1.4	Comportement symptotique de fonctions usuelles . . . . .	7
<b>2</b>	<b>Complexité des boucles</b>	<b>9</b>
2.1	Complexité de boucles “pour” . . . . .	9
2.2	Complexité de boucles “tant que” . . . . .	10
<b>3</b>	<b>Complexité d’algorithmes définis par réccurence</b>	<b>11</b>
<b>4</b>	<b>Structure de données et complexité</b>	<b>12</b>
<b>A</b>	<b>Exercices</b>	<b>13</b>
A.1	TD 1 . . . . .	13

# Introduction

## 1.1 Complexité

On cherche à estimer le temps de calcul d'un algorithme  $A$  en fonction d'un paramètre  $n$ . Pour avoir une mesure indépendante de la machine, on identifie le temps de calcul avec le nombre d'instructions exécutées.

**Ex** Le paramètre  $n$  pourrait être la taille d'un tableau, par exemple.

Soit  $D_i$  l'ensemble des données possibles telle que  $n = i$ . Pour  $d \in D_i$  on notera  $T(A, d)$  le nombre d'instructions exécutées pendant l'exécution de  $A(d)$ .

On notera  $\text{prob}(d|i)$  la probabilité que les données soit  $d$  étant donné qu'elles sont de taille  $i$ .

### 1.1.1 La complexité temporelle maximale

La complexité temporelle maximale<sup>1</sup> d'un algorithme  $A$  :

$$T_{\max}(i) = \max_{d \in D_i} \{T(A, d)\}$$

### 1.1.2 La complexité temporelle moyenne

La complexité temporelle moyenne<sup>2</sup> d'un algorithme  $A$  :

$$T_{\text{moy}} = \sum_{d \in D_i} \text{prob}(d|i) \times T(A, d)$$

**R** Pour pouvoir calculer  $T_{\text{moy}}$ , il faut connaître la distribution des données, ce qui n'est pas toujours évident (par exemple en traitement d'image)

---

1. Complexité dans le pire des cas  
2. Complexité dans le cas moyen

### 1.1.3 La complexité temporelle minimale

La complexité temporelle minimale<sup>3</sup> d'un algorithme A :

$$T_{\min}(i) = \min_{d \in D_i} \{T(A, d)\}$$

**R** Peu utilisé, sauf pour prouver qu'un algorithme est mauvais. Si la complexité temporelle minimale est mauvaise même dans le meilleur des cas, alors l'algorithme n'est pas bon.

### 1.1.4 Comparaison de complexités en fonction de la machine

Complexité	Nombre d'instructions pouvant exécuter la machine	
	1 000 000	1 000 000 000 000
$n$	1 000 000	1 000 000 000 000
$n \log_2 n$	64 000	32 000 000 000
$n^2$	1 000	1 000 000
$n^3$	100	10 000
$2^n$	20	40

## 1.2 Complexité asymptotique

Pour comparer des algorithmes, on ne s'intéresse qu'à leur comportements pour  $n$  grand. On cherche une mesure de complexité qui soit indépendante du langage de programmation et de la vitesse de la machine.

⇒ On ne doit pas perdre en compte des facteurs constants.

⇒ Ordre de grandeur

### 1.2.1 La complexité asymptotique

La complexité asymptotique<sup>4</sup> est l'ordre de grandeur de sa limite lorsque  $n \rightarrow \infty$

### 1.2.2 Notation

Soient  $T, f$  des fonctions positives ou nulles. Rotations de grandeur de fonction asymptotiques.

**Grand O**  $T = O(f)$  si  $\exists c \in \mathbb{R}^{>0}$  et  $n_0 \in \mathbb{N}$  tels que  $\forall n \geq n_0, T(n) \leq cf(n)$ .

3. Complexité dans le meilleur des cas

4. Que ce soit maximale, moyenne ou minimale

**Grand Oméga**  $T = \Omega(f)$  si  $\exists c \in \mathbb{R}^{>0}$  et  $n_0 \in \mathbb{N}$  tels que  $\forall n \geq n_0, T(n) \geq cf(n)$

**Petit O**  $T = o(f)$  si  $\frac{T(n)}{f(n)} \rightarrow 0$  lorsque  $n \rightarrow \infty$ .

**R** T est négligeable devant f

**Ex**

1.  $2n^2 + 5n + 10 = O(n^2)$   
 Dans la définition  $n_0 = 5, c = 4$  :  
 $\forall n \geq 5, 2n^2 + 5n + 10 \leq 4n^2$
2.  $2n^2 + 5n + 10 = \Omega(n^2)$   
 Dans la définition,  $n_0 = 1, c = 2$   
 $\forall n \geq 1, 2n^2 + 5n + 10 \geq 2n^2 \dots$   
 Donc  $2n^2 + 5n + 10 = \Theta(n^2)$
3.  $\frac{1}{5} + n = O(n \log_2 n)$  ( $n_0 = 2, c = 2$ )
4.  $\frac{1}{5}n \log_2 n + n = \Omega(n \log n)$  ( $n_0 = 1, c = \frac{1}{5}$ )
5.  $\forall k \geq 0, n^k = O(n^{k+1})$  mais  $n^k \neq \Omega(n^{k+1})$
6.  $\forall a, b > 1, \log_a n = \Theta(\log_b n)$  car  $\log_a n = \frac{\log_b n}{\log_b a}$  et  $\log_b a$  est une constante.  
 $\Rightarrow$  On a pas besoin de préciser la base de logarithme dans une complexité asymptotique
7.  $2n^2 + 5n + 10 = 2n^2 + o(n^2)$
8. Pour toute constante  $c > 0, C = \Theta(1)$
9.  $2^n = o(3^n)$

**R**

1. O et  $\Omega$  sont des pré-ordres<sup>a</sup> :  
 $f = O(f)$  et  $f = O(g)$  et  $g = O(h) \Rightarrow f = O(h)$
2.  $\Theta$  est une relation d'équivalence<sup>b</sup> :  $f = \Theta(g) \Leftrightarrow g = \Theta(f)$

a. Relations réflexives et transitives

b. relation réflexives, symétrique et transitive

**Proposition**

Si  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = a > 0$  Alors  $f = \Theta(g)$

**R** La réciproque est fausse

**Notation**

$$f \sim g \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$$

$$\text{Ex } (3n + 1)^3 \sim 27n^3$$

## 1.3 Exemple de complexités d'algorithmes

### 1.3.1 Le tri à bulles

$$\begin{aligned} T_{\min}(n) &= \Theta(n) \text{ Si le tableau est déjà trié} \\ T_{\max}(n) &= \Theta(n^2) \text{ Si le tableau est trié en ordre décroissant} \\ T_{\text{moy}}(n) &= T_{\max}(n) = \Theta(n^2) \end{aligned}$$

### 1.3.2 Tri par fusion

$$T_{\min}(n) = T_{\max}(n) = T_{\text{moy}}(n) = \Theta(n \log n)$$

### 1.3.3 Tri rapide

$$\begin{aligned} T_{\min}(n) = T_{\text{moy}}(n) &= \Theta(n \log n) \\ T_{\max}(n) &= \Theta(n^2) \end{aligned}$$

## 1.4 Comportement asymptotique de fonctions usuelles

Il y a quatre groupes importants de fonction positives croissantes.

**Logarithmiques**  $(\log n)^\sigma$  (où  $\sigma > 0$ ),  $\log \log n, \dots$

**Polynomiales**  $n^\gamma$  (où  $\gamma > 0$ ),  $n^\gamma (\log n)^\gamma$  (où  $\gamma > 0$ )

**Exponentielles**  $2^{\alpha n^\beta}$  (où  $\alpha > 0$  et  $0 < \beta \leq 1$ ), par exemple  $2^n, 4^n, 2^{\sqrt{n}}$

**Supraexponentielles**  $n!, n^n, 2^{n^2}, \dots$

**R** Il existe des fonctions intermédiaires (par exemple  $n^{\log_2 n}$ ) mais ces fonctions se rencontrent très rarement dans l'analyse de complexité d'algorithmes

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n^b}{a^n} &= 0 \text{ Pour toutes constantes } a, b \text{ avec } a > 1) \\ n^b &= o(a^n) \\ \lim_{n \rightarrow \infty} \frac{(\log_2 n)^\sigma}{n^\sigma} &= 0 \\ \Rightarrow (\log_2 n)^\sigma &= o(n^\sigma) \end{aligned}$$

### 1.4.1 La formule de Stirling

$$\begin{aligned} n! &\sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \\ \Rightarrow n! &= o(n^n) \text{ et } n! = \Omega(2^n) \end{aligned}$$

On peut aussi en déduire :

$$\log(n!) \sim n \log n$$



# Complexité des boucles

## 2.1 Complexité de boucles “pour”

```

1  pour i := 1 a n faire
2      -- Corps de la boucle
3  fin pour;
```

Notions  $I_i$  la  $i^{\text{e}}$  itération (les instructions exécutées lors du  $i^{\text{e}}$  passage dans la boucle) et  $T(I_i)$  sa complexité temporelle. :

Par exemple,  $T_{\text{moy}}(n) = T_{\text{max}}(n) = \Theta(n)$  si  $T(I_i)$  constant et  $= \Theta(n^2)$  si  $T(I_i) = an + b$  (boucle imbriquée).

### 2.1.1 Exemple

Calculer  $A = BC$ , le produit de 2 matrices. Rappel :

$$a_{ik} = \sum_{j=1}^n b_{ij} C_{ji}$$

```

1  pour i = 1 a n faire
2      pour k = 1 a n faire
3          aik  0
4          pour j = 1 a n faire
5              aik = aik + bij * cjk;
6          fin pour;
7      fin pour;
8  fin pour;
9  fin pour;
```

$$T_{\text{moy}}(n) = T_{\text{max}}(n) = \sum_{i=1}^n \sum_{k=1}^n (1 + n) = \Theta(n^3)$$

## 2.2 Complexité de boucles “tant que”

```
1  tantque C faire
2      -- Corps de la boucle
3  fin tantque;
```

$$T_{\text{moy}} = 1 + \sum_{i=1}^{\infty} \text{Prob}$$

On ajoute 1 pour le test de la condition C lorsque C = faux.

Soit  $E_i$  l'événement C = Vrai au début de  $i_i$

Si  $\forall i, j, E_i, E_j$  sont indépendantes et  $\text{prob}(E_i) = p < 1$ , où p est une constante, alors  
 $\text{prob}(\text{on exécute } I_i) = \text{prob}(E_1 \cdots E_i) = p^i$  d'où

$$T_{\text{moy}}(n) = 1 + \sum_{i=1}^{\infty} p^i * T(I_i)$$

Si  $T(I_i)$  est constante, alors

$$T_{\text{moy}}(n) = \Theta\left(1 + \frac{p}{1-p}\right) = \Theta\left(\frac{1}{1-p}\right) = \Theta(1)$$

### 2.2.1 Exemple

Comparaison de 2 suites  $\{A_i\}, \{b_i\}$ .

```
1  i := 1;
2  tantque (ai = bi et i <= n) faire
3      i := i + 1;
4  fin tantque;
```

$T_{\text{moy}}(n) = \Theta(1)$  si les suites sont indépendantes et aléatoires.

---

# Complexité d'algorithmes définis par récurrence

---

# Structure de données et complexité

CHAPITRE

4

# Exercices

## A.1 TD 1

### A.1.1 Lesquelles des affirmations suivantes sont vraies ?

1.  $n^2.5 = \Theta(n^3)$  : Faux
2.  $n^2.5 = O(n^3)$  : Vrai
3.  $n^2.5 = \Omega(n^3)$  : Faux
4.  $\log_2(2n) = \Theta(\log n)$  : Vrai
5. Vrai
6. Faux

### A.1.2 Une seule des affirmations suivantes est vraie. Laquelle ?

Réponse D

### A.1.3 Une seule des affirmations suivantes est vraie. Laquelle ?

Réponse C  $n + n \log_2 n \leq 2n \log_2 n = \Theta(n \log n)$

**R** On ne s'occupe pas des facteurs constants

### A.1.4

Réponse D

### A.1.5 Laquelle des affirmations suivantes sont vraies

1.  $\max(f(n), g(n)) = \Theta(f(n) + g(n))$  Vrai :  $\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \max(f(n), g(n))$
2. Vrai :  $\frac{1}{c}f(n) \leq g(n)$  et  $g(n) \leq \frac{1}{2}f(n)$

3. Vrai :  $\forall n \geq n_0 : f(n) \leq cg(n)$
4. Faux
5. Vrai
6. Faux

$$g(n) = 2n, f(n) = ng(n) = O(f(n))2^{g(n)} = 2^{2n} = (2^n)^2$$

### A.1.6 Lesquelles des affirmations suivantes sont vraies ?

Réponse D.

$$\begin{aligned} f(n) &\leq c_1 g(n) \quad , \quad g(n) \leq c_2 f(n) \\ \frac{1}{c_2} &\leq \frac{f(n)}{g(n)} \leq c_1 \cdot 1 \Rightarrow \frac{f(n)}{g(n)} = \Theta(1) \end{aligned}$$

### A.1.7 Simplifiez les expressions suivantes

1.  $O(4n^2 + 3n^2 + 7 \log_2(n^n)) = O(n^3)$
2.  $\Theta(n \log_2 n + 17n + 2n^3) = \Theta(n^3)$
3.  $\Omega(4n^2 + 3n^3) = \Omega(n^3)$
4.  $O(2^{n \log_3 n} + 3 \log_2 n!) = O(n^2)$
5.  $O(2 \log_3 n + 3 \log_2 n + 6) = O(\log n)$

### A.1.8 Classez les fonctions suivantes dans l'ordre croissant d'ordre de grandeur

1.  $4n \log_2 n + 4n$
2.  $2n \log_2 n + 4n$
3.  $n^2 \log_e n$