Université Toulouse III – Paul sabatier
L2 Informatique

# Complexité des algorithmes

Semestre 3

Toulouse, le 8 novembre 2012

# Table des matières


**1 Introduction** **4**

1.1 Complexité . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.2 Complexité asymptotique . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.3 Exemple de complexités d'algorithmes . . . . . . . . . . . . . . . . . . . . . . 7

1.4 Comportement symptotique de fonctions usuelles . . . . . . . . . . . . . . . . . 8

**2 Complexité des boucles** **9**

2.1 Complexité de boucles "pour" . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.2 Complexité de boucles "tant que" . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.3 Approximation asymptotique de sommes partielles . . . . . . . . . . . . . . . . 11

2.4 Analyse de cas particuliers de boucles . . . . . . . . . . . . . . . . . . . . . . 13

**3 Complexité d'algorithmes définis par réccurence** **13**

**4 Structure de données et complexité** **14**

**A Exercices** **15**

A.1 TD 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

CHAPITRE 1

# Introduction

## Sommaire


1.1 Complexité . . . 4
1.2 Complexité asymptotique . . . 5
1.3 Exemple de complexités d'algorithmes . . . 7
1.4 Comportement symptotique de fonctions usuelles . . . 8


## 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 exécutée 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[1] 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[2] d'un algorithme A :

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

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

**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)

### 1.1.3 La complexité temporelle minimale

La complexité temporelle minimale[3] 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 Compairaison de complexités en fonction de la machine

| **Complexité** | **Nombre d'instructions pouvant executer 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.
$\Rightarrow$ On ne doit pas perdre en compte des facteurs constants.
$\Rightarrow$ Ordre de grandeur

### 1.2.1 La complexité asymptotique

La complexité asymptotique[4] est l'ordre de grandeur de sa limite lorsque $n \to \infty$

### 1.2.2 Notation

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

3. Complexité dans le meilleur des cas
4. Que ce soit maximale, moyenne ou minimale

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

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

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

**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, $n_0 = 1$, $c = 2$
   $\forall n \geq 1,\ 2n^2 + 5n + 10 \geq 2n^2 \cdots$
   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 dnas une complexité asymptotique
7. $2n^2 + 5n + 10 = 2n^2 + 0(n^2)$
8. Pour toute constante $c > 0$, $C = \Theta(1)$
9. $2^n = o(3^n)$

**R**

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

a. Relations reflexives et transitives
b. relation reflexives, symétrique et transitive

**Proposition**

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

**R** La réciproque est fausse

**Notation**

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

**Ex** $(3n+1)^3 \sim 27n^3$

## 1.3 Exemple de complexités d'algorithmes

### 1.3.1 Le tri à bulles

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

### 1.3.2 Tri par fusion

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

### 1.3.3 Tri rapide

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

## 1.4 Comportement symptotique de fonctions usuelles

Il y a quatre groupes importants de fonction positives croissantes.

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

**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** $n!$, $n^n$, $2^{n^2}$, ...

**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

$$\lim_{n\to\infty} \frac{n^b}{a^n} = O \text{ Pour toutes constantes} a, b \text{ avec } a > 1)$$
$$n^b = o(a^n)$$
$$\lim_{n\to\infty} \frac{(\log_2 n)^\sigma}{n^\sigma} = O$$
$$\Rightarrow (\log_2 n)^\sigma = o(n^\sigma)$$

### 1.4.1 La formule de Stisling

$$n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$$
$$\Rightarrow n! = o(n^n) \text{ et } n! = \Omega(2^n)$$

On peut aussi en déduire :

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

 

Chapitre 2

# Complexité des boucles

## Sommaire


2.1 Complexité de boucles “pour” . . . . . . . . . . . . . . . . . . . . . . . 9
2.2 Complexité de boucles “tant que” . . . . . . . . . . . . . . . . . . . . . 11
2.3 Approximation asymptotique de sommes partielles . . . . . . . . . . . 11
2.4 Analyse de cas particuliers de boucles . . . . . . . . . . . . . . . . . . . 13


## 2.1 Complexité de boucles “pour”

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

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

Par exemple, $T_{\text{moy}}(n) = T\max(n) = \Theta(n)$ 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 matrics. Rappel :

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

```
pour i = 1 a n faire
  pour k = 1 a n faire
      aik   0
      pour j = 1 a n faire
        aik = aik + bij * cjk;
      fin pour;
    fin pour;
  fin pour;
fin pour;
```

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

## 2.2 Complexité de boucles “tant que”

```
tantque C faire
  -- Corps de la boucle
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

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

### 2.2.1 Exemple

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

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

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

## 2.3 Approximation asymptotique de sommes partielles

**Exemples de sommes partielles**

$$\sum_{i=1}^{n} \frac{1}{i} \quad \sum_{i=1}^{n} i^k \quad \sum_{i=1}^{n} \log_2 i$$

### 2.3.1 Principe de la méthode

Pour calculer une approximation asymptotique de $\sum_{i=1}^{n} f(i)$ où f est une fonction monotone on l'encadre par $\int f(n)du$.

**Proposition** Si $f$ est décroissante, alors

$$\int_{p}^{n+1} f(u)du \leq \sum_{i=p}^{n} f(i) \leq \int_{p-1}^{n} f(u)du$$

**Ex** f(u) = $\frac{1}{u}.H_n = \sum_{i=1}^{n} \frac{1}{i}$ est la série harmonique. On ne peut intégrer $\frac{1}{u}$ qu'à partir de 1 donc on choisit $p = 2$.

$$\int_{2}^{n+1} \frac{1}{u}du \leq H_n - 1 \leq \int_{1}^{n} \frac{1}{u}du$$

$$\begin{aligned}
[\log_e u]_2^{n+1} &\leq H_n - 1 \leq [\log_e u]_1^n \\
\log_e(n+1) - \log_e 2 &\leq H_n - 1 \leq \log_e n - \log_e 1 \\
\log_e n - \log_e 2 + 1 &< H_n \leq (\log_e n) + 1
\end{aligned}$$

Donc $H_n = \Theta(\log n)$

## 2.3.2 Application

Étude de complexité d'un algorithme de génération d'une permutation aléatoire des entiers $1, 2, \cdots, n$ dans un tableau `perm`

**R** Il existe un algorithme de complexité $\Theta(N)$ pour ce problème : pour chaque $i \in \{1, 2, \cdots, n\}$, échanger `perm`[i] et `perm`[`random(i)`].

```
pour i = 1 a n faire
  vu[i] = faux;
fin pour;
pour i = 1 a n faire
  x = random(n);
  tantque vu[x] faire
    x = random(n);
  fin tantque;
  perm[i] = x;
  vu[x] = vrai;
fin pour;
```

Listing 2.1 – Génération d'une permutation aléatoire

**R** $T_{\max} = \infty$ car il n'y a aucune garantie de terminaison. C'est un exemple d'algorithme de type *Las Vegas* la probabilité de non terminaison est nulle.

On suppose que la complexité de `perm`(n) est $\Theta(1)$.

Pour $i, n$ fixe, à chaque itération de la boucle "tantque", la probabilité de rentrer dans la boucle est une constante pour $p = \frac{i-1}{n}$ et $p < 1$ pour $1 \leq i \leq n$.

Par l'analyse de la complexité d'une boucle "tantque" (section 2.2), la complexité moyenne de la boucle "tantque" est $\Theta(\frac{1}{1-p}$ donc

$$\begin{aligned} T_{\text{moy}} &= \Theta(\sum_{i=1}^{n} \frac{1}{1-\frac{i-1}{n}}) \\ &= \Theta(\sum_{i=1}^{n} \frac{n}{n-(i-1)} \\ &= \Theta(n \sum_{k=1}^{n} \frac{1}{k}) \\ &= \Theta(nH_n) = \Theta(n \log n) \text{ car } H_n = \Theta(\log n) \end{aligned}$$

# 2.4 Analyse de cas particuliers de boucles

Parfois, il est possible de trouver un majorant de la complexité d'un algorithme en identifiant une variable monotone croissante dont la valeur est majorée.

## 2.4.1 Algorithme gourmand pour trouver une semengation optimale

### 2.4.1.1 Problème

Décomposer une suite d'entiers $A_1 A_2 \cdots A_n$ en un nombre minimum de segments tels que les valeurs dans un même segment ne différent que par au plus $k$.

**Ex** $k = 1$, $A = 1\ 1\ 1\ 2\ 3\ 4\ 4\ 4\ 4\ 3\ 3\ 1\ 1\ 1\ 2\ 2$

Cet exemple possède au moins deux segmentations possibles

| 111 | 23 | 44433 | 11122 | 4 segments. |
|---|---|---|---|---|
| 1112 | 344433 | 11122 | 3 segments. | |

**Application** Nettoyage de signal, compactage de données (avec perte d'informations)

**Algorithme gourmand**

1. Trouver le plus long préfixe $A_1 \cdots A_{i_1}$, de la suite $A_1 \cdots A_n$ telle que que $\forall i, j \in \{1, \cdots, i_1\}$, $|A_i - A_j| \leq k$
2. Appel récursif du même algorithme sur la suite $A_{i_1+1} \cdots A_n$

#### 2.4.1.2 Démonstration que l'algorithme trouve toujours une segmentation optimale

Supposons que l'algorithme gourmand trouve une segmentation $\sigma$ dont les segments se terminent aux positions $i_1, i_2, \cdots, i_\sigma$, mais qu'il existe une segmentation optimale $\sigma_opt$ dont les segments se terminent aux positions $j_1, j_2, \cdots, j_t$ avec $t < r$.

Soient $i_0 = j_0 = 1$. Nous avons $i_t < i_r = n = j_t$.

Soit $m$ le plus petit indice tel que $i_m < j_m$. Donc $i_{m-1} \geq j_{m-1}$. Un tel indice existe car $i_0 = j_0$ et $i_t < j_t$

Par définition de la segmentation «gourmande», $\sigma$, il y a une valeur $j$ dans le segment $S_m$ telle que $|y - x| > k$. Mais dans ce cas, $\sigma_{opt}$ n'est pas une segmentation valide. Cette contradiction montre que la segmentation gourmande est toujours optimale.

```
i   = 1;
m   = 0;
i0  = 1;
tantque (i <= n) faire
  m = m + 1; --On cherche le segment Sm
  max = min = A[i] -- max et min sont les valeurs max et min de Sm
  i = i + 1;
  tantque (i <= n et A[i]-min <= k et max-A[I] <= k) faire
    si A[i] > max alors
      max = A[i];
    fin si;
    si A[i] < min alors
      min = A[i];
    fin si;

    i = i + 1;
  fin tantque
  im = i-1; --im = fin du segmetn de Sm
fin tantque;
```

Chaque itération des deux boucles `tantque` incrémente $i$. Puisque $i \leq n$, on peut en déduire $T_{\max}(n) = \Theta(n)$ malgré la présence de deux boucles imbriquées.

Annexe A

# 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 afirmations 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 : $\max(f(n), g(n)) \leq f(n)+n(n) \leq 2\max(f(n), g(n))$
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.

$$\begin{array}{ccc} f(n) \leq c_1 g(n) & , & g(n) \leq c_2 f(n) \\ \frac{1}{c_2} \leq \frac{f(n)}{g(n)} \leq c_1.1 & \Rightarrow & \frac{f(n)}{g(n)} = \Theta(1) \end{array}$$

### A.1.7 Simplifiez les expressions suivantes

1. $O(4n^2 + 3n^2 + 7\log_2(n^n)) = O(n^3)$
2. $\Theta(n \log_2 n + 17n + 2n^3 = \Theta(n^2)$
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. $n^2 \log_e n$

### A.1.9

$$\Theta(\frac{1}{1-p})$$

$$p = 1 - (\frac{1}{6})^{n-1}$$

$$\Theta(\frac{1}{1-(1-\frac{1}{6^{n-1}})}) = \Theta(\frac{1}{\frac{1}{6^{n-1}}}) = \Theta(6^{n-1}) = \Theta(6^n)$$

Donc réponse D.

### A.1.10

a $\Theta(1)$
b $\Theta(1)$
c $\Theta(\log n)$
d $\Theta(n \log n)$
e $\Theta(n^3)$
f $\Theta(n^4)$

### A.1.11

### A.1.12