Complexité des algorithmes

Remise à niveau en Informatique

Valérie Gillot

Master Informatique & Mathématiques

Fonctionnement

Découpage des heures

1. 4 h de CM-TD
2. 4 séances de TD-TP de 4 h : pour 4 énoncés
3. 1 séance de TP de 3 h : pour 2 énoncés

Évaluation

1. une note de TD-TP par trinôme pour ma partie
2. une note de contrôle final pour l'ensemble de la partie info (CC le 4 octobre 2024)

Composition des équipes de TD-TP

Pour composer les équipes, je vous ai regroupés en 3 groupes comme suit :

A1. Jeanne Barastier
A2. Enzo Berger
A3. Loïs Hermelin
A4. André Marcais
A5. Quentin Lavit
A6. Tom Bartier

B1. Emad Ba Gubair
B2. Mattéo Bonavita
B3. Yacine Haouas
B4. Lilian Laure
B5. Pierre Nicolas
B6. Jean-Baptiste Nguyen

C1. Yann Roblin
C2. Florian Audouard
C3. Mattéo Pegliasco
C4. Shawn Pelerin
C5. Stéphane Rossi
C6. (Lucie Quarta)

Chaque équipe est constituée de 3 étudiants, un par groupe.
Les équipes sont :

Équipe 1 : ('A2', 'B2', 'C5')
Équipe 2 : ('A3', 'B3', 'C4')
Équipe 3 : ('A5', 'B6', 'C3')
Équipe 4 : ('A1', 'B4', 'C2')
Équipe 5 : ('A4', 'B1', 'C6')
Équipe 6 : ('A6', 'B5', 'C1')

Références

Ce cours est principalement une extration et compilation des cours

- *Algorithmique I*, de N. Méloni, niveau L1 informatique
- *Algorithmique II*, de J.-P. Zanotti, niveau L2 informatique

1. Algorithmique
2. Analyse des algorithmes
3. Analyse asymptotique de la complexité
4. Ordres de grandeur
5. Analyse de boucles

Section 1

Algorithmique

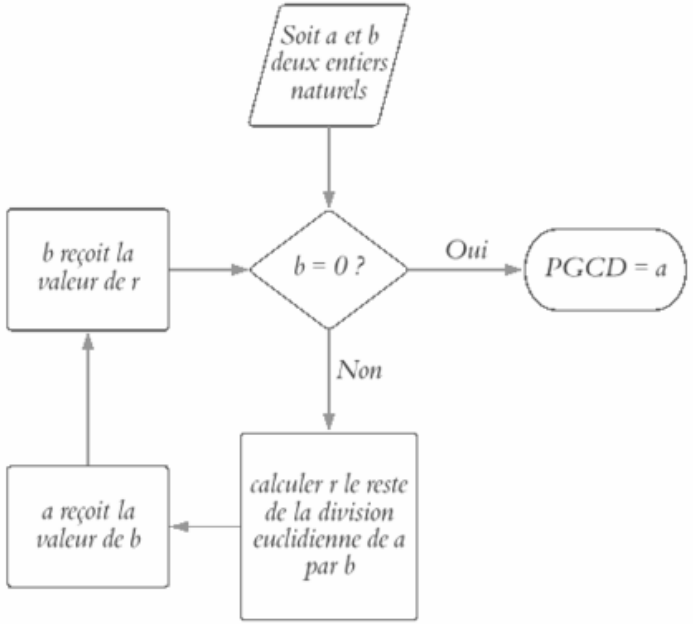
Algorithme [1] : Késako ?

Définition informelle

On appelle algorithme tout procédé de résolution d'un problème en un nombre fini d'étapes par application d'une série de règles prédéfinies.

Les premiers algorithmes

- 1600 avant JC : algorithmes de factorisation ou d'extraction de racines carrées des babyloniens
- 300 avant JC : algorithme d'Euclide pour calculer le pgcd.



1. Le mot « algorithme » vient du nom du mathématicien Al-Khwârizmî1 (latinisé au Moyen Âge en Algoritmi), qui, au IXe siècle écrivit le premier ouvrage systématique donnant des solutions aux équations linéaires et quadratiques.

Algorithmique : Késako ?

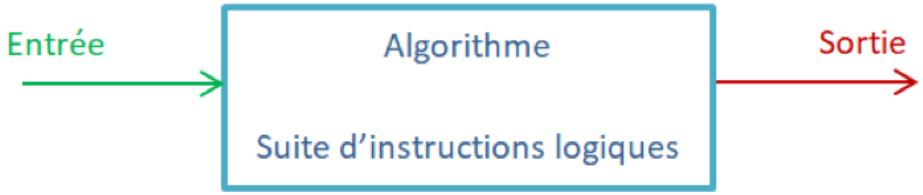
L'algorithmique

est le domaine des sciences qui étudie les règles et les techniques impliquées dans

- la conception
  et
- l'analyse des algorithmes.

1. concevoir des méthodes efficaces pour automatiser la résolution d'un problème.
2. s'assurer de son efficacité avec une économie des ressources principalement en temps et en espace.

Conception des algortithmes



On décrit les algorithmes grâces aux structures suivantes :

Les structures de contrôles

- séquence
- sélection/alternative
- itération

Les structures de données

- constante
- variable
- tableau
- arbre, liste, pile, file,
- ...

Exemple d'algorithme

Problème : recherche d'un élément dans un tableau

Entrée : un tableau de n élément et un élément e

Sortie : l'indice du tableau où se trouve l'élément ou 0 s'il ne s'y trouve pas

```
ALGORITHME
 DEBUT
   i ← 1
   TQ i ⩽ n FAIRE
     SI e = T[i] ALORS
       RENVOYER i
     FSI
     i ← i+1
   FTQ
   RENVOYER 0
 FIN
```

```
DEBUT
  i ← 1
  TQ i ⩽ n ET e≠T[i] FAIRE
    i ← i+1
  FTQ
  SI i>n ALORS
    RENVOYER 0
  SINON
    RENVOYER i
  FSI
FIN
```

Section 2

Analyse des algorithmes

Analyse des algorithmes

Terminaison, correction et complétude

1. La terminaison est l'assurance que l'algorithme terminera en un temps fini (exhiber une fonction entière positive strictement décroissante à chaque étape de l'algorithme).
2. La preuve de correction c'est prouver que si l'algorithme termine en donnant un résultat, alors ce résultat est effectivement une solution au problème posé (à l'aide de spécification logique - preuve de programme)
3. La preuve de complétude garantit que, pour un espace de problèmes donné, l'algorithme, s'il termine, donnera l'ensemble des solutions de l'espace du problème (identification de l'espace du problème et l'espace des solutions, puis montrer que l'algorithme produit bien le second à partir du premier)

Complexité des algorithmes ✓

Analyse du coût de l'algorithme : prévoir les ressources qui sont nécessaires à son exécution.

Complexité d'un algorithme

Les ressources nécessaires dépendent du contexte

- temps
- espace mémoire
- consommation électrique
- coût financier

Indépendance

L'analyse ne doit pas dépendre :

- de la machine sur laquelle s'exécute l'algorithme
- du langage de programmation utilisé pour l'implanter

Nécessité de concevoir un modèle d'étude indépendant : le modèle RAM.

La modèle RAM

(R)andom (A)ccess (M)achine

Machine hypothétique pour laquelle :

- les opérations simple (+,-,*,/,`if`, appels) consomment une unité de temps
- les boucles sont des compositions d'opération simples, leur temps d'exécution dépend du nombre d'itérations et de la nature des opérations à l'intérieur de la boucle
- un accès mémoire consomme une unité de temps
- la quantité de mémoire n'est pas limitée

- Mesurer le temps d'exécution = compter le nombre d'étapes effectuées pour une instance donnée.

Malgré sa simplicité, le modèle permet une analyse très juste du comportement d'un algorithme sur une machine réelle.

Section 3

Analyse asymptotique de la complexité

Complexité en temps

Mesurer le nombre d'opérations en fonction de la taille n des données

- A priori, plus la taille de l'entrée est grande plus longue est la résolution du problème.
- Pour étudier l'éfficacité d'un algorithme on considère toujours des instances du problème à taille fixée.
- La complexité d'un algorithme nous renseigne sur comment évolue le temps d'exécution avec la taille de l'entrée.
- C'est une fonction de n souvent notée $C(n)$ ou $T(n)$.

Analyse asymptotique

Somme des éléments d'un tableau

Entrée : T un tableau de n éléments
Sortie : la somme des éléments du tableau

```
ALGORITHME Somme(T)
DONNEES
    T un tableau de n entiers
VARIABLES
    S,i des entiers
DEBUT
   S ← 0
   i ← 1
   TQ i ⩽ n FAIRE
     S ← S + T[i]
     i ← i+1
   FTQ
   RENVOYER S
FIN
```

Terminaison

Pour $1 \leqslant i \leqslant n$, $n - i$ est une fonction strictement décroissante

Complexité en temps

Pour un tableau T de taille n

- il faut toujours n passages dans la boucle pour terminer l'algorithme
- indépendamment de l'instance

Analyse asymptotique

Recherche séquentielle d'un élément

Entrée : un tableau de n élément et un élément e
Sortie : VRAI si l'élément e a été trouvé et FAUX sinon

```
1  ALGORITHME Recherche(e,T)
2  DONNEES
3      T un tableau de n elements
4      e un element
5  DEBUT
6     i ← 1
7     TQ i ⩽ n ET e≠T[i] FAIRE
8       i ← i+1
9     FTQ
10    RENVOYER  (i ⩽ n)
11 FIN
```

Même à taille fixée, certaines instances peuvent être plus faciles à résoudre que d'autres

Complexité en temps

- meilleur cas : l'élement recherché est au début du tableau 1 tour de boucle
- pire cas : l'élement recherché n'est pas dans le tableau n tours de boucle
- cas moyen : l'élement recherché est au milieu environ $n/2$ tours de boucle

Analyse asymptotique

On considère traditionnelement trois cas :

Meilleur des cas : $\check{T}(n)$

- Le nombre minimal d'étapes effectuées par l'algorithme pour une instance instance de taille n du problème. C'est le cas d'exécution le plus favorable possible.

Pire des cas : $\widehat{T}(n)$

- Le nombre maximal d'étapes effectuées par l'algorithme pour une instance de taille n du problème. C'est le cas d'exécution le plus défavorable possible.

Cas moyen : $\overline{T}(n)$

- Le nombre moyen d'étapes effectuées par l'algorithme pour l'ensemble des instances de taille n du problème.

Familles de complexité

- Etude des fonctions de complexité des algorithmes nécessitent des outils d'analyse efficaces
- Intérêt pour le comportement asymptotique (pour des tailles de données conséquentes)
- L'expression exacte de la fonction de complexité souvent difficile à obtenir
- Déterminer le comportement général de la fonction avec un ordre de grandeur comme par exemple le O de Landau

Section 4

Ordres de grandeur

Ordre de grandeur : O de Landau [2]

La complexité des algorithmes

- s'exprime en fonction de la taille des données $n \in \mathbb{N}$ et à valeurs dans $\mathbb{R}$
- notons $\mathcal{F}$ l'ensemble des fonctions de $\mathbb{N}$ dans $\mathbb{R}$

$f \in O(g)$ — **f est en grand-O de g** — **$f = O(g)$**

- $O(g) = \{f \in \mathcal{F} \mid \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, 0 \leqslant f(n) \leqslant c.g(n)\}$
- f est dominée par g à une constante près

Exercice

Vérifier avec la définition formelle que

1. $3n^2 + 1 = O(n^2)$
2. $3n^2 - n + 6 = O(n^2)$
3. $3n^2 - n + 6 = O(n^3)$
4. $3n^2 - n + 6 \neq O(n)$

Correction

1. il suffit de prendre $c = 4$ et $n_0 = 1$
2. il suffit de prendre $c = 3$ et $n_0 = 6$
3. il suffit de prendre $c = 1$ et $n_0 = 4$
4. $\forall c, cn < 3n^2 - n + 6$ quand $n > c + 1$

2. Edmund Landau (1877-1938) mathématicien allemand, contributions en analyse complexe et théorie analytique des nombres

Ordre de grandeur : O de Landau [2]

La complexité des algorithmes

- s'exprime en fonction de la taille des données $n \in \mathbb{N}$ et à valeurs dans $\mathbb{R}$
- notons $\mathcal{F}$ l'ensemble des fonctions de $\mathbb{N}$ dans $\mathbb{R}$

$f \in O(g)$ — **f est en grand-O de g** — **$f = O(g)$**

- $O(g) = \{f \in \mathcal{F} \mid \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, 0 \leqslant f(n) \leqslant c.g(n)\}$
- f est dominée par g à une constante près

Exercice

Vérifier avec la définition formelle que

1. $3n^2 + 1 = O(n^2)$
2. $3n^2 - n + 6 = O(n^2)$
3. $3n^2 - n + 6 = O(n^3)$
4. $3n^2 - n + 6 \neq O(n)$

Correction

1. il suffit de prendre $c = 4$ et $n_0 = 1$
2. il suffit de prendre $c = 3$ et $n_0 = 6$
3. il suffit de prendre $c = 1$ et $n_0 = 4$
4. $\forall c, cn < 3n^2 - n + 6$ quand $n > c + 1$

2. Edmund Landau (1877-1938) mathématicien allemand, contributions en analyse complexe et théorie analytique des nombres

Ordre de grandeur : Ω de Knuth [3]

$f \in \Omega(g)$ — f est en grand-Omega de g — $f(n) = \Omega(g(n))$

- $\Omega(g) = \{f \in \mathcal{F} \mid \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, 0 \leqslant c.g(n) \leqslant f(n)\}$
- f est minorée par g à une constante près

$$f = \Omega(g) \Longleftrightarrow g = O(f)$$

Exercice

Vérifier que

- $3n^3 - 2n + 1 = \Omega(n^3)$
- $3n^2 - n + 6 = \Omega(n)$
- $3n^2 - n + 6 \neq \Omega(n^3)$

Correction

1. en prenant $c = 2$ et $n_0 = 0$
2. en prenant $c = 1$ et $n_0 = 0$
3. $\forall c, 3n^2 - n + 6 < c.n^3$ quand $c.n > 3$ et $n > 6$

3. Donald Knuth (1938-) informaticien et mathématicien américain de renom, pionner de l'algorithmique (*The Art of Computer Programming*,*Logiciel libre TEX*)

Ordre de grandeur : Ω de Knuth [3]

$f \in \Omega(g)$ — f est en grand-Omega de g — $f(n) = \Omega(g(n))$

- $\Omega(g) = \{f \in \mathcal{F} \mid \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, 0 \leqslant c.g(n) \leqslant f(n)\}$
- f est minorée par g à une constante près

$$f = \Omega(g) \Longleftrightarrow g = O(f)$$

Exercice

Vérifier que

- $3n^3 - 2n + 1 = \Omega(n^3)$
- $3n^2 - n + 6 = \Omega(n)$
- $3n^2 - n + 6 \neq \Omega(n^3)$

Correction

1. en prenant $c = 2$ et $n_0 = 0$
2. en prenant $c = 1$ et $n_0 = 0$
3. $\forall c, 3n^2 - n + 6 < c.n^3$ quand $c.n > 3$ et $n > 6$

3. Donald Knuth (1938-) informaticien et mathématicien américain de renom, pionner de l'algorithmique (*The Art of Computer Programming*,*Logiciel libre TEX*)

Ordre de grandeur : Θ

$f \in \Theta(g)$ — **f est en grand-Thêta** — **$f(n) = \Theta(g(n))$**

- f et g sont du même ordre de grandeur asymptotiquement f est à la fois majorée et minorée par g à une constante près
- $\Theta(g) = \{f \in \mathcal{F} \mid \exists c_1, c_2 > 0, \exists n_0 \in N, \forall n \geqslant n_0 \quad 0 \leqslant c_1.g(n) \leqslant f(n) \leqslant c_2.g(n)\}$
- $f = \Theta(g) \Longleftrightarrow (f = O(g) \wedge f = \Omega(g))$

Exercice

1. Montrer que

$$f = \Theta(g) \Longleftrightarrow (f = O(g) \wedge f = \Omega(g))$$

2. Vérifier que
   1. $3n^2 - n + 6 = \Theta(n^2)$
   2. $3n^2 - n + 6 \neq \Theta(n^3)$
   3. $3n^2 - n + 6 \neq \Theta(n)$

Correction

1. en prenant $c1 = 1$, $c_2 = 3$ et $n_0 = 6$
2. $3n^2 - n + 6 = O(n^3)$, mais $3n^2 - n + 6 \neq \Omega(n^3)$
3. $3n^2 - n + 6 = \Omega(n)$ mais $3n^2 - n + 6 \neq O(n)$

Ordre de grandeur : Θ

$f \in \Theta(g)$ — f est en grand-Thêta — $f(n) = \Theta(g(n))$

- f et g sont du même ordre de grandeur asymptotiquement f est à la fois majorée et minorée par g à une constante près
- $\Theta(g) = \{f \in \mathcal{F} \mid \exists c_1, c_2 > 0, \exists n_0 \in N, \forall n \geqslant n_0 \quad 0 \leqslant c_1.g(n) \leqslant f(n) \leqslant c_2.g(n)\}$
- $f = \Theta(g) \Longleftrightarrow (f = O(g) \wedge f = \Omega(g))$

Exercice

1. Montrer que

$$f = \Theta(g) \Longleftrightarrow (f = O(g) \wedge f = \Omega(g))$$

2. Vérifier que
   1. $3n^2 - n + 6 = \Theta(n^2)$
   2. $3n^2 - n + 6 \neq \Theta(n^3)$
   3. $3n^2 - n + 6 \neq \Theta(n)$

Correction

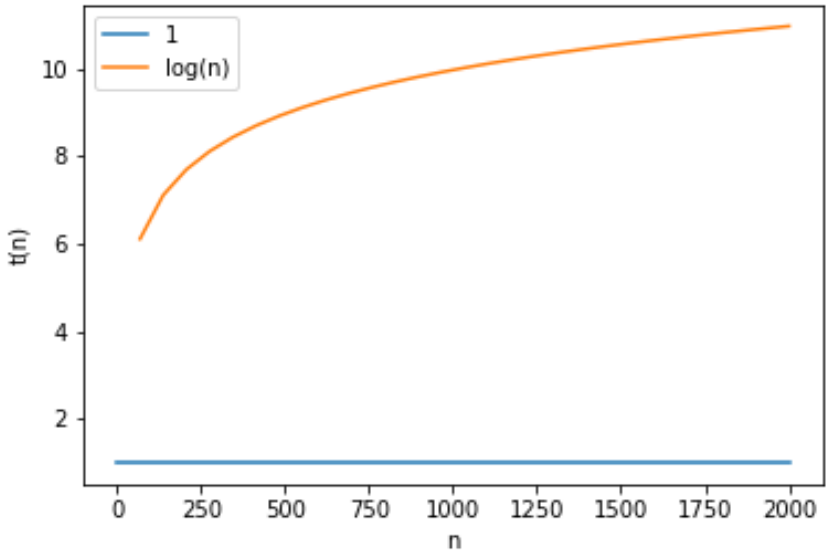
1. en prenant $c1 = 1$, $c_2 = 3$ et $n_0 = 6$
2. $3n^2 - n + 6 = O(n^3)$, mais $3n^2 - n + 6 \neq \Omega(n^3)$
3. $3n^2 - n + 6 = \Omega(n)$ mais $3n^2 - n + 6 \neq O(n)$

Les principales classes de complexité

Ordres de grandeur usuels

Pour n la taille d'un instance de l'algorithme

- constante : $O(1)$
- logarithmique : $O(\log n)$
- linéaire : $O(n)$
- linéarithmique : $O(n \log n)$
- quadratique : $O(n^2)$
- polynomiale : $O(n^c)$ $(c > 1)$
- exponentielle : $O(c^n)$ $(c > 1)$
- factorielle : $O(n!)$

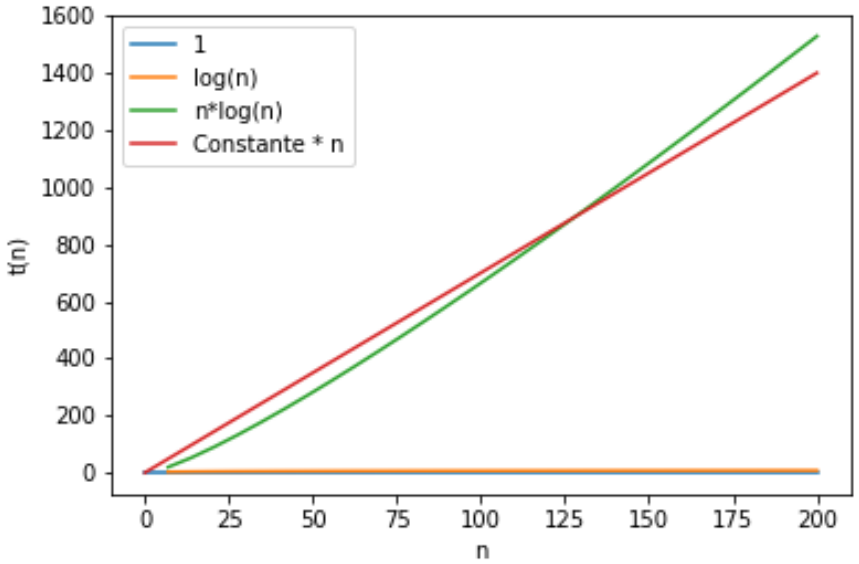


Les principales classes de complexité

Ordres de grandeur usuels

Pour n la taille d'un instance de l'algorithme

- constante : $O(1)$
- logarithmique : $O(\log n)$
- linéaire : $O(n)$
- linéarithmique : $O(n \log n)$
- quadratique : $O(n^2)$
- polynomiale : $O(n^c)$ $(c > 1)$
- exponentielle : $O(c^n)$ $(c > 1)$
- factorielle : $O(n!)$

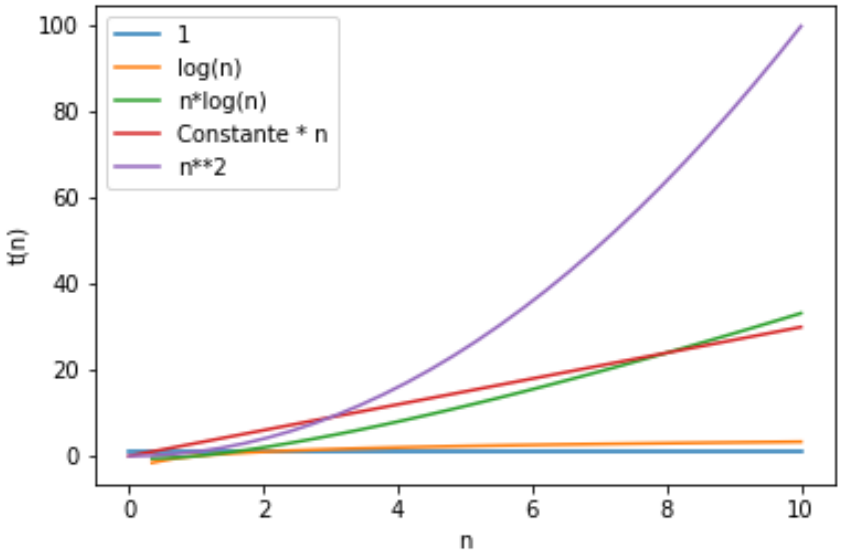


Les principales classes de complexité

Ordres de grandeur usuels

Pour n la taille d'un instance de l'algorithme

- constante : $O(1)$
- logarithmique : $O(\log n)$
- linéaire : $O(n)$
- linéarithmique : $O(n \log n)$
- quadratique : $O(n^2)$
- polynomiale : $O(n^c)$ $(c > 1)$
- exponentielle : $O(c^n)$ $(c > 1)$
- factorielle : $O(n!)$

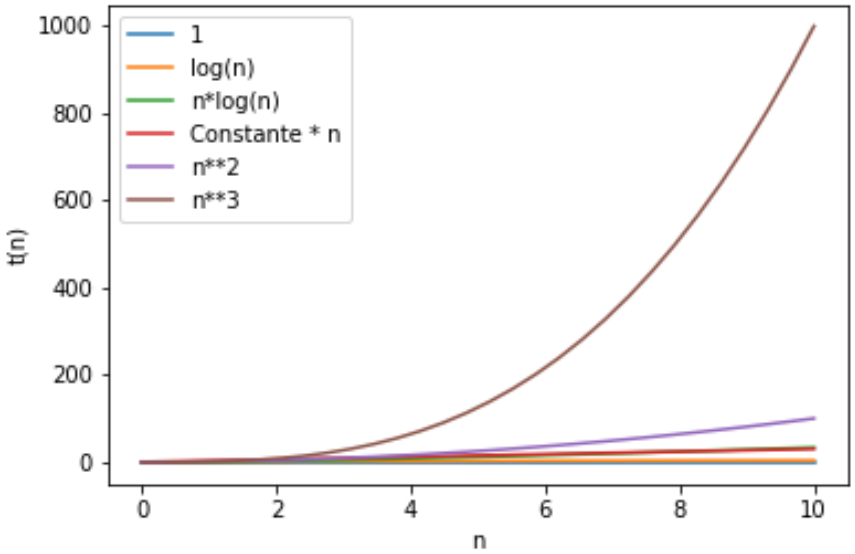


Les principales classes de complexité

Ordres de grandeur usuels

Pour n la taille d'un instance de l'algorithme

- constante : $O(1)$
- logarithmique : $O(\log n)$
- linéaire : $O(n)$
- linéarithmique : $O(n \log n)$
- quadratique : $O(n^2)$
- polynomiale : $O(n^c)$ $(c > 1)$
- exponentielle : $O(c^n)$ $(c > 1)$
- factorielle : $O(n!)$

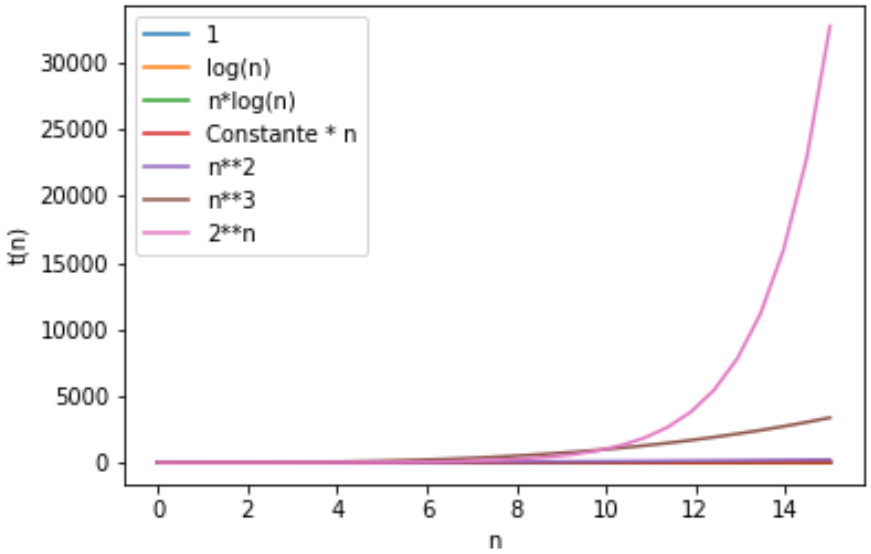


Les principales classes de complexité

Ordres de grandeur usuels

Pour n la taille d'un instance de l'algorithme

- constante : $O(1)$
- logarithmique : $O(\log n)$
- linéaire : $O(n)$
- linéarithmique : $O(n \log n)$
- quadratique : $O(n^2)$
- polynomiale : $O(n^c)$ $(c > 1)$
- exponentielle : $O(c^n)$ $(c > 1)$
- factorielle : $O(n!)$



Ordre de grandeur : après les grands les petits !

$f(n) = o(g(n))$ — f est en petit-o de g

- f est négligeable devant g asymptotiquement
- $o(g) = \{f \in \mathcal{F} \mid \forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, 0 \leqslant f(n) \leqslant c.g(n)\}$
- $\lim\limits_{n \to +\infty} \frac{f(n)}{g(n)} = 0$

$f(n) = \omega(g(n))$ — f est en petit-omega de g

- f domine g asymptotiquement
- $\omega(g) = \{f \in \mathcal{F} \mid \forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geqslant n_0, 0 \leqslant c.g(n) \leqslant f(n)\}$
- $\lim\limits_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$

$$f = \omega(g) \Longleftrightarrow g = o(f)$$

Ordre de grandeur : après les grands les petits !

$f(n) \sim g(n)$ — f est équivalent à g

- f et g sont asymptotiquement équivalentes
- $\forall \epsilon \in \mathbb{R}_+, \exists n_0 \in N, \forall n \geqslant n_0, |f(n) - g(n)| \leqslant \epsilon g(n)$
- $\lim\limits_{n \to +\infty} \frac{f(n)}{g(n)} = 1$

Exercice

Montrer que

1. $n^2 + 2n + 1 \sim n^2 - 50n + 75$

Calculer avec les ordres de grandeurs

Les propriétés de O, Ω, Θ

1. $f(n) + g(n) = O(\max(f(n), g(n)))$
2. $c \times f(n) = O(f(n)), \forall c > 0$
3. si $f_1(n) = O(g_1(n))$ et $f_2(n) = O(g_2(n))$ alors $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Les propriétés sont aussi vraies pour Ω et Θ.

Exercice

Montrer que

1. $O(n) + O(n) = O(n)$
2. $n \times O(g(n)) = O(ng(n))$
3. $k \times n^p = O(n^p)$

Calculer/simplifier

- $\Theta(n) + \Theta(n)$
- $O(n) + \Theta(n)$
- $\Theta(n) + \Theta(1)$
- $O(n) + \Omega(n)$
- $O(n)O(n)$
- $n\Theta(1)$
- $O(n) + O(n^2)$

Famille de complexité : logarithmique

$f(n) = \log(n)$

- $\log_b(n) = \ln(n)/\ln(b)$
- $\log_b(b) = 1$ et $\log_b(1) = 0$
- $\log_b(xy) = \log_b(x) + \log_b(y)$
- $\log_b(x/y) = \log_b(x) - \log_b(y)$
- $\log_b(b^n) = n$
- $a^{\log_b(n)} = n^{\log_b(a)}$

Exercice

Vérifier que

1. $a^{\ln(b)} = b^{\ln(a)}$
2. $a^{\log_b(n)} = n^{\log_b(a)}$
3. $\log(n^c) = O(\log n)$

Correction

On utilise la définition de $a^x := e^{x \ln(a)}$

1. $a^{\ln(b)} = e^{\ln(a)\ln(b)} = b^{\ln(a)}$
2. $a^{\log_b(n)} = e^{\ln(a)\log_b(n)} = e^{\ln(a)\frac{\ln(n)}{\ln(b)}} = e^{\ln(n)\frac{\ln(a)}{\ln(b)}} = e^{\log_b(a)\ln(n)} = n^{\log_b(a)}$

Famille de complexité : logarithmique

$f(n) = \log(n)$

- $\log_b(n) = \ln(n)/\ln(b)$
- $\log_b(b) = 1$ et $\log_b(1) = 0$
- $\log_b(xy) = \log_b(x) + \log_b(y)$
- $\log_b(x/y) = \log_b(x) - \log_b(y)$
- $\log_b(b^n) = n$
- $a^{\log_b(n)} = n^{\log_b(a)}$

Exercice

Vérifier que

1. $a^{\ln(b)} = b^{\ln(a)}$
2. $a^{\log_b(n)} = n^{\log_b(a)}$
3. $\log(n^c) = O(\log n)$

Correction

On utilise la définition de $a^x := e^{x \ln(a)}$

1. $a^{\ln(b)} = e^{\ln(a)\ln(b)} = b^{\ln(a)}$
2. $a^{\log_b(n)} = e^{\ln(a)\log_b(n)} = e^{\ln(a)\frac{\ln(n)}{\ln(b)}} = e^{\ln(n)\frac{\ln(a)}{\ln(b)}} = e^{\log_b(a)\ln(n)} = n^{\log_b(a)}$

Famille de complexité : polynomiale

$f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0,\ a_k > 0$

- $f(n) = \Theta(n^k)$
- $k = 1$ on parle de complexité linéaire
- $k = 2$ on parle de complexité quadratique
- On peut étendre la définition aux puissances réelles : $n\sqrt{n} = n^{1.5} = O(n^2)$

Exercice

Montrer que

- $\forall a > 0, b > 0, \lim\limits_{n \to +\infty} \frac{(\log n)^a}{n^b} = 0$
- $\forall a > 0, b > 0, \lim\limits_{n \to +\infty} \frac{\log(n^a)}{n^b} = 0$
- En déduire que $(\log n)^a = O(n^b)$ et $\log(n^a) = O(n^b)$

Famille de complexité : exponentielle

$f(n) = a^n$

- $a^0 = 1, a^1 = a, a^{-1} = 1/a$
- $a^{m+n} = a^m a^n$
- $(a^m)^n = (a^n)^m = a^{mn}$

Exercice

1. Montrer que $\forall a > 1, b > 0, \lim_{n \to +\infty} \frac{n^b}{a^n} = 0$
2. En déduire que $n^b = O(a^n)$

Famille de complexité : factorielle

$f(n) = n!$

- $0! = 1$
- $n! = n \times (n-1) \times \cdots \times 2 \times 1$
- $(n+1)! = (n+1)n!$
- $\ln n! = n \ln n - n + \frac{\ln n}{2} + O(1)$

Exercice

1. Montrer que $\forall a > 0, \lim_{n \to +\infty} \frac{a^n}{n!} = 0$
2. En déduire que $a^n = O(n!)$

Ordre de grandeur

Estimation des temps de calcul pour quelques complexités standards (vit. de calcul : 3.5×10^9 op/s)

n	$\log(n)$	n	$n\log(n)$	n^2	2^n	$n!$
10	$0.001\mu s$	$0.003\mu s$	$0.007\mu s$	$0.029\mu s$	$0.293\mu s$	$0.001s$
20	$0.001\mu s$	$0.006\mu s$	$0.017\mu s$	$0.114\mu s$	$0.3ms$	$22ans$
30	$0.001\mu s$	$0.009\mu s$	$0.029\mu s$	$0.257\mu s$	$0.307s$	$2.4 \times 10^{15} ans$
40	$0.001\mu s$	$0.011\mu s$	$0.042\mu s$	$0.457\mu s$	$5.2min$	$7.3 \times 10^{30} ans$
50	$0.001\mu s$	$0.014\mu s$	$0.056\mu s$	$0.714\mu s$	$3.7jours$	$2.7 \times 10^{47} ans$
100	$0.001\mu s$	$0.029\mu s$	$0.132\mu s$	$0.003ms$	$1.1 \times 10^{13} ans$	
1000	$0.002\mu s$	$0.286\mu s$	$0.002ms$	$0.286ms$		
10000	$0.003\mu s$	$0.003ms$	$0.026ms$	$0.029s$		
100000	$0.003\mu s$	$0.029ms$	$0.329ms$	$2.8s$		
1000000	$0.004\mu s$	$0.286ms$	$0.004s$	$4.7min$		
10000000	$0.005\mu s$	$0.003s$	$0.046s$	$7.9h$		

Section 5

Analyse de boucles

Analyse de boucle : méthodologie

- Identifier les variables qui contrôlent la boucle
- Identifier les valeurs initiales des variables impliquées
- Identifier la condition d'arrêt
- Identifier les instructions où sont modifiées les variables dont dépend la condition d'arrêt
- Compter le nombres d'exécutions
- Evaluer le coût d'un passage dans la boucle
- Utiliser des techniques de sommation sur les entiers

Formule de sommation

Suite arithmétique : $\forall n \geqslant 0, \quad u_{n+1} = u_n + r$

- $\forall n \geqslant 0, \quad u_n = u_0 + nr$
- $\sum_{i=0}^{n} u_i = (n+1)\frac{u_0+u_n}{2}$, où $(n+1)$ est le nombre de termes dans la somme

Exercice

Calculer le terme général et la somme pour les suites

1. $(u_n)_{n\geqslant 0} = (0, 1, 2, 3, \dots)$
2. $(u_n)_{n\geqslant 0} = (1, 2, 3, \dots)$
3. $(u_n)_{n\geqslant 0} = (1, 3, 5, 7 \dots)$
4. $(u_n)_{n\geqslant 0} = (0, 2, 4, 6 \dots)$

1. $u_0 = 0, r = 1, \forall n \geqslant 0, u_n = n, \sum_{i=0}^{n} i = \frac{(n+1)n}{2}$
2. $u_0 = 1, r = 1, \forall n \geqslant 0, u_n = n+1, \sum_{i=0}^{n}(i+1) = \frac{(n+1)(n+2)}{2} \quad \sum_{i=0}^{n-1}(i+1) = \frac{(n+1)n}{2}$
3. $u_0 = 1, r = 2, \forall n \geqslant 0, u_n = 2n+1 \quad \sum_{i=0}^{n}(2i+1) = (n+1)^2$
4. $u_0 = 2, r = 2, \forall n \geqslant 0, u_n = 2n, \sum_{i=0}^{n}(2i) = \frac{(n+1)(2n+0)}{2} = n(n+1)$

Formule de sommation

Suite arithmétique : $\forall n \geqslant 0, \quad u_{n+1} = u_n + r$

- $\forall n \geqslant 0, \quad u_n = u_0 + nr$
- $\sum_{i=0}^{n} u_i = (n+1)\frac{u_0+u_n}{2}$, où $(n+1)$ est le nombre de termes dans la somme

Exercice

Calculer le terme général et la somme pour les suites

1. $(u_n)_{n\geqslant 0} = (0, 1, 2, 3, \dots)$
2. $(u_n)_{n\geqslant 0} = (1, 2, 3, \dots)$
3. $(u_n)_{n\geqslant 0} = (1, 3, 5, 7 \dots)$
4. $(u_n)_{n\geqslant 0} = (0, 2, 4, 6 \dots)$

1. $u_0 = 0, r = 1, \forall n \geqslant 0, u_n = n, \sum_{i=0}^{n} i = \frac{(n+1)n}{2}$
2. $u_0 = 1, r = 1, \forall n \geqslant 0, u_n = n+1, \sum_{i=0}^{n}(i+1) = \frac{(n+1)(n+2)}{2} \quad \sum_{i=0}^{n-1}(i+1) = \frac{(n+1)n}{2}$
3. $u_0 = 1, r = 2, \forall n \geqslant 0, u_n = 2n+1 \quad \sum_{i=0}^{n}(2i+1) = (n+1)^2$
4. $u_0 = 2, r = 2, \forall n \geqslant 0, u_n = 2n, \sum_{i=0}^{n}(2i) = \frac{(n+1)(2n+0)}{2} = n(n+1)$

Formule de sommation

Suite géométrique : $\forall n \geqslant 0, \quad u_{n+1} = qu_n \quad q \neq 1$

- $\forall n \geqslant 0, \quad u_n = q^n u_0$
- $\sum_{i=0}^{n} u_i = u_0 \frac{1-q^{n+1}}{1-q}$ où $n+1$ est le nombre de termes dans la somme

Exercice

1. Calculer le terme général et la somme pour la suite $(u_n)_{n\geqslant 0} = (1, 2, 4, 8, 16, \dots)$
2. Proposer une preuve en utilisant la réprésentation binaire de $2^{n+1} - 1$

1. $u_0 = 1, q = 2, \forall n \geqslant 0, u_n = 2^n, \sum_{i=0}^{n} 2^i = \frac{1-2^{n+1}}{1-2} = 2^{n+1} - 1$
2. $2^{n+1} - 1 = (\underbrace{111\dots11}_{n+1})_2 = \sum_{i=0}^{n} 2^i$

Formule de sommation

Suite géométrique : $\forall n \geqslant 0, \quad u_{n+1} = qu_n \quad q \neq 1$

- $\forall n \geqslant 0, \quad u_n = q^n u_0$
- $\sum_{i=0}^{n} u_i = u_0 \frac{1-q^{n+1}}{1-q}$ où $n+1$ est le nombre de termes dans la somme

Exercice

1. Calculer le terme général et la somme pour la suite $(u_n)_{n\geqslant 0} = (1, 2, 4, 8, 16, \dots)$
2. Proposer une preuve en utilisant la réprésentation binaire de $2^{n+1} - 1$

1. $u_0 = 1, q = 2, \forall n \geqslant 0, u_n = 2^n, \sum_{i=0}^{n} 2^i = \frac{1-2^{n+1}}{1-2} = 2^{n+1} - 1$
2. $2^{n+1} - 1 = (\underbrace{111\dots11}_{n+1})_2 = \sum_{i=0}^{n} 2^i$

Formule de sommation : linéarité pour les sommes finies

Pour les suites

$$\sum_{i=a}^{b}(u_i + v_i) = \sum_{i=a}^{b} u_i + \sum_{i=a}^{b} v_i$$

Pour les ordres de grandeurs

Pour une portion de code de complexité $C(i) = \Theta(f(i))$

$$\sum_{i=1}^{n} \Theta(f(i)) = \Theta(\sum_{i=1}^{n} f(i))$$

en effet $\exists c_1, c_2 > 0$

$$c_1 f(i) \leqslant C(i) \leqslant c_2 f(i)$$

$$c_1 \sum_{i=1}^{n} f(i) \leqslant C(n) = \sum_{i=1}^{n} C(i) \leqslant c_2 \sum_{i=1}^{n} f(i)$$

Analyse de boucle

```
1 DEBUT
2   i ← 1
3   TQ i ⩽ n FAIRE
4     i ← i+1
5   FTQ
6 FIN
```

Terminaison

$n - i$ est une fonction strictement décroissante

Complexité en temps

La variable `i` contrôle la boucle

- initialisation : $i \leftarrow 1$
- condition d'arrêt : $i > n$
- i est modifié à l'instruction 4 : $i \leftarrow i + 1$, coût de l'instruction $\Theta(1)$
- les valeurs de i sont $1, 2, 3, \ldots$
- la condition d'arrêt est réalisée quand $i = n + 1$
- nb de passages dans la boucle : n

La complexité est en $\Theta(n)$

Analyse de boucle

```
DEBUT
  i ← 1
  TQ i ⩽ n FAIRE
    bloc instructions
    i ← i+1
  FTQ
FIN
```

Terminaison

$n - i$ est une fonction strictement décroissante

Complexité en temps

La variable i contrôle la boucle,

- le nombre de passage dans la boucle est le même que précedemment : n
- Coût du bloc d'instructions : $f(i, n)$

La complexité est en $\sum_{i=1}^{n} f(i, n)$

Analyse de boucles imbriquées

`n,m` des entiers donnés

```
1  DEBUT
2    i ← 1
3    TQ i ⩽ n FAIRE
4      j ← 1
5      TQ j ⩽ m FAIRE
6        j ← j+1
7      FTQ
8      i ← i+1
9    FTQ
10 FIN
```

Boucle interne : j contrôle la boucle

- les valeurs de j sont $1, 2, 3, \ldots$
- la condition d'arrêt est réalisée quand $j = m + 1$
- nb de passages dans la boucle : m
- coût de chaque passage (ligne 6) : $\Theta(1)$

La complexité est $C_{\text{int}}(m) = \sum_{j=1}^{m} \Theta(1) = \Theta(m)$

Boucle externe : i contrôle la boucle

- les valeurs de i sont $1, 2, 3, \ldots$
- la condition d'arrêt est réalisée quand $i = n + 1$
- nb de passages dans la boucle : n
- coût de chaque passage : $C_{\text{int}}(m) + \Theta(1) = \Theta(m)$

La complexité est $C_{\text{ext}}(n, m) = \sum_{i=1}^{n} \Theta(m) = n \times \Theta(m) = \Theta(nm)$

Analyse de boucles imbriquées

```
1  DEBUT
2    i ← 1
3    TQ i ⩽ n FAIRE
4      j ← 1
5      TQ j ⩽ i FAIRE
6        j ← j+1
7      FTQ
8      i ← i+1
9    FTQ
10 FIN
```

Boucle intérieure : j contrôle la boucle

- $j = 1, 2, \ldots i$
- la condition d'arrêt est réalisée quand $j = i + 1$
- coût de chaque passage (ligne 6) : $\Theta(1)$

La complexité est
$C_{\text{int}}(i) = \sum_{j=1}^{i} \Theta(1) = \Theta(\sum_{j=1}^{i} 1) = \Theta(i)$

Boucle externe : i contrôle la boucle

- $i = 1, 2, \ldots, n$
- la condition d'arrêt est réalisée quand $i = n + 1$
- nb passage dans la boucle : n
- coût de la boucle : $C_{\text{int}}(j) + \Theta(1) = \Theta(i)$
- complexité : $\sum_{i=1}^{n} C_{\text{int}}(j) = \sum_{i=1}^{n} \Theta(i) = \Theta(\sum_{i=1}^{n} i) = \Theta(\frac{(n+1)n}{2}) = \Theta(n^2)$

Analyse de boucles imbriquées

Exercice

Faire l'analyse de l'algorithme suivant :

```
DEBUT
  i ← 1
  TQ i ⩽ n FAIRE
    j ← 1
    TQ j ⩽ 2^i FAIRE
      j ← j+1
    FTQ
    i ← i+1
  FTQ
FIN
```

- complexité boucle intérieure : $C_{\text{int}}(i) = \sum_{j=1}^{2^i} \Theta(1) = \Theta(2^i)$
- complexité boucle extérieure : $C_{\text{ext}}(i,n) = \sum_{i=1}^{n} C_{\text{int}}(i) = \sum_{j=1}^{n} \Theta(2^i) = \Theta(\sum_{j=1}^{n} 2^i) = \Theta(2^n - 1) = \Theta(2^n)$

Analyse de boucles imbriquées

Exercice

Faire l'analyse de l'algorithme suivant :

```
DEBUT
  i ← 1
  TQ i ⩽ n FAIRE
    j ← 1
    TQ j ⩽ 2^i FAIRE
      j ← j+1
    FTQ
    i ← i+1
  FTQ
FIN
```

- complexité boucle intérieure :
  $C_{\text{int}}(i) = \sum_{j=1}^{2^i} \Theta(1) = \Theta(2^i)$
- complexité boucle extérieure :
  $C_{\text{ext}}(i,n) = \sum_{i=1}^{n} C_{\text{int}}(i) = \sum_{j=1}^{n} \Theta(2^i) = \Theta(\sum_{j=1}^{n} 2^i) = \Theta(2^n - 1) = \Theta(2^n)$

Analyse de boucle : dichotomie

```
1 DONNEES n entier
2 DEBUT
3   i ← n
4   TQ i ⩾ 1 FAIRE
5     i ← ⌊i/2⌋
6   FTQ
7 FIN
```

Majoration de la complexité : O

- on ne connait pas les valeurs exactes prises par i
- on les majore par les valeurs de la suite $(n, n/2, n/4, \dots)$
- après k passage dans la boucle, on a $i \leqslant n/2^k$
- la condition d'arrêt est satisfaite quand $n/2^k < 1$, i.e. $n < 2^k \Leftrightarrow \log_2(n) < k$
- nb de passage dans la boucle k est majoré par $\lfloor \log_2(n) \rfloor + 1$

La complexité est

$$C(n) = O(\log(n))$$

Analyse de boucle

```
1 DEBUT
2   i ← n
3   TQ i ⩾ 1 FAIRE
4     i ← ⌊i/2⌋
5   FTQ
6 FIN
```

Minoration de la complexité : Ω

- on minore par les valeurs prise par i par $(n/2, n/4, n/8, \dots)$
- après k passage dans la boucle, on a $i \leqslant n/2^{k+1}$
- la condition d'arrêt est satisfaite quand $n/2^{k+1} < 1$, i.e. $n < 2^{k+1} \Leftrightarrow \log_2(n) < k+1$
- nb de passage dans la boucle k est minoré par $1/2 \log_2(n)$ à partir d'un certain rang

La complexité est

$$C(n) = \Omega(\log(n))$$

La complexité est

$$C(n) = \Theta(\log(n))$$

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME A(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ i ⩽ n ET j ⩽ m FAIRE
        i ← i+1
        j ← j+1
    FTQ
 FIN
```

```
ALGORITHME B(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ i ⩽ n OU j ⩽ m FAIRE
        i ← i+1
        j ← j+1
    FTQ
FIN
```

Correction

$A(n,m) = \Theta(\min(n,m))$ et $B(n,m) = \Theta(\max(n,m))$

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME A(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ i ⩽ n ET j ⩽ m FAIRE
        i ← i+1
        j ← j+1
    FTQ
 FIN
```

```
ALGORITHME B(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ i ⩽ n OU j ⩽ m FAIRE
        i ← i+1
        j ← j+1
    FTQ
FIN
```

Correction

$A(n,m) = \Theta(\min(n,m))$ et $B(n,m) = \Theta(\max(n,m))$

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME C(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ j ⩽ m FAIRE
        SI i ⩽ n ALORS
            i ← i+1
        SINON
            j ← j+1
        FSI
    FTQ
 FIN
```

```
ALGORITHME D(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ j ⩽ m FAIRE
        SI i ⩽ n ALORS
            i ← i+1
        SINON
            j ← j+1
            i ← 1
        FSI
    FTQ
FIN
```

Correction

$C(n,m) = \Theta(n+m)$ et $D(n,m) = \Theta(\sum_{j=1}^{m} n) = \Theta(nm)$

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME C(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ j ⩽ m FAIRE
        SI i ⩽ n ALORS
            i ← i+1
        SINON
            j ← j+1
        FSI
    FTQ
 FIN
```

```
ALGORITHME D(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← 1
    TQ j ⩽ m FAIRE
        SI i ⩽ n ALORS
            i ← i+1
        SINON
            j ← j+1
            i ← 1
        FSI
    FTQ
FIN
```

Correction

$C(n,m) = \Theta(n+m)$ et $D(n,m) = \Theta(\sum_{j=1}^{m} n) = \Theta(nm)$

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME E(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    TQ i ⩽ n FAIRE
        j ← 1
        TQ j ⩽ m FAIRE
            j ← j+1
        FTQ
        i ← i+1
    FTQ
 FIN
```

```
ALGORITHME F(n)
DONNEES
    n un entier
DEBUT
    TQ n> 0 FAIRE
        n ← ⌊n/2⌋
    FTQ
FIN
```

```
ALGORITHME G(n)
DONNEES
    n un entier
DEBUT
    TQ n> 0 FAIRE
        n ← ⌊n/10⌋
    FTQ
FIN
```

Correction

$E(n,m) = \Theta(nm)$, $F(n) = \Theta(\log_2(n))$ et $G(n) = \Theta(\log_{10}(n))$

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME E(n,m)
DONNEES
    n,m des entiers
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    TQ i ⩽ n FAIRE
        j ← 1
        TQ j ⩽ m FAIRE
            j ← j+1
        FTQ
        i ← i+1
    FTQ
 FIN
```

```
ALGORITHME F(n)
DONNEES
    n un entier
DEBUT
    TQ n> 0 FAIRE
        n ← ⌊n/2⌋
    FTQ
FIN
```

```
ALGORITHME G(n)
DONNEES
    n un entier
DEBUT
    TQ n> 0 FAIRE
        n ← ⌊n/10⌋
    FTQ
FIN
```

Correction

$E(n,m) = \Theta(nm)$, $F(n) = \Theta(\log_2(n))$ et $G(n) = \Theta(\log_{10}(n))$

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME H(n)
DONNEES
    n entier
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    TQ i ⩽ n FAIRE
        j ← 2
        TQ j ⩽ √n FAIRE
            j ← j+1
        FTQ
        i ← i+1
    FTQ
 FIN
```

```
ALGORITHME I(n)
DONNEES
    n entier
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    j ← n
    TQ i*j ⩽ n² FAIRE
            i ← i+1
            j ← j+1
    FTQ
FIN
```

Estimer la complexité des algorithmes

Exercice

```
ALGORITHME J(n)
DONNEES
    n entier
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    TQ i*i ⩽ n^3 FAIRE
        j ← 1
        TQ j ⩽ n FAIRE
            j ← j+2
        FTQ
        i ← i+1
    FTQ
 FIN
```

```
ALGORITHME K(n)
DONNEES
    n entier
VARIABLES
    i,j des entiers
DEBUT
    i ← 1
    TQ i ⩽ n FAIRE
        j ← n
        TQ j ⩾ 1 FAIRE
            j ← ⌊j/2⌋
        FTQ
        i ← i+1
    FTQ
FIN
```