IN 303 – Structures de données

Sandrine Vial
sandrine.vial@uvsq.fr

Septembre 2020

Fonctionnement

- Cours : Tous les lundis matin à 08h00 sur moodle.
- TD : Par demi-groupes, une semaine sur 2 en présentiel et une semaine sur 2 à distance.
- Demi-groupes **A** : Début des TD : semaine du 21/09.
- Demi-groupes **B** : Début des TD : semaine du 28/09.
- Chaque sujet de TD sera en 2 parties :
  - Première partie présentielle
  - Deuxième partie : travail à la maison.

Organisation

- Même chargé de TD pour IN301 et IN303.
- Sujets de TD mélangeant Langage C et Algorithmique
- Tous les TDs en présentiel sont avec ordinateur
- QCM en présentiel régulièrement qui porteront sur travail présentiel et distanciel.
- un CC en présentiel en fin de semestre.

Introduction

3 niveaux d'abstraction

- **Les problèmes** décrits en langage naturel
- **Les algorithmes** décrits dans un pseudo-langage de programmation, proche du langage naturel
- **Les programmes** décrits dans un langage de programmation (C, Caml, Pascal, C++, Java, Python ...)

Niveaux de difficulté

Conceptuel : Problème

- Difficulté du problème ?
- Comment le résoudre ?
- Quelle démarche utiliser ?

Résolution du problème => Algorithme

Technique : Programme

- Comment mettre en œuvre mon algorithme sur une machine ?
- Quelles sont les ressources à ma disposition ?
- Quel est le langage le plus adapté ?

Questions relatives aux algorithmes

- Les sorties correspondent-elles à la solution de mon problème ?

  Preuve de l'algorithme
- Combien de calculs élémentaires doit-on faire pour produire la sortie ?

  Complexité (en temps et en espace) de l'algorithme

Complexité d'un algorithme

- Mesure intrinsèque de la complexité de l'algorithme indépendamment de l'implémentation.
- Permet la comparaison entre différents algorithmes pour un même problème.

Complexité d'un algorithme (2)

Somme des nombres de 1 à n

Recherche de l'algorithme avec le moins d'étapes élémentaires

Idée 1 : Utilisation d'une boucle

Algorithme 1

$i \leftarrow 1$

$som \leftarrow 0$

Tant que $i \leq n$ *Faire*

$\quad som \leftarrow som + i$

$\quad i \leftarrow i + 1$

Fin Tant que

Coût : $n \times 2$ additions

Complexité d'un algorithme (3)

Somme des nombres de 1 à n

Idée 2 : Utilisation des mathématiques

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

Algorithme 2

$som \leftarrow n + 1$

$som \leftarrow som * n$

$som \leftarrow som / 2$

Coût : 1 addition, 1 multiplication et 1 division.

Complexité d'un algorithme (4)

Différentes Mesures

- Complexité en temps
- Complexité en espace

But

« Sur toute machine, et quel que soit le langage utilisé, l'algorithme α est meilleur que l'algorithme β pour des données de grande taille. »

Complexité d'un algorithme (5)

- **Mesure élémentaire** :
  - nombre de comparaisons
  - nombre d'affectations
  - nombre d'opérations arithmétiques
  - ...
- On cherche la complexité d'un algorithme $\mathcal{A}$ en fonction d'un paramètre représentatif des entrées (taille).
- Attention : pas de système complet de règles.

Quelques règles

$cout(x)$: nbre d'op. élémentaires de l'ens. d'instructions x.

- **Séquence d'instructions** : $x_1; x_2;$

$$cout(x_1; x_2;) = cout(x_1;) + cout(x_2;)$$

Exemple

Mesure : nombre d'opérations arithmétiques.

Algorithme $\mathcal{A}$
Début
    $som \leftarrow n + 1$
    $som \leftarrow som * n$
    $som \leftarrow som/2$
Fin

$$cout(\mathcal{A}) = cout(som \leftarrow n + 1) + cout(som \leftarrow som * n) + cout(som \leftarrow som/2) = 3$$

Quelques règles

- **Les boucles simples** : *tant que condition faire* x_i ;

$$cout(boucle) = \sum_{i=1}^{n} (cout(x_i) + cout(condition))$$

Exemple

Mesure : nombre de comparaisons.

Algorithme $\mathcal{A}$
Début
❶ $i \leftarrow 1$
❷ $som \leftarrow 0$
Tant que $i \leq n$ *Faire*
❸ $som \leftarrow som + i$
❹ $i \leftarrow i + 1$
Fin Tant que
Fin

$$cout(\mathcal{A}) = cout(❶ ; ❷) + \sum_{i=1}^{n} (cout(i \leq n) + cout(❸ ; ❹)) = n$$

Quelques règles

- **Conditionnelle** : *Si condition alors* x_{vrai}; *sinon* x_{faux};

  $$cout(conditionnelle) \leq cout(condition) + \max(cout(x_{vrai}); cout(x_{faux}$$

Exemple

Mesure : nombre d'affectations.

Algorithme $\mathcal{A}$
Début
❶ $u \leftarrow 0$
Si $i \mod 2 = 0$ *Alors*
❷ $u \leftarrow i/2$
Sinon
❸ $u \leftarrow i - 1$
❹ $u \leftarrow u/2$
Fin Si
Fin

$$cout(\mathcal{A}) \leq cout(❶) + cout(i \mod 2 = 0) + \max(cout(❷); cout(❸;❹)) = 3$$

Grandeurs

- Caractérisation du comportement d'un algorithme $\mathcal{A}$ sur l'ensemble des données D_n de taille n.
- $Cout_{\mathcal{A}}(d)$: coût de l'algorithme $\mathcal{A}$ sur la donnée d.

- **Complexité dans le meilleur cas :**

$$Min_{\mathcal{A}}(n) = \min\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

Grandeurs

- Caractérisation du comportement d'un algorithme $\mathcal{A}$ sur l'ensemble des données D_n de taille n.
- $Cout_{\mathcal{A}}(d)$: coût de l'algorithme $\mathcal{A}$ sur la donnée d.

- **Complexité dans le meilleur cas :**

$$Min_{\mathcal{A}}(n) = \min\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

- **Complexité dans le pire cas :**

$$Max_{\mathcal{A}}(n) = \max\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

Grandeurs

- Caractérisation du comportement d'un algorithme $\mathcal{A}$ sur l'ensemble des données D_n de taille n.
- $Cout_{\mathcal{A}}(d)$: coût de l'algorithme $\mathcal{A}$ sur la donnée d.

- **Complexité dans le meilleur cas :**

$$Min_{\mathcal{A}}(n) = \min\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

- **Complexité dans le pire cas :**

$$Max_{\mathcal{A}}(n) = \max\{Cout_{\mathcal{A}}(d), d \in D_n\}$$

- **Complexité en moyenne :**

$$Moy_{\mathcal{A}}(n) = \sum_{d \in D_n} p(d) \times Cout_{\mathcal{A}}(d)$$

Remarques

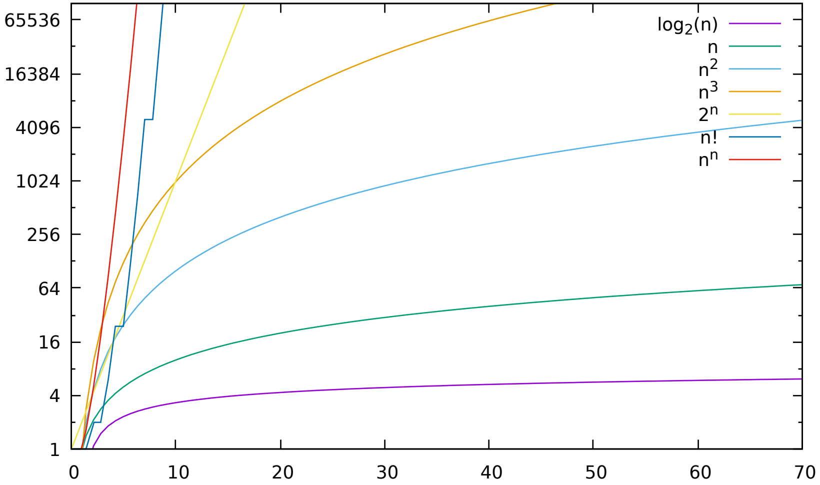
- Comportement à l'extrême pour les complexités dans le meilleur et le pire cas.
- Complexité en moyenne : comportement de l'algorithme en général avec un modèle probabiliste sur les données.

$$Min_{\mathcal{A}}(n) \leq Moy_{\mathcal{A}}(n) \leq Max_{\mathcal{A}}(n)$$

Ordres de grandeurs

- Une approximation de la fonction de complexité est suffisante.
- Utilisation d'une échelle de comparaison avec les fonctions $n^n, n!, 2^n, n^3, n^2, n \log n, n, \log n$

Ordres de grandeurs



Notations O

O « Borne Supérieure »

Soient f et $g : \mathbb{N} \rightarrow \mathbb{R}_+$: $f = O(g)$ ssi $\exists c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N}$ tels que :

$$\forall n > n_0, f(n) \leq c \times g(n)$$

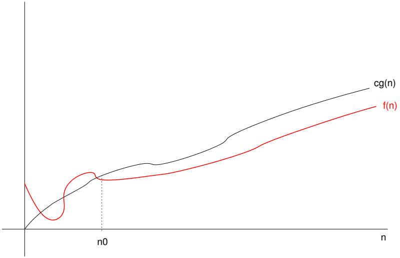


Figure – $f(n) = O(g(n))$

La complexité de l'algorithme $\mathcal{A}$ est en $O(n^2)$

Notations Ω

Ω « Borne Inférieure »

Soient f et g : $\mathbb{N} \rightarrow \mathbb{R}_+$: $f = \Omega(g)$ ssi $\exists c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N}$ tels que :

$$\forall n > n_0, 0 \leq c \times g(n) \leq f(n)$$

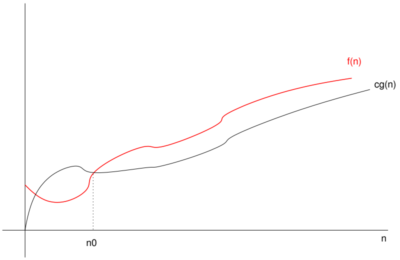


Figure – $f(n) = \Omega(g(n))$

La complexité de l'algorithme $\mathcal{A}$ est en $\Omega(\log_2 n)$

Notation Θ

Θ

$$f = \Theta(g) \text{ ssi } f = O(g) \text{ et } f = \Omega(g)$$

$\exists c, d \in \mathbb{R}_+, \exists n_0 \in \mathbb{N}$ tels que :

$$\forall n > n_0, d \times g(n) \leq f(n) \leq c \times g(n)$$

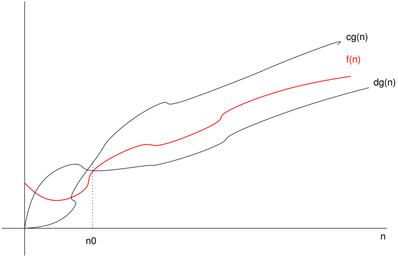


Figure – $f(n) = \Theta(g(n))$

La complexité de l'algorithme $\mathcal{A}$ est en $\Theta(n^3)$

Exemples

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

$$2n = O(n)$$

$$2n = \Theta(n)$$

$$2n \neq \Theta(n^2)$$

Un algorithme ...

- de complexité $O(1)$ effectue un nombre constant d'opérations
- de complexité $O(n)$ est un algorithme linéaire
- de complexité $O(n^k)$ est un algorithme polynomial

Quelques chiffres ...

		Complexité						
		1	$\log n$	n	$n \log n$	n^2	n^3	2^n
Taille des données	$n = 10^2$	$1\mu s$	$6.6\mu s$	$0.1ms$	$0.6ms$	$10ms$	$1s$	$4.10^{16}a$
	$n = 10^3$	$1\mu s$	$9.9\mu s$	$1ms$	$9.9ms$	$1s$	$16.6min$	∞
	$n = 10^4$	$1\mu s$	$13.2\mu s$	$10ms$	$0.1s$	$100s$	$11.5j$	∞
	$n = 10^5$	$1\mu s$	$16.6\mu s$	$0.1s$	$1.66s$	2.7h	$31.7a$	∞
	$n = 10^6$	$1\mu s$	$19.9\mu s$	$1s$	$19.9s$	$11.5j$	$31.7 * 10^3 a$	∞

Temps d'exécution en fonction de la complexité d'un algorithme et de la taille des données.

$\infty =$ « $> 10^{25}$ années »

Nombre d'opérations par seconde $= 10^6$

Quelques chiffres ...

		Complexité						
		1	$\log n$	n	$n \log n$	n^2	n^3	2^n
Taille des données	$n = 10^2$	$1ns$	$6.6ns$	$0.1\mu s$	$6.6\mu s$	$10\mu s$	$0.001s$	$4.10^{13}a$
	$n = 10^3$	$1ns$	$9.9ns$	$1\mu s$	$9.9\mu s$	$0.001s$	$1s$	∞
	$n = 10^4$	$1ns$	$13.2ns$	$10\mu s$	$1.32ms$	$0.1s$	$1min\ 40s$	∞
	$n = 10^5$	$1ns$	$16.6ns$	$1ms$	$1.66ms$	$10s$	$11j\ 13h\ 46min\ 40s$	∞
	$n = 10^6$	$1ns$	$19.9ns$	$0.001s$	$0.019s$	$1min\ 40s$	$> 31a$	∞

Temps d'exécution en fonction de la complexité d'un algorithme et de la taille des données.

$\infty =$ « $> 10^{25}$ années »

Nombre d'opérations par seconde $= 10^9$

Quelques chiffres ...

		Complexité						
		1	$\log n$	n	$n \log n$	n^2	n^3	2^n
Taille des données	$n = 10^2$	$1ps$	6.64 ps	0.1 ns	0.66 ns	$0.01\mu s$	$1\mu s$	3.99 10^{10} a
	$n = 10^3$	$1ps$	9.96 ps	1 ns	9.96 ns	$1\mu s$	$0.001s$	∞
	$n = 10^4$	$1ps$	13.28 ps	10 ns	132.87 ns	10 ms	1 s	∞
	$n = 10^5$	$1ps$	16.6 ps	$0.1\mu s$	$1.6\mu s$	0.01 s	$>$ 16 min	∞
	$n = 10^6$	$1ps$	19.93 ps	$1\mu s$	$19.93\mu s$	$1s$	$>$ 11 jours	∞

Temps d'exécution en fonction de la complexité d'un algorithme et de la taille des données.

$\infty =$ « $> 10^{25}$ années »

Nombre d'opérations par seconde $= 10^{12}$