Algorithmique Avancée : Séance 1

Frédéric A. Hayek

Jeudi 26 septembre 2024

Overview

Algorithmique

- Algorithme ?

1: $i \leftarrow 10$
2: **if** $i \geq 5$ **then**
3: $\quad i \leftarrow i - 1$
4: **else**
5: $\quad$ **if** $i \leq 3$ **then**
6: $\quad\quad i \leftarrow i + 2$
7: $\quad$ **end if**
8: **end if**


```c
#include<stdio.h>
int main ()
{
  printf("1");
  return 0;
}
```

Algorithmique

- Un algorithme est une suite d'instructions

Algorithmique

- Un algorithme est une suite d'instructions
- Un programme est une instanciation d'un algorithme

Algorithmique

- Un algorithme est une suite d'instructions
- Un programme est une instanciation d'un algorithme
- L'algorithmique est l'étude des algorithmes en termes de :
  - temps
  - espace
  - terminaison
  - correction

Algorithmique

- Un algorithme est une suite d'instructions
- Un programme est une instanciation d'un algorithme
- L'algorithmique est l'étude des algorithmes en termes de :
  - temps
  - espace
  - terminaison
  - correction
- Fun Fact : Muhammad ibn Musa Al-Khwârizmî

Algorithmique

Travail Pratique

- Par binôme, écrire un programme `python` qui introduit n fois le nombre 1 dans une liste initialement vide, avec chacune des fonctions suivantes :
  `insert(0, 1)`, `insert(-1, 1)`, `append(1)`
- Lancer et chronometrer le programme avec les valeurs de n suivantes : $100'000, 200'000, 300'000, 400'000, 500'000, 10'000'000, 20'000'000, 30'000'000, 40'000'000, 50'000'000$
- Comparer les résultats
- Faire des hypothèses

Le Fonctionnement des listes en python

- Explications au tableau

Complexité Temporelle

Exercice 1

Pour `append(1)` et `insert(-1, 1)` (pseudocode ci-dessous), calculer le nombre d'opérations (affectations) en fonction de n

Require: $\ell[n]$

1: **for** $i \in 0, \ldots, n-1$ **do**
2: $\quad \ell[i] \leftarrow 1$

Complexité Temporelle

Exercice 1

Pour `append(1)` et `insert(-1, 1)` (pseudocode ci-dessous), calculer le nombre d'opérations (affectations) en fonction de n

Require: $\ell[n]$

1: **for** $i \in 0, \ldots, n-1$ **do**
2: $\quad \ell[i] \leftarrow 1$

Solution

Le nombre d'affectations est : n

Complexité Temporelle

Exercice 2

Pour `insert(0, 1)` (pseudocode ci-dessous), calculer le nombre d'opérations en fonction de n

Require: $\ell[n]$

1: **for** $i \in 0, \ldots, n-1$ **do**
2: **for** $j \in i-1, i-2 \ldots, 0$ **do**
3: $\ell[j+1] \leftarrow \ell[j]$
4: $\ell[0] \leftarrow 1$

Complexité Temporelle

Exercice 2

Pour `insert(0, 1)` (pseudocode ci-dessous), calculer le nombre d'opérations en fonction de n

Require: $\ell[n]$

1: **for** $i \in 0, \ldots, n-1$ **do**
2: **for** $j \in i-1, i-2 \ldots, 0$ **do**
3: $\ell[j+1] \leftarrow \ell[j]$
4: $\ell[0] \leftarrow 1$

Solution

Le nombre d'affectations est : $\frac{n^2+n}{2}$

Pseudocode

- Pas de syntaxe rigide (contrairement aux langages de programmation)
  - Affectation : $a \leftarrow 0, a < -0, a = 0$
  - Test égalité : $a == b, a \stackrel{?}{=} b, a = b$
- Différents niveaux d'abstraction
  - Haut niveau : utilisation de `append` acceptée (proche de `python`)
  - Bas niveau : il faut tout expliciter (proche du `C`)

Complexité Temporelle

Exercice 3

Écrire un algorithme qui prend en paramètre un tableau à deux dimensions de taille $n \times n$ et qui calcule la somme des éléments du tableau. Donner le nombre d'opérations de l'algorithme.

Complexité Temporelle

Exercice 3

Écrire un algorithme qui prend en paramètre un tableau à deux dimensions de taille $n \times n$ et qui calcule la somme des éléments du tableau. Donner le nombre d'opérations de l'algorithme.

Solution

Require: $tab[n][n]$

1: $somme \leftarrow 0$
2: **for** $i \in 0, \ldots, n-1$ **do**
3: **for** $j \in 0, \ldots, n-1$ **do**
4: $somme \leftarrow somme + tab[i][j]$

Complexité Temporelle

Exercice 3

Écrire un algorithme qui prend en paramètre un tableau à deux dimensions de taille $n \times n$ et qui calcule la somme des éléments du tableau. Donner le nombre d'opérations de l'algorithme.

Solution

Require: $tab[n][n]$

1: $somme \leftarrow 0$
2: **for** $i \in 0, \ldots, n-1$ **do**
3: **for** $j \in 0, \ldots, n-1$ **do**
4: $somme \leftarrow somme + tab[i][j]$

Le nombre d'opérations est : n^2 additions ET $n^2 + 1$ affectations !

Complexité Temporelle

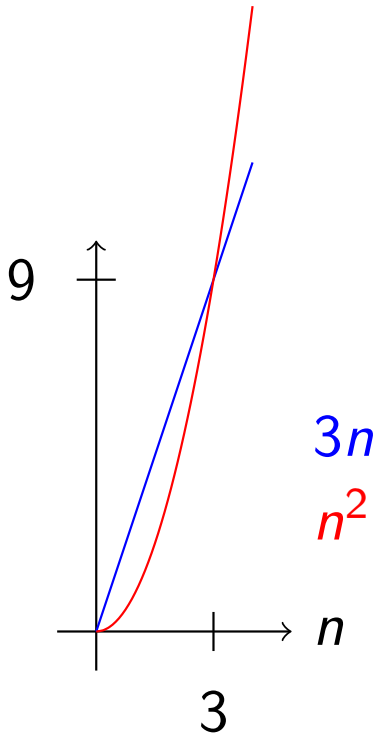
Exercice 4

Supposons deux algorithmes A et B, qui exécutent respectivement $3n$ opérations et n^2 opérations. Lequel est le plus efficace ?

Complexité Temporelle

Exercice 4

Supposons deux algorithmes A et B, qui exécutent respectivement $3n$ opérations et n^2 opérations. Lequel est le plus efficace ?



Complexité Temporelle

Exercice 5

Soit un algorithme qui fait n insertions dans une liste initialement vide. Quel est le nombre d'opérations exécutées par l'algorithme ?

Complexité Temporelle

Exercice 5

Soit un algorithme qui fait n insertions dans une liste initialement vide. Quel est le nombre d'opérations exécutées par l'algorithme ?

Solution

- L'algorithme fait n insertions.
  Cela peut se traduire soit par n affectations, soit par $\frac{n^2+n}{2}$ affectations.

Complexité Temporelle

Exercice 5

Soit un algorithme qui fait n insertions dans une liste initialement vide. Quel est le nombre d'opérations exécutées par l'algorithme ?

Solution

- L'algorithme fait n insertions.
  Cela peut se traduire soit par n affectations, soit par $\frac{n^2+n}{2}$ affectations.

Remarque

La plupart du temps, on comptera en terme d'opérations élémentaires :
addition, multiplication, division, test

Complexité Temporelle

Exercice 5

Soit un algorithme qui fait n insertions dans une liste initialement vide. Quel est le nombre d'opérations exécutées par l'algorithme ?

Solution

- L'algorithme fait n insertions.
  Cela peut se traduire soit par n affectations, soit par $\frac{n^2+n}{2}$ affectations.

Remarque

La plupart du temps, on comptera en terme d'opérations élémentaires :
addition, multiplication, division, test, incrémentation...

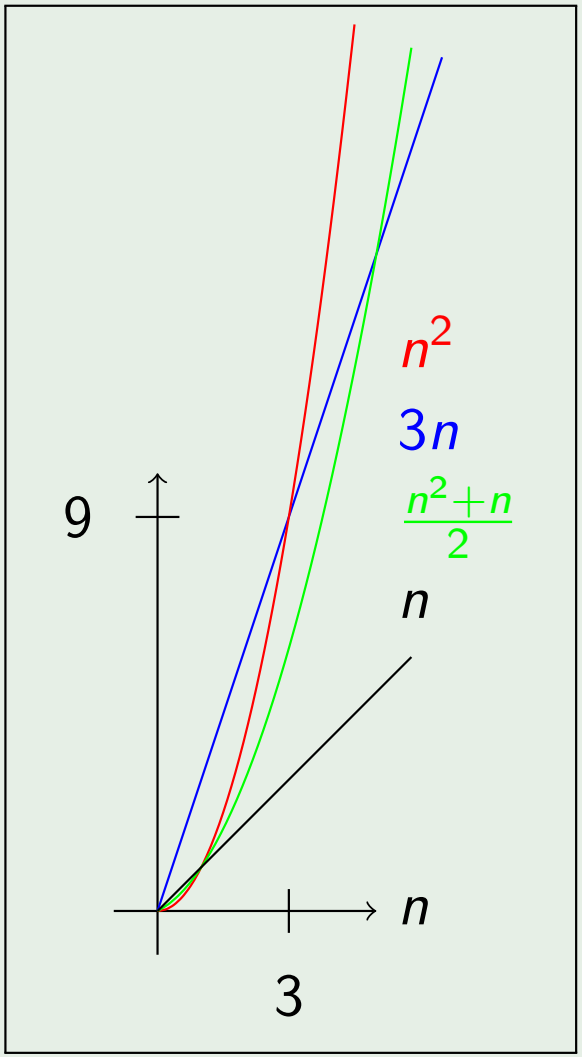
Complexité Temporelle

Remarque

- En pratique, la plupart du temps, on tiendra compte que du pire cas, ainsi que du comportement asymptotique.
- Ce qui importe c'est de combien augmentera le temps d'éxécution en fonction de l'augmentation de la taille de l'entrée

Complexité Temporelle

Exemple



n	$3n$	n^2	$\frac{n^2+n}{2}$
0	0	0	0
1	3	1	1
2	6	4	3
3	9	9	6
4	12	16	10
5	15	25	15
6	18	36	21

Complexité Temporelle

Gros Flemmards

$3n$ se comporte "comme" n, de façon linéaire.
$\frac{n^2+n}{2}$ se comporte "comme" n^2, de façon quadratique.

Complexité Temporelle

Gros Flemmards

$3n$ se comporte "comme" n, de façon linéaire.
$\frac{n^2+n}{2}$ se comporte "comme" n^2, de façon quadratique.

Exemple

$\frac{3x^3-150x^2+\pi x-5}{\frac{x^4}{1000}-12x^3-9x^2-2022x}$ se comporte "comme" $\frac{1}{x}$ quand $x \to \infty$.
Calculer :

$$\lim_{x \to +\infty} \frac{3x^3 - 150x^2 + \pi x - 5}{\frac{x^4}{1000} - 12x^3 - 9x^2 - 2022x}$$

Notation

Définition : domination

$f(x) = O(g(x))$ quand $x \to \infty$ si
$\exists M \in \mathbb{R}$ et $\exists x_0 \in \mathbb{R}$ tels que $\forall x \geq x_0, |f(x)| \leq M \cdot g(x)$

Notation

Définition : domination

$f(x) = O(g(x))$ quand $x \to \infty$ si
$\exists M \in \mathbb{R}$ et $\exists x_0 \in \mathbb{R}$ tels que $\forall x \geq x_0, |f(x)| \leq M \cdot g(x)$

Exemple

$2n = O(n)$
Preuve : soit $M = 2$, on a : $2n \leq M \cdot n$

Notation

Exemple

$\frac{x^2+x}{2} = O(x^2)$

Preuve :

$$\lim_{x\to+\infty} \frac{\frac{x^2+x}{2}}{x^2} = \lim_{x\to+\infty} \frac{x^2\left(\frac{1}{2}+\frac{1}{2x}\right)}{x^2} = \lim_{x\to\infty} \frac{1}{2}+\frac{1}{2x} = \frac{1}{2}$$

Donc $\forall \epsilon > 0, \exists x_0$ t. q. $x \geq x_0 \implies \frac{\frac{x^2+x}{2}}{x^2} \in \left[\frac{1}{2}-\epsilon, \frac{1}{2}+\epsilon\right]$

Soit $\epsilon = \frac{1}{2}$, soit x_0 t. q. $x \geq x_0 \implies \frac{\frac{x^2+x}{2}}{x^2} \in \left[\frac{1}{2}-\frac{1}{2}, \frac{1}{2}+\frac{1}{2}\right]$

Soit $M = 1$. On a $x \geq x_0 \implies \frac{\frac{x^2+x}{2}}{x^2} \leq M$.

Notation

Exercice 6 – Théorème

Prouver que pour f et g deux fonctions positives :

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = k \in \mathbb{R} \implies f(x) = O(g(x))$$

Exercice 7

Prouver que $n^2 = O(\frac{n^2+n}{2})$

Notation

Définition

$f(x) = \Omega(g(x))$ si
$g(x) = O(f(x))$

Notation

Définition

$f(x) = \Omega(g(x))$ si
$g(x) = O(f(x))$

Définition

$f(x) = \Theta(g(x))$ si
$f(x) = O(g(x))$ et $g(x) = O(f(x))$

Notation

Définition

$f(x) = \Omega(g(x))$ si
$g(x) = O(f(x))$

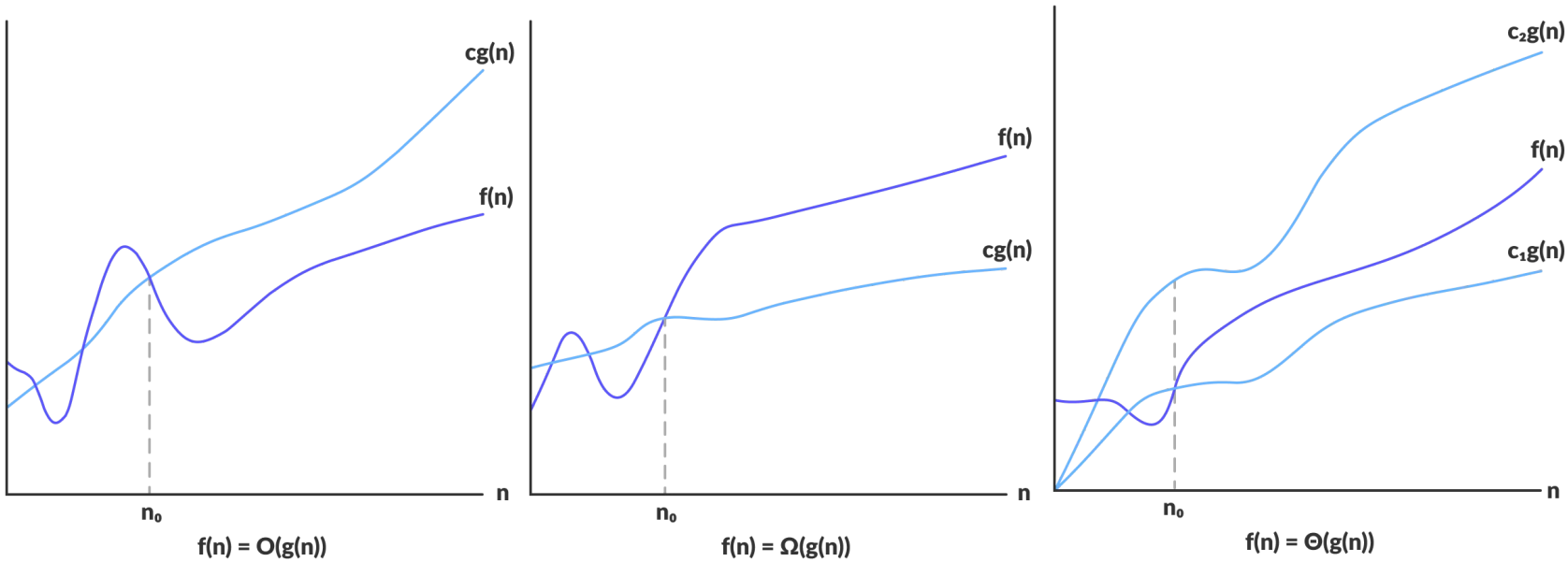
Définition

$f(x) = \Theta(g(x))$ si
$f(x) = O(g(x))$ et $g(x) = O(f(x))$

Propriété

$f(x) = \Theta(g(x)) \iff g(x) = \Theta(f(x))$

Notation



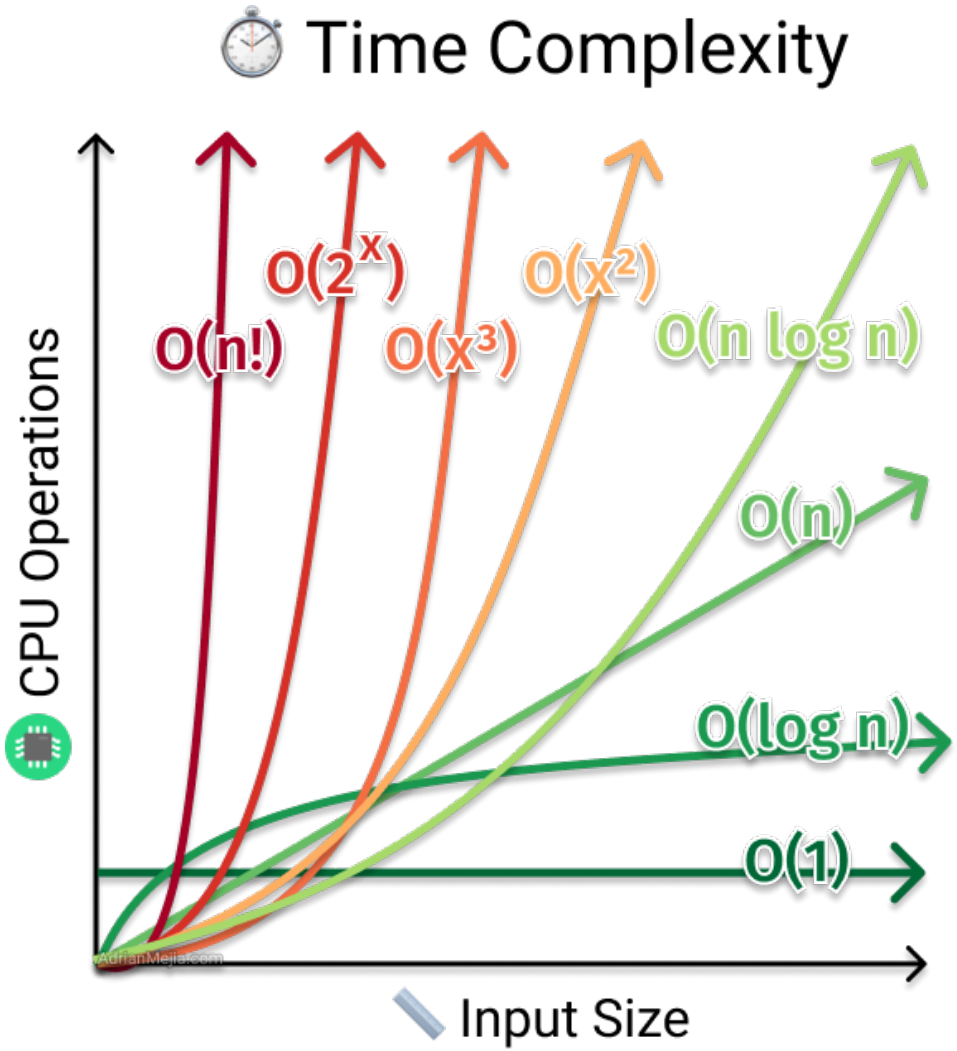
Exercices

Exercice 8

Comparer avec les notations $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ les pairs de fonctions suivantes :

a. $1, 1000$
b. $1, n$
c. $\log\left(n^2\right), \log(n)$
d. n, e^n
e. $2^n, 3^n$
f. $e^n, n!$

Notation


Time Complexity
CPU Operations
Input Size
O(n!)
O(2^x)
O(x^3)
O(x^2)
O(n log n)
O(n)
O(log n)
O(1)
AdrianMejia.com

Tri par insertion

Exercice 9

Écrire le pseudocode du tri par insertion, et calculer le nombre d'opérations au meilleur des cas et le nombre d'opérations au pire des cas.