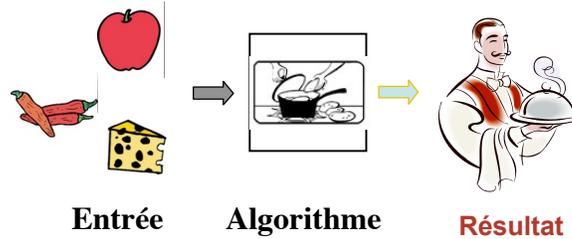
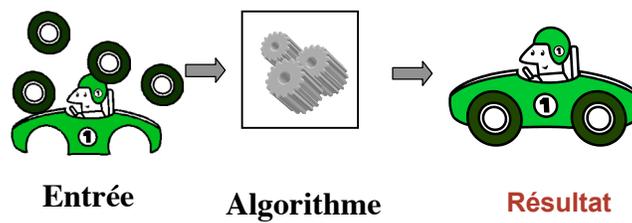


# Analyse d'algorithmes

---



Un **algorithme** est une suite d'instructions qui sert à résoudre un problème donné dans un temps fini.



Un **algorithme** est une suite d'instructions qui sert à résoudre un problème donné dans un temps fini.

**Analyser un algorithme = évaluer son efficacité**

## Effacité ?

---

Temps d'exécution ...

Espace mémoire occupé ...

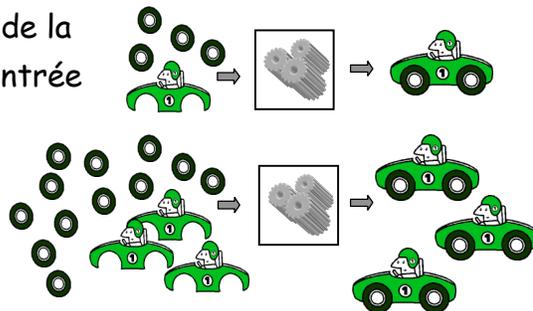
Qualité du résultat ...

Simplicité ....

## Temps d'exécution d'un algorithme

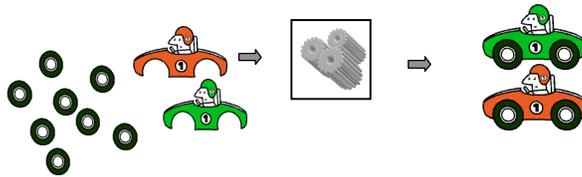
---

Le temps d'exécution d'un algorithme dépend de la **taille** des données d'entrée



## Temps d'exécution d'un algorithme

Il dépend aussi de la nature des données à traiter (des entrées différentes peuvent avoir des temps d'exécution différents)

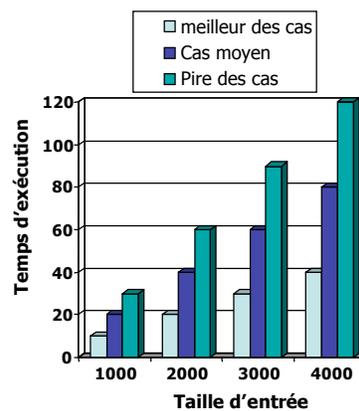


CSI2510 - Paola Flocchini

5

## Temps d'exécution d'un algorithme

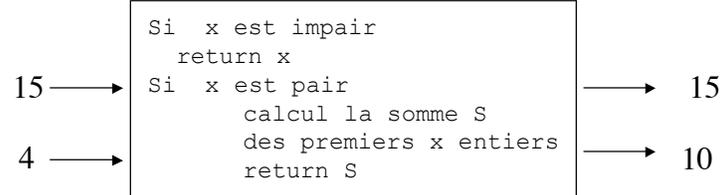
- Le temps d'exécution d'un algorithme dépend de la taille des données d'entrée
- Il dépend aussi des données (diverses exécutions peuvent avoir temps d'exécutions diverses)



CSI2510 - Paola Flocchini

6

## Exemple

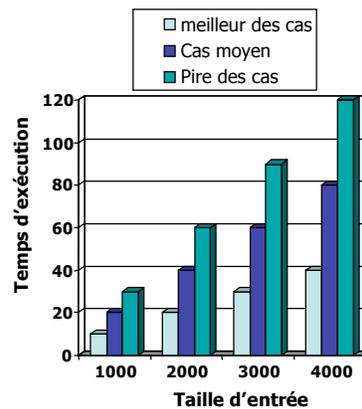


CSI2510 - Paola Flocchini

7

## Temps d'exécution d'un algorithme

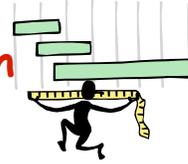
- Trouver le **cas moyen** peut être difficile
- On se concentre souvent sur le **pire des cas**.
  - plus facile a analyser
  - d'importance cruciale dans certaines applications (par ex. contrôle aérien, chirurgie, gestion de réseau)



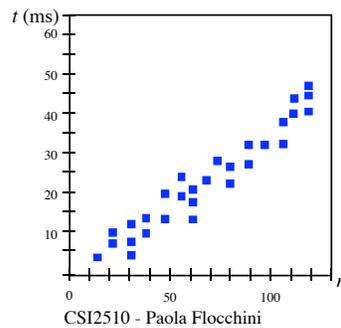
CSI2510 - Paola Flocchini

8

## Mesurer le temps d'exécution



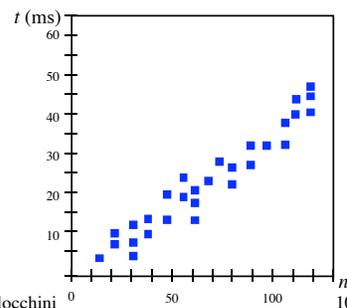
- Comment on devrait mesurer le temps d'exécution d'un **algorithme**?
- Approche 1: Etude Expérimental



9

### • Étude expérimentale:

- Implémenter l'algorithme
- Exécuter le programme avec des ensembles de données de taille et de contenu variés.
- Mesurer précisément le temps d'exécution pour chaque cas



## Etudes expérimentales

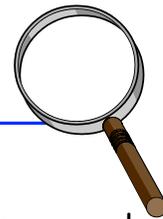
---

### Limitations des mesures expérimentales:

- Il est nécessaire de **implémenter**
- Lors des tests, l'ensemble des données d'entrée est réduit et ne couvre pas la totalité des cas possibles
- Afin de comparer deux algorithmes, les mêmes **environnements matériel et logiciel** devraient être utilisés.

## Analyse Théorique

---



Nous avons besoin d'une **méthodologie générale** pour analyser le temps d'exécution d'algorithmes qui:

- Utilise une **description de haut niveau** de l'algorithme (indépendant de l'implémentation)
- Caractérise le temps d'exécution comme **une fonction de la taille des données d'entrée**
- Considère tous les entrées
- Est **indépendant des environnements matériels et logiciels**

## Analyse des algorithmes

---

### Opérations primitives:

opérations de bas niveau qui sont indépendantes du langage de programmation, par exemple:

- Appel et retour d'une méthode
- Effectuer une opération arithmétique
- Comparer deux nombres, etc.
- Affectation d'une variable...

## Analyse des algorithmes

---

### Opérations primitives:

opérations de bas niveau qui sont indépendantes du langage de programmation, par exemple:

- Appel et retour d'une méthode
- Effectuer une opération arithmétique
- Comparer deux nombres, etc.
- Affectation d'une variable...

En observant le pseudo-code d'un algorithme on peut compter le nombre d'opérations primitives exécutées par cet algorithme et par la suite analyser son temps d'exécution et son efficacité.

## Pseudo-code

---

- Un mélange de langage naturel et de concepts de programmation de haut niveau qui décrit les idées générales derrière la réalisation générique d'une structure de données ou d'un algorithme.
- Le **pseudo-code** est une description d'algorithme qui est plus structurée que la prose ordinaire mais moins formelle qu'un langage de programmation.
- Le pseudo-code est notre notation de choix pour la description d'algorithmes.
- Cependant, le pseudo-code cache plusieurs problèmes liés à la conception de programmes.

CSI2510 - Paola Flocchini

15

## Analyse d'algorithmes - Exemple1

---

**Trouver l'élément maximal d'un tableau d'entiers**

**Algorithme TabMax(A, n):**

Entrée: un tableau A contenant n entiers

Sortie: L'élément maximal de A

```
currentMax ← A[0]
for i ← 1 to n -1 do
{
if currentMax < A[i] then
    currentMax ← A[i]
}
return currentMax
```

CSI2510 - Paola Flocchini

16

## Analyse d'algorithmes - Exemple1

A

5	20	4	7	6	2	3	8	1	9
---	----	---	---	---	---	---	---	---	---

currentMax

```
currentMax ← A[0]
for i ← 1 to n -1 do
{
  if currentMax < A[i] then
    currentMax ← A[i]
}
return currentMax
```

CSI2510 - Paola Flocchini

17

## Analyse d'algorithmes - Exemple1

A

5	20	4	7	6	2	3	8	1	9
---	----	---	---	---	---	---	---	---	---

↑  
currentMax

```
currentMax ← A[0]
for i ← 1 to n -1 do
{
  if currentMax < A[i] then
    currentMax ← A[i]
}
return currentMax
```

CSI2510 - Paola Flocchini

18

## Analyse d'algorithmes - Exemple1

A

5	20	4	7	6	2	3	8	1	9
---	----	---	---	---	---	---	---	---	---

currentMax 5

```
currentMax ← A[0]
for i ← 1 to n-1 do
{
  if currentMax < A[i] then
    currentMax ← A[i]
}
return currentMax
```

CSI2510 - Paola Flocchini

19

## Analyse d'algorithmes - Exemple1

Quelles sont les opérations primitives à compter ?

A

5	20	4	7	6	2	3	8	1	9
---	----	---	---	---	---	---	---	---	---

currentMax 20

Comparaisons  
Affectations à currentMax

```
currentMax ← A[0]
for i ← 1 to n-1 do
{
  if currentMax < A[i] then
    currentMax ← A[i]
}
return currentMax
```

CSI2510 - Paola Flocchini

20

## Analyse d'algorithmes - Exemple1

Meilleur cas

A

20	2	3	4	5	6	7	8	9	1
----	---	---	---	---	---	---	---	---	---

```
currentMax ← A[0] ← 1 affectation
for i ← 1 to n-1 do
{
  if currentMax < A[i] then ← n-1 comparaisons
    currentMax ← A[i] ← 0 affectation
}
return currentMax
```

CSI2510 - Paola Flocchini

21

## Analyse d'algorithmes - Exemple1

Pire cas

A

1	2	3	4	5	6	7	8	9	20
---	---	---	---	---	---	---	---	---	----

```
currentMax ← A[0] ← 1 affectation
for i ← 1 to n-1 do
{
  if currentMax < A[i] then ← n-1 comparaisons
    currentMax ← A[i] ← n-1 affectations
}
return currentMax
```

CSI2510 - Paola Flocchini

22

## Analyse d'algorithmes - Exemple1

Algorithme de recherche du max  $\text{TabMax}(A,n)$

Meilleur cas  
1 affectation+(n-1) comparaisons

Pire cas  
n affectations+(n-1) comparaisons

## Exemple2:

Chercher l'indice d'une valeur  
dans un tableau A de taille sizeA

Opérations: affectations et verifications

```
i ← 0 -----> 1 affectation
while (A[i] ≠ element) -----> sizeA verifications & affectation
    i ← i+1                      (dans le cas pire)
return i
```

---

## Notation asymptotique

---

CSI2510 - Paola Flocchini

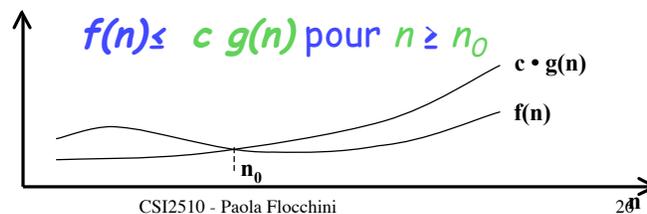
25

## Big-Oh (Grand Oh)

### Limite supérieure

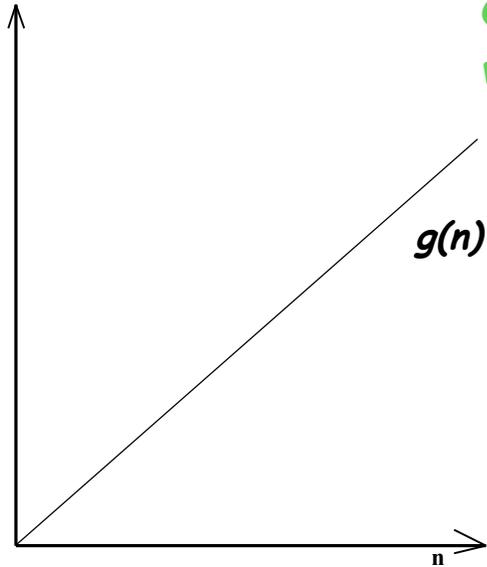
---

- Soit les fonctions  $f(n)$  et  $g(n)$ , nous disons que  $f(n)$  est  $O(g(n))$  (ou  $f(n) = O(g(n))$  ou  $f(n) \in O(g(n))$ )
- si et seulement si il y a des constantes positives  $c$  et  $n_0$  tel que



*Quest-ce que ça  
veut dire c  $g(n)$  ?*

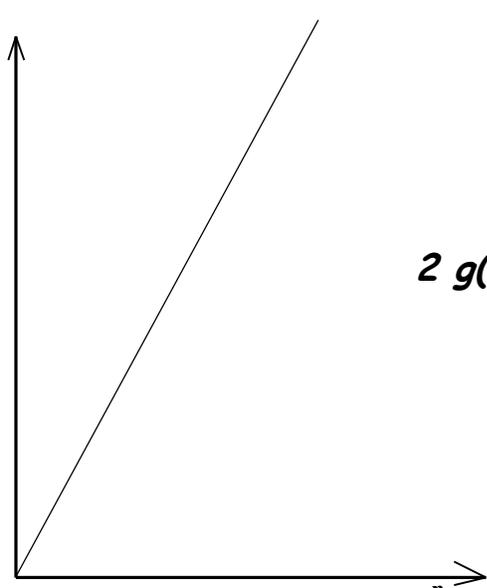
Example:



$g(n) = n$

CSI2510 - Paola Flocchini

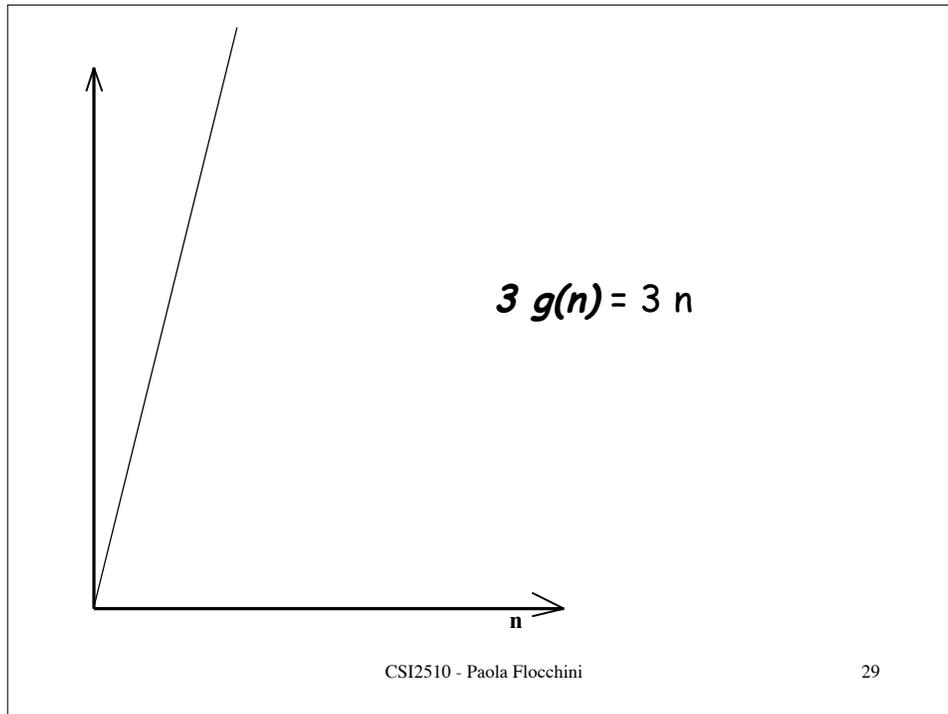
27



$2g(n) = 2n$

CSI2510 - Paola Flocchini

28



## Big-Oh (Grand Oh)

### Limite supérieure

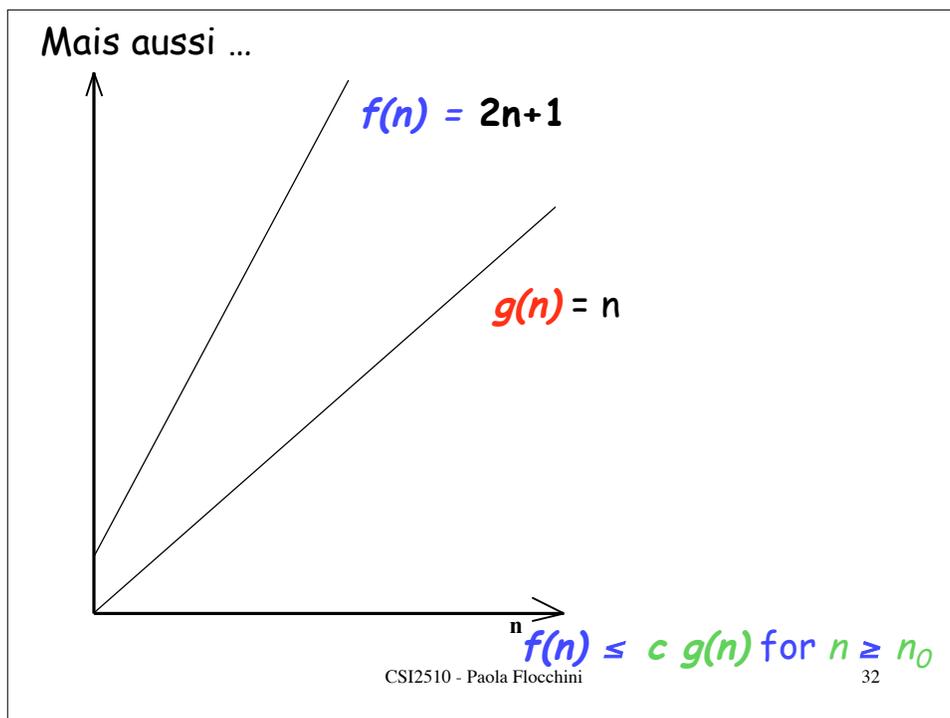
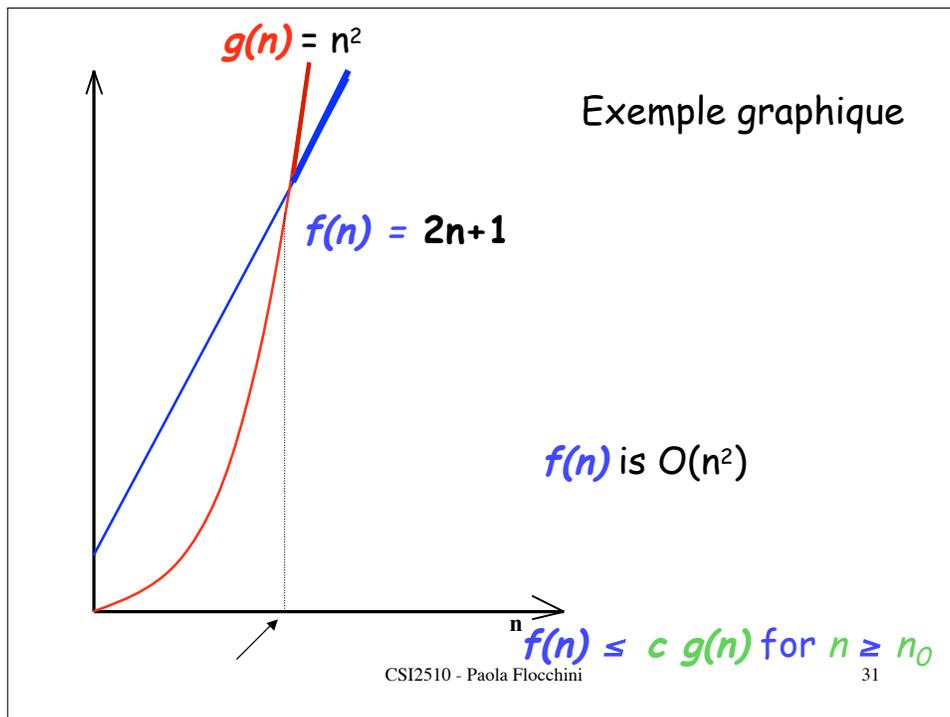
---

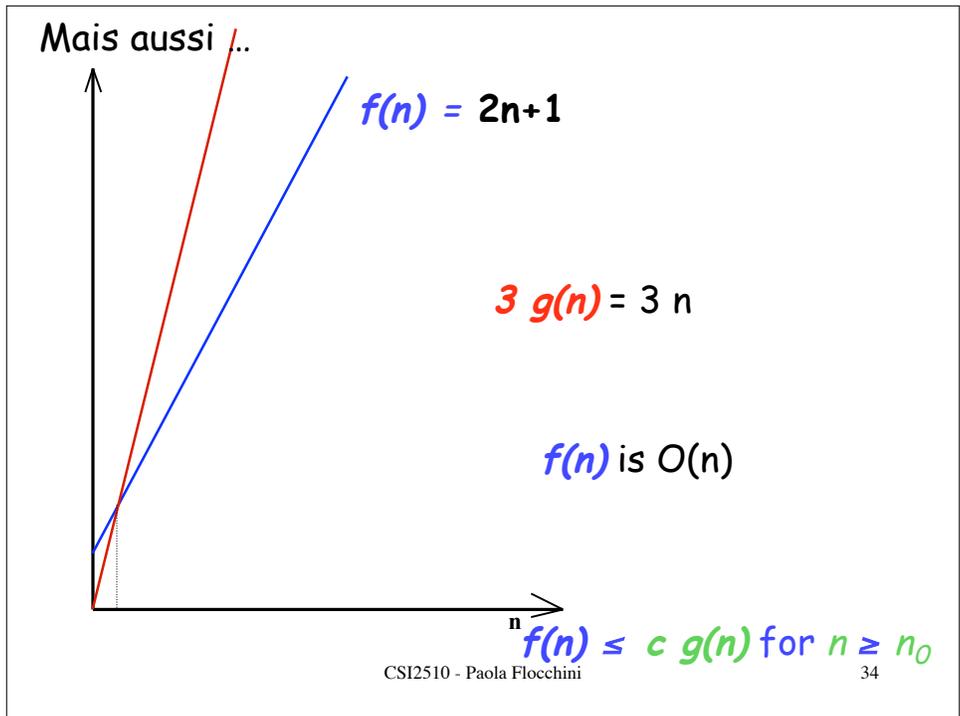
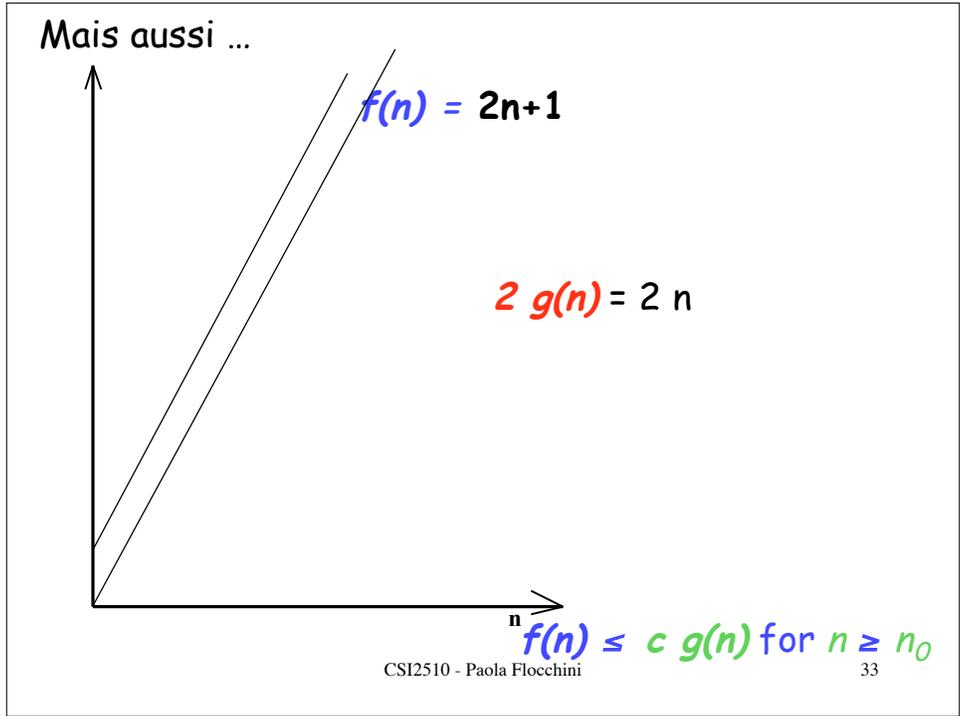
- Soit les fonctions  $f(n)$  et  $g(n)$ , nous disons que  $f(n)$  est  $O(g(n))$  (ou  $f(n) = O(g(n))$  ou  $f(n) \in O(g(n))$ )
- si et seulement si il y a des constantes positives  $c$  et  $n_0$  tel que

$f(n) \leq c g(n)$  pour  $n \geq n_0$

c • g(n)  
f(n)  
n<sub>0</sub>

CSI2510 - Paola Flocchini 30

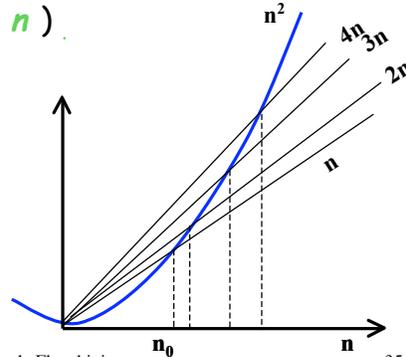




Mais ...

$n^2$  n'est pas  $O(n)$  parce que nous ne pouvons pas trouver  $c$  et  $n_0$  tel que  $n^2 \leq cn$  pour  $n \geq n_0$

( n'importe comment grand un  $c$  est choisi il y a un  $n$  assez grand tel que  $n^2 > cn$  ).



CSI2510 - Paola Flocchini

35

Exemple

Limite supérieure -  $O$  (Grand  $O$  ou Big -  $O_h$ )

Prouver que  $f(n) = 60n^2 + 5n + 1$  est  $O(n^2)$

Il faut trouver un nombre  $c$  et un nombre  $n_0$  tel que:

$$60n^2 + 5n + 1 \leq c n^2 \text{ pour tout } n \geq n_0$$

$$5n \leq 5n^2 \text{ pour tout } n \geq 1$$

$$1 \leq n^2 \text{ pour tout } n \geq 1$$



$$f(n) \leq 60n^2 + 5n^2 + n^2 \text{ pour tout } n \geq 1$$

$$f(n) \leq 66n^2 \text{ pour tout } n \geq 1$$

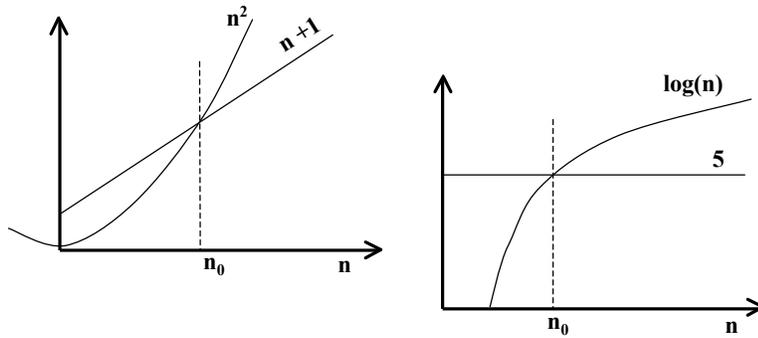
$$c = 66 \text{ et } n_0 = 1 \Rightarrow f(n) = O(n^2)$$

CSI2510 - Paola Flocchini

36

Limite supérieure - O (Grand O ou Big - Oh)

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) \dots$  *mémorisez !!*



CSI2510 - Paola Flocchini

37

$n =$	2	16	256	1024
$\log \log n$	0	2	3	3.32
$\log n$	1	4	8	10
$n$	2	16	256	1024
$n \log n$	2	64	448	10 200
$n^2$	4	256	65 500	$1.05 * 10^6$
$n^3$	8	4 100	16 800 800	$1.07 * 10^9$
$2^n$	4	35 500	$11.7 * 10^6$	$1.80 * 10^{308}$

CSI2510 - Paola Flocchini

38

**Théorème:**

**Limite supérieure - O (Grand O ou Big - Oh)**

Si  $g(n)$  est  $O(f(n))$ , alors pour n'importe quelle constante  $c > 0$   $g(n)$  est aussi  $O(c f(n))$

**Théorème:**

si  $f_1(n) = O(g_1(n))$  et  $f_2(n) = O(g_2(n))$  alors  
 $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$

Ex 1:

$$\begin{aligned} 2n^3 + 3n^2 &= O(\max(2n^3, 3n^2)) \\ &= O(2n^3) = O(n^3) \end{aligned}$$

Ex 2:

$$\begin{aligned} n^2 + 3 \log n - 7 &= O(\max(n^2, 3 \log n - 7)) \\ &= O(n^2) \end{aligned}$$

**Limite supérieure - O (Grand O ou Big - Oh)**

**Faire l'approximation la plus proche possible; utiliser la plus petite classe possible:**

Ex.: Il est correct de dire que  $5n-3$  est  $O(n^3)$  mais la meilleure formulation est de dire  $5n-3$  est  $O(n)$

**Utiliser l'expression la plus simple de la classe :**

Ex.: Dire  $10n+15$  est  $O(n)$  au lieu de  $10n+15$  est  $O(10n)$

## Limite supérieure - O (Grand O ou Big - Oh)

Laisser tomber les termes d'ordre inférieur ainsi que les coefficients

Ex.1:  $7n-3$  est  $O(n)$

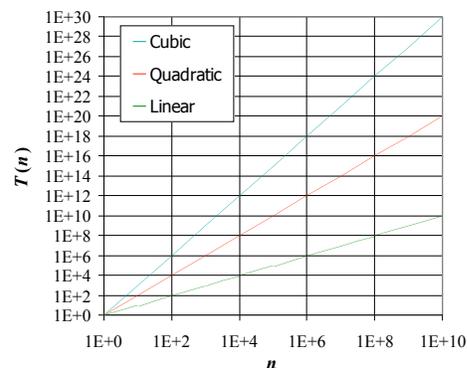
Ex.2:  $6n^2\log(n) + 3n^2 + 5n$  est  $O(n^2\log n)$

Ex.3:  $n^5+1000n^4+20n^3 - 8$  est  $O(n^5)$

## Notation asymptotique (terminologie)

Types de complexité :

<i>constant:</i>	$O(1)$
<i>logarithmique:</i>	$O(\log n)$
<i>linéaire:</i>	$O(n)$
<i>quadratique:</i>	$O(n^2)$
<i>cubique:</i>	$O(n^3)$
<i>polynomial:</i>	$O(n^k), k \geq 1$
<i>exponentiel:</i>	$O(a^n), n > 1$

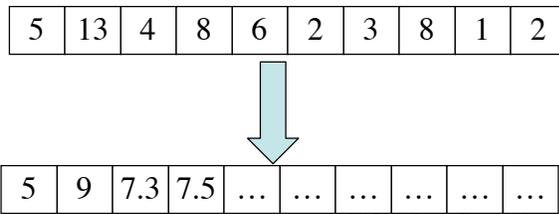


## Exemple d'analyse asymptotique

Un algorithme pour calculer les moyennes préfixes

La moyenne de préfixe  $i$ -ième d'un tableau  $X$  est la moyenne des premiers  $(i + 1)$  éléments de  $X$

$$A[i] = \frac{X[0] + X[1] + \dots + X[i]}{(i + 1)}$$



CSI2510 - Paola Flocchini

43

## Exemple d'analyse asymptotique

**Algorithm** *prefixAverages1*( $X, n$ )

**entrée** Un vecteur de nombre  $X$  à  $n$  éléments  
**sortie** un vecteur  $A$  avec la moyenne des préfixes de  $X$

```
 $A \leftarrow$  new array of  $n$  integers
for  $i \leftarrow 0$  to  $n - 1$  do
     $s \leftarrow X[0]$ 
    for  $j \leftarrow 1$  to  $i$  do
         $s \leftarrow s + X[j]$ 
     $A[i] \leftarrow s / (i + 1)$ 
return  $A$ 
```

CSI2510 - Paola Flocchini

44

i



5	13	4	7	6	2	3	8	1	2
---	----	---	---	---	---	---	---	---	---

j = 0

5									
---	--	--	--	--	--	--	--	--	--

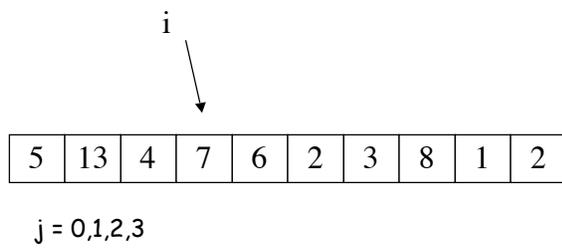
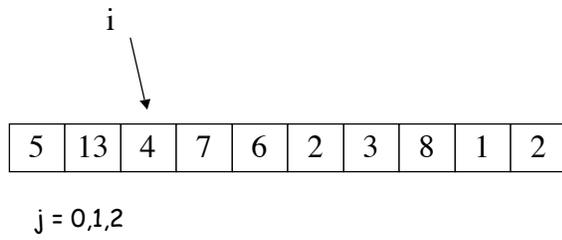
i



5	13	4	7	6	2	3	8	1	2
---	----	---	---	---	---	---	---	---	---

j = 0,1

5	8								
---	---	--	--	--	--	--	--	--	--



## Exemple d'analyse asymptotique

Algorithm `prefixAverages1(X, n)`

**Entrée:** Tableau **X** de **n** chiffres

**Sortie:** Tableau **A** de 'prefix averages'

**A** ← new array of size **n**

for **i** ← 0 to **n - 1** do

{ **s** ← **X[0]**

for **j** ← 1 to **i** do

{**s** ← **s + X[j]**}

**A[i]** ← **s / (i + 1)**

}

return **A**

#operations

**n**

**0+1+2+...+(n-1)**

**n**

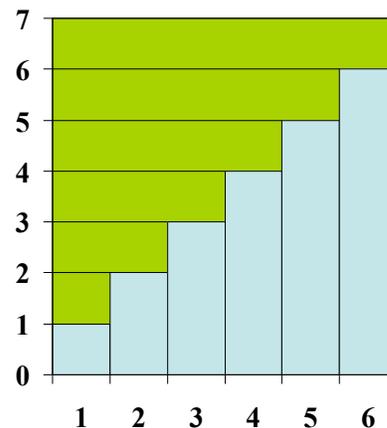
**1**

$$1 + 2 + \dots + (n - 1) = ?$$

CSI2510 - Paola Flocchini

49

- Le temps d'exécution de *prefixAverages1* est  $O(1 + 2 + \dots + n)$
- Le sommation de les premières  $n$  éléments est  $n(n + 1) / 2$ 
  - Il y a représentations visuelle
- Alors, l'algorithme *prefixAverages1* est  $O(n^2)$



CSI2510 - Paola Flocchini

50

Donc l'algorithme *prefixAverages1* a un temps d'execution de:

$$O(n(n+1)/2)$$

=

$$O(n^2)$$

**NE PAS OUBLIER**

$$1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

51

**Un autre exemple:**

un meilleur algorithme pour calculer les moyennes  
préfixes.

**Algorithm prefixAverages2(X, n)**

**Entrée:** Tableau **X** de **n** chiffres

**Sortie:** Tableau **A** de 'prefix averages'

**A** ← new array of size **n**

**s** ← 0

for **i** ← 0 to **n - 1** do

{

**s** ← **s** + **X**[**i**]

**A**[**i**] ← **s** / (**i** + 1)

}

return **A**

CSI2510 - Paola Flocchini

52

i



5	13	4	7	6	2	3	8	1	2
---	----	---	---	---	---	---	---	---	---

s=0

5									
---	--	--	--	--	--	--	--	--	--

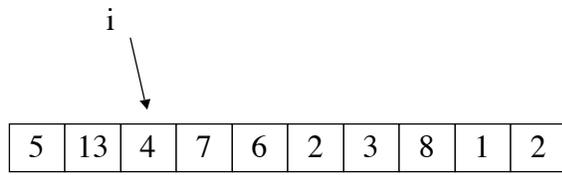
i



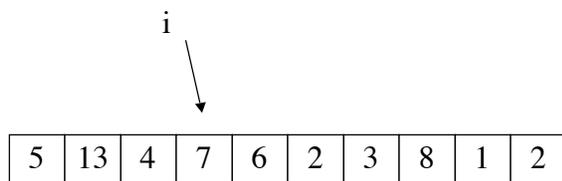
5	13	4	7	6	2	3	8	1	2
---	----	---	---	---	---	---	---	---	---

s=5

5	8								
---	---	--	--	--	--	--	--	--	--



s=18



s=22



### Algorithm prefixAverages2(X, n)

**Entrée:** Tableau **X** de **n** chiffres

**Sortie:** Tableau **A** de 'prefix averages'

**A** ← new array of size **n**

**s** ← 0

for **i** ← 0 to **n - 1** do

{

**s** ← **s** + **X**[**i**]

**A**[**i**] ← **s** / (**i** + 1)

}

return **A**

#operations

← ..... **n**  
← ..... **n**  
← ..... **1**

**O(n)**

**O(n)**

CSI2510 - Paola Flocchini

57

## big-Omega (Grand Omega)

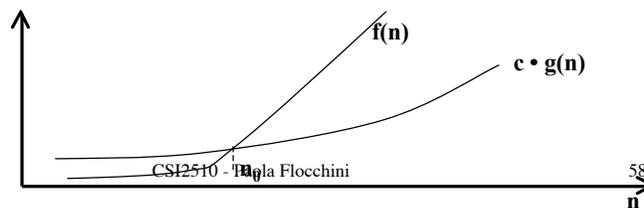
(Limite inferieure)

$f(n)$  est  $\Omega(g(n))$

Si il y a  $c > 0$  et  $n_0 > 0$  tel que

$f(n) \geq c \cdot g(n)$  pour tout  $n \geq n_0$

$f(n)$  est  $\Omega(g(n))$  si et seulement si  $g(n)$  est  $O(f(n))$



## big-Thêta (Grand Thêta)

$g(n)$  est  $\Theta(f(n))$

$\iff$

si  $g(n) \in O(f(n))$

et

$f(n) \in O(g(n))$

CSI2510 - Paola Flocchini

59

## Exemple

Nous avons déjà vu:

$f(n) = 60n^2 + 5n + 1$  est  $O(n^2)$

mais  $60n^2 + 5n + 1 \geq 60n^2$  pour  $n \geq 1$

Donc: avec  $c = 60$  et  $n_0 = 1$

$f(n) \geq c \cdot n^2$  pour tout  $n \geq 1$

$f(n)$  est  $\Omega(n^2)$

Alors:

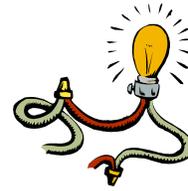
$f(n)$  est  $O(n^2)$   
et  
 $f(n)$  est  $\Omega(n^2)$

$f(n)$  est  $\Theta(n^2)$

CSI2510 - Paola Flocchini

60

## Intuition pour les notations asymptotiques



### Big-Oh

- $f(n)$  est  $O(g(n))$  si  $f(n)$  est **plus petite ou égal** à  $g(n)$  quand  $n$  est grand

### big-Omega

- $f(n)$  est  $\Omega(g(n))$  si  $f(n)$  est **plus grand ou égal** à  $g(n)$  quand  $n$  est grand

### big-Theta

- $f(n)$  est  $\Theta(g(n))$  si  $f(n)$  est **à peu près égal** à  $g(n)$  quand  $n$  est grand

## Mathématiques à réviser Logarithmes et exposants



### Propriétés des logarithmes:

$$\log_b(xy) = \log_b x + \log_b y$$

$$\log_b(x/y) = \log_b x - \log_b y$$

$$\log_b x^a = a \log_b x$$

$$\log_b a = \frac{\log_x a}{\log_x b}$$

### Propriétés des exposants:

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$a^b / a^c = a^{(b-c)}$$

$$b = a^{\log_a b}$$

$$b^c = a^{c \cdot \log_a b}$$

## Mathématiques à réviser

---

- Plancher (Floor):  $\lfloor x \rfloor$  = le plus grand entier  $\leq x$      $\lfloor 2.3 \rfloor = 2$
- Plafond (Ceiling):  $\lceil x \rceil$  = le plus petit entier  $\geq x$      $\lceil 2.3 \rceil = 3$
- Sommations:
  - Définition général:

$$\sum_{i=s}^t f(i) = f(s) + f(s+1) + f(s+2) + \dots + f(t)$$

- Où  $f$  est une fonction,  $s$  est l'index de départ, et  $t$  est l'index d'arrivée

CSI2510 - Paola Flocchini

63

## Progression géométrique

---

$$S = \sum_{i=0}^n r^i = 1 + r + r^2 + \dots + r^n$$
$$rS = r + r^2 + \dots + r^n + r^{n+1}$$
$$rS - S = (r-1)S = r^{n+1} - 1$$
$$S = (r^{n+1}-1)/(r-1)$$

If  $r=2$ ,     $S = (2^{n+1}-1)$

CSI2510 - Paola Flocchini

64

## Progression aritmetique

---

$$S = \sum_{i=0}^n di = 0 + d + 2d + \dots + nd$$
$$= nd + (n-1)d + (n-2)d + \dots + 0$$

$$2S = nd + nd + nd + \dots + nd$$
$$= (n+1)nd$$

$$S = d/2 n(n+1)$$

$$\text{for } d=1, S = 1/2 n(n+1)$$

CS2510 - Paola Picchini

65