Matière: Algorithmique Avancée et langage C

Préliminaires

Rappels mathématiques et algorithmique

Enseignant(e)s: **Feyrouz HAMDAOUI & Sofiane Ben Ahmed**

Niveau: **3ème Génie Informatique**

2024/2025

SU2

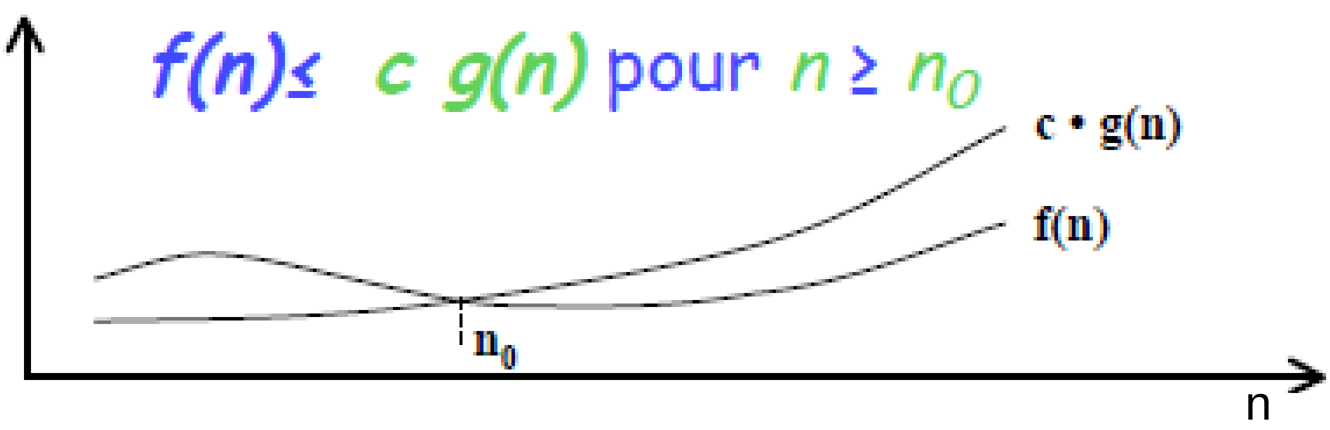
Plan

- Notation asymptotique
- Fonctions mathématiques importantes
- Instructions algorithmiques élémentaires.

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

$$f(n) \leq c\, g(n) \text{ pour } n \geq n_0$$



Big Oh (Grand Oh)
Limite supérieure

- on note **f = O(g)** ou f(x) = O(g(x))
- on dit que f est **dominée asymptotiquement** par g
- cette notation est appelée **notation de Landau**
- ***Attention :*** il ne s'agit que d'une borne supérieure, et à partir d'un certain rang, cela indique juste que g ne croit pas plus vite que f à partir de ce rang (mais rien n'indique qu'elle croit moins vite, ni qu'elle croit aussi vite)

Big Oh (Grand Oh)
Limite supérieure

- ***<u>Propriétés:</u>*** La notation O, dite notation de Landau, vérifie les propriétés suivantes :

➢ si f=O(g) et g=O(h) alors f=O(h)

➢ si f=O(g) et k un nombre, alors k*f=O(g)

➢ si f1=O(g1) et f2=O(g2) alors f1+f2 = O(g2+g2)

➢ si f1=O(g1) et f2=O(g2) alors f1*f2 = O(g2*g2)

- ***<u>Exemples de domination asymptotique:</u>***

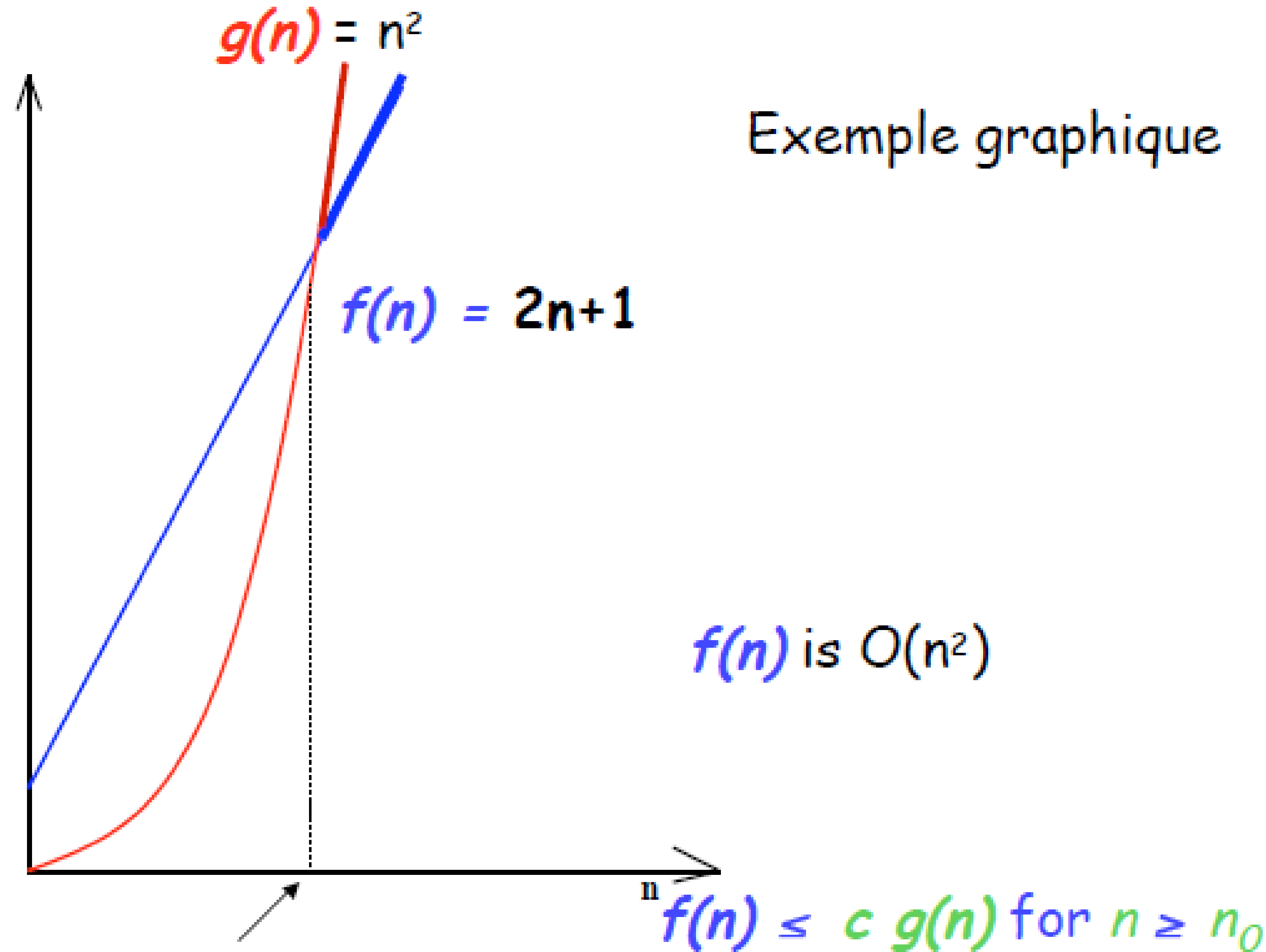
$x = O(x^2)$ car pour $x>1$, $x<x^2$ $x^2 = O(x^3)$ car pour $x>1$, $x^2<x^3$

$100*x = O(x^2)$ car pour $x>100$, $x<x^2$ $ln(x) = O(x)$ car pour $x>0$, $ln(x)<x$

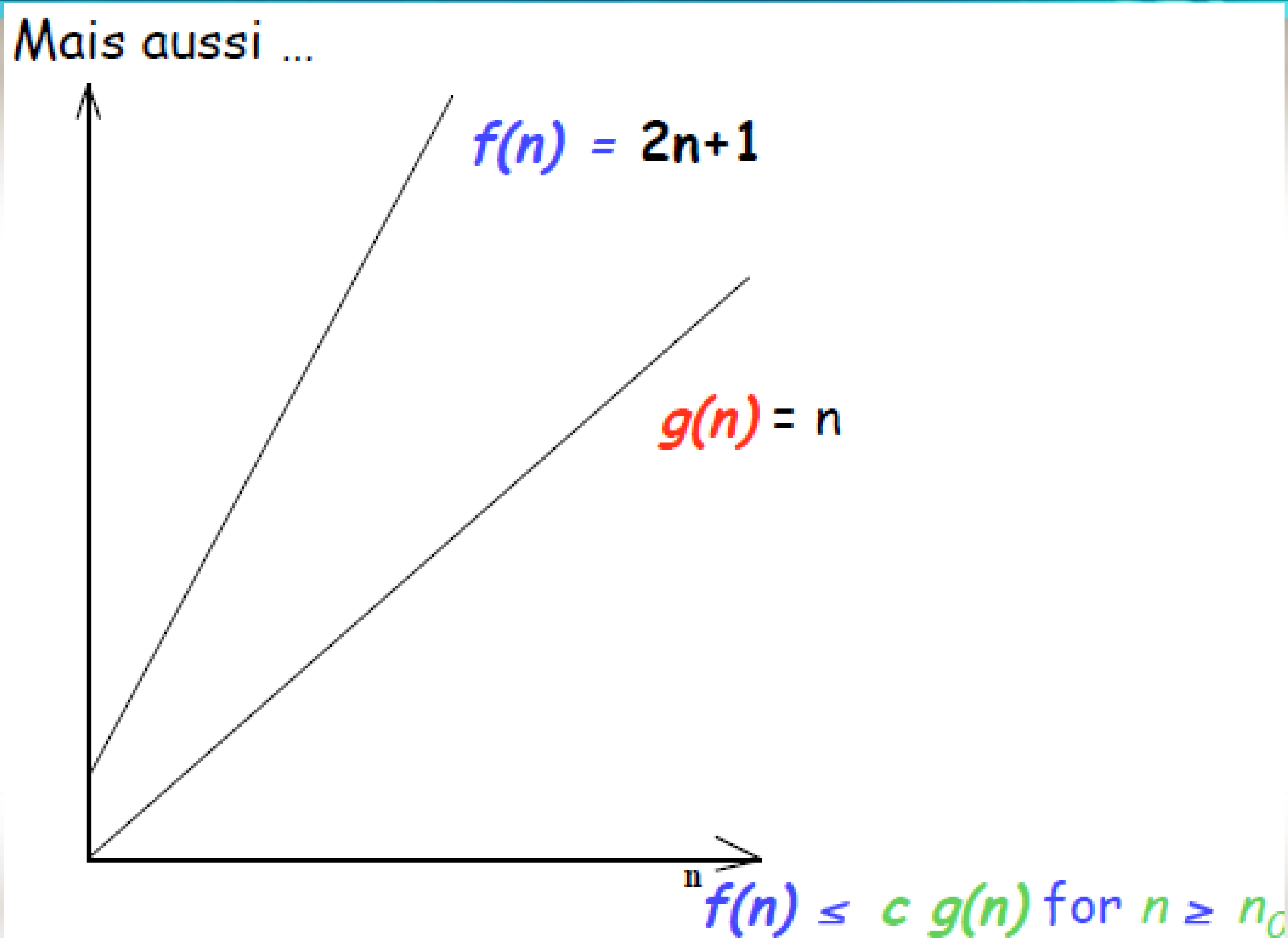
si $i>0$, $x^i = O(e^x)$ car pour x tel que $x/ln(x)>i$, $x^i<e^x$

EUR-ACE®

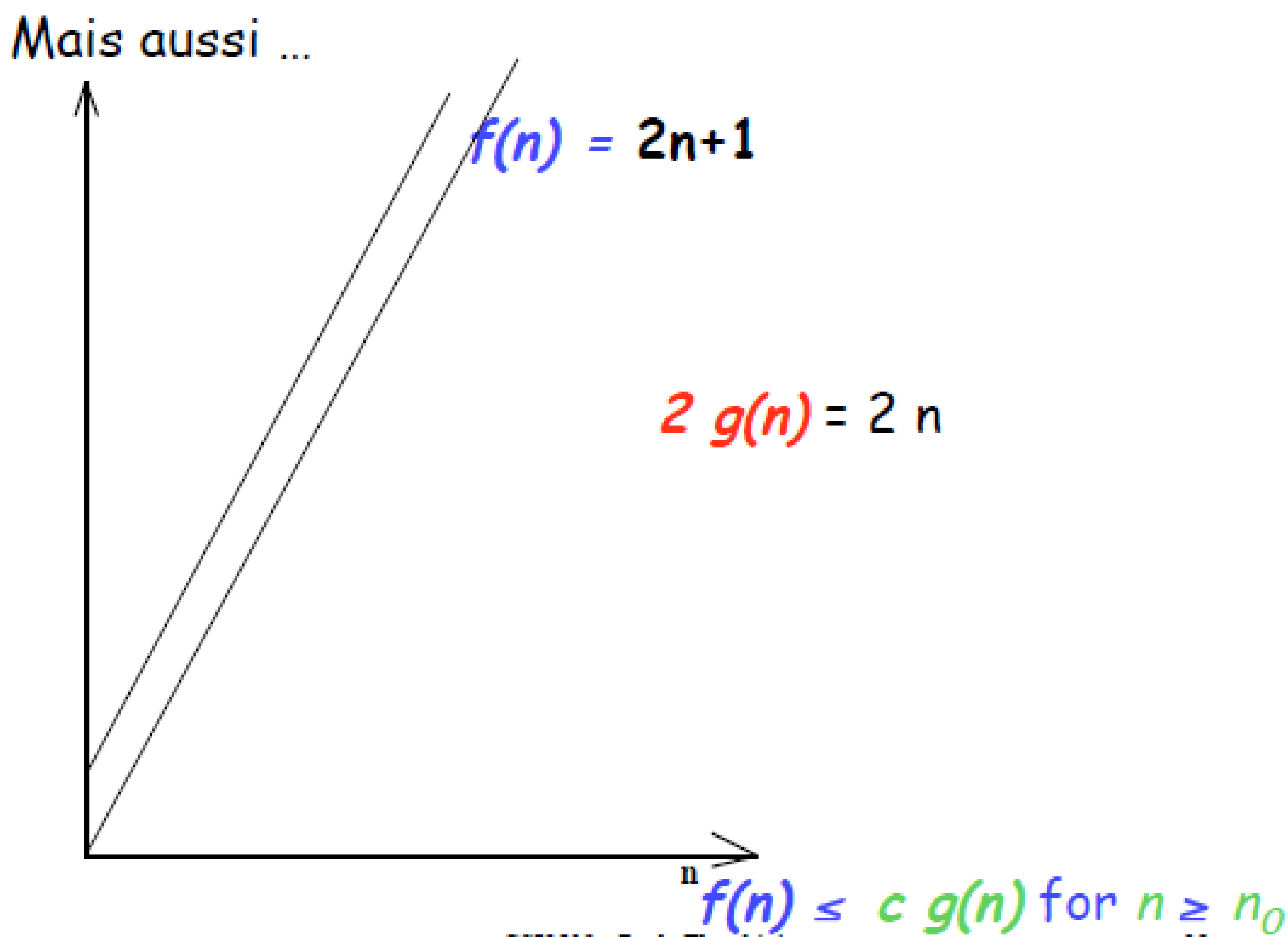
Big Oh (Grand Oh)
Limite supérieure



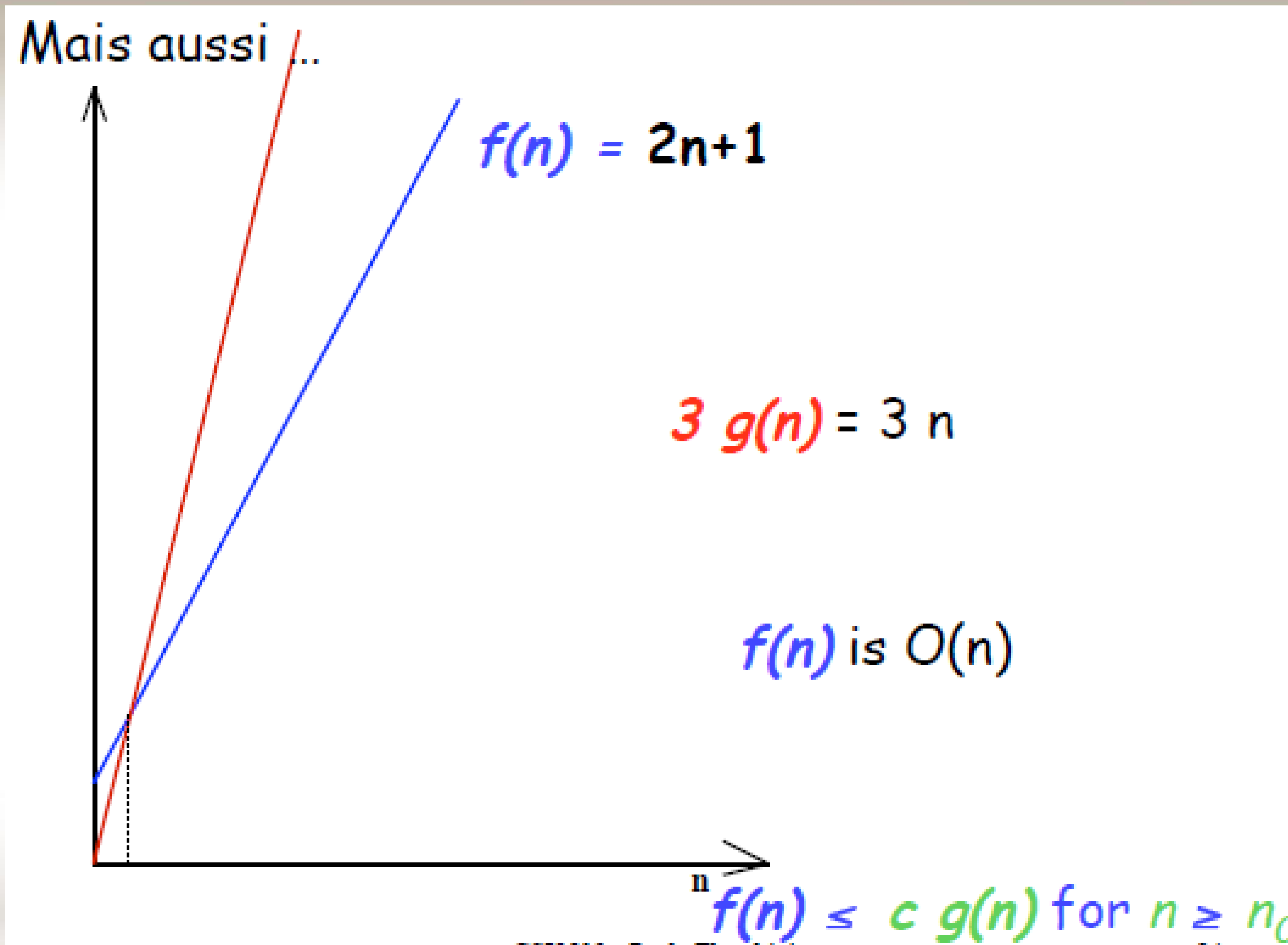
Big Oh (Grand Oh)
Limite supérieure



Big Oh (Grand Oh)
Limite supérieure



Big Oh (Grand Oh)
Limite supérieure

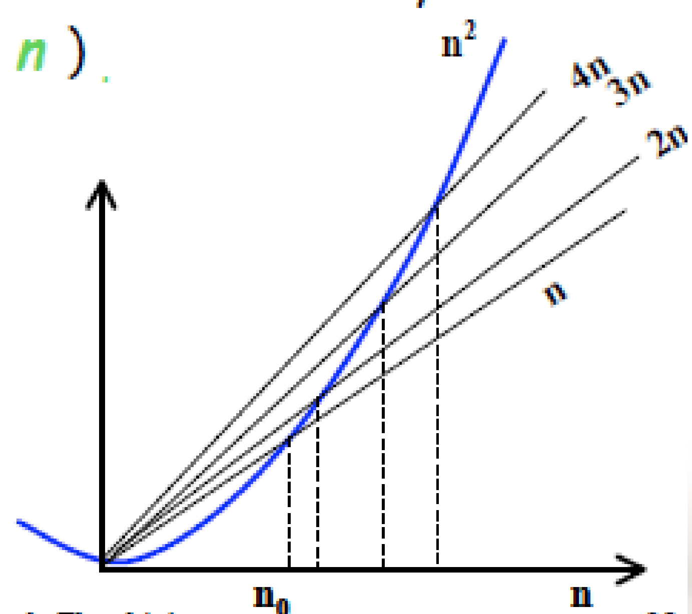


Big Oh (Grand Oh)
Limite supérieure

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 $n^2 > c\,n$).



Big Oh (Grand Oh)
Limite supérieure

Exemple

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

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:

$$60n^2 + 5n + 1 \le c\, n^2 \text{ pour tout } n \ge n_0$$

$5n \le 5n^2$ pour tout $n \ge 1$

$1 \le n^2$ pour tout $n \ge 1$

$f(n) \le 60n^2 + 5n^2 + n^2$ pour tout $n \ge 1$

$f(n) \le 66n^2$ pour tout $n \ge 1$

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

Remarques

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

➢ Ex.: Il est correct de dire que 5n-3 est O(n3) 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)

Remarques

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

➢ Ex.1: 7n-3 est O(n)

➢ Ex.2: 6n2log(n) + 3n2 +5n est O(n2log n)

➢ Ex.3: n5+1000n4+20n3 - 8 est O(n5)

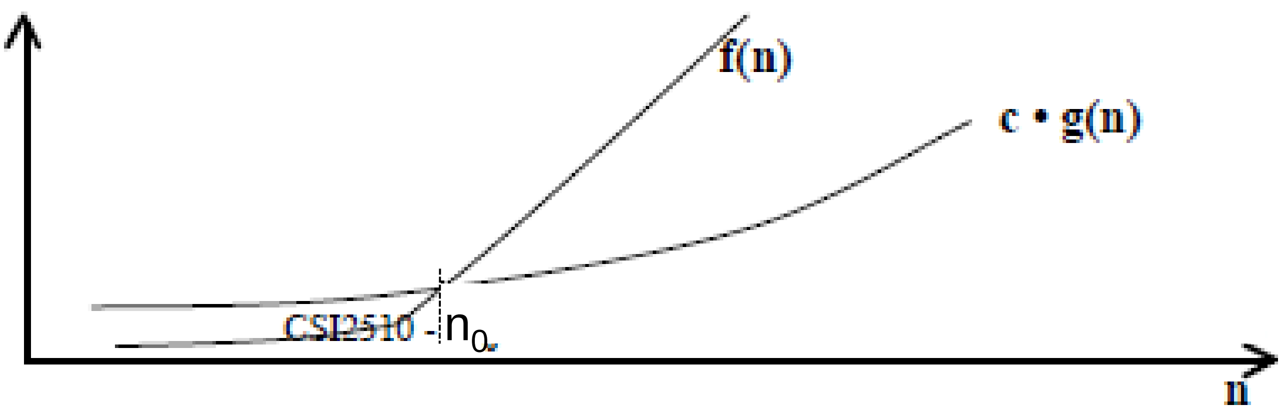
big-Omega (Grand Omega) (Limite inferieure)

$f(n)$ 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)

$$g(n) \text{ est } \Theta(f(n))$$

<===>

$$\text{si } g(n) \in O(f(n))$$

et

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

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 Ω(g(n)) si f(n) est **plus grand ou égal** à g(n) quand n est grand

big-Theta

– f(n) est Θ(g(n)) si f(n) est **a peu près égal à** g(n) quand n est grand

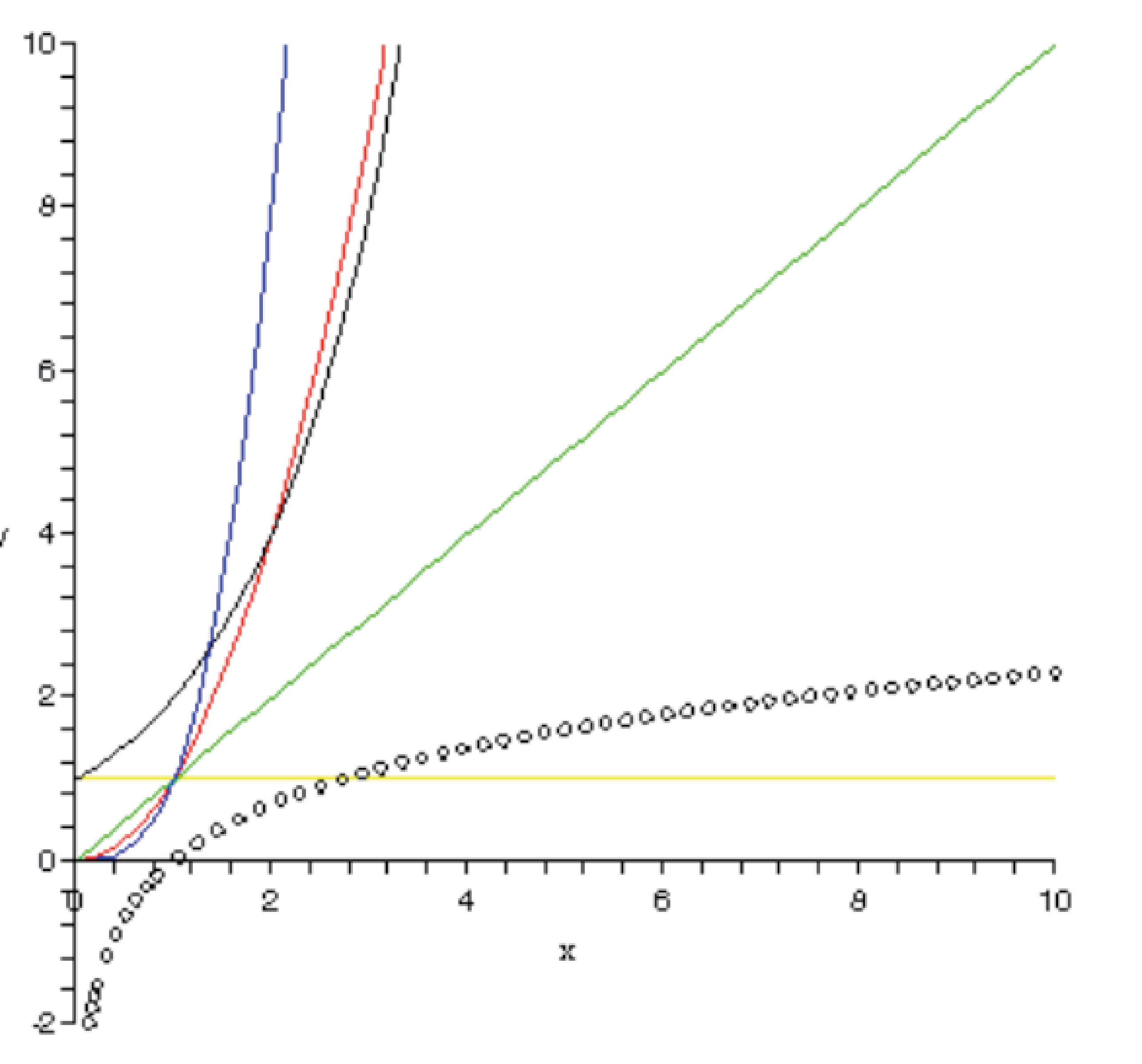
Plan

- Notation asymptotique
- Fonctions mathématiques importantes
- Instructions algorithmiques élémentaires.

Fonctions importantes

- fonction constante, $f(x)=1$
- fonction linéaire, $f(x)=x$
- fonction logarithmique, $f(x)=\log(x)$
- fonction quadratique, $f(x)= x^2$
- fonction cubique, $f(x)=x^3$
- fonction exponentielle, $f(x)= 2^x$

Comparaison des fonctions



- fonction constante, $f(x)=1$
- fonction linéaire, $f(x)=x$
- fonction logarithmique, $f(x)=\log(x)$
- fonction quadratique, $f(x)= x^2$
- fonction cubique, $f(x)=x^3$
- fonction exponentielle, $f(x)= 2^x$

Comparaison des fonctions

log(n)	$\sqrt{n}$	n	n log(n)	n^2
3	3	10	33	100
7	10	100	664	10 000
10	32	1000	9966	1 000 000
13	100	10 000	132 877	100 000 000
17	316	100 000	1 660 964	10 000 000 000
20	1000	1 000 000	19 931 569	1 000 000 000 000

Valeurs des fonctions communément rencontrées, tableau modifié de © 2004, Sedgewick

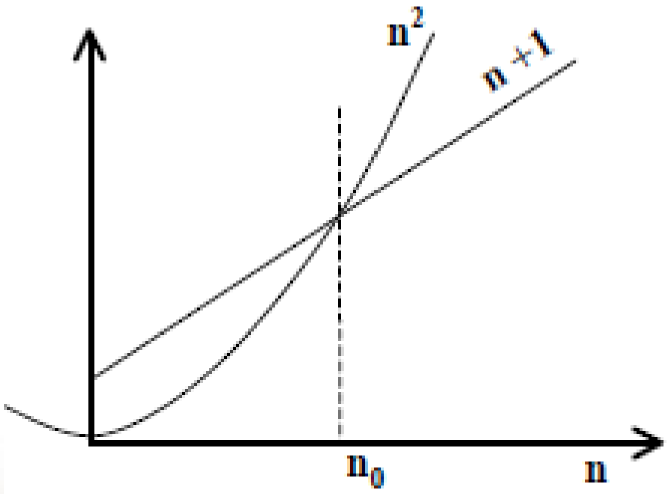
Notation asymptotique (terminologie)

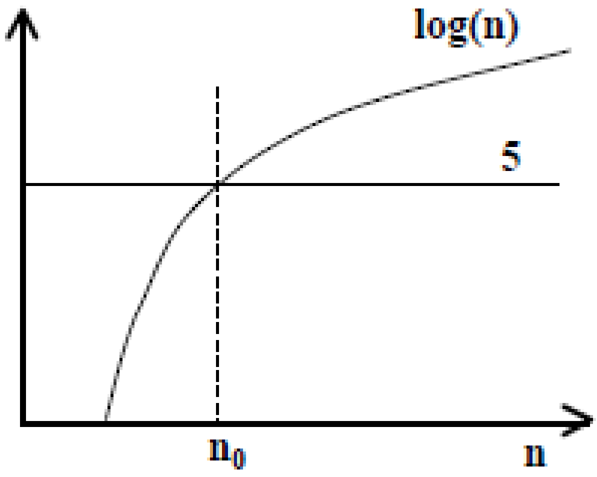
Types de complexité :

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

Notation asymptotique (terminologie)

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





Plan

- Notation asymptotique
- Fonctions mathématiques importantes
- Instructions algorithmiques élémentaires

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…
- On assume que leur temps d’exécution est **constant**

Compter les opérations primitives

- En inspectant le pseudocode d'un algorithme, on peut déterminer le nombre maximum d'opérations élémentaires exécuté par un algorithme, comme une fonction de la taille de l'entrée et par la suite analyser son temps d'exécution et son efficacité.
- Le pseudo-code est une description d'algorithme qui est plus structurée que la prose ordinaire mais moins formelle qu'un langage de programmation.

Exemple

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
```

Exemple

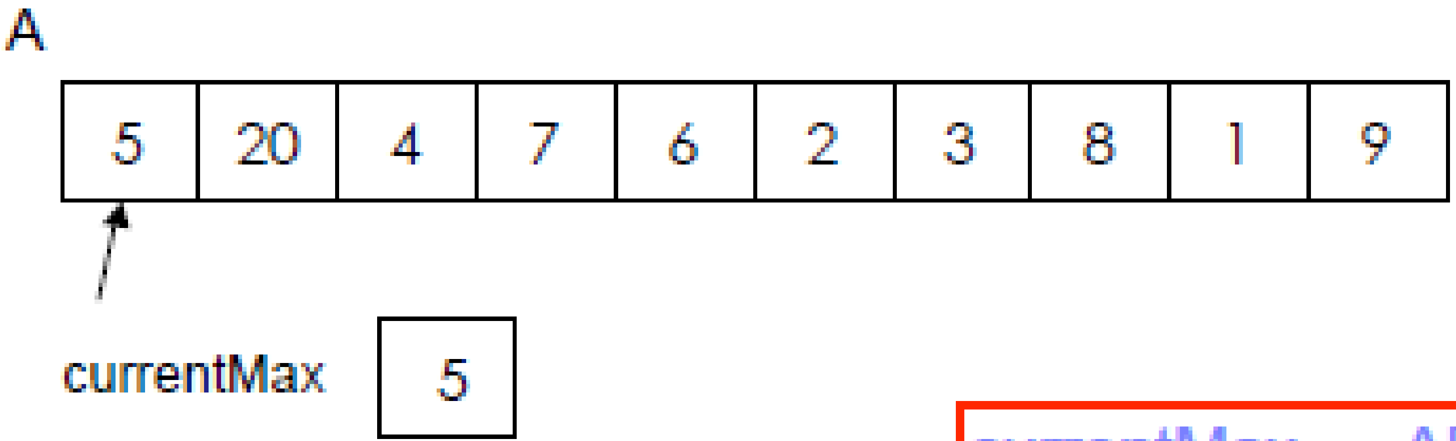
A

5	20	4	7	6	2	3	8	1	9

currentMax $\boxed{}$

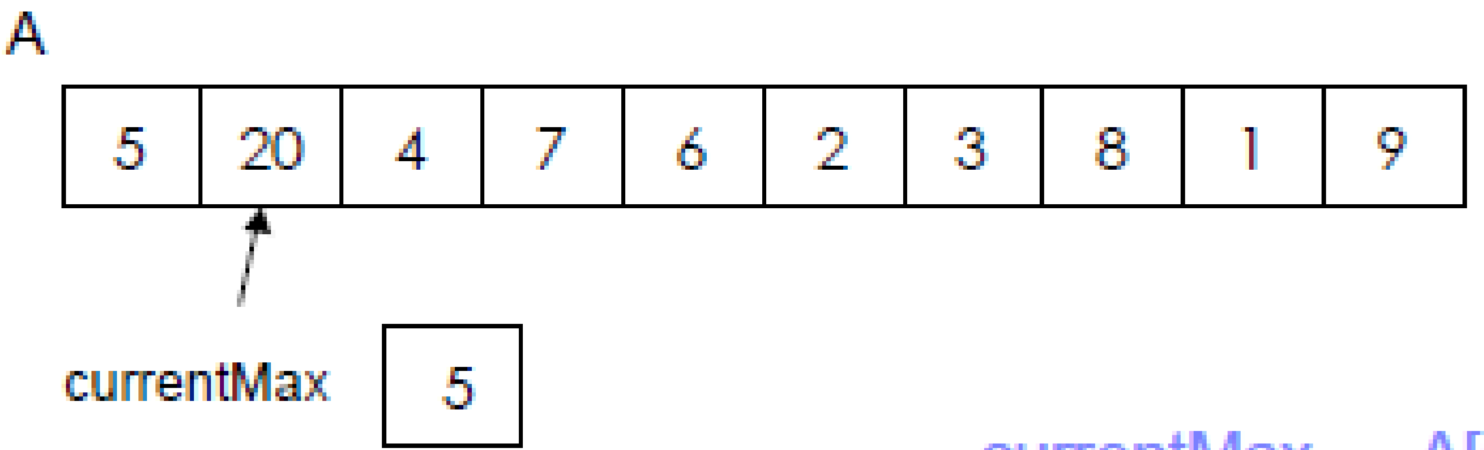
```
currentMax ← A[0]
for i ← 1 to n -1 do
{
if currentMax < A[i] then
        currentMax ← A[i]
}
return currentMax
```

Exemple




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

Exemple



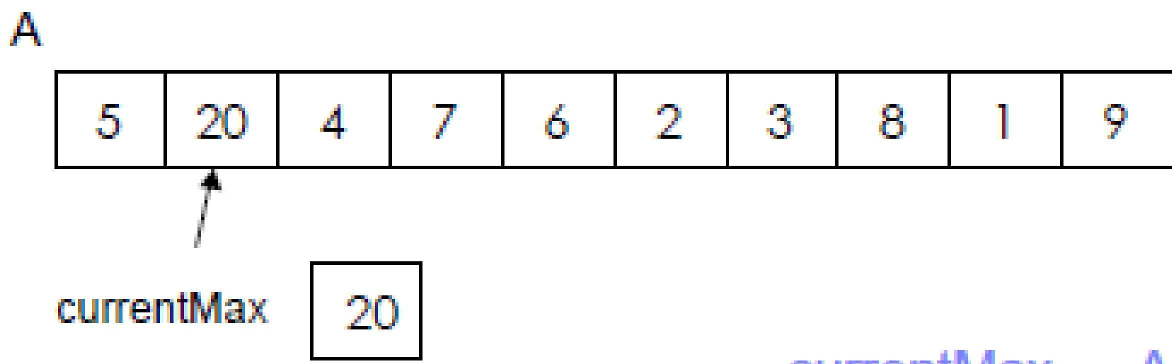

```
currentMax ← A[0]
for i ← 1 to n -1 do
{
if currentMax < A[i] then
            currentMax ← A[i]

}

return currentMax
```

Exemple

Quelles sont les opérations primitives à compter ?



Comparaisons
Affectations à currentMax

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

Exemple

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
```

Exemple

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
```

Exemple

Algorithme de recherche du max TabMax(A,n)

Meilleur cas	Pire cas
1 affection+(n-1) comparaisons	n affections+(n-1) comparaisons

Bibliographie

- Sylvie Hamel. «Analyse et complexité des algorithmes ». Université de Montréal; IFT2810, A2009
- Frédéric Fürst. « complexité ». Cours «Algorithmique et programmation », Licence Informatique, Université de Picardie. 2015/2016
- Paola Flocchini. « Structures de données et algorithmes».Université d'Ottawa, 2010.