# 3 Croissance de fonctions

## 3.1 Mesures de performance

- ⋆ usage de mémoire («espace»)
- ⋆ vitesse d'exécution («temps»)
- ⋆ d'autres mesures possibles, selon l'application (p.e., quantité de données transmises sur un réseau, ou nombre de *cache miss*)

Pourquoi ?

- ⋆ comparer l'efficacité de différentes solutions au même problème
- ⋆ donner un pronostic des performances
- ⋆ initialiser les valeurs des paramètres de la structure ou de l'algorithme

## 3.2 Expériences

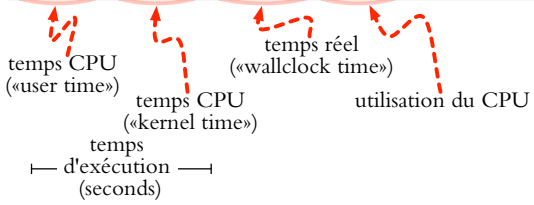
Le temps d'exécution peut être mesuré avec des entrées différentes pour l'analyse empirique.

- ⋆ Dans le programme (Java) :

```
long T0 = System.currentTimeMillis(); // temps de début
...
long dT = System.currentTimeMillis()-T0; // temps (ms) dépassé
```

- ⋆ Par les outils du système d'exploitation (shell Linux/Unix)

```
% time java -cp Monjar.jar mabelle.Application
0.283u 0.026s 0:00.35 85.7% 0+0k 0+53io 0pf+0w
```




## 3.3 Analyse formel du temps

L'analyse formel doit assumer un modèle mathématique qui spécifie les **instructions élémentaires**, et le temps d'exécution associé avec chaque instruction. Les instructions élémentaires forment le vocabulaire pour décrire un algorithme.

**Cycles CPU.** En principe, on peut calculer explicitement le nombre de cycles pour un algorithme donné en assembleur. Les instructions élémentaires sont celles du CPU : chaque instruction prend un nombre connu de cycles. Mais ce genre de

W(fr)

calcul est difficile (il faut décrire l'algorithme en un langage primitif), non-général (l'ensemble d'instructions du langage et les cycles d'exécution varient entre CPUs), et moins précis en pratique qu'on le pense (le temps actuel d'exécution dépend de détails de l'architecture de CPU, p.e., pipelining, nombre de cœurs).

```
[Java]
int X = (Y+4)*3;
[Assembly]
mov eax, Y    ; affectation registre EAX (2-4 cycles)
add eax, 4    ; EAX = EAX + 4:           (2-4 cycles)
mov ebx, 3    ; EBX = 3:                 (2-4 cycles)
imul ebx      ; EAX = EAX*EBX:           (6-9 cycles)
mov X, eax    ;                          (2-4 cycles)
```

**Modèles mathématiques.** La **machine de Turing** comprend seulement un ruban «magnétique», et une tête de lecture-écriture. La gestion de mémoire est difficile avec une machine de Turing (la tête glisse par une case à la fois). Dans une **machine RAM** (*Random Access Machine* — machine à accès direct) on peut adresser le mémoire directement pour stocker les données. L'ensemble d'instructions est proche des langages assembleurs des CPUs courants. W(fr) W(fr)

**Pseudocode.** On décrit souvent un algorithme en «pseudocode» : programme de lignes, mémoire infini, nombre fini de variables, instructions d'affectation, arithmétiques, et de contrôle. Le syntaxe dépend de l'auteur, mais les instructions en pseudocode sont faciles à implanter sur une machine RAM (ou en un langage de programmation), et s'exécutent typiquement en un temps constant.

## 3.4 Notation asymptotique

Comparaison de croissances : qu'est-ce qui se passe avec $\frac{f(n)}{g(n)}$ quand $n \to \infty$ ? W(fr)

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \qquad f = o(g) \tag{3.1a}$$

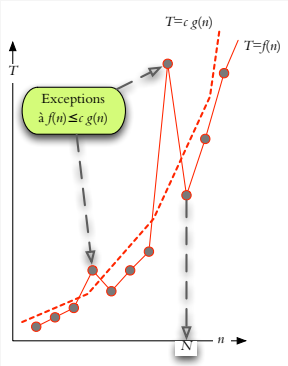
$$\liminf_{n\to\infty} \frac{f(n)}{g(n)} > 0 \qquad f = \Omega(g) \tag{3.1b}$$

$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty \qquad f = O(g) \tag{3.1c}$$

$$0 < \liminf_{n\to\infty} \frac{f(n)}{g(n)} \leq \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty \qquad f = \Theta(g) \tag{3.1d}$$

Il existe plusieurs définitions équivalentes.

**Définition 3.1** (**grand O**). $f = O(g)$ *si et seulement s'il existe* $c$ *et* $N$ *tels que* $f(n) \leq c \cdot g(n)$ *pour tout* $n \geq N$.

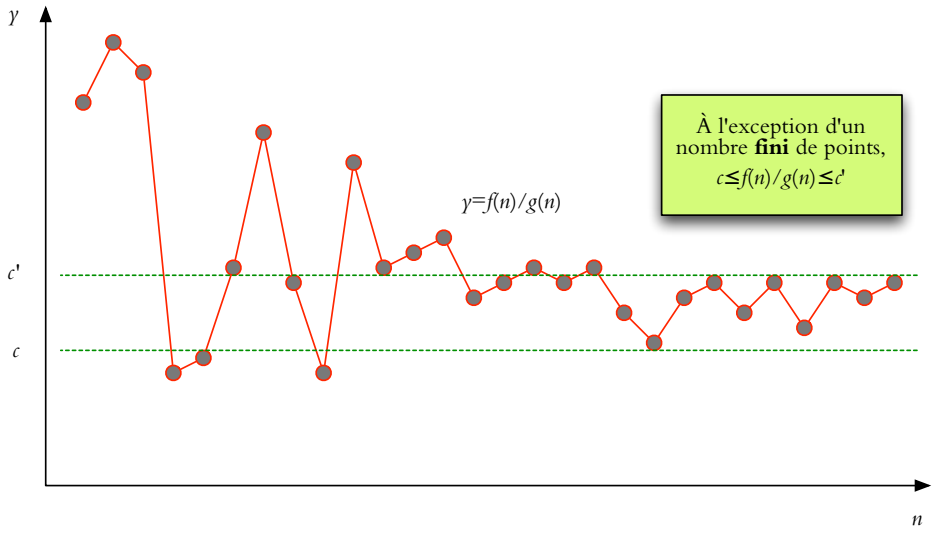



Dans d'autres mots, $f = O(g)$ ssi $f(n) \leq c \cdot g(n)$ pour **presque tout**$^\star$ $n$.

$^\star$ "presque tout" = tout sauf pour un nombre fini d'exceptions

**Définition 3.2** (**petit O**). $f = o(g)$ *si et seulement si pour tout* $c$*, il existe* $N$ *tel que* $f(n) \leq c \cdot g(n)$ *pour tout* $n \geq N$.

**Définition 3.3** (**Theta**). $f = \Theta(g)$ *si et seulement s'il existe* $0 < c \leq c' < \infty$ *tels que* $c \cdot g(n) \leq f(n) \leq c' \cdot g(n)$ *pour presque tout* $n$.



## 3.5 Règles d'arithmétique

$$c \cdot f = O(f) \tag{3.2a}$$

$$\underbrace{O(f) + O(g)}_{\underline{\text{pour tout}}\ h = O(f)\ \text{et}\ h' = O(g)} = \underbrace{O(f+g)}_{\underline{\text{il existe}}\ h'' = O(f+g)\ \text{t.q.}\ h + h' = h''} \tag{3.2b}$$

$$= O\big(\max\{f, g\}\big) \tag{3.2c}$$

$$O(f) \cdot O(g) = O(f \cdot g) \tag{3.2d}$$

## 3.6 Quelques fonctions notables

**Logarithmes.**

$$\lg x = \log_2 x \qquad \text{et} \qquad \ln x = \log_e x \quad \{x > 0\}$$

$$\log(xy) = \log x + \log y; \log(x^a) = a \log x$$

$$2^{\lg n} = n; n^n = 2^{n \lg n}; \quad \log_a n = \frac{\lg n}{\lg a} = \Theta(\lg n) \quad ; a^{\lg n} = n^{\lg a}$$

**Factorielle.** $n! = 1 \cdot 2 \cdots n$. Approximation de Stirling : W(fr)

$$n! = \big(1 + o(1)\big)\sqrt{2\pi n}\Big(\frac{n}{e}\Big)^n = \Theta\big(n^{n+1/2}e^{-n}\big)$$

D'où $\sum_{j=1}^{n} \ln k = \ln(n!) = \big(1 + o(1)\big) n \ln n$, donc $\log(n!) = \Theta(n \log n)$.

**Nombre harmonique.**

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n} \approx \ln n + \gamma = \big(1 + o(1)\big) \ln n = \Theta(\log n)$$

où $\gamma = \lim_{n\to\infty}(H_n - \ln n) = 0.5772\cdots$ est la constante d'Euler. W(fr)

## 3.7 Hiérarchie de croissances

$\Theta$ définit des classes d'équivalence entre fonctions séparées par $o$ [petit-o] (et il y a aussi des classes non-comparables)
Croissance polynomiale : $n^{O(1)}$, exponentielle : $2^{O(n)}$

$$\underbrace{\underbrace{\Theta(1)}_{\text{constante}} \xrightarrow{o()} \underbrace{\Theta(\log n)}_{\text{logarithmique}} \xrightarrow[\{0<a<1\}]{o()} \Theta(n^a) \xrightarrow{o()} \underbrace{\Theta(n)}_{\text{linéaire}} \xrightarrow{o()} \Theta(n \log n) \xrightarrow[\{a>1\}]{o()} \Theta(n^a)}_{\text{polynomiale}}$$

$$\xrightarrow{o()} \underbrace{\Theta(a^n) \xrightarrow[\{b>a\}]{o()} \Theta(b^n)}_{\text{exponentielle}} \xrightarrow{o()} \underbrace{\Theta(n!) \xrightarrow{o()} \Theta(n^n)}_{\text{superexponentielle}} \xrightarrow{o()} \cdots$$