## Question

- Peut-on comparer les complexités des différents algorithmes ?
  oui s'ils comptent la même chose
- Peut-on dire si deux algorithmes ont des complexités semblables ?
  oui s'il existe une taille d'exemplaire telle que pour tout exemplaire de plus grande taille le rapport de leurs complexités tend vers une constante

On s'intéresse au comportement asymtotique de la fonction de complexité

## Comportement asymptotique - illustration 1/2

Supposons avoir deux algorithmes A et B, de complexité en temps respective $c_A(n) = n$ et $c_B(n) = 3n + 100$, ont-ils un comportement réellement différent ?

| exemplaire de petite taille | | | | exemplaire de grande taille | | | |
|---|---|---|---|---|---|---|---|
| n | A | B | A/B | n | A | B | A/B |
| 10 | 10 | 130 | 0.07 | $10^6$ | 1000000 | 3000100 | 0.33 |
| 100 | 100 | 400 | 0.25 | $10^9$ | 1000000000 | 3000000100 | 0.33 |

Lorsque $n$ devient grand, le rapport des complexités tend vers une constante.

Les algorithmes ont un comportement similaire.



## Comportement asymptotique - illustration 2/2

Supposons avoir deux algorithmes A et B, de complexité en temps respective $c_A(n) = n$ et $c_B(n) = n^2$, ont-ils un comportement réellement différent ?

| exemplaire de petite taille | | | | exemplaire de grande taille | | | |
|---|---|---|---|---|---|---|---|
| n | A | B | A/B | n | A | B | A/B |
| 10 | 10 | 100 | 0.1 | $10^6$ | $10^6$ | $10^{12}$ | $1^{-6}$ |
| 100 | 100 | 10000 | 0.01 | $10^9$ | $10^9$ | $10^{18}$ | $1^{-9}$ |

Même si $n$ devient grand, le rapport des complexités ne tend pas vers une constante.

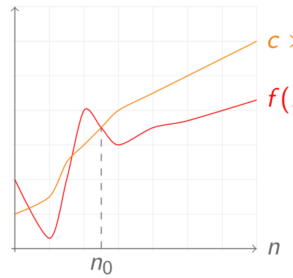
Les algorithmes n'ont pas un comportement similaire : B sera considéré moins efficace que A.



## La notation $\mathcal{O}$

**Definition ($\mathcal{O}$)**

Pour une fonction $g(n)$ donnée, on note $\mathcal{O}(g(n))$ l'ensemble de fonctions suivant : $\mathcal{O}(g(n)) = \{f(n)$ : il existe des constantes positives $c$ et $n_0$ telles que $0 \leq f(n) \leq c \times g(n)$ pour tout $n \geq n_0\}$




On note $f(n) = \mathcal{O}(g(n))$.

On dit "en grand o de g(n)".

Exemples :
$n = \mathcal{O}(n)$,
$3n + 10^{3700} = \mathcal{O}(n)$,
$3n^3 + 2n + 1 = \mathcal{O}(n^3)$

## Notation $\mathcal{O}$ et complexité des algorithmes

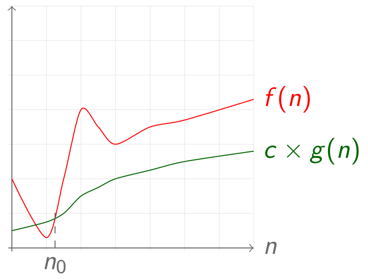
- Si les complexités de deux algorithmes ont même borne asymptotique supérieure, alors pour des exemplaires de grande taille ils mettront le même temps (à une constante multiplicative près) au maximum.
- Si on dispose d'un algorithme A dont on connaît la fonction de complexité en temps $f$ dans le pire des cas, on dira qu'il s'exécute en $\mathcal{O}(f)$



## La notation $\Omega$

**Definition ($\Omega$)**

Pour une fonction $g(n)$ donnée, on note $\Omega(g(n))$ l'ensemble des fonctions suivant : $\Omega(g(n)) = \{f(n)$ : il existe des constantes positives $c$ et $n_0$ telles que $0 \leq c \times g(n) \leq f(n)$ pour tout $n \geq n_0\}$




On note $f(n) = \Omega(g(n))$.

On dit "en grand omega de g(n)".

Exemples :
$n = \Omega(n)$,
$3n + 10^{3700} = \Omega(n)$,
$3n^3 + 2n + 1 = \Omega(n)$

Rmq : toute fonction positive est en $\Omega(1)$.



## Notation $\Omega$ et complexité des algorithmes

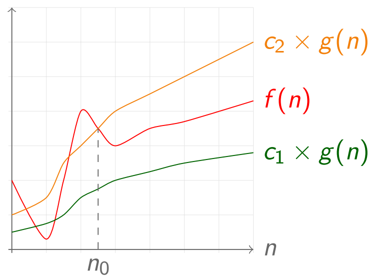
- Si les complexités de deux algorithmes ont même borne asymptotique inférieure, alors pour des exemplaires de grande taille ils mettront le même temps (à une constante multiplicative près) au minimum.
- Si on dispose d'un algorithme A dont on connaît la fonction de complexité en temps $f$ dans le meilleur des cas, on dira qu'il s'exécute en $\Omega(f)$



## La notation $\Theta$

**Definition ($\Theta$)**

Pour une fonction donnée $g(n)$, on note $\Theta(g(n))$ l'ensemble des fonctions $\Theta(g(n)) = \{f(n)$, il existe des constantes positives $c_1, c_2$ et $n_0$ telles que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ pour tout $n \geq n_0\}$




On note $f(n) = \Theta(g(n))$.
On dit "en theta de g(n)".
Exemples :
$n = \Theta(n)$,
$3n + 10^{3700} = \Theta(n)$,
$3n^3 + 2n + 1 = \Theta(n^3)$

C'est bien la même fonction mais avec deux constantes différentes de chaque côté de l'inégalité

## Rapport avec la complexité des algorithmes

- Si on dispose d'un algorithme A dont la fonction de complexité en temps $f$ est la même dans le meilleur des cas et dans le pire des cas, on dira qu'il s'exécute en $\Theta(f)$.
- Si on dispose d'un algorithme $A$, pour lequel on ne sait pas calculer exactement la complexité, mais seulement un encadrement : $f(n) \leq c(n) \leq g(n)$ et que $f(n) = \Theta(g(n))$ alors on pourra dire que la complexité est en $\Theta(f(n))$.
- Si $c_m(n) = \Theta(f(n))$ alors $c(n) = \Omega(f(n))$.
- Si $c_p(n) = \Theta(g(n))$ alors $c(n) = \mathcal{O}(g(n))$.



## Complexité des tris

| Tri | Nombre d'opérations de comparaison | | Complexité |
|---|---|---|---|
| | pire des cas | meilleur des cas | |
| Bulle | $\frac{n(n-1)}{2}$ | $\frac{n(n-1)}{2}$ | $\Theta(n^2)$ |
| Sélection | $\frac{n(n-1)}{2}$ | $\frac{n(n-1)}{2}$ | $\Theta(n^2)$ |
| Insertion | $\frac{n(n-1)}{2}$ | $n-1$ | $\Omega(n)$,$\mathcal{O}(n^2)$ |



## Petites nuances

Les notations asymptotiques décrivent avant tout le comportement d'une fonction.
Dans le cas du tri insertion :

- si on note $c_m(n)$ la complexité dans le meilleur des cas, $c_m(n) = n - 1$
- si on note $c_p(n)$ la complexité dans le pire des cas, $c_p(n) = \frac{n(n-1)}{2}$
- si on note $c(n)$ la complexité sans préciser le cas

alors :

- $c_m(n) = \Theta(n)$
- $c_p(n) = \Theta(n^2)$
- $c(n) = \Omega(n)$, et $c(n) = \mathcal{O}(n^2)$



## Calcul sur les relations de comparaison

R0 $g = \mathcal{O}(g)$

R1 $f = \Theta(g) \Rightarrow g = \Theta(f)$

R2 $f = \mathcal{O}(g)$ et $g = \mathcal{O}(h) \Rightarrow f = \mathcal{O}(h)$

R3 $f = \mathcal{O}(g) \Rightarrow \lambda f = \mathcal{O}(g), \lambda \in \mathrm{IR}^{+*}$

R4 $f_1 = \mathcal{O}(g_1)$ et $f_2 = \mathcal{O}(g_2) \Rightarrow f_1 + f_2 = \mathcal{O}(\max(g_1, g_2))$

R5 soient $f_1$ et $f_2$ telles que $f_1 - f_2 \geq 0$,
$f_1 = \mathcal{O}(g_1)$ et $f_2 = \mathcal{O}(g_2) \Rightarrow f_1 - f_2 = \mathcal{O}(g_1)$

R6 soient $f_1$ et $f_2$ telles que $f_1 - f_2 \geq 0$,
$f_1 = \Theta(g_1)$ et $f_2 = \Theta(g_2)$ et si $g_2 = \mathcal{O}(g1)$ et $g_1$ n'appartient pas à $\mathcal{O}(g_2)$, alors $f_1 - f_2 = \mathcal{O}(g_1)$

R7 $f_1 = \mathcal{O}(g_1)$ et $f_2 = \mathcal{O}(g_2) \Rightarrow f_1 \times f_2 = \mathcal{O}(g_1 \times g_2)$

Sauf R5, les règles énoncées pour $\mathcal{O}$ sont aussi valables pour $\Theta$.

## Classes d'équivalence

| Nom de la classe | Comportement asymptotique |
|---|---|
| constant | $\Theta(1)$ |
| logarithmique | $\Theta(\log n)$ |
| linéaire | $\Theta(n)$ |
| | $\Theta(n \log n)$ |
| quadratique | $\Theta(n^2)$ |
| polynomiale | $\Theta(n^k)$ , $k > 0$ fixé |
| exponentiel | $\Theta(k^n)$ , $k > 0$ fixé |



## II Faisons le point

d'un point de vue pratique

- tris non récursifs

d'un point de vue théorique

- notations asymptotique
- expression de la complexité avec ces notations