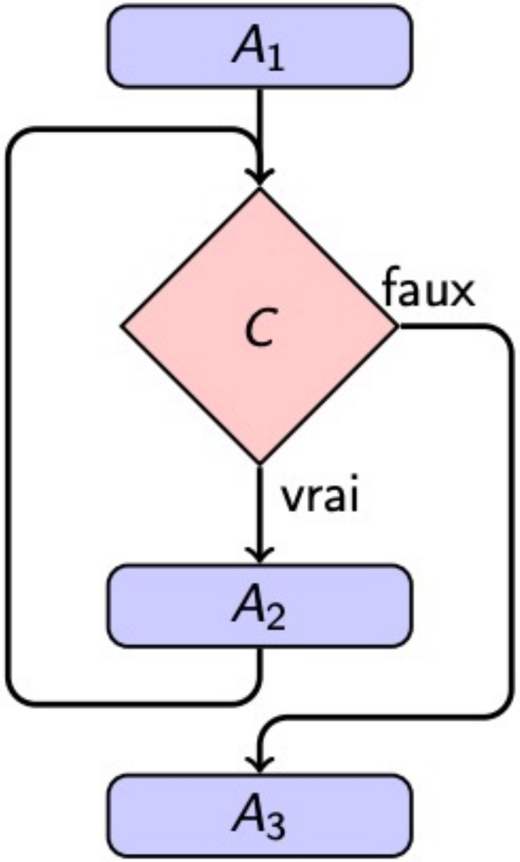
Analyse des algorithmes

Calcul des complexités temporelles pratiques des solutions (0), (1), (2) et (3) :

Les algorithmes ont la même structures :

A_1
tant que C : A_2
A_3



Hypothèse : boucle compilée par un branchement conditionnel et un branchement inconditionnel.

Analyse des 3 algorithmes

Calcul des complexités temporelles pratiques des différentes solutions :

Les quatres algorithmes ont la même structures (for = tant que) :

A_1
tant que C : A_2
A_3

Pour un algorithme donné, soient t_1, t_C, t_2 et t_3 les temps d'exécution respectifs des actions A_1, C, A_2 et A_3.

Temps branchement conditionnel et inconditionnel : t_{BC} et t_{BI}.

Le temps d'exécution est :

$$t_1 + (t_C + t_{\mathrm{BC}} + t_2 + t_{\mathrm{BI}})B(n) + t_C + t_{\mathrm{BC}} + t_3,$$

où $B(n)$ est le nombre de boucles exécutées.

Analyse des algorithmes

- Sur une machine où les opérations sur les entiers s'effectuent en temps constant, le temps d'exécution est donc de la forme :

$$a\,B(n) + b$$

- où a et b sont des constantes.

Analyse des algorithmes

Borne maximale :

Pour les solution (0), (1) et (2) $B(n) \leq n - 2$

Pour la solution (3) $B(n) \leq \lfloor \sqrt{n} \rfloor - 1$

Complexité temporelle maximale :

Pour les solution (0), (1) et (2) $a' n + b'$

Pour la solution (3) $a' \lfloor \sqrt{n} \rfloor + b'$

Analyse des algorithmes

Voici les temps d'exécution mesurés pour quelques nombres à la fois premiers et proches de puissances de 10 :

n	solution 0	solution 1	solution 2	solution 3
11	$1.26 \cdot 10^{-7}$	$1.12 \cdot 10^{-7}$	$1.01 \cdot 10^{-7}$	$8.93 \cdot 10^{-8}$
1 009	$2.14 \cdot 10^{-6}$	$2.26 \cdot 10^{-6}$	$2.41 \cdot 10^{-6}$	$1.55 \cdot 10^{-7}$
100 003	$1.86 \cdot 10^{-4}$	$1.96 \cdot 10^{-4}$	$2.15 \cdot 10^{-4}$	$1.05 \cdot 10^{-6}$
10 000 019	$1.88 \cdot 10^{-2}$	$1.95 \cdot 10^{-2}$	$2.15 \cdot 10^{-2}$	$9.26 \cdot 10^{-6}$
1 000 000 007	$1.85 \cdot 10^{0}$	$1.92 \cdot 10^{0}$	$2.08 \cdot 10^{0}$	$8.88 \cdot 10^{-5}$
$\approx \times 100$	$\approx \times 100$	$\approx \times 100$	$\approx \times 100$	$\approx \times 10$
Théorique	$an + b$	$an + b$	$an + b$	$a\sqrt{n} + b$

Bilan

- Il est très difficile de prévoir le temps de calcul d'un programme.
- En revanche, on peut très bien prévoir comment ce temps de calcul augmente quand la donnée augmente.

Complexité fonction de la Taille

- Recherche d’une notion de complexité robuste,
  - Indépendante de l’ordinateur,
  - du langage de programmation,
  - du compilateur ou de l’interpréteur,
  - etc.
- Complexité exprimée en fonction de la **Taille** de la donnée à traiter.

Opérations élémentaires

- Utilisation d'un modèle de machine simplifiée
- Définition :
  - opération élémentaire est une opération qui prend un temps constant (ou presque).
- Opérations suivantes considérées comme élémentaires :
  - Opérations arithmétiques (additions, multiplications,
  - comparaisons) ;
  - Accès aux données en mémoire ;
  - Sauts conditionnels et inconditionnels ;

Note : discutable en réalité.

Complexité d'un algorithme

- Cout de A sur x :
  - exécution de l'algorithme A sur la donnée x requiert $C_A(x)$opérations élémentaires
- Définitions (cas le pire ; cas moyen)
  - Soit n la taille de la donnée à traiter
  - Dans le pire des cas :
    - $C_A(n) = \max C_A(x)$
  - En moyenne :
    - $C_A^{Moy}(n) = \sum_{|x|\ni n} p_n(x)\ C_A(x)$
    - p_n : distribution de probabilité sur les données de taille n.

Complexité d'un algorithme : PRINCIPE

- Pas de comparaisons du nombre exact d'opérations.
- Comparaison aux constantes près de la **vitesse de croissance** des fonctions qui comptent les nombres d'opérations.

Notations asymptotiques

- Les constantes sont ignorées
- Définitions :
  - $\boldsymbol{O(g)}$ est l'ensemble des fonctions positives f pour lesquelles il existe une constante strictement positive α et un entier n_0 tels que :
    $$f(n) \leq \alpha g(n), \text{ pour tout } n \geq n_0$$
  - $\boldsymbol{\Omega(g)}$ est l'ensemble des fonctions positives f pour lesquelles il existe une constante strictement positive α et un entier n_0 tels que :
    $$f(n) \geq \alpha g(n), \text{ pour tout } n \geq n_0$$
  - $\boldsymbol{\Theta(g)} = O(g) \cap \Omega(g)$

Notations asymptotiques

Notation	Signification	Exemple
$O(f(n))$	Borne supérieure asymptotique	temps $\leq c \cdot f(n)$
$\Omega(f(n))$	Borne inférieure asymptotique	temps $\geq c \cdot f(n)$
$\Theta(f(n))$	Encadrement asymptotique	temps $\approx f(n)$

Notations asymptotiques

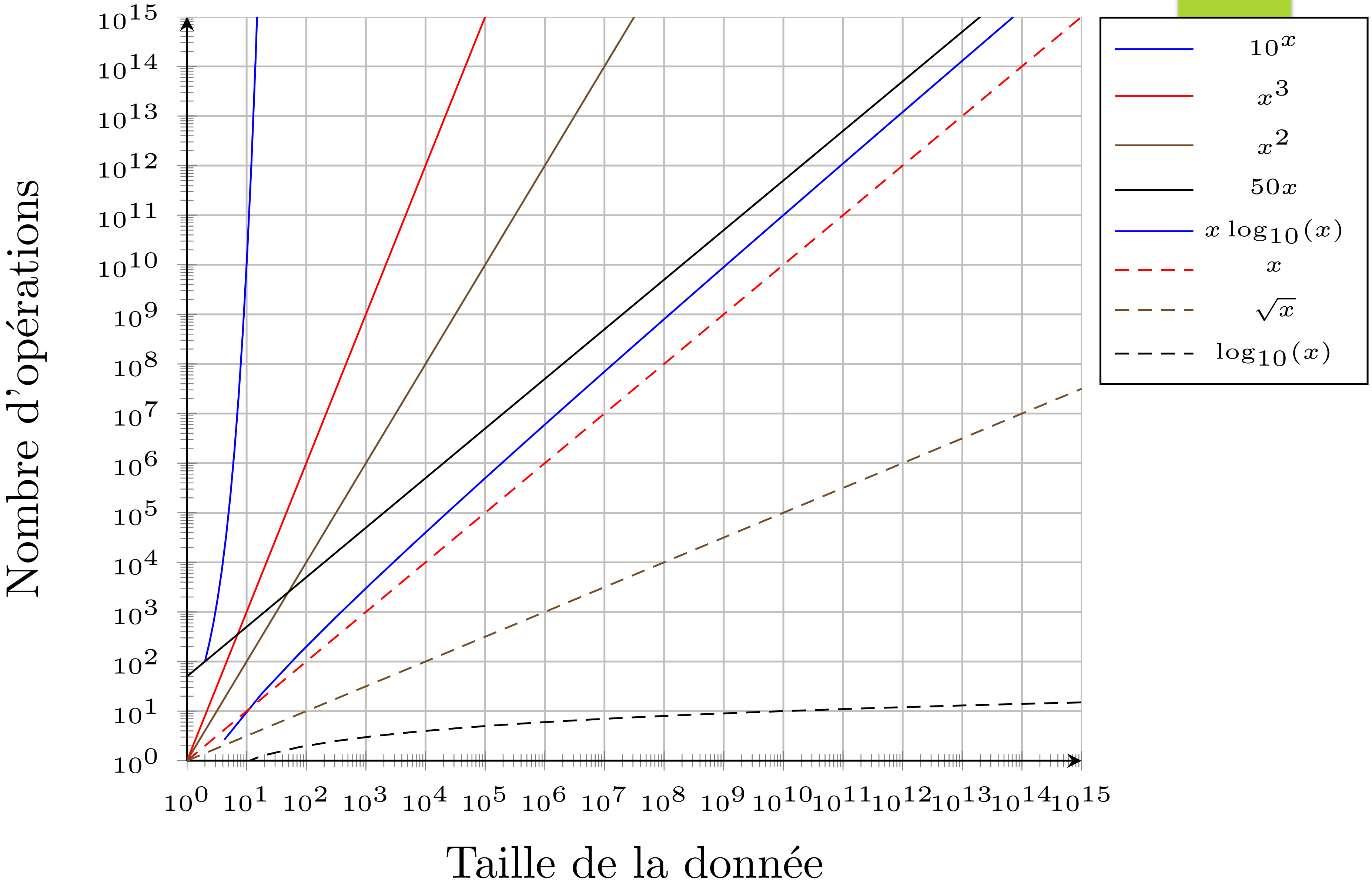
- Par commodité, les expressions « image » des fonctions sont souvent utilisées dans les notations plutôt que leurs symboles.
- On écrit ainsi « $f(n) \epsilon C(g(n))$ » plutôt que « $f \in C(g)$ », où C signifie O, Ω ou Θ.
- Par commodité toujours, on écrit souvent « est » plutôt que « ∈ »
- et on dit souvent « est » plutôt que «appartient ».

Fonctions de référence

- Définitions (désignations des complexités courantes)

notation	désignation	notation	désignation
$\Theta(1)$	*constante*	$\Theta(n^2)$	*quadratique*
$\Theta(\log n)$	*logarithmique*	$\Theta(n^3)$	*cubique*
$\Theta(\sqrt{n})$	*racinaire*	$\Theta(n^k),\ k \in \mathbb{N}, k \geq 2$	*polynomiale*
$\Theta(n)$	*linéaire*	$\Theta(a^n),\ a > 1$	*exponentielle*
$\Theta(n \log n)$	*quasi-linéaire*	$\Theta(n!)$	*factorielle*

- Aucun progrès technologique (modèle de machine standard) ne permet à un algorithme de changer de classe de complexité.



Nombre d'opérations
Taille de la donnée
10^x
x^3
x^2
$50x$
$x\log_{10}(x)$
x
$\sqrt{x}$
$\log_{10}(x)$
10^0 10^1 10^2 10^3 10^4 10^5 10^6 10^7 10^8 10^9 10^{10} 10^{11} 10^{12} 10^{13} 10^{14} 10^{15}

Fonctions de référence

- La complexité d'un algorithme ne parle pas du **temps de calcul absolu** des implémentations de cet algorithme mais de la **vitesse avec laquelle ce temps de calcul augmente** quand la taille des entrées augmente.
- Effacement des constantes : $O(n) = O(2n)$.
- La complexité asymptotique ne permet pas de comparer deux algorithmes différents de même complexité.
- Une telle comparaison ne serait d'ailleurs pas forcément utile, en raison des différences entre les ordinateurs.

Propriétés des notations asymptotiques

Proposition

Soient f, g, h, l *des fonctions positives et* $a, b > 0$.
X *désigne n'importe lequel des opérateur* O, Ω *ou* Θ

- *Si* $f \in X(g)$ *et* $g \in X(h)$ *alors* $f \in X(h)$;
- *Si* $f, g \in X(h)$ *alors* $af + bg \in X(h)$;
- *Si* $f \in X(h)$ *et* $g \in X(l)$ *alors* $fg \in X(hl)$;
- *Si* $f \in \Omega(h)$ *alors pour tout* g *on a* $af + bg \in \Omega(h)$;

Cas des polynômes

Proposition

Un polynôme est de l'ordre de son degré. Plus précisément si

$$P = \sum_{i=0}^{d} c_i x^i$$

avec $c_d \neq 0$ (c'est-à-dire que d est le degré de P) alors

$$P \in \Theta(x^d)$$

- Par exemple, $5x^3 + 3x^2 + 100x + 12 \in \Theta(x^3)$

Récapitulatif

Complexité	Vitesse	Temps	Formulation	Exemple
Factorielle	très lent	proportionnel à N^N	$N!$	Résolution par recherche exhaustive du problème du voyageur de commerce.
Exponentielle	lent	proportionnel à une constante à la puissance N	K^N	Résolution par recherche exhaustive du Rubik's Cube.
Polynomiale	moyen	proportionnel à N à une puissance donnée	N^K	Tris par comparaison, comme le tri à bulle (N^2).
Quasi-linéaire	assez rapide	intermédiaire entre linéaire et polynomial	$N \log(N)$	Tris quasi-linéaires, comme le Quicksort.
Linéaire	rapide	proportionnel à N	N	Itération sur un tableau.
Logarithmique	très rapide	proportionnel au logarithme de N	$\log(N)$	Recherche dans un arbre binaire.
Constante	le plus rapide	indépendant de la donnée	1	recherche par index dans un tableau.

Calcul de complexité dans les structures de contrôle

- Les instructions élémentaires (arithmétique affectations, comparaisons) sont en temps constant, soit en $\Theta(1)$.
- Tests : si $a \in O(A)$, $b \in O(B)$ et $c \in O(C)$ alors

(<u>`Si`</u> `a` <u>`alors`</u> `b` <u>`sinon`</u> `c`) $\in O(A + \max(B, C))$

- Tests : si $a \in \Omega(A)$, $b \in \Omega(B)$ et $c \in \Omega(C)$ alors

(<u>`Si`</u> `a` <u>`alors`</u> `b` <u>`sinon`</u> `c`) $\in \Omega(A + \min(B, C))$

Calcul de complexité dans les structures de contrôle

- Cas des boucles imbriquées
  - Boucles si $a_i \in O(A_i)$ (idem Ω, Θ) alors

    (<u>`Pour`</u> `i` <u>`de`</u> `1 à n` <u>`faire`</u> `a`$_i$) $\in O\sum_{i=1}^{n} A_i$
  - Lorsque A_i est constant égal à A, alors

    (<u>`Pour`</u> `i` <u>`de`</u> `1 à n` <u>`faire`</u> `a`$_i$) $\in nO(A)$
  - Cas particulier : si $A_i \in O(i^k)$ (idem Ω, Θ) alors

    (<u>`Pour`</u> `i` <u>`de`</u> `1 à n` <u>`faire`</u> `a`$_i$) $\in O(n^{k+1})$