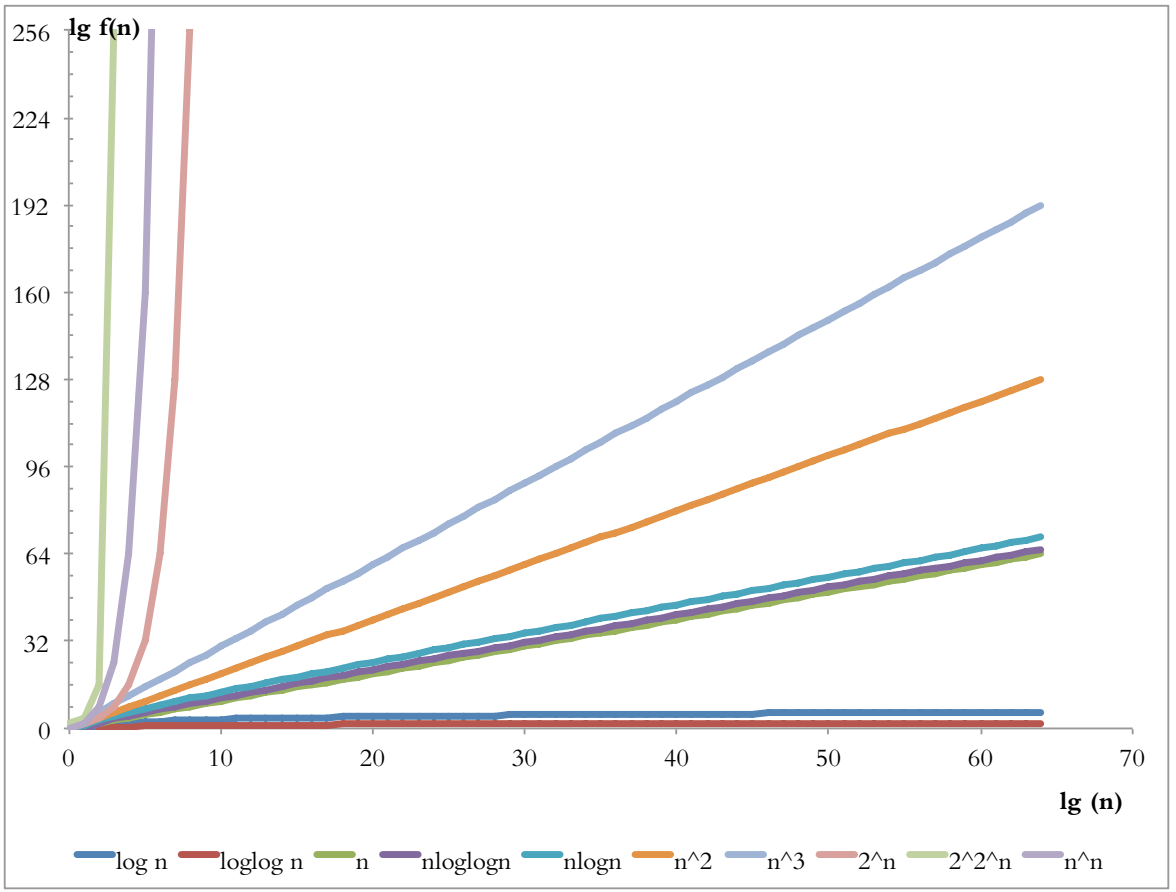
Au de-là de l'échelle standard

Échelle standard

ordre	$f(n)$
constante	1
logarithmique	$\log n$
linéaire	n
linéarithmique	$n \log n$
quadratique	n^2
cubique	n^3
polynomial	$n^a \log^b n$
exponentiel	A^n ; $A > 1$

Ordres de croissance

épère log-log :



Expansion asymptotique

on a une fonction $f(n)$ et on veut caractériser sa croissance sur l'échelle standard :

$$f(n) \sim c_0 g_0(n) + c_1 g_1(n) + c_2 g_2(n) + \cdots$$

avec $g_{k+1} = o(g_k)$ (g_k «négligeable» par rapport à g_k)

précision croissante selon le but de la caractérisation :

$$\begin{aligned} f &= O(g_0) \\ f &= c_0 g_0 + O(g_1) \\ f &= c_0 g_0 + c_1 g_1 + O(g_2) \\ &\cdots \end{aligned}$$

O : info sur la grandeur de termes omis

$\Rightarrow$ approximation des croissances

Logarithme itéré

composition :

$$\lg \lg n = \lg(\lg(n))$$

logarithme itéré

$$\lg^* n = \begin{cases} 0 & \text{si } n \leq 1 \\ 1 + \lg^*(\lg n) & \text{si } n > 1 \end{cases}$$

c'est une fonction très lente (mais à croissance monotone)

Observation pragmatique : $\lg^* n \leq 5$ pour tout $n \leq 2^{65536}$

Coût amorti de l'union-find

Thm. En utilisant union-par-rang et compression de chemin (ou compression par réduction à moitié), une séquence de m opérations sur n éléments prend $O(m \lg^* n)$ temps, où $\lg^*$ dénote le logarithme itéré.

Hyperopérateurs

Fonction de tour / tétration : $2 \uparrow\uparrow p$ = exponentiation p fois :

$$2 \uparrow\uparrow p = 2^{2\uparrow\uparrow p-1}$$

Pentation : $2 \uparrow\uparrow\uparrow p = 2 \uparrow\uparrow (2 \uparrow\uparrow\uparrow p - 1)$

Ackermann

Ackermann $A(i,j)$ avec $i,j \geq 1$ [définition de Tarjan]

$$A(i,j) = \begin{cases} 2^j & \text{si } i = 1; \\ A(i-1,2) & \text{si } j = 1 \text{ et } i \geq 2 \\ A(i-1,A(i,j-1)) & \text{si } i,j \geq 2 \end{cases}$$

Ackermann inverse ($m \geq n \geq 1$)

$$\alpha(m,n) = \min\{i : A(i, \lfloor m/n \rfloor) \geq \lg n\}$$

c'est une fonction de croissance très lente

Observation pragmatique : $\alpha(m,n) \leq 3$ pour tout $n < 65536$ et $\alpha(m,n) \leq 4$ pour tout $n < 2^{2^{\cdots^2}}$ (exponentiation 16 fois)

Union-find

Thm. En utilisant union-par-rang et compression de chemin (ou compression par réduction à moitié), une séquence de m opérations sur n éléments prend $O(m\alpha(m,n))$ temps.

Fonction de Radó

Un algorithme implementé en un langage de programmation (ou par une machine de Turing universelle) correspond à un encodage sur L octets (p.e., Java bytecode avec toutes dépendances).

On considère les algorithmes encodés sur L octets qui affiche un nombre entier : algorithme a, encodé sur $|a| = L$ octets, affiche $x(a)$.

On considère le plus grand entier qu'un programme de longueur L peut afficher

$$X(L) = \max_{|a| \leq L} \Big\{ x(a) \Big\}.$$

```
public abstract BigInt plusGrandNombre(long L); // à implémenter??
```

Croissance incalculable

On peut alors construire un autre programme qui exécute $X(L)$ et affiche $X(N)+1$.

```
private static final long L = ... ;
public BigInt champion()
{
    BigInt x = plusGrandNombre(L);
    BigInt x1 = add1(x1);
    return x1;
}

private BigInt add1()
{
...
}
```

BigInt

on veut travailler avec des entiers ≤ 0 de taille arbitraire

comme bitstream : poids moindre significatif vers plus significatif,
bits doublés jusqu'au plus lourd :

$19 = \overline{10011}$ *représentation binaire*

$\rightarrow$ 1 1 | 1 1 0 0 0 0 | 1 0 *un encodage de bitstream*

(1 1 : doublés ; 1 0 : fin)

add1 : avec bit carry

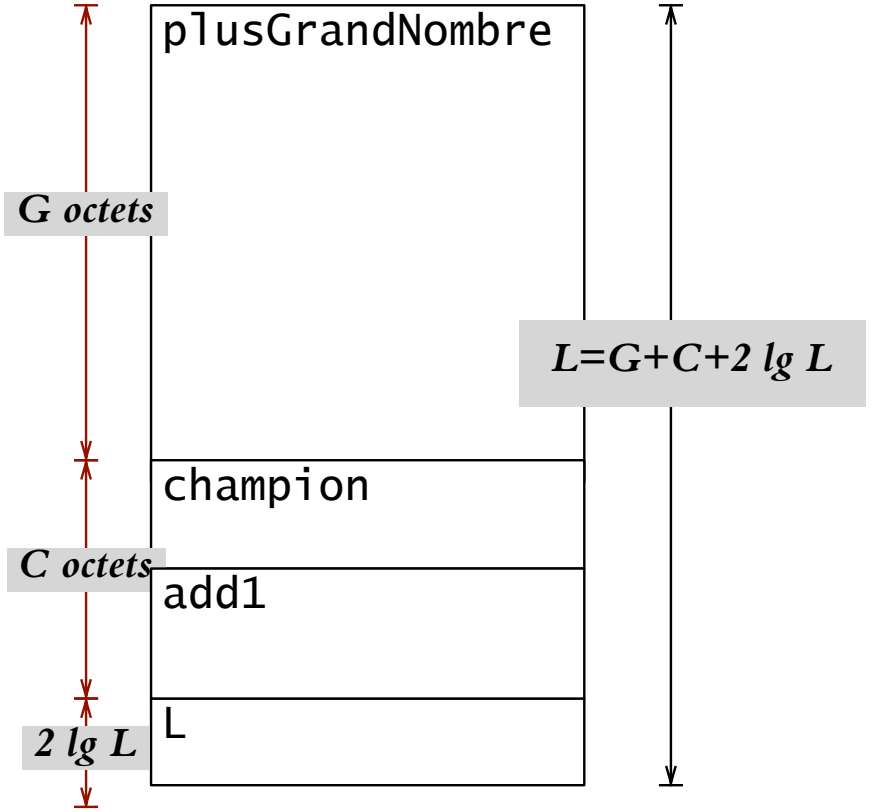
$19 + 1 \rightarrow$ 1 1 1 1 0 0 0 0 1 0 $+ 1$

$=$ 0 0 0 0 1 1 | 0 0 1 0

$\rightarrow$ carry bit jusqu'ici

$\Rightarrow x$ representé sur $\sim 2 \lg(n)$ bits ($\sim \lg n$ aussi possible)

Incalculabilité



Paradoxe : champion est un programme encodé sur L octets qui affiche un nombre plus grand que tout programme de longueur L,
incluant soi-même

$\Rightarrow$ plusGrandNombre(n) est une fonction incalculable !

construction de paradoxe similaire pour croissance trop lente : il n'existe pas de programme qui peut décider si une fonction calculable reste constante ou non

Gödel-Turing

Gödel : dans tout système formel (logique), il existe des énoncés dont on ne peut décider la validité

Church-Turing : fonctions incalculables

⋆ peut-être le cerveau humain est un ordinateur quantique (Roger Penrose)
⋆ réseaux neuronaux aussi puissant que machine de Turing
⋆ ordinateur quantique en réalité (Giles Brassard+. . .)

Turing/Gödel reste valide même pour ordinateur quantique, . . .
et il y a des problèmes indécidables dans tout modèle de calcul / système de logique formel

⇒ limites de l'inférence de connaissances du monde réel !

Chaitin's Ω

on considère tout algorithme dans l'ordre lexicographique de leur encodage :

$$a_0, a_1, a_2, \ldots$$

$$\omega = \overline{0.b_0 b_1 b_2 \cdots}$$

$$b_i = \begin{cases} 1 & \text{si } a_i \text{ est «fonctionnel»} \\ 0 & \text{sinon} \end{cases}$$

«fonctionnel» : calcule quelque chose / a_i se termine (aucune boucle infinie)

ω est bien défini mais incalculable
$\longrightarrow$ un seul nombre $0 < \omega < 1$ d'omniscience :-)