Introduction à l’algorithmique
- première partie -

Marian SCUTURICI

Analyse d'un algorithme

- Exemple : calculer la somme de n premiers entiers positifs $\sum_{i=1}^{n} i$

Procédure $somme(n, sum)$

  Entrée : entier n

  Sortie : entier sum

  début

    $sum \leftarrow 1;$

    pour $(i \leftarrow 2; i \leq n; i \leftarrow i + 1)$ **faire**

      $sum \leftarrow sum + i;$

Coût : ?

Coût : 1+1+(n-1)*3+1 = 3*n

Analyse d'un algorithme

- Calculer la somme de n premiers entiers positifs

Procédure $somme(n, sum)$
    Entrée : entier n
    Sortie : entier sum
    début
        $sum \leftarrow n * (n + 1);$
        $sum \leftarrow sum/2;$

Coût : ? **Coût : 2** **2 << 3*n !**

Analyse d’un algorithme

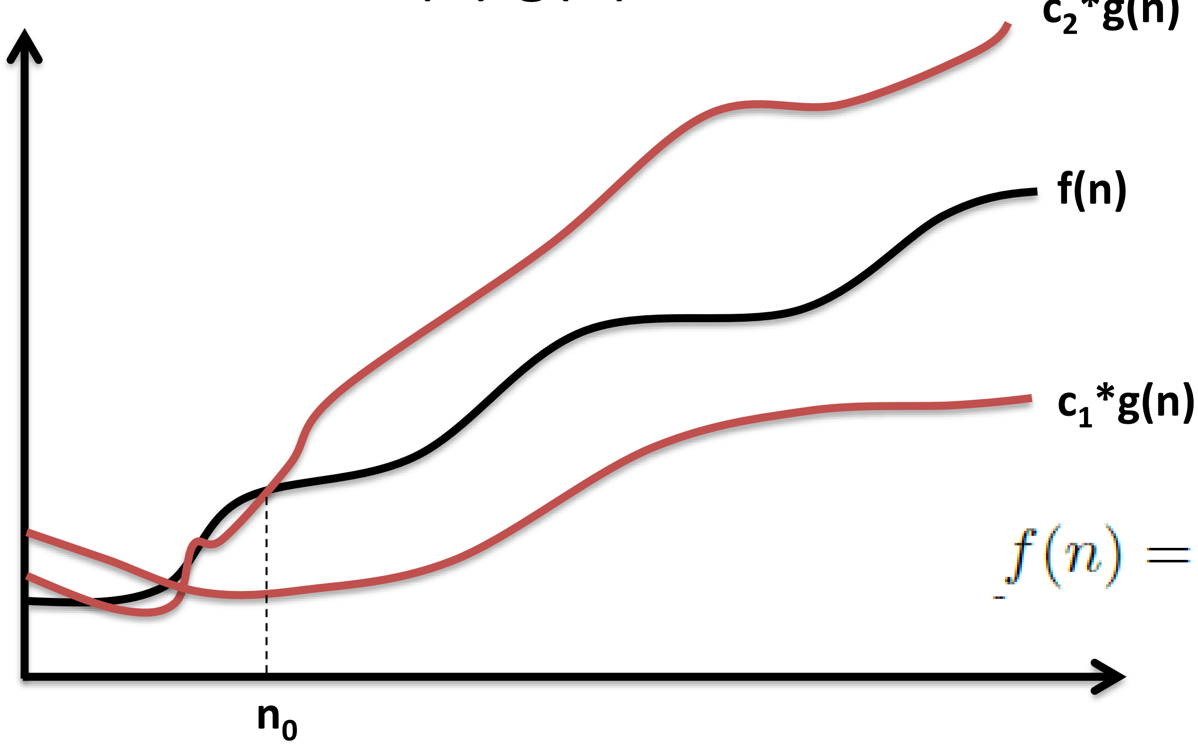
- Paramètres de qualité
  - Algorithme correct !!! – éventuellement :
    - Pseudo-correct
    - Approximatif
  - Analyse du coût temporel (complexité)
  - Analyse du coût mémoire

Analyse du temps de calcul

- Basée sur le nombre d'instructions exécutées par un algorithme par rapport aux paramètres en entrée
  - Fonction f(entrée)
  - Exemples :
    - 3*n+1
    - 24
    - 2*n+m+p
- Problème : comment comparer ces fonctions ?

Notation Θ

- Donne un ordre de grandeur pour le temps d'exécution d'un algorithme
  - #instructions = f(n),g(n)



$$f(n) = \Theta(g(n))$$

Notation Θ

- D'une manière formelle :

$$\Theta(g(n)) = \{f(n) | \exists c_1, c_2 > 0, n_0 \in \mathbb{N} \; telles \; que$$
$$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n), \forall n \geq n_0\}$$

- Convention

  si $f(n) \in \Theta(g(n))$ alors $f(n) = \Theta(g(n))$

Exemple

$$\frac{1}{2} * n^2 - 3 * n = \Theta(n^2)$$

- Il faut trouver c_1, c_2, n_0 telles que :

$$c_1 * n^2 \leq \frac{1}{2} * n^2 - 3 * n \leq c_2 * n^2, \forall n \geq n_0$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

$$c_1 = \frac{1}{14}, c_2 = \frac{1}{2} \text{ et } n_0 = 7$$

Exemple

$$6 * n^3 \neq \Theta(n^2)$$

- Si c_2, n_0 existent tel que :

$$6 * n^3 \leq c_2 * n^2$$

- alors :

$$n \leq \frac{c_2}{6}$$

- Impossible ! (c_2 constante)

Notation Θ

- Propriétés :

transitivité : $f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
réflexivité : $f(n) = \Theta(f(n))$
symétrie : $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

Exemples

$$2 + 15 = \Theta(1)$$
$$2n + 1 = \Theta(n)$$
$$n^2 - 2n + 1 = \Theta(n^2)$$
$$\sum_{i=0}^{k} c_i * n^i = \Theta(n^k)$$
$$2^n + n^2 = \Theta(2^n)$$

Exercices

- Quelle complexité (Θ) pour :

$$n^3 + 8 * n^5 - 2n + 1$$

$$n^3 + n! + 12$$

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$n^2 + m^3 + p$$

$$\log(n) * n^2 + n^3$$

$$\log(n) + \log(n^2)$$

$$\log(n) * \log(n) * n^2 + 2^n$$

$$2^n + 2^{4*n} + n^2$$

Conception d'un algorithme

- Objectif : trouver des algorithmes avec un Θ le plus petit …

$$\leftarrow$$

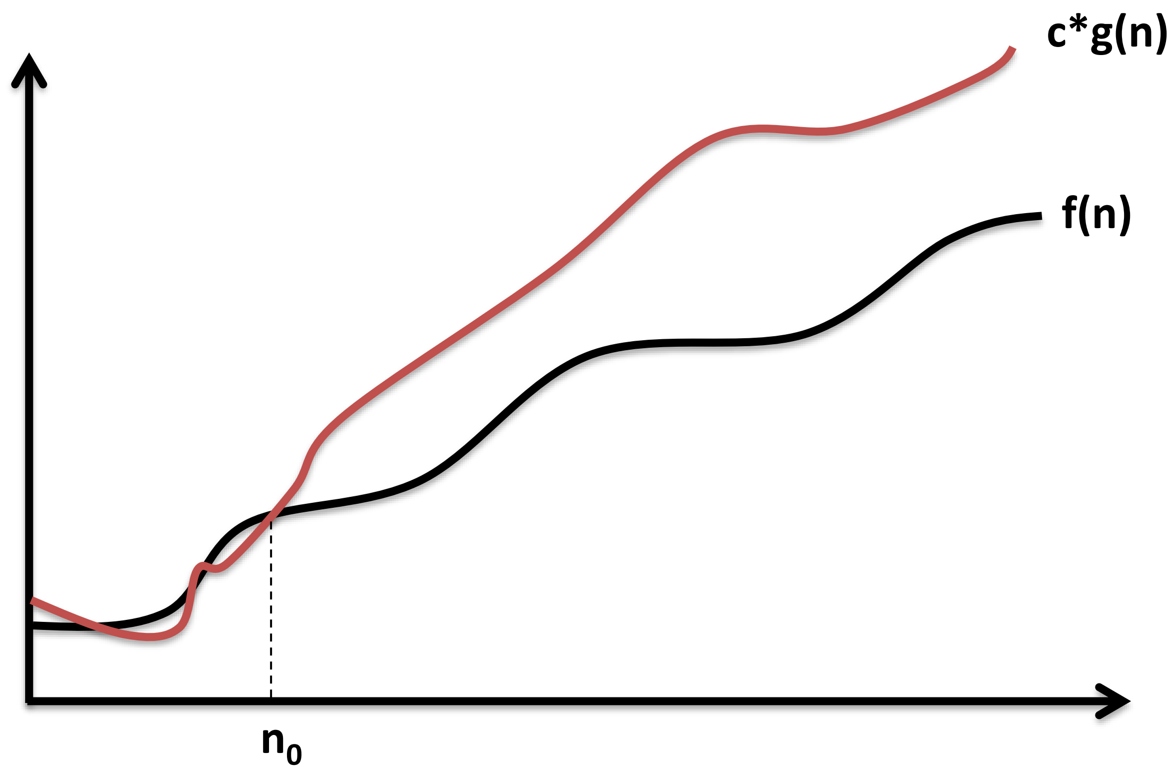
$$\Theta(1) < \Theta(log(n)) < \Theta(n) < \Theta(n * log(n)) < \ldots$$

$$< \Theta(n^2) < \ldots < \Theta(n^k) < \ldots < \Theta(2^n) < \ldots$$

- Problème : souvent très difficile de trouver Θ !

Notation O

- Version moins stricte que la notation Θ



Notation O

- D'une manière formelle :

$$\mathcal{O}(g(n)) = \{f(n) | \exists c > 0, n_0 \in \mathbb{N}\ telles\ que$$
$$0 \leq f(n) \leq c * g(n), \forall n \geq n_0\}$$

- Convention

si $f(n) \in \mathcal{O}(g(n))$ alors $f(n) = \mathcal{O}(g(n))$

Exemples

$$n^3 - 2n + 1 = \mathcal{O}(n^3)$$

Mais aussi :

$$n^3 - 2n + 1 = \mathcal{O}(2^n)$$

Propriétés

: si un algorithme à une complexité $\Theta(f(n))$,

alors il est aussi de complexité $\mathcal{O}(f(n))$,

mais la réciproque n'est pas valable.

Exercices

- Vrai ou faux ?

A $n^3 + 8 * n^2 - 2n + 1 = \mathcal{O}(n^4)$

B $n^3 + 8 * n^2 - 2n + 1 = \mathcal{O}(n^3)$

C $n^3 + 8 * n^2 - 2n + 1 = \mathcal{O}(n^2)$

D $n^3 + 8 * n^2 - 2n + 1 = \mathcal{O}(2^n)$

Exercices

- Vrai ou faux ?

A $n^2 + m^3 + p = \mathcal{O}(2^n)$

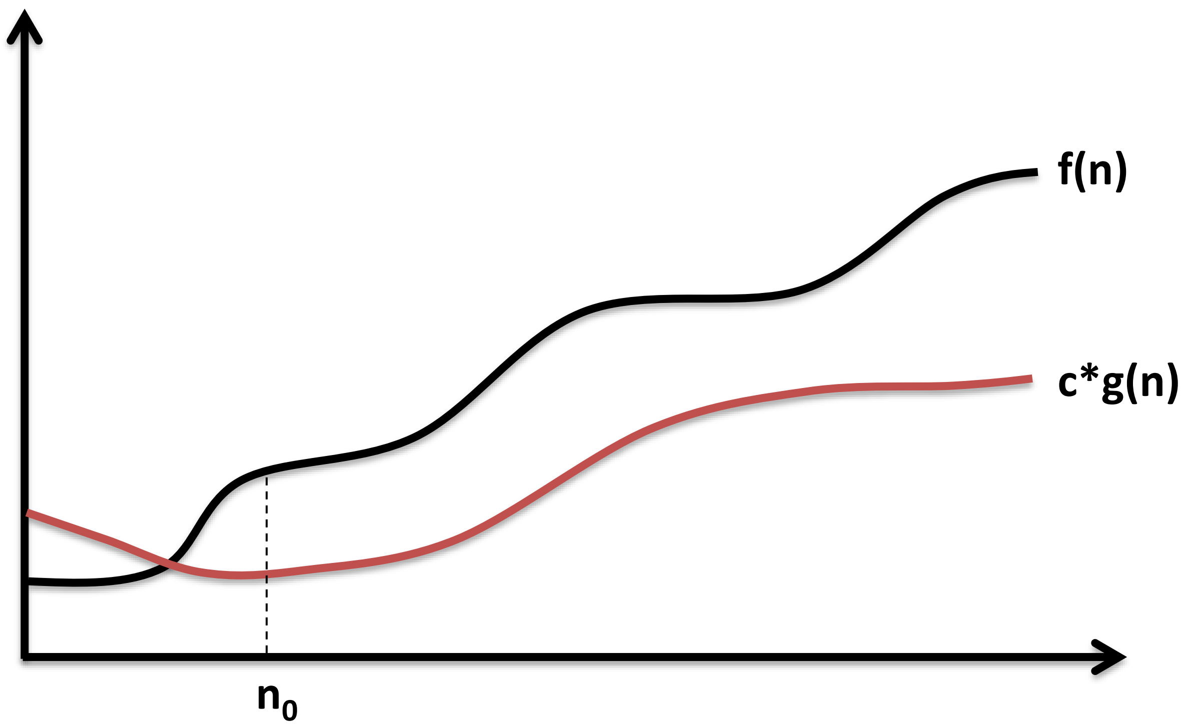
B $n^2 + m^3 + p = \mathcal{O}(n^2 + m^4 + p * \log(p))$

C $n^2 + m^3 + p = \mathcal{O}(2^n + 2^m + 2^p)$

D $n^2 + m^3 + p = \mathcal{O}(n^2 * m^3 * p)$

Notation Ω

- Version moins stricte que la notation Θ



$$f(n) = \Omega(g(n))$$

Notation Ω

- D'une manière formelle :

$$\Omega(g(n)) = \{f(n) | \exists c > 0, n_0 \in \mathbb{N}\, telles\ que$$
$$0 \leq c * g(n) \leq f(n), \forall n \geq n_0\}$$

Exemples

$$n^3 + 8 * n^2 - 2n + 1 = \Omega(n^3)$$

Mais aussi :

$$n^3 + 8 * n^2 - 2n + 1 = \Omega(n^2)$$

Exercices

- Vrai ou faux ?

A $\quad n^3 + 8 * n^2 - 2n + 1 = \Omega(n^4)$

B $\quad n^3 + 8 * n^2 - 2n + 1 = \Omega(n^3)$

C $\quad n^3 + 8 * n^2 - 2n + 1 = \Omega(n^2)$

D $\quad n^3 + 8 * n^2 - 2n + 1 = \Omega(2^n)$

Exercices

- Vrai ou faux ?

A $n^2 + m^3 + p = \Omega(2^n)$

B $n^2 + m^3 + p = \Omega(n^2 + m^4 + p * \log(p))$

C $n^2 + m^3 + p = \Omega(2^n + 2^m + 2^p)$

D $n^2 + m^3 + p = \Omega(n + m + p)$

Différents ordres de complexité …

Function $f(n)$		Input size n			
		5	$5^2 = 25$	$5^3 = 125$	$5^4 = 625$
Constant	1	1	1	1	1
Logarithmic	$\log_5 n$	1	2	3	4
Linear	n	$\mathbf{1}$	$\mathbf{5}$	$\mathbf{25}$	$\mathbf{125}$
Linearithmic	$n \log_5 n$	1	10	75	500
Quadratic	n^2	1	$5^2 = 25$	$5^4 = 625$	$5^6 = 15,625$
Cubic	n^3	1	$5^3 = 125$	$5^6 = 15,625$	$5^9 = 1,953,125$
Exponential	2^n	$\mathbf{1}$	$\mathbf{2^{20} \approx 10^6}$	$\mathbf{2^{120} \approx 10^{36}}$	$\mathbf{2^{620} \approx 10^{187}}$

Différents ordres de complexité …

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Dominance asymptotique

f(n) >> g(n) si : $\lim_{n\to\infty} g(n)/f(n) = 0$

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n\log n \gg n \gg \log n \gg 1$$

Un détail concernant la complexité …

- Les différents mesures (Θ, Ω, O) sont utiles pour des valeurs (très) importantes de différents paramètres d’entrée !

  – Il est possible que le choix du n_0 soit critique …

Rappel

- Un algorithme a une bonne qualité si :
  - Il est correct
  - Il a une « bonne » complexité temporelle
    - Mesurée avec les notations O ou Θ
  - Il a un coût mémoire « raisonnable »

?

Tableaux

- Déclaration :

  type-elem nom-tableau[d..f]

- Exemple :

  Déclaration : entier tab1[1..10]
  réel tab2[12..44]

- Accès à un élément :

  $$vec[1]$$

- Chaîne de caractères

  Déclaration : car chaine[0..1024]

Exemple – min(tableau)

Procédure *min*($vec, result$)

Entrée : entier[1..taille] vec

Sortie : entier $result$, correspondant à $min(vec[1], vec[2], ..., vec[taille])$

Précondition : $taille \geq 1$

Postcondition : $result \in vec$

début

$result \leftarrow vec[1];$

pour $i \leftarrow 2; i \leq taille; i \leftarrow i + 1$ **faire**

si $vec[i] < result$ **alors**

$result \leftarrow vec[i];$

Coût : ?

Complexité : ?

Problème - Fibonacci

- Complétez l'algorithme …

1 **Procédure** *fibonacci*($n, fibo$)

Entrée : entier n

Sortie : entier $fibo[0..n]$

Précondition : $n \geq 1$

Postcondition : $fibo[0] = 1$

$fibo[1] = 1$

$fibo[n] = fibo[n-1] + fibo[n-2], \forall n \geq 2$

Déclaration : entier i

Fibonacci

Procédure *fibonacci*(n, $fibo$)

Entrée : entier n

Sortie : entier $fibo[0..n]$

Précondition : $n \geq 1$

Postcondition : $fibo[0] = 1$

$fibo[1] = 1$

$fibo[n] = fibo[n-1] + fibo[n-2], \forall n \geq 2$

Déclaration : entier i

début

$fibo[0] \leftarrow 1$

$fibo[1] \leftarrow 1$

$i \leftarrow 1$

tant que $i < n$ **faire**

// invariant : $\forall j \in [2..i],\ fibo[j] = fibo[j-1] + fibo[j-2]$

$i \leftarrow i + 1$

$fibo[i] \leftarrow fibo[i-1] + fibo[i-2]$

Coût : ? **Complexité : ?**

Problème - Crible d'Eratosthène

- 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
- 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~
- 2 3 ~~4~~ 5 ~~6~~ 7 ~~8 9 10~~ 11 ~~12~~ 13 ~~14 15 16~~ 17 ~~18~~ 19 ~~20 21 22~~ 23 ~~24~~ 25 ~~26 27 28~~ 29 ~~30~~
- 2 3 ~~4~~ 5 ~~6~~ 7 ~~8 9 10~~ 11 ~~12~~ 13 ~~14 15 16~~ 17 ~~18~~ 19 ~~20 21 22~~ 23 ~~24 25 26 27 28~~ 29 ~~30~~
- 2 3 5 7 11 13 17 19 23 29

Crible d'Eratosthène

Procédure *crible*($n, crible$)

Entrée : entier n

Sortie : logique $crible[1..n]$

Précondition : $n \geq 2$

Postcondition : pour tout $i \in [1..n]$, $crible[i] = vrai$ ssi i est un nombre premier

Déclaration : entier i, j

Crible d'Eratosthène

Procédure *crible*$(n, crible)$

Entrée : entier n

Sortie : logique $crible[1..n]$

Précondition : $n \geq 2$

Postcondition : pour tout $i \in [1..n]$, $crible[i] = vrai$ ssi i est un nombre premier

Déclaration : entier i, j

début

  pour $(i \leftarrow 1\,;\, i \leq n\,;\, i \leftarrow i + 1$ **faire**

    $crible[i] \leftarrow vrai$

  pour $(i \leftarrow 2\,;\, i < n\,;\, i \leftarrow i + 1)$ **faire**

    si $crible[i]$ **alors**

      pour $(j \leftarrow i + i\,;\, j \leq n\,;\, j \leftarrow j + i)$ **faire**

        $crible[j] \leftarrow faux$

Coût : ? **Complexité : ?**

Crible d'Eratosthène

- Complexité :

$$n * (1 + 1/2 + 1/3 + 1/5 + \ldots + 1/n)$$

- mais :

$$\lim_{n \to +\infty} (\sum_{p \leq n} (\frac{1}{p}) - \log\log(n)) = M$$

$$M = 0.26147\ldots$$

- donc :

$$\mathcal{O}(n * \log\log(n))$$

?

Problème

- Proposez un algorithme pour calculer l'aire de l'intersection de deux intervalles [a1; b1] et [a2; b2].
  - a1; b1; a2; b2 sont des nombres réels, et le résultat est un réel.

- Le même problème pour l'aire de l'intersection des deux ensembles d'intervalles (n et m).
  - complexité ?