Complexité algorithmique

Complexité

- Exécution d ’un programme →utilisation des ressources de l ’ordinateur
  - **temps de calcul** pour exécuter les opérations
  - **occupation mémoire** (programme + données)
- Mesurer ces 2 grandeurs pour comparer entre eux différents algorithmes
- « Sur toute machine, quelque soit le langage de programmation, l ’algorithme A1 est meilleur que A2 pour des données de grande taille »

Temps d ’exécution d ’un programme

- Mettre en évidence les opérations fondamentales: **le temps est proportionnel au nombre de ces opérations**
  - Recherche d ’un élément E dans une liste L:
    nombre de comparaisons entre E et les éléments de L
  - Recherche d ’un élément sur disque:
    nombre d ’accès à la mémoire secondaire
  - Multiplication de matrices:
    nombre de multiplications et d ’additions

Temps d ’exécution d ’un programme

- Soit n la taille des données soumise au programme
  - taille de la liste dans laquelle on cherche E
  - taille de la liste à trier
  - dimensions d ’une matrice dont on calcule le carré
  - …

- **T(n) : temps d ’exécution du programme**
  - T(n) fonction de N dans R

Décompte du nombre d ’opérations fondamentales

- P: nombre d ’instructions dans un bloc de programme
- P(séquence) = $\sum$ P(E) ; E: élément de la séquence
- P(*if C then I1 else I2*) P(C) + max(P(I1), P(I2))
- P(boucle) = $\sum$ P(I_i) ; I_i = Ième itération de la boucle
- P(F(n)) s ’écrit en fonction de P(F(k)) pour une procédure ou fonction récursive → résolution d ’équations de récurrences

Temps d ’exécution d ’un programme

- Pour certains problèmes, le temps ne dépend pas seulement de la taille de la donnée mais aussi de la donnée elle-même

- Complexité au mieux
  $T_{min}(n) = \min(\text{Temps}(d) |\ d \text{ donnée de taille } n)$
- Complexité au pire
  $T_{max}(n) = \max(\text{Temps}(d) |\ d \text{ donnée de taille } n)$
- Complexité en moyenne
  $T_{moy}(n) = \sum_{d:\text{donnée de taille } n} p(d) * \text{Temps}(d)$

Temps d ’exécution d ’un programme

- $T_{min}(d) <= T_{moy}(d) <= T_{max}(d)$

- Un exemple de calcul: recherche séquentielle d'un élément X dans un tableau L

L:tableau [1..n] de Element

X: élément

Début

i ← 1

Tantque i ≤ n et (L[i] ≠X) faire

    i ← i+1

Fintantque

Si i>n alors i ← 0

Fin

Ordre de grandeur asymptotique: la notation *O*

- T(n) pour n grand
- Comparaison des ordres de grandeur asymptotique
- *Définition:*

  Soit f et g deux fonctions de N dans R^{*+},

  f= *O* (g) (f est en grand O de g) ssi

  $\exists\ c \in R^{*+}, \exists\ n_0$ **tq** $\forall\ n > n_0, f(n) \leq c\ g(n)$

- Ex: f(n)=10n et g(n)=$O(n^2)$

Equivalent: la notation Θ

- *Définition:*

  f= Θ(g) ssi f= $\mathcal{O}$(g) et g= $\mathcal{O}$(f)

  c'est-à-dire

  $\exists\ c \in \mathbf{R}^{*+}, \exists\ d \in \mathbf{R}^{*+}, \exists\ n_0$ **tq**

  $\forall\ n > n_0$, **d g(n) ≤ f(n) ≤ c g(n)**

Ex: 2n = Θ(n) mais 2n n'est pas en Θ(n²)

Propriétés de le notation *O*

- Les constantes ne sont pas importantes
- Les termes d 'ordre inférieur sont négligeables

  Si $T(n) = a_k n^k + \ldots + a_1 n + a_0$ avec $a_k > 0$

  alors $T(n)$ est en (n^k)

$$\text{Si } \frac{h(n)}{g(n)} \xrightarrow[n \to \infty]{} 0 \text{ alors } g(n) + h(n) = O(g)$$

- Si $T1(n) = O(f1)$ et $T2(n) = O(f2)$

  alors $T1(n) + T2(n) = O(f1 + f2)$

  et $T1(n) * T2(n) = O(f1 * f2)$

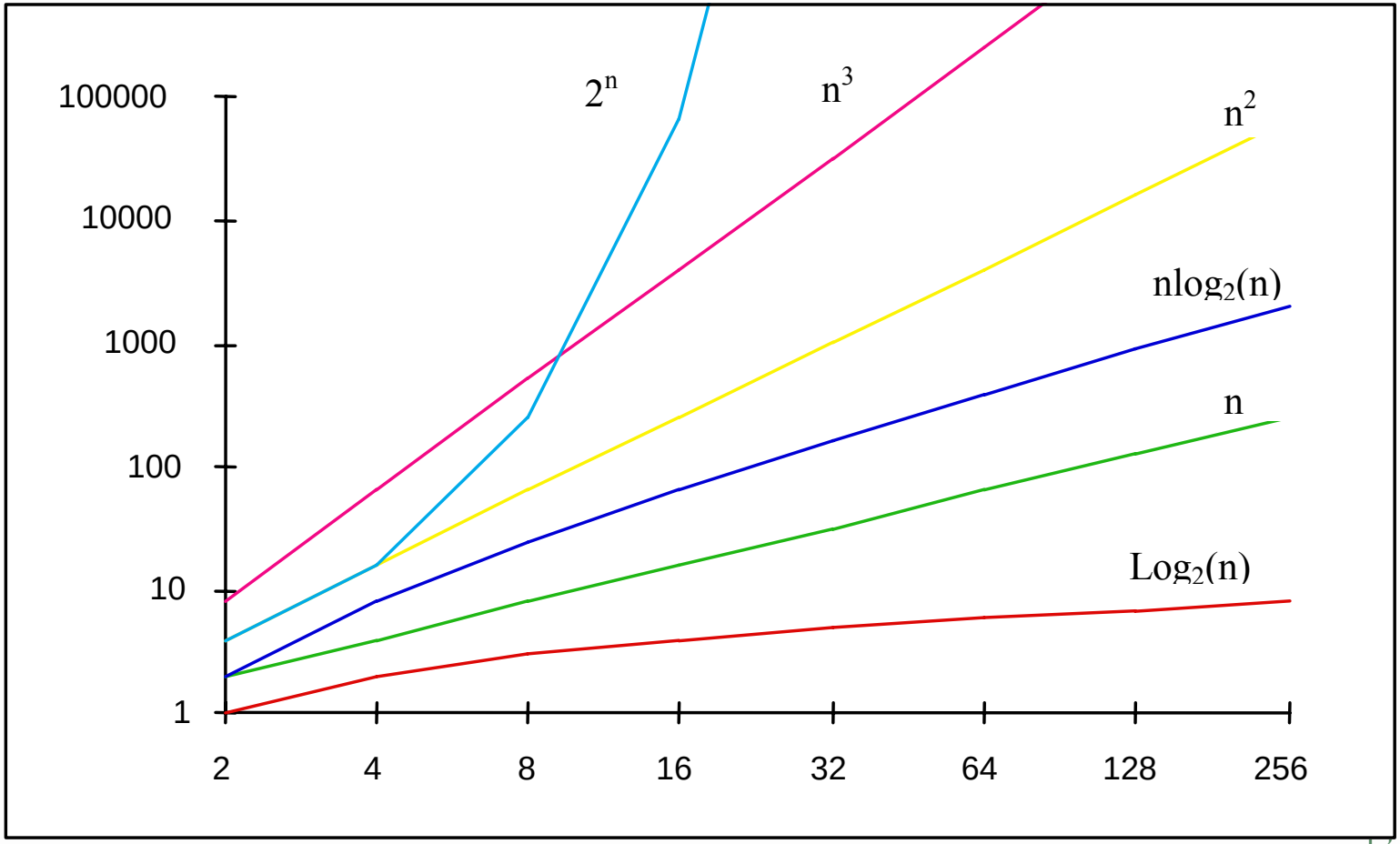
- Si $f(n) = O(g)$ et $g(n) = O(h)$

  alors $f(n) = O(h)$

Appellation des ordres de grandeur courant

- $\mathcal{O}(1)$: constante
- $\mathcal{O}(\log(n))$: logarithmique
- $\mathcal{O}(n)$: linéaire
- $\mathcal{O}(n \log n)$: n log n
- $\mathcal{O}(n^2)$: quadratique
- $\mathcal{O}(n3)$: cubique
- $\mathcal{O}(2^n)$: exponentielle

Croissance comparée des fonctions fréquemment utilisées en complexité



Comparaisons des fonctions usuelles

- $\log(n) = O(n)$
- $n^{1/2} = O(n)$
- $n^k = O(n^{k+1})$ pour tout $k \geq 0$
- $P(n) = O(n^k)$ pour tout polynôme P de degré $\leq k$
- $n^k = O(2^n)$ pour tout $k \geq 0$
- $2^n = O(n^n)$

(ouvrons une parenthèse

Rappels d'analyse pour classer les fonctions

Etudier de la limite de f /g pour comparer f et g

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & \mathrm{f(n)}\in\mathrm{O(g(n))\ mais\ g(n)}\notin\mathrm{O(f(n))}\\ c\ \text{implique} & \mathrm{f(n)}\in\mathrm{O(g(n))\ et\ g(n)}\in\mathrm{O(f(n))}\\ \infty & \mathrm{f(n)}\notin\mathrm{O(g(n))\ mais\ g(n)}\in\mathrm{O(f(n))}\end{cases}$$

Règle de L'Hôpital:

Si f et g sont dérivables et si $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} g(x)$

Alors $\lim_{x\to\infty}\frac{f(x)}{g(x)} = \lim_{x\to\infty}\frac{f'(x)}{g'(x)}$ si cette limite existe

Et aussi la formule de Stirling

fin de la parenthèse)

- à propos de n!

$$n! \sim \sqrt{2\pi n}\, n^n\, e^{-n}$$

Rapport Temps / Taille des données

Complexités	1	$\log_2(n)$	n	$n\log_2(n)$	n^2	n^3	2^n
Evolution du temps t quand la taille des données est multipliée par 10	t	t+3,32	10 t	(10+ε)t	100 t	1000 t	t^{10}
Evolution de la taille quand le temps alloué est multiplié par 10	∞	n^{10}	10 n	(10-ε)t	3,16 n	2,15 n	n+3,32

Calcul de la complexité

- Toute lecture, écriture, affectation (sans appel de fonction) a un temps en $O(1)$
- Complexité d 'une séquence=complexité de l 'instruction de plus forte complexité
- Si C alors A1 sinon A2

  $O(c(n))$ $O(f1(n))$ $O(f2(n))$

  $T(n) = O(c(n) + \max(f1(n), f2(n)))$

Calcul de la complexité

- Pour i de A à B faire
  Iter$_i$
  Finpour
- Si T(Iter$_i$) indépendant de i en Θ(f(n)), alors T(n) = O((B-A+1) f(n))
- Si T(Iter$_i$) dépendant de i: $T_{iter}(n,i)$ alors

$$T(n)=O(\sum_{i=A}^{B} T_{\text{iter}}(n,i))$$

Calcul de la complexité

- Tant que C faire Iter Fintantque

- La difficulté est de déterminer une borne sup pour le nombre d ’itérations:NB
- Si T(Iter)= O(f(n))

  $T(n) = O(NB*(C(n) + f(n)))$
- Si T(Iter) dépend de l ’itération i

  $T(n) = O(NB*C(n) + \sum T_{iter}(n,i)$

Calcul de la complexité

- Procédures et fonctions récursives
- T(n) calculé grâce à des relations de récurrence

Exemple

- function fact(n:integer) :integer;
- begin
- if n<=1 then fact := 1
  else fact := n* fact(n-1)

  end;

- T(n) temps d ’exécution en fonction de l ’argument n
- base
  $T(1) = a$
- récurrence
  $T(n) = b + T(n-1)$, pour $n>1$
- On démontre par récurrence que
  $T(n) = a + (n-1)\, b$ pour $n \geq 1$
- Donc $T(n) = O(n)$

Un exemple: Recherche dichotomique

Un exemple: Recherche dichotomique

- On cherche dans un tableau TRIE dans l'ordre croissant *L[1..n]* l'indice d'un élement *X*
- Version récursive: recherche entre les indices *g* et *d*
  - Comparer *X* avec l'élément du milieu: *L[m]*
  - Si *X= L[m]* , fin de le recherche
  - Si *X> L[m]* , rechercher *X* entre les places *m+1* et *d*
  - Si *X< L[m]* , rechercher X entre *g* et *m-1*

Un exemple: Recherche dichotomique

```
Procedure dicho(X, L, g, d, resultat res: 0..n)
    var m: 1..n
Début
    Si g<= d alors
            m := (g+d) div 2
            Si X = L[m] alors res := m
                        sinon si X < L[m] alors dicho(X,L,g,m-1,res)
                                          sinon dicho(X,L,m+1,d,res)
       sinon res :=0
Fin
```

Un exemple: Recherche dichotomique

- On étudie le cas le pire

- $T_{max}(n) = O(\log_2 n)$