Algorithmique P2

La complexité

Ulg, 2009-2010

R.Dumont

Sources supplémentaires

- Ouvrages
  - Data Structures in Java, T. Standish, 1998
  - Data Structures and Algorithms in Java (4th ed), Michael T. Goodrich & Roberto Tamassia, 2006
- Cours
  - Techniques for the Design of Algorithms, Paul E. Dunne
  - Algorithmique, Duret-Lutz Alexandre, 2009
  - Complexité des algorithmes et notation grand-O, G. Savard, 2009
  - Algorithmique et Programmation, C. Herpson, 2009
  - Algorithmique, L. Brun, 2009

Complexité algorithmique

- Deux dimensions
  - Complexité en temps
    - nombre d'opérations explicites effectuées par l'algorithme
  - Complexité en espace
    - taille en mémoire nécessaire pour stocker les structures de données utilisées
- Attention
  - Améliorer la complexité en temps peut se faire aux détriments de l'espace utilisé (et inversement)

Complexité temps vs espace : un exemple

- Echanger les valeurs de deux variables

```
// échange des valeurs de deux variables
entier x, y, z;
... // initialisation de x et y
z <- x;
x <- y;
y <- z;
```

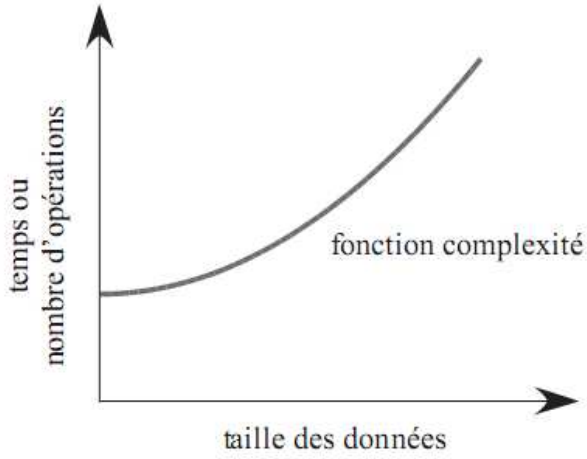
```
// échange des valeurs de deux variables
entier x, y;
... // initialisation de x et y
x <- y-x;
y <- y-x;
x <- y+x;
```

  - Cas 1 : 3 variables et 3 affectations
  - Cas 2 : 2 variables, 3 affections mais 3 opérations supplémentaires

- Aujourd'hui, on considère presque exclusivement la complexité en temps
  - Exception faite des situations où l'espace mémoire est restreint et/ou onéreux.

Complexité : pourquoi ?

- Mesure des performances d'un algorithme
  - En vue de le comparer à d'autres algorithmes
  - Afin de savoir s'il est ou non utilisable (en un temps *raisonnable* ou non)
    - Notion d'efficacité
    - L'objectif est de savoir comment croissent les besoins en temps lorsque la taille des entrées augmente.



Evaluation de la complexité

- Etude expérimentale
  - On teste l'algorithme et on mesure les temps d'exécution pour différentes tailles en entrée.
  - Mais problème car
    - Difficile de comparer deux algorithmes alors qu'un même algorithme s'exécutera à différentes vitesses selon
      - l'architecture matérielle,
      - le langage,
      - l'implémentation,
      - le compilateur,…

Type d'ordinateur	Temps (ms)
Ordinateur familial	52
Serveur	11,5
Supercalculateur	0,085

    - → Etude peu pertinente et fastidieuse

Evaluation de la complexité

- Etude de la complexité *algorithmique*
  - Etude "générale" de la complexité
  - On n'étudie plus les résultats obtenus mais la manière avec laquelle ces résultats évoluent par rapport à la taille de l'entrée

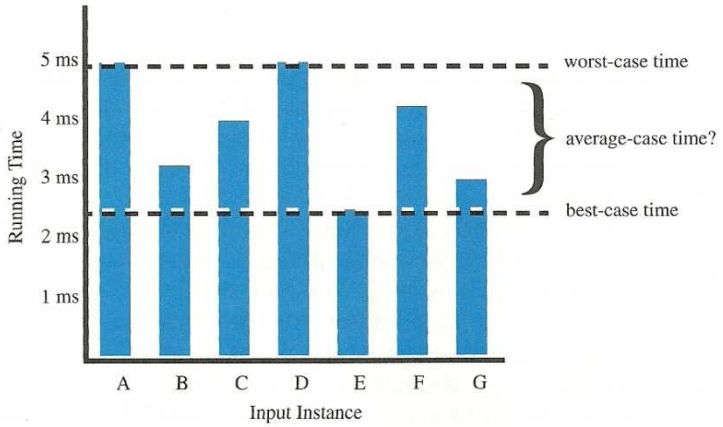
Taille d'un tableau	Ordi familial	PC Serveur
125	12,5	2,8
250	49,3	11,0
500	195,8	43,4
1000	780,3	172,9
2000	3114,9	690,5

x2 ≅ x4

xN ≅ xN^2

Source : Data structures in Java, T. Standish, p.457, Addison-Wesley, 1998

Types d'approche



- 3 approches possibles :
  - Complexité dans le meilleur des cas (*best-case*)
    - Complexité "minimale", cas le plus favorable
    - Situation souvent rare en pratique
  - Complexité dans le cas moyen (*average-case*)
    - Moyenne des complexités
    - Difficile à déterminer
  - Complexité dans le pire des cas (*worst-case*)
    - Complexité "maximale", cas défavorable
    - Réaliste et pertinent pour l'obtention d'une borne

Calculs simplifiés

- Mesurer la complexité exacte est peu pertinent
  - Souvent trop complexe vu la taille des programmes
- Pour éviter de calculer en détails la complexité d'un algorithme, on repère l'opération fondamentale
- Ces opérations fondamentales peuvent être
  - Une assignation
  - Une comparaison entre deux variables
  - Une opération arithmétique entre deux variables
  - …

Complexité – compter le nombre d'opérations fondamentales

- Exemple

```
n:=5;
Loop
   get(m);
   n:=n-1;                    ← op. fond.
Until (m=0 or n=0)
```

  - Complexité Pire des cas ?
    - 5 opérations
  - (et complexité meilleur des cas ?)

Techniques for the Design of Algorithms, Paul E. Dunne

Complexité – compter le nombre d'opérations fondamentales

- Evaluation dans un contexte plus général

```
get(n);
Loop
        get(m);
        n:=n-1;
Until (m=0 or n=0)
```

  - Dans le pire des cas ?
    - n itérations

Complexité - Exemple

```
for i in 1..n loop
   for j in 1..n-1 loop
         if i < j then
                  swap (a(i,j), a(j,i)); ← Opération fond.
         end if;
   end loop;
end loop;
```

→itérations < n*(n-1)*1 = n(n-1)

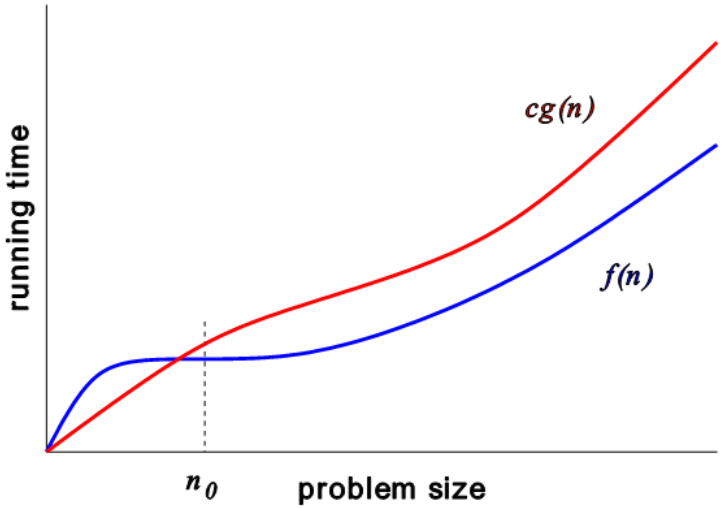
- On considère la complexité du pire des cas :
  - Obtention d'une borne maximale
  - Notation O, *Big-Oh*

Grand-O (Big-Oh)

- Soient f(n) et g(n) deux fonctions de N vers R.
- $f(n) \in O(g(n))$ s'il existe une constante réelle *(facteur)* $c>0$ et une constante entière *(seuil)* $n_0 \geq 1$ tels que

$$f(n) \leq cg(n) \ \forall n \geq n_0$$

- Borne supérieur pour un seuil donné



Utilisation de la notation O

- On écrit
  - $f(n) \in O(g(n))$
    - Voire parfois $f(n) \leq O(g(n))$, mais superflu car "O" signifie par définition "plus petit ou égal", "borné supérieurement par".
- On dit
  - f(n) est de l'ordre de g(n)
  - f(n) est proportionnel à g(n)
  - f(n) est de complexité <dénomination> en temps (voir tableau par la suite)

Exercices

- Soit $f(n)=5n^4+3n^3+2n^2+4n+1$
- Complexité ?
  - $5n^4+3n^3+2n^2+4n+1 < (5+3+2+4+1)n^4=cn^4$
  - Donc $f(n) \in O(n^4)$ pour $c=15$ et $n \geq n_0=1$
- Soit $f(n)=5n^2+3n.\log n+2n+5$
- Complexité ?
  - $5n^2+3n \log n+2n+5 < (5+3+2+5)n^2=cn^2$
  - Donc $f(n) \in O(n^2)$ pour $c=15$ et $n \geq n_0=2$
- Soit $f(n)=20n^3+10n \log n+5$
- Complexité ?
  - $20n^3+10n \log n+5 < (20+10+5)n^3=cn^3$
  - Donc $f(n) \in O(n^3)$ pour $c=35$ et $n \geq n_0=2$

Exercices

- Soit $f(n)=3\log n+2$
- Complexité ?
  - $3\log n+2 < (3+2)\log n=c\log n$
  - Donc $f(n) \in O(\log n)$ pour $c=5$ et $n \geq n_0=2$
- Soit $f(n)=2n+100\log n$
- Complexité ?
  - $2n+100\log n < (2+100)n=cn$
  - Donc $f(n) \in O(n)$ pour $c=102$ et $n \geq n_0=2$
- Soit $f(n)=2^{n+2}$
- Complexité ?
  - $2^{n+2} = 2^n 2^2 = 4.2^n = c.2^n$
  - Donc $f(n) \in O(2^n)$ pour $c=4$ et $n \geq n_0=1$

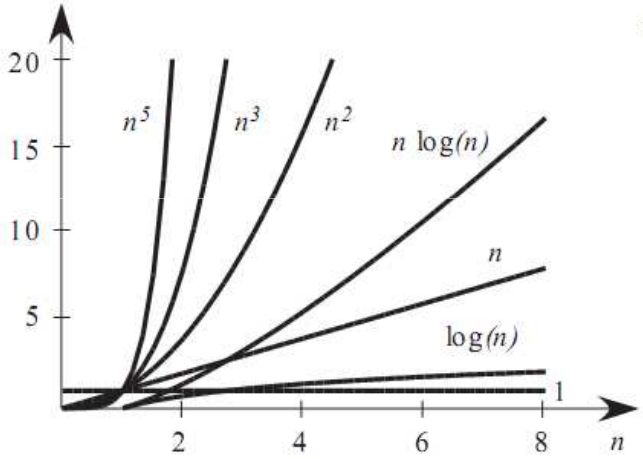
Propriétés de Big-Oh

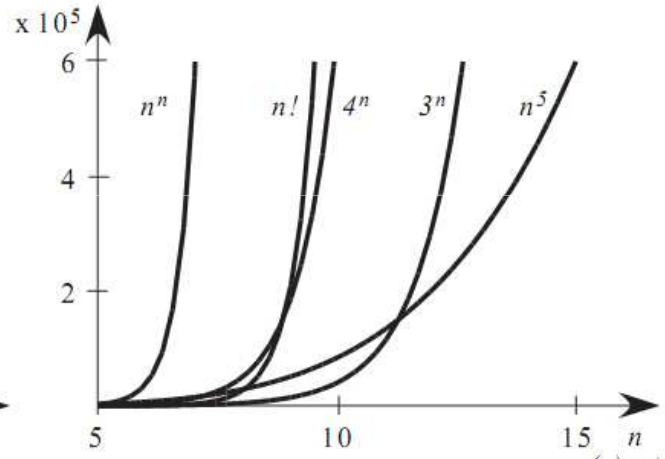
- Si $f(n) \in O(g(n))$, alors
  $$kf(n) \in O(g(n))$$
  - En particulier, $a^b \in O(1)$ et $a^{n+b} \in O(a^n)$
- Si $e(n) \in O(g(n))$ et $f(n) \in O(h(n))$ et si $g(n) \in O(h(n))$ alors
  $$e(n) + f(n) \in O(h(n))$$
  - En particulier, $f(n) = \sum_{d=0}^{D} a_d n^d \in O(n^D)$
- Si $e(n) \in O(g(n))$ et $f(n) \in O(h(n))$, alors
  $$e(n)f(n) \in O(g(n)h(n))$$

Classement des complexités

- Dans l'ordre, pour $0<p<q<1<r<s<2<j<k$, $1<a<b$

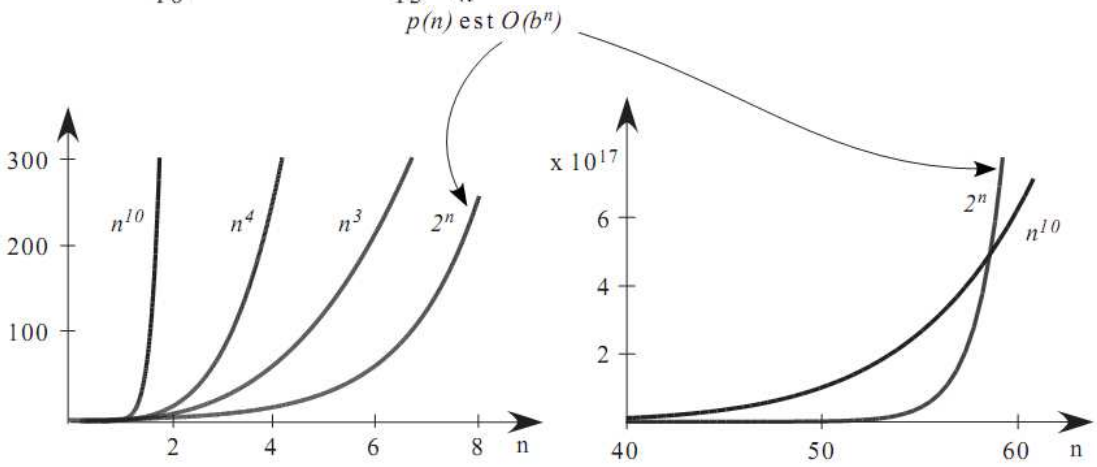
$O(1) \subset O(\log n) \subset O(n^p) \subset O(n^q) \subset O(n) \subset O(n.\log n)$
$\subset O(n^r) \subset O(n^s) \subset O(n^2) \subset O(n^j) \subset O(n^k) \subset O(a^n) \subset O(b^n) \subset O(n!) \subset O(n^n)$





G. Savard, voir Sources Supp

- Si $a > 1$, alors
  $O(\log_a n) \subset O(\log_2 n)$
- $O(\log n!) \subset O(n.\log n)$



O(log n)⊂O(n)

- Si $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$
- Alors $f(x) \in O(g(x))$ et $g(x) \notin O(f(x))$

Soit $f(n) = \log_2(n)$ et $g(n) = n$.

$$\begin{aligned} \lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{\log_2(n)}{n} && \text{qui est de la forme } \frac{\infty}{\infty} \\ &= \lim_{n\to\infty} \frac{\frac{1}{n\ln(2)}}{1} && \text{par la règle de l'Hopital} \\ &= 0 \end{aligned}$$

donc

$$\log_2(n) \in O(n) \text{ mais } n \notin O(\log_2(n))$$

G. Savard, voir Sources Supp

Exercices

- Soit $f(n)=n^4+25n^3+4$
- Quelle est sa borne supérieure ?

- Soit $f(n)=14n^2+3^n$
- Quelle est sa borne supérieure ?

- Soit $f(n)=4n^2+10n+5.3^n+2\log(n)$
- Quelle est sa borne supérieure ?

- Soit $f(n)=(3n+1)\log(n)+3^n$
- Quelle est sa borne supérieure ?

Complexité : cas généraux

- Instructions isolées (=op.fond.)
  - O(1)
- Séquence

  Soient P et Q, deux parties d'algorithme de complexité respective O(n) et O(m):

  P;
  Q;

  Complexité=O(m+n)

Complexité : cas généraux

- Boucles simples :
  - O(nf(n)) pour un corps de complexité O(f(n))

    Pour i allant de 1 à n

    … }O(f(n))
- Boucles d'incrément exponentiel :
  - O(log n)

    Pour i allant de 1 à n

    i←2i

Complexité : cas généraux

- Boucles imbriquées identiques :
  - $O(n^2)$

    Pour i allant de 1 à n
      Pour j allant de 1 à n
        …
  - 3 boucles imbriquées identiques : $O(n^3)$
  - …
- Boucles imbriquées indépendantes :
  - $O(mn)$

    Pour i allant de 1 à n
      Pour j allant de 1 à m
        …
- Boucles imbriquées incrémentées :
  - $O(n^2)$

    Pour i allant de 1 à n
      Pour j allant de 1 à i
        …

Complexité : combinaisons

- Soit le fonction strings(p,n) avec p, une chaine initialement vide et n, une longueur.
- La fonction retourne une liste des chaines de taille n comprenant les caractères inférieurs à k.

```
If n>0 then
        for i←1 to k do
                strings(p+alphabet[i], n-1)
else
        return p
```

- Complexité ?
  - $O(k^n)$ si + est O(1)
  - $O(nk^n)$ si + est O(n)

Nomenclature des fonctions

Notation	Dénomination	Exemples d'application
O(1)	Constante	Opération indépendante de l'entrée
O(log n)	Logarithmique	Hauteur d'un arbre équilibré
O(n)	Linéaire	Parcours d'une liste liée
O(n log n)	"N log n", Quasi-linéaire	Algorithme de tri
$O(n^2)$	Quadratique	Algorithme de tri
$O(n^3)$	Cubique	Triple boucle imbriquée
$O(n^k)$	Polynomiale	/ pour n↗
$O(a^n)$	Exponentielle (pour a>1)	/ pour n↗
O(n!)	Factorielle	/

- Il existe des fonctions spécifiques qui croissent encore plus vite (fct d'Ackermann, Stirling,…)
  - → benchmarks de supercalculateurs

$$A(x,y) \equiv \begin{cases} y+1 & \text{if } x = 0 \\ A(x-1,1) & \text{if } y = 0 \\ A(x-1, A(x,y-1)) & \text{otherwise.} \end{cases}$$

$$A(0,y) = y+1$$
$$A(1,y) = y+2$$
$$A(2,y) = 2y+3$$
$$A(3,y) = 2^{y+3} - 3$$
$$A(4,y) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{y+3} - 3.$$

Complexité : illustration

- Evolution du temps de calcul selon la complexité, la taille des données et la puissance de la machine (Jaguar ≅ 2.10^{15})

© Cédric Herpson

flops	Taille des données : 1 million				Taille des données : 1 milliard			
	n	n log(n)	n^2	2^n	n	n log(n)	n^2	2^n
10^6	secondes	secondes	semaines	10000 ans	heures	heures	10000 ans	une éternité
10^9	millisecondes	millisecondes	heures	10 ans	secondes	secondes	décennies	10 billions d'années
10^{12}	microsecondes	microsecondes	secondes	semaines	millisecondes	millisecondes	semaines	10 milliards d'années

- Il vaut mieux optimiser qu'attendre "La" machine
  - Sans compter que la masse de données à traiter ne cesse de croitre.

Complexité : en pratique

- Complexité exprimée en fonction de la taille de la structure étudiée
  - Pour une structure de données linéaire de taille n (liste liée,…), la complexité repose sur ce n
  - Pour une structure à plusieurs dimensions (tableaux multi., arbres,…), on doit préciser la dimension de référence utilisée pour n
- On s'attarde sur la forme générale de la complexité
  - On supprime les facteurs constants
  - On ne conserve que le terme croissant le plus rapidement
    - On supprime les termes d'ordre inférieur

Termes et facteurs

- Soit $f(n) = an^2+bn+c$
  avec $a = 0.0001724$, $b = 0.0004$ et $c=0.1$

n	f(n)	an^2	Importance du terme en n^2 sur le total
125	2.8	2.7	94.7
250	11.0	10.8	98.2
500	43.4	43.1	99.3
1000	172.9	172.4	99.7
2000	690.5	689.6	99.9

Source : Data structures in Java, T. Standish, p.457, Addison-Wesley, 1998

- Ici, $(bn+c)$ est insignifiant, même si $c > 250b$ et $b > 2a$

- La contribution des termes d'ordre inférieur est négligeable.
- Le facteur a du terme an^2 peut différer d'une implémentation à l'autre (langage, architecture,…)
  - On omet ce facteur

Termes et facteurs

- Soient 3 algorithmes P, Q, R de complexités 2^n, $5n^2$ et $100n$
- Pour les tailles d'entrée variables suivantes, il vient

n	P	Q	R
1	1	5	100
10	1024	500	1000
100	2^{100}	50,000	10,000
1000	2^{1000}	5 million	100,000

- Si on exécute ces algorithmes sur un processeur opérant à 10^6 opérations/s

n	P	Q	R
1	1μs	5μs	100μs
5	1 millisec	0.5 millisec	1 millisec
100	2^{70} years	0.05 secs	0.01 secs
1000	2^{970} years	5 secs	0.1 secs

- L'expression de la complexité est donc plus représentative sous forme de n (2^n, n^2, n) que de valeurs (1;5;100)

Techniques for the Design of Algorithms, Paul E. Dunne

Comportement asymptotique

- Dans notre cas, la définition de la complexité sera le résultat d'une analyse **asymptotique**
  - C'est à dire quand n tend vers l'infini
    - On n'analyse pas la complexité pour de petites valeurs de n
  - Ceci explique pourquoi les constantes pourront être ignorées ici.

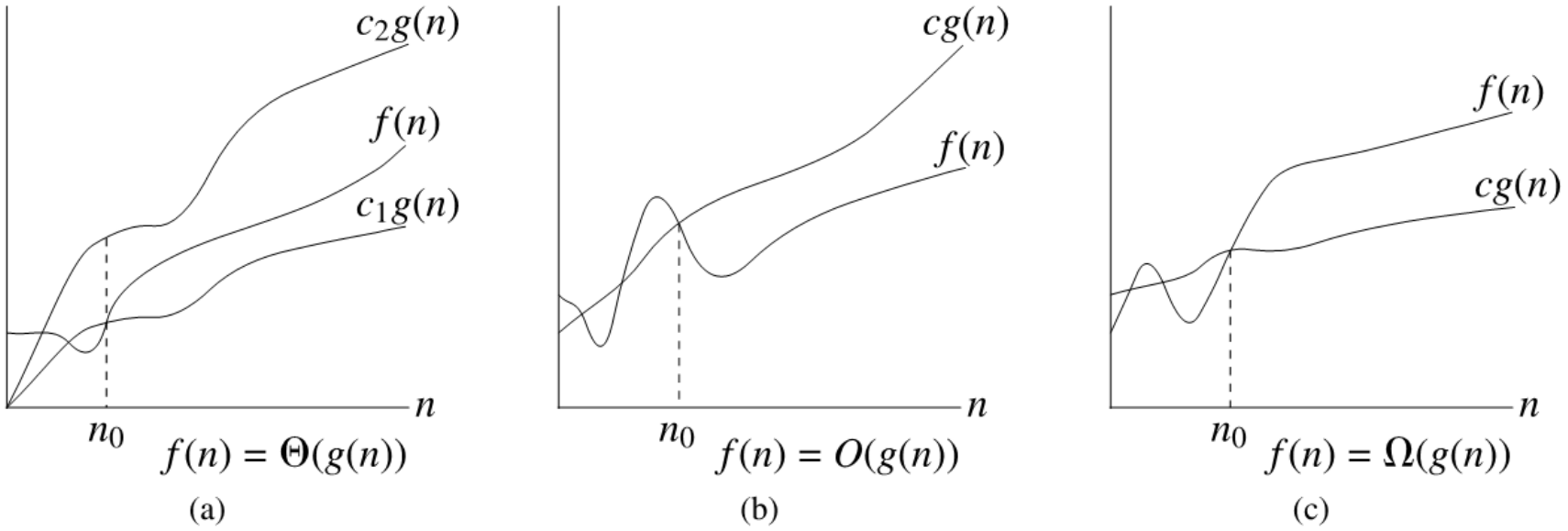
- Attention, parfois, on peut devoir en tenir compte
  - Applications où l'analyse de la complexité demande plus de précision
  - Lorsque ces facteurs sont *extraordinairement* différents ($c' << c$)
    - Comparez $f(n) = 10^{100}n$ et $g(n) = 5n.\log n$

Autres notations (de Landau)

- D'autres écritures existent
  - Bornée inférieurement (c)

    $f(n) \in \Omega(g(n))$ si $g(n) \in O(f(n))$
  - Bornée supérieurement et inférieurement (a)

    $f(n) \in \Theta(g(n))$ si $f(n) \in O(g(n)) \cap \Omega(g(n))$



Exercices rapides

Pour i = 1 à n
  Pour j = 1 à n
  x = x+1
	Pour i = 1 à n
	  Pour j = 1 à n
	  Pour k = 1 à n
	  x = x+1

Pour i = 1 à n
  Pour j = 1 à i
  x = x+1
	Pour i = 1 à n
	  Pour j = 1 à i
	  Pour k = 1 à j
	  x = x+1

Pour i = 5 à n-5
  Pour j = i-5 à i+5
  x = x+1
	Pour (i = n; i>1;i=i/2)
	  Pour (j = 0; j<i; j++)
	  x = x+1

Pour i = 1 à n
  Si T[i]>a Alors
  x = x+ T[i]
	Si a>b Alors
	  Pour i = 1 à n
	  x=x+1
	Sinon x = x+2