Université Clermont Auvergne
ISIMA FISA 1
Algorithmique
Année 2025-2026

Florent Foucaud

TD1 - Complexité en temps d'un algorithme

Les exercices commencent à la page 2

1 Rappel de cours : complexité en temps d'un algorithme

Pour un algorithme donné, on souhaite déterminer le nombre d'instructions de base $T(n)$ qui seront effectuées *dans le pire des cas*, pour une entrée de taille n (où $T : \mathbb{N} \to \mathbb{N}$). La taille est le nombre de bits nécessaires pour représenter l'entrée. On considère généralement qu'un nombre (par exemple) a une taille constante, un tableau a donc une taille proportionnelle à sa longueur.

Les instructions de base sont les affectations, les opérations mathématiques, les comparaisons, l'accès/écriture d'un élément de tableau... et on considère que chacune prend une unité de temps.

En général, on s'intéresse surtout à l'ordre de grandeur de $T(n)$, pour cela on utilise les *notations asymptotiques* suivantes (étant données deux fonctions $f, g : \mathbb{N} \to \mathbb{N}$) :

- $f(n) \in O(g(n))$ si il existe une constante $c \in \mathbb{R}^+$ $(c > 0)$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \leq c \cdot g(i)$. Cela veut dire que f ne croît pas de façon plus rapide que g (à un facteur constant près) lorsque le paramètre n est suffisamment grand.
- $f(n) \in \Omega(g(n))$ si il existe une constante $c \in \mathbb{R}^+$ $(c > 0)$ et un rang $n_0 \in \mathbb{N}$ tels que, pour tout entier $i \geq n_0$, on a $f(i) \geq c \cdot g(i)$. Cela veut dire que f ne croît pas de façon moins rapide que g (à un facteur constant près) lorsque le paramètre n est suffisamment grand.
- $f(n) \in \Theta(g(n))$ si $f(n) \in O(g(n))$ et $f(n) \in \Omega(g(n))$. Cela signifie que f et g se comportent de la même façon (à facteurs constants près) lorsque le paramètre n est suffisamment grand.

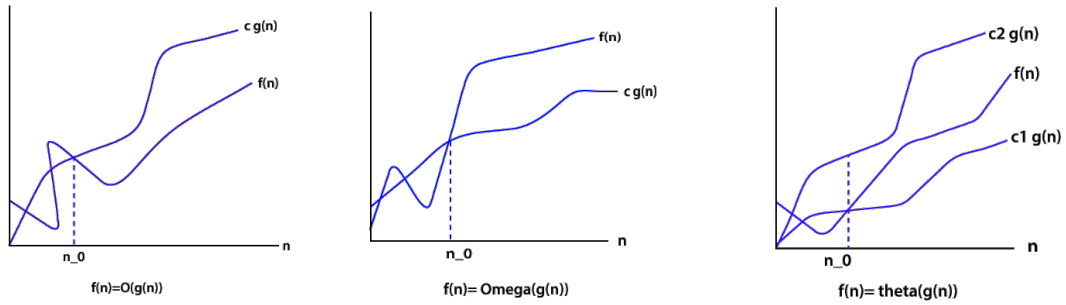

FIGURE 1 – Schéma des trois notations asympototiques

Remarque : si $f(n) \in O(g(n))$, alors $g(n) \in \Omega(f(n))$, et inversement.

Un *problème algorithmique* est défini par le type d'entrée attendu (par exemple : un entier, un tableau d'entiers, etc) et une tâche à effectuer (généralement, une sortie à renvoyer). Plusieurs algorithmes qui résolvent un même problème algorithmique donné peuvent exister, on cherche bien entendu à trouver l'algorithme le plus efficace possible parmi ceux-ci.

Une *classe de complexité* est un ensemble de problèmes algorithmiques pour lesquels le meilleur algorithme possible se comporte de manière similaire (par exemple, en termes de temps de calcul).

Les notations asymptotiques vues ci-dessus permettent de définir des *classes de complexité* classiques pour les problèmes algorithmiques, en utilisant des fonctions usuelles, par exemple (du meilleur au pire) :

- $T(n) \in \Theta(1)$ (complexité constante)
- $T(n) \in \Theta(\log(\log(n)))$
- $T(n) \in \Theta(\log(n))$ (complexité logarithmique)
- $T(n) \in \Theta(\sqrt{n})$
- $T(n) \in \Theta(n)$ (complexité linéaire)
- $T(n) \in \Theta(n \log(n))$
- $T(n) \in \Theta(n^2)$ (complexité quadratique)
- $T(n) \in \Theta(n^c)$ pour une constante $c \geq 1$ (complexité polynomiale)
- $T(n) \in \Theta(c^n)$ pour une constante $c > 1$ (complexité exponentielle)
- $T(n) \in \Theta(n!)$ (qui est dans $O(n^n)$)
- ...

2 Premières estimations de complexité

Exercice 1 (Complexité naïve)**.**

La fonction `f` s'exécute en 1 jour de calcul.

1. Est-ce que les programme 1 et 2 calculent-ils la même chose ?

Programme 1 :
```
a = f(x) + f(x)
```

Programme 2 :
```
a = 2 * f(x)
```

2. Quel est le programme le plus rapide ? Justifier votre réponse.
3. Est-ce que les programme A, B, C, et D calculent la même chose ?

Programme A :
```
Si y == f(x) + f(x) :
   Alors z = f(x)
Sinon z = f(x) + f(x)
```

Programme B :
```
a = 2 * f(x)
Si y == a :
   Alors z = f(x)
Sinon z = a
```

Programme C :
```
a = f(x)
Si y == 2 * a  :
   Alors z = a
Sinon z = f(x) + f(x)
```

Programme D :
```
a = 2*f(x)
Si y == a :
   Alors z = a/2
Sinon z = a
```

4. Classer ces programmes du plus rapide au plus lent et justifier votre réponse en jours de calcul.

Exercice 2 (Boucles imbriquées de trucs et bidules)**.**

`truc()` et `bidule()` sont deux fonctions quelconques, sans argument.

Programme 1 :

```
Pour i allant de 1 à n:
    truc()
Pour i allant de 1 à n:
    bidule()
```

Programme 2 :

```
Pour i allant de 1 à n:
    truc()
    Pour j allant de 1 à n:
        bidule()
```

Programme 3 :

```
Pour i allant de 1 à n:
    truc()
    Pour j allant de 1 à i:
        bidule()
```

Programme 4 :

```
Pour i allant de 1 à n:
    truc()
    Pour j allant de 1 à n:
        Pour k allant de 1 à n:
            bidule()
```

A) Donner le nombre d'exécutions de `truc()` et `bidule()`. On suppose que les deux fonctions `truc()` et `bidule()` s'exécutent chacune en temps constant : 1 — donner (sans justification) la complexité *asymptotique* de ces scripts avec la notation grand O, si les deux fonctions s'exécutent chacun en temps $O(1)$.

On pourra utiliser la formule suivante lorsque nécessaire :

$$\sum_{i=0}^{s} i = \frac{s(s+1)}{2}$$

B) On suppose maintenant que `truc()` prend un temps n et `bidule()` prend un temps n^2. Que deviennent les complexités ?

Exercice 3 (Maximum itératif)**.**

Quelle est la complexité en temps $T(n)$ *à l'instruction élémentaire près* de l'algorithme ci-dessous, appelé sur un tableau à $n \geq 1$ éléments ? On rappelle que l'accès $TAB[i]$ à une case du tableau est une instruction élémentaire, une comparaison, une affectation et une addition sont aussi des instructions élémentaires.

`maxiter(`TAB`): Recherche du maximum d'un tableau` TAB

`Entrée : un tableau` $TAB[1...n]$ `non trié de` n `entiers`

`Sortie : le plus grand entier de` TAB

- `max` $=$ $TAB[1]$
- $i = 2$
- `Tant que` $i \leq n$ `:`
  - `Si` $TAB[i] > $ `max alors :`
    `max` $=$ $TAB[i]$
  - $i = i + 1$
- `Retourner max`

Exercice 4 (Tri par insertion)**.**

Dans l'algorithme de tri par insertion, on trie le tableau de gauche à droite, en insérant la valeur courante à la bonne place dans la partie déjà triée.

Quelle est la complexité en temps $T(n)$ *à l'instruction élémentaire près* de l'algorithme, appelé sur un tableau à n éléments ?

`triInsertion(`TAB`): tri par ordre croissant du tableau` TAB `d'entiers`

`Entrée : un tableau` $TAB[1...n]$ `non trié de` n `entiers`

`Sortie : le tableau` TAB `trié`

- $i = 2$
- `Tant que` $i \leq n$ `faire :`
  - $j = i - 1$
  - `valeurInsertion` $= TAB[i]$
  - `Tant que` $j > 0$ `et` $TAB[j] >$ `valeurInsertion faire :`
    - $TAB[j+1] = TAB[j]$
    - $j = j - 1$
  - $TAB[j+1] =$ `valeurInsertion`
  - $i = i + 1$
- `Retourner` TAB

3 Notations asymptotiques, classes de complexité

Exercice 5 (Notations asymptotiques)**.**

1. Montrer que $100n \in O(n^2)$.
2. Montrer que $100n \in \Theta(n)$.
3. Montrer que $n^2 \in \Omega(3n \log_2(n))$.
4. Montrer que $2n^2+n+10 \in \Theta(n^2)$. Plus généralement, comment déterminer la classe de complexité $\Theta(...)$ d'une somme de fonctions, si on connait la complexité de ses termes ?
5. Montrer que $4n - 3\sqrt{n} \in \Theta(n)$.
6. Montrer que $1 + \frac{1}{n} \in \Theta(1)$.
7. Montrer que si $f(n) \in O(g(n))$, alors $g(n) \in \Omega(f(n))$.
8. Montrer que $100n \notin \Omega(n^2)$. *Indice : supposer par l'absurde que* $100n \in \Omega(n^2)$ *et en déduire une contradiction.*

4 Complexité avancée

Exercice 6 (Récursivité et Fibonacci)**.**

On considère le programme récursif suivant, permettant d'approximer la taille de la $n$ème génération d'une population de lapins.

```
Fibo(n):
Entrée : un entier n
Sortie : la nème valeur de la suite de Fibonacci
  • Si n == 0 ou n == 1:
    Retourner 1
  • Sinon:
    Retourner Fibo(n − 1) + Fibo(n − 2)
```

1. Dessiner l'arbre des appels récursifs pour $Fibo(6)$. Intuitivement, quelle est la complexité du programme ?
2. Écrire la complexité en temps $T(n)$ en fonction de $T(n-1)$ et $T(n-2)$, pour $n > 1$, en supposant qu'une instruction élémentaire coûte 1.
3. Supposons que $T(n) = x^n$ pour un certain $x > 0$. Remplacer $T(n)$ dans l'équation précédente, et en déduire une équation (qui ressemble à une équation du second degré) ayant x comme variable.
4. Considérer cette équation comme une équation du second degré (en approximant les termes qui sont très petits quand n est grand, à 0), la résoudre, [1] et en déduire une valeur approximative pour $T(n)$.
5. Pourrait-on calculer $Fibo(n)$ de façon plus efficace ? Si oui, comment ?

5 Pour aller plus loin

Exercice 7 (Notations asymptotiques, suite)**.**

1. Montrer que pour tout entier fixé $k > 0$, $n^k \in O(2^n)$.
2. Montrer que $2^n \notin \Theta(3^n)$.

Exercice 8 (Complexités de sommes et produits de fonctions)**.**

A. Classer les fonctions suivantes en fonction des types de complexité proposés :
$\log(n) + 2n^2 - 5$ $\quad 3n^4 + 2n$ $\quad 33\log(n) + 56$ $\quad 8^n + n^2$ $\quad 348n - \sqrt{n} + \log(n)$ $\quad 1 + \frac{1}{n^2}$

1. temps constant $\Theta(1)$
2. temps logarithmique $\Theta(\log n)$
3. temps linéaire $\Theta(n)$
4. temps quadratique $\Theta(n^2)$
5. temps polynomial $\Theta(n^c)$ pour une constante $c \geq 1$
6. temps exponentiel $\Theta(c^n)$ pour une constante $c > 1$

B. De manière générale, si on sait que $f_1 \in \Theta(g_1)$ et $f_2 \in \Theta(g_2)$, comment détermine-t-on la complexité asymptotique de la somme $f_1 + f_2$? Montrez-le rigoureusement.

C. Même question pour le produit $f_1 \times f_2$.

1. Rappel : pour résoudre l'équation $ax^2 + bx + c = 0$, on calcule le discriminant $\Delta = b^2 - 4ac$, si $\Delta < 0$ il n'y a pas de solution, si $\Delta = 0$ il y a une solution unique à $\frac{-b}{2a}$, sinon il y a deux solutions : $\frac{-b-\sqrt{\Delta}}{2a}$ et $\frac{-b+\sqrt{\Delta}}{2a}$.