Université Clermont Auvergne
ISIMA FISA 1
Algorithmique
Année 2024-2025

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 à 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 $r \in \mathbb{N}$ tels que, pour tout entier $i \geq r$, 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 $r \in \mathbb{N}$ tels que, pour tout entier $i \geq r$, 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.

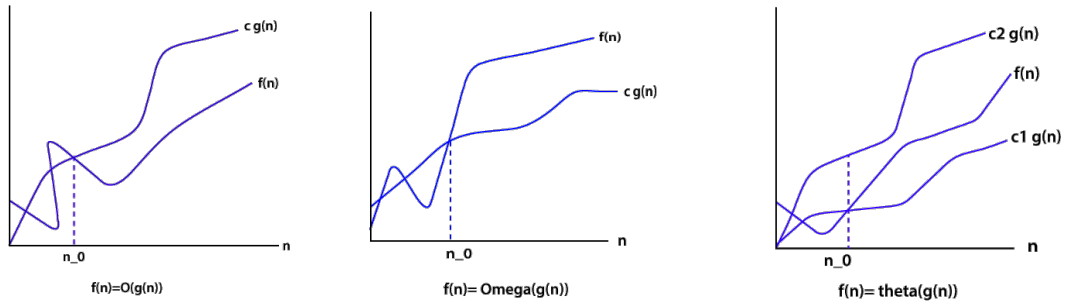

FIGURE 1 – Schéma des trois notations asympototiques

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 Exercices : 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 font-ils 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()`, ainsi que la complexité asymptotique de ces scripts (notation grand O), en supposant que les deux fonctions `truc()` et `bidule()` s'exécutent en temps constant $O(1)$. On pourra utiliser la formule suivante lorsque nécessaire :
$\sum_{i=0}^{s} i = \frac{s(s+1)}{2}$

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

Exercice 3 (Maximum itératif)**.**

Quelle est la complexité en temps $T(n)$ *à l'opération 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 opération élémentaire, une comparaison, une affectation et une addition sont aussi des opérations 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 =` $TAB[1]$
- `Pour` i `allant de 2 à` n `:`
  - `Si` $TAB[i]$ `> max alors :`
    `max =` $TAB[i]$
- `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'opération près* de l'algorithme, appelé sur un tableau à n éléments ?

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

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

`Sortie : le tableau` TAB `trié`

- `Pour` i `allant de 2 à` 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`
- `Retourner` TAB

Exercice 5 (Tri à bulles)**.**

Dans le tri à bulles, on fait remonter vers la fin du tableau (comme une bulle dans un liquide) la valeur la plus grande par échanges successifs des valeurs consécutives du tableau.

Exprimer le plus précisément possible, la complexité en temps $T(n)$ de l'algorithme du tri à bulles, où n est la taille du tableau.

`triBulles(`TAB`): tri d'un tableau` TAB `d'entiers par ordre croissant`

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

`Sortie : le tableau` TAB `trié`

- `Pour` i `allant de` n `à 1 faire :`
  - `Pour` j `allant de 1 à` $i-1$ `faire :`
    - `Si` $TAB[j+1] \ < \ TAB[j]$ `alors :`
      $x \ = \ TAB[j]$
      $TAB[j] \ = \ TAB[j+1]$
      $TAB[j+1] \ = \ x$
- `Retourner` TAB

3 Exercices : Notations asymptotiques, classes de complexité

Exercice 6 (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 $n^2 + n + 10 \in \Theta(n^2)$

Exercice 7 (Complexités de sommes de fonctions)**.**

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

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 $f_1 + f_2$? Démontrez-le rigoureusement.

Exercice 8 (Notations asymptotiques, suite)**.**

1. Montrer que $100n \notin \Omega(n^2)$.
2. Montrer que pour tout entier fixé $k > 0$, $n^k \in O(2^n)$.
3. Montrer que $2^n \notin \Theta(3^n)$.

4 Exercices : exemples de complexités avancées

Exercice 9 (Sous-tableaux)**.**

Quelle est la complexité asymptotique en temps $T(n)$ de l'algorithme ci-dessous, appelé sur un tableau à n éléments ?

`soustableautriemax(`TAB`): Recherche du plus grand sous-tableau d'éléments croissants d'un tableau` TAB

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

`Sortie : la taille d'un plus grand sous-tableau (pas forcément consécutif) de` TAB `dont tous les éléments sont rangés dans l'ordre croissant`

- `max =` 0
- `Pour chaque sous-tableau` S `de` TAB `:`
  - ⋆ `test = Vrai`
  - ⋆ `Pour` i `allant de 1 à` $|S| - 1$`:`
    `si` $S[i] > S[i+1]$`:`
    `test = Faux`
  - ⋆ `si test == Vrai et` $|S| >$ `max:`
    `max =` $|S|$
- `Retourner max`

Exercice 10 (Colorations de graphes)**.**

On rappelle le problème de *coloration de graphes* : étant donné un graphe non orienté $G = (V, E)$, attribuer un entier $c(v)$ à chaque sommet v (appelé *couleur*), de telle façon que pour chaque arête xy, $c(x) \neq c(y)$. On cherche à trouver le plus petit nombre possible de couleurs permettant de colorer le graphe selon cette contrainte.

1. Proposer une fonction `verif_col(G,c)` qui, étant donné un graphe $G = (V, E)$ à n sommets et une coloration potentielle $c : V \to \mathbb{N}$, vérifie que c est une coloration valide de G.
2. Quelle est sa complexité en fonction du nombre n de sommets ?
3. Quel est le nombre de colorations (pas forcément valides) possibles pour un graphe à n sommets, avec au plus k couleurs ?
4. On propose l'algorithme suivant pour le problème de coloration de graphes :

```
coloration(G): coloration du graphe G
Entrée : un graphe G = (V,E) à n sommets
Sortie : une coloration optimale de G
  • nb_opt = n
  • col_opt = []
  • Pour chaque coloration c possible de G:
      - nb_c = nombre de couleurs de c
      - Si verif_col(G,c) et nb_c < nb_opt alors:
        nb_opt = nb_c
        col_opt = c
  • Retourner col_opt
```

Quelle est la complexité en temps $T(n)$ de `coloration(G)` en fonction du nombre n de sommets de G ?