Université Clermont Auvergne
ISIMA 1A par apprentissage
Algorithmique

Florent Foucaud
2022-2023

TD2 - Complexité en temps

1 Rappel de cours

1.1 Complexité en temps d'un algorithme

Pour un algorithme donné, on souhaite déterminer le nombre d'instructions de base $C(n)$ qui seront effectuées *dans le pire des cas*, pour une entrée de taille n (où $C : \mathbb{N} \to \mathbb{N}$). 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.

On s'intéresse surtout à l'ordre de grandeur de $C(n)$, 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 $c \in \mathbb{R}$ 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 sufisamment grand.
- $f(n) \in \Omega(g(n))$ si il existe une constante $c \in \mathbb{R}$ 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 sufisamment 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 sufisamment grand.

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) :

- $C(n) \in \Theta(1)$ (complexité constante)
- $C(n) \in \Theta(\log(\log(n)))$
- $C(n) \in \Theta(\log(n))$ (complexité logarithmique)
- $C(n) \in \Theta(\sqrt{n})$
- $C(n) \in \Theta(n)$ (complexité linéaire)

- $C(n) \in \Theta(n\log(n))$
- $C(n) \in \Theta(n^c)$ pour une constante $c > 1$ (complexité polynômiale)
- $C(n) \in \Theta(c^n)$ pour une constante $c > 0$ (complexité exponentielle)
- $C(n) \in \Theta(n!)$ (qui est dans $O(n^n)$)...

2 Exercices

Exercice 1 (Maximum itératif)**.**

Quelle est la complexité en temps $C(n)$ de l'algorithme ci-dessous, appelé sur un tableau à $n \geq 1$ éléments ? On considère que l'accès $T[i]$ à une case du tableau est une opération élémentaire.

```
maxiter(T): Recherche du maximum d'un tableau T
Entrée : un tableau T[1...n] non trié de n entiers
Sortie : le plus grand entier de T
  • max = T[1]
  • Pour i allant de 2 à n :
      ⋆ Si T[i] > max alors :
        max = T[i]
  • Retourner max
```

Exercice 2 (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 $C(n)$ de l'algorithme, appelé sur un tableau à n éléments ?

```
triInsertion(T): tri d'un tableau T d'entiers par ordre croissant
Entrée : un tableau T[1...n] non trié de n entiers
Sortie : le tableau T trié
  • Pour i allant de 2 à n faire :
      ⋆ j = i−1
      ⋆ valeurInsertion = T[i]
      ⋆ Tant que j > 0 et T[j] > valeurInsertion faire :
          − T[j+1] = T[j]
          − j = j−1
      ⋆ T[j+1] = valeurInsertion
  • Retourner T
```

Exercice 3 (Notations asymptotiques)**.**

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

Exercice 4 (Sous-ensembles)**.**

Quelle est la complexité asymptotique en temps $C(n)$ de l'algorithme ci-dessous, appelé sur un tableau à n éléments ?

`soustableautriemax(`T`): Recherche du plus grand sous-tableau d'éléments croissants d'un tableau` T

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

`Sortie : la taille d'un plus grand sous-tableau (pas forcément consécutif) de` T `dont tous les éléments sont rangés dans l'ordre croissant`

- `max` $= 0$
- `Pour chaque sous-tableau` S `de` T `:`
  - ⋆ `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 5 (Recherche dichotomique récursive)**.**

Quelle est la complexité en temps $C(N)$ de l'algorithme de recherche dichotomique ci-dessous, appelé avec $m=1$ et $n=N=2$? On peut supposer que $N=2^k$ est une puissance de deux.

`rechercheDicho(`T`,`m`,`n`,`x`): calcul d'une position de` x `dans` T

`Entrée : Un tableau d'entiers` $T[m...n]$ `triés par ordre croissant, un entier` x`, deux positions` m `et` n

`Sortie :` 0 `si` $x \notin T$`, et une position de` x `dans` T `(entre` m `et` n`) sinon`

- `Si` $m == n$ `:`
  - ⋆ `Si` $T[m] == x$`:`
    `Retourner` m
  - ⋆ `Sinon :`
    `Retourner` 0
- `Sinon :`
  - ⋆ $k = \lfloor \frac{m+n}{2} \rfloor$
  - ⋆ `Si` $T[k] < x$ `:`
    `Retourner rechercheDicho(`T`,`$k+1$`,`n`,`x`)`
  - ⋆ `Sinon :`
    `Retourner rechercheDicho(`T`,`m`,`k`,`x`)`