Travaux Dirigés Algorithmique n°1

Cours d'algorithmique

—Licence CFA / Troisième Année—

Complexité, fonctions usuelles

Cette séance de Travaux dirigée est consacrée à la notion de complexité et à l'étude du comportement asymptotique des fonctions élémentaires.

Formulaire sur la notion de sommation

Dans les formules ci-dessous, $(u_i)_i$ et $(v_i)_i$ sont des suites de nombre et c ne dépend pas de i.

- $\sum_{i=1}^{n} 1 = n$ et $\sum_{i=0}^{n} 1 = n+1$
- $\sum_{i=m}^{n} 1 = n - m + 1$
- $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=m}^{n} c\,u_i = c \sum_{i=m}^{n} u_i$
- $\sum_{i=m}^{n} (u_i + v_i) = \sum_{i=m}^{n} u_i + \sum_{i=m}^{n} v_i$
- si $m < n$: $\sum_{i=m}^{n} u_i = \left(\sum_{i=m}^{n-1} u_i\right) + u_n$
- si $m < n$: $\sum_{i=m}^{n} u_i = u_m + \left(\sum_{i=m+1}^{n} u_i\right)$

▸ **Exercice 1. (Nombres d'opérations de programmes élémentaires).**

Pour chacun des patrons de programmes suivants, dire en fonction de n quel est le nombre d'opérations `op` que le programme effectue. Justifier votre résultat par un calcul de sommation :

1.
```
pour i de 1 à n faire op
```

2.
```
op; op; op
pour i de 1 à n faire
    op; op
```

3.
```
pour i de 1 à n faire
   pour j de 1 à n faire op
```

4.
```
op; op
pour i de 1 à n faire
   op; op
   pour j de 1 à n faire
       op
       pour k de 1 à n faire op
```

5.
```
pour i de 1 à n faire
    pour j de 1 à i faire op
```

6.
```
pour i de 1 à n faire
    op; op
    pour j de i à n faire op
```

7.
```
pour i de 1 à n faire
    pour j de 1 à i faire
        op
        pour k de i à n faire op
```

8.
```
pour i de 1 à n faire
    pour j de 1 à i faire
        pour k de 1 à j faire op
```

9.
```
pour i de 1 à n faire
    pour j de 1 à i faire
        op
        pour k de j à i faire op
```

▸ **Exercice 2. (Nombres d'opérations de programmes élémentaires).**

Pour chacun des patrons de programmes suivants, dire en fonction de n quel est le nombre d'opérations `op` que le programme effectue. On pourra d'abord par tâtonnements chercher une formule que l'on prouvera par récurrence :

1.
```
tant que n > 0 faire
    op
    n <- floor(n / 2)
```

2.
```
tant que n > 1 faire
    op
    n <- ceil(n / 2)
```

▸ **Exercice 3. (Ordres de grandeurs).**

On dispose d'une machine qui est capable de faire 1000000 opérations par secondes. On considère des algorithmes dont les complexités sont les suivantes : $\log(n), \sqrt{n}, n, 50 * n, n\log(n), n^2, n^3, 10^n$.

1. Dessiner chacune de ces fonctions sur une échelle doublement logarithmique. Pour simplifier, on prendra $\log = \log_{10}$. Quelle est la différence si l'on avait pris $\log = \log_2$?
2. Quel est pour chaque algorithme, la taille des problèmes que l'on peut résoudre en une seconde ?
3. Même question si l'on dispose d'une machine 1000 fois plus rapide et de 1000s.

▸ **Exercice 4.**

On donne dans le tableau ci-dessous les mesures des temps de calculs en secondes de trois programmes pour différentes tailles de données. On rappelle que la notation scientifique 1.2e-3 signifie $1.2 \cdot 10^{-3}$.

n	1	10	100	1000	10000	100000
Algo 1	1.1e-7	5.2e-4	6.1e-2	6.2	6.3e+2	6.2e+4
Algo 2	2.1e-3	3.2e-3	4.1e-2	4.2e-1	4.3	4.3e+1
Algo 3	8.1e-5	9.5e-5	1.1e-4	1.3e-4	1.5e-4	1.7e-5

1. Quelle semble être la complexité de chacun de ces programmes ? Expliquer pourquoi en une phrase où deux ?
2. Pouvez vous donner un ordre de grandeur pour les temps de calcul de ces programmes pour des données de taille $n = 10^6$, $n = 10^9$, $n = 10^{12}$?

▸ **Exercice 5. (Notation O et Θ)**

On considère les fonctions suivantes :

$$f_1 = 4x^2, \quad f_2 = \log(x), \quad f_3 = 3^x + 4, \quad f_4 = 2\log(x) + 1, \quad f_5 = 5x^3 + 2.$$

1. Pour chacune des fonctions, écrire la fonction de même complexité la plus simple possible.
2. Remplir le tableau suivant en mettant dans la case à l'intersection de la ligne i et de la colonne j le symbole qui convient selon la règle :

   • Θ si $f_i \in \Theta(f_j)$, • O si $f_i \in O(f_j)$ mais $f_i \notin \Theta(f_j)$, • $\times$ sinon.

	1	2	3	4	5
1					
2					
3					
4					
5					