TD/TP 3 : Complexités algorithmiques Théorie et pratique

CPES « Sciences des données, arts et cultures » : Introduction à l'algorithmique

Antoine GAUQUIER antoine.gauquier@ens.fr
Pierre SENELLART pierre@senellart.com

03 octobre 2023

Dans ce troisième TD / TP, nous étudions la complexité des algorithmes. Ce sujet proposera donc deux types de questions au travers de plusieurs exercices. D'abord des questions de type TD, où vous allez devoir calculer des complexités algorithmiques via les méthodes vues en cours, à partir de fonctions ou de pseudo-codes qui vous seront fournis. Ces questions ne nécessitent pas d'écrire de code ni d'utiliser d'ordinateur : il vous faut résoudre ces exercices sur feuille.

D'autres questions vous demanderont d'implémenter des algorithmes en Python, et d'en mesurer la complexité par des méthodes empiriques de mesure de temps, qu'on appelle *profiling*. Il sera alors intéressant de comparer les résultats calculés sur papier avec ce qui est mesuré en pratique.

Les exercices proposés dans ce TD / TP seront similaires sur le fond à ce que vous trouverez dans le premier devoir maison (même si ce dernier intégrera tout le contenu étudié à la fois dans les cours et dans les séances de TD / TP jusqu'à aujourd'hui). Nous vous invitons donc à particulièrement bien les travailler.

Exercice 1: Manipulation des notations asymptotiques.

Donner les (meilleures) notations asymptotiques O des fonctions suivantes :

- $f(n) = 2 \times n^2 + 1000 \times n$
- $f(n) = n \times \log(n) + n^2$
- $f(n) = \log(n) + 2 \times \sqrt{n}$
- $f(n) = 2 \times 2^n + n \times 2^n$
- $f(n) = n \times (8 \times n + \sqrt{n})$
- $f(n) = n^3 + 3^n$
- $f(n) = n^2 \times \log(n) + n \times \log(n)^2$

Exercice 2: Démonstration de propriétés notables.

Soient quatre fonctions f, g, ϕ et ψ de $\mathbb{N} \to \mathbb{R}_+$, telles que $f(n) \in O(\phi(n))$ et $g(n) \in O(\psi(n))$.

Question 1. Montrer qu'on a $f(n) + g(n) \in O(\phi(n) + \psi(n))$.

Question 2. Montrer qu'on a $f(n) \times g(n) \in O(\phi(n) \times \psi(n))$.

Algorithme 1

Entrée: $n \in \mathbb{N}^*$

```
k ← 0
pour i allant de 1 à n faire
    pour j allant de 1 à i faire
        k ← k + 1
    fin pour
fin pour
retourne k
```

Exercice 3: Déterminer la complexité d'un algorithme.

Décrire ce que réalise l'algorithme 1. Calculer ensuite la complexité de l'algorithme (la fonction $f(n)$ décrivant le nombre d'*opérations élémentaires* réalisées avec une taille d'entrée n), puis déterminer la notation asymptotique associée.

Exercice 4: Vérification empirique des complexités asymptotiques : le *profiling* en Python

Dans cet exercice, vous allez observer empiriquement (c'est à dire, par l'expérience) la complexité asymptotique théorique que vous avez calculée dans l'exercice précédent.

Question 1. Implémentez en Python l'algorithme 1.

Question 2. En utilisant la fonction `time` de la bibliothèque `time` (qui retourne l'heure en secondes à un moment précis de l'exécution d'un morceau de code), mesurez le temps que prend l'exécution de cet algorithme pour différentes valeurs d'entrée n.

Question 3. Ensuite, dans un *Notebook*, écrivez un script affichant la courbe des temps mesurés en fonction de la taille d'entrée. Affichez une deuxième courbe correspondant à la complexité asymptotique théorique.

Question 4. Si le temps le permet, proposez une version plus efficace de cet algorithme (c'est à dire, un autre algorithme permettant d'obtenir le même résultat mais avec une complexité asymptotique plus faible). Reprenez les trois premières questions avec ce dernier.