Série 8

Sauf si spécifié autrement, tous les algorithmes demandés sont à écrire sous forme d'une fonction Python.

Exercice 1

Ordonnez sans justification les fonctions suivantes par ordre de croissance, c'est-à-dire pour deux fonctions données f et g, faites apparaître f avant g si $f = \mathcal{O}(g)$. Groupez ensemble les fonctions qui ont le même ordre de croissance (c'est-à-dire groupez ensemble f et g si $f = \Theta(g)$).

$$n + 50,\ n\sqrt{n},\ 2n,\ 100,\ 10n^{0.1},\ n^2,\ n^3 + n^2,\ n^3,\ n^{1.1},\ |\sin(n)| + 1,\ n^{10}.$$

Pour vous aider dans cette tâche, n'hésitez pas à utiliser un outil de plotting en ligne comme Wolfram Alfa, Geogebra, ou Desmos pour observer la vitesse de croissance de diverses fonctions.

Exercice 2

Pour $n \in \mathbb{N}$, prouvez les affirmations suivantes en exhibant une constante C (ou des constantes C_1 et C_2) et un rang N appropriés :

(a) $2n + 100 = \Theta(n)$
(b) $an + b = \Theta(n)$ pour a, b des réels strictement positifs
(c) $100n\sqrt{n} = \mathcal{O}(n^2)$.

Exercice 3

(a) Soit $n \in \mathbb{N}$, et f, g, h des fonctions positives de n telles que $f = \mathcal{O}(g)$ et $g = \mathcal{O}(h)$. Prouvez que $f = \mathcal{O}(h)$.
(b) Soient $T_s(n)$ et $T_\ell(n)$ les temps de parcours respectifs des algorithmes `max_somme` et `max_somme_lineaire` tels qu'ils ont été définis au cours.
  (i) En utilisant le point (a), déduire que $T_\ell(n) = \mathcal{O}(T_s(n))$.
  (ii) En utilisant le fait que $n^2 \neq \mathcal{O}(n)$, déduire que $T_s(n) \neq \mathcal{O}(T_\ell(n))$. Regardez en bas de page pour un indice. [1]

1. C'est peut-être l'heure d'être un peu *absurde*...

Exercice 4

(a) Donnez un algorithme `carre` qui prend un entier positif n en entrée et affiche un carré de côté n. Par exemple, l'appel `carre(5)` doit afficher :

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

L'espaçage entre les astérisques n'est pas important.

(b) Combien d'astérisques est-ce que votre code affiche, pour l'entrée n ?

(c) Donnez, en notation $\Theta(\cdot)$, l'ordre de croissance du temps de parcours de votre algorithme en fonction de n.

Remarque : si votre algorithme contient des instructions de la forme `print("*" * i)`, sachez qu'une telle instruction ne prend pas temps constant ! En effet la création et l'affichage d'une chaîne de caractère de taille n prendra un temps au moins linéaire en n.
Un algorithme contenant deux boucles `for` imbriquées prendra le même temps de parcours mais ce temps sera plus facile à analyser.

(d) Donnez un algorithme `triangle` qui prend un entier positif n en entrée et affiche un triangle de côté n de la forme ci-dessous (dans ce cas, ce triangle a été affiché par `triangle(5)`) :

```
*
* *
* * *
* * * *
* * * * *
```

Puis répondez aux mêmes questions (b) et (c) pour l'algorithme `triangle`.

Exercice 5

Soit `L` une liste de $n \geq 3$ nombres. On s'intéresse à la plus grande somme de trois éléments distincts de la liste, c'est-à-dire au maximum de la valeur de `L[i] + L[j] + L[k]`, pour `i, j, k` des indices de `L` distincts deux à deux.

(a) Modifiez chacun des algorithmes `max_somme` et `max_somme_lineaire` vus en cours pour calculer le maximum demandé.

(b) Utilisez le module `time` comme vu en cours pour mesurer le temps de parcours de chacun de vos algorithmes sur une liste de nombre aléatoires dont vous ferez varier la taille. Est-ce que c'est soutenable de donner une liste de taille 1000 à ces deux algorithmes ? Une liste de taille $10,000$?

(c) Donnez, en notation $\Theta(\cdot)$, l'ordre de croissance du temps de parcours de chacun des deux algorithmes en fonction de la taille de l'entrée.
Indication : vous pouvez utiliser les identités

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \text{ et } \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}.$$

Exercice 6

On considère le problème de la multiplication de deux matrices numériques de dimensions $n \times n$, pour $n \geq 2$.
Pour rappel, soient A et B des matrices de dimensions $n \times n$, et $C = AB$ la matrice produit de A et B. On dénote par a_{ij} l'élément de la $i$ème ligne et $j$ème colonne de A (et de même pour B et C). Alors

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

C'est-à-dire que c_{ij} est obtenu en calculant le produit scalaire de la $i$ème ligne de A et la $j$ème colonne de B. Pour calculer l'entrée c_{ij} de la matrice produit C, il faut donc calculer une somme où chaque terme est le produit d'une entrée de A et d'une entrée de B.
Par exemple, si

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & -1 & 2 \\ 5 & -2 & 1 \end{pmatrix} \text{ et } B = \begin{pmatrix} 2 & -1 & -2 \\ 0 & 3 & 2 \\ 1 & 4 & 3 \end{pmatrix},$$

alors

$$c_{11} = 1{\cdot}2{+}2{\cdot}0{+}3{\cdot}1 = 5,\ c_{12} = 1{\cdot}{-}1{+}2{\cdot}3{+}3{\cdot}4 = 17, \text{ et } C = \begin{pmatrix} 5 & 17 & 11 \\ 10 & 1 & -4 \\ 11 & -7 & -11 \end{pmatrix}.$$

(a) Ecrivez un algorithme qui prend en entrée deux matrices numériques A et B de dimensions $n \times n$ et retourne la matrice $C = A \cdot B$. On représentera une matrice de dimensions $n \times n$ en Python par une liste de taille n dont chaque élément est une liste de taille n. Par exemple, la matrice

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

sera représentée par `A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`.
N'oubliez pas de tester votre code sur plusieurs instances de petite taille.

(b) Quelle est la taille de l'entrée en fonction de n ? Quel est le temps de parcours de votre algorithme en fonction de n ?

(c) Donnez une borne inférieure (triviale) sur le temps de parcours de n'importe quel algorithme qui résoud le problème de la multiplication de deux matrices de dimensions $n \times n$.

Exercice 7 (facultatif)

(a) Pour f et g des fonctions positives de n, prouvez que

$$f = \Omega(g) \iff g = \mathcal{O}(f).$$

(b) Pour f et g des fonctions positives de n, prouvez que

$$f = \Theta(g) \iff f = \mathcal{O}(g) \text{ et } f = \Omega(g).$$

(c) Soient p, q rationnels tels que $0 < p < q$. Prouver que

$$n^p = \mathcal{O}(n^q) \text{ et } n^q \neq \mathcal{O}(n^p).$$

Regardez en bas de page pour un indice. [2]

2. L'heure sonne à nouveau ! (cf. Exercice 3(b)(ii))