EA4 – Éléments d'algorithmique
TD n° 2 : notations de Landau et complexité

Exercice 1 : notations O, Ω, Θ et limites

Dans cet exercice et le suivant on considère des fonctions f et g sur les entiers naturels $\mathbb{N}$, à valeurs dans $\mathbb{N}$ et strictement positives presque partout. Si on utilise des fonctions comme la racine carrée alors on suppose implicitement que le nombre réel $\sqrt{n}$ est arrondi à un entier naturel.
Argumenter pour ou contre les assertions suivantes :

1. $f \in \Omega(g)$ ssi $g \in O(f)$.
2. $f \in O(g)$ ssi la suite $\{\frac{f(n)}{g(n)}\}_{n\in\mathbb{N}}$ a une limite finie (on suppose $g(n)$ non-nul).
   Suggestion : considérer les fonctions f et g telles que : $f(0) = g(0) = 1$, $f(n+1) = 2 \cdot f(n)$, $g(2k+1) = g(2k)$, $g(2k+2) = 4 \cdot g(2k)$, pour $k, n \in \mathbb{N}$.
3. $\forall f, g$ ($f \in \Theta(g)$ ssi $g \in \Theta(f)$).
4. $\forall f, g,\ \max(f, g) \in \Theta(f + g)$.
5. $\forall f, g\ \min(f, g) \in \Theta(f + g)$.

Exercice 2 : classement

On dit que deux fonctions f et g sont *équivalentes* si $f \in \Theta(g)$ (est-ce une *bonne* définition d'équivalence ?). Soit F un ensemble dont les éléments sont les 10 fonctions suivantes :

$$n^2 - 5n, \quad n + 2n^3 \log n, \quad n^2 + n \log n, \quad n^2\sqrt{n}, \quad \sqrt{n}, \quad 2^n, \quad e^n, \quad 2^{n+4}, \quad \log n, \quad \log(n^4)\ .$$

1. Partitionner F en classes de fonctions équivalentes.
2. Ordonner par ordre croissant les classes obtenues. La classe C est plus petite ou égale à la classe C' si on peut trouver $f \in C$ et $g \in C'$ telles que $f \in O(g)$.

Exercice 3 : complexité de boucles et fonctions python

1. Quel est l'ordre de grandeur du nombre d'additions effectuées par les deux boucles suivantes ?

```
for i in range(n):
   for j in range(n):
       for k in range(n):
           val1 = val1 + 1
```

```
for i in range(n):
    for j in range(i):
        val1 = val1 + 1
```

2. On rappelle que la complexité de la multiplication de deux nombres à n bits est en $O(n^2)$. Proposer une borne supérieure à la complexité de la fonction suivante en fonction des entiers naturels s et k.

```
def iterated_square(s, k):
    for i in range(k):
       s = s * s
    return s
```

3. On considère deux fonctions python `A` et `B`, qui prennent en argument une liste de flottants et renvoient une liste de flottants. On sait que `A` est de complexité $O(n^2)$, et `B` de complexité $O(n^{3/2})$, où n désigne la taille de la liste passée en argument. Soit `C` la composition de `A` par `B` : `C(liste)` renvoie `B(A(liste))`. Que pouvez-vous dire sur la complexité de la fonction `C` ?

Exercice 4 : Euclide rencontre Fibonacci

D'après Euclide, le calcul du plus grand commun diviseur $pgcd(a, b)$, pour $a \geqslant b > 0$, peut être organisé récursivement comme suit où $a \ mod \ b$ est le reste de la division entière de a par b :

$$pgcd(a, b) = \begin{cases} b & \text{si } a \ mod \ b = 0, \\ pgcd(b, a \ mod \ b) & \text{sinon.} \end{cases}$$

1. Donner une borne supérieure en fonction de a du nombre d'appels récursifs de la fonction *pgcd*. *Suggestion :* déplier deux fois la définition.
2. On admet qu'une division euclidienne sur des nombres de n bits peut être calculée en $O(n^2)$. En déduire une borne supérieure pour la complexité du calcul de $pgcd(a, b)$ en fonction du nombre de bits qu'il faut pour représenter le nombre a.
3. Soit F_n le n^{e} terme de la suite de Fibonacci. Que se passe-t-il dans le calcul de $pgcd(F_{n+1}, F_n)$?
4. Estimer en fonction de n le nombre de bits nécessaires à représenter F_n.
   Suggestion : $log_2(1, 6) \approx 0, 68$.
5. Supposons qu'on manipule des nombres représentés sur 10^3 bits (ce qui est ordinaire en cryptographie). Donner une borne supérieure et une borne inférieure au nombre d'appels récursifs (dans le pire des cas) de la fonction *pgcd* sur des nombres de cette taille.

Exercice 5 : un algorithme récursif

On considère la fonction suivante :

```
def F(n) :
    if n < 3 :
        return 1
    else :
        return 2 * F(n - 1) + F(n - 3)
```

Soit $A(n)$ le nombre d'additions effectuées lors de l'exécution de `F(n)`.

1. Donner une définition de $A(n)$ par récurrence. En déduire que A est croissante.
2. Montrer que $A(n) \in \Omega(2^{n/3})$ où `/` est la division euclidienne.
3. Proposer un algorithme qui fait un nombre linéaire d'additions pour calculer les mêmes valeurs que `F(n)`.
4. Quelle est la complexité de cet algorithme ?
5. Un petit challenge : proposer un point de vue matriciel (voir CM n° 2) sur le calcul de `F(n)` et en déduire un algorithme alternatif. Quelle est sa complexité ?