EA4 – Éléments d'algorithmique
TD n° 2 : calculs de complexité

Exercice 1 : ordres de grandeur

1. Montrer que $\frac{1}{5}n^3 - 3n^2 \in \Theta(n^3)$.
2. Montrer que $\frac{1}{2}n^4 + 2n + 3 \in \Theta(n^4)$.
3. Montrer que $5n^3 \notin \Theta(n^4)$.
4. Comparer $n!$ à 2^n et à n^n.

Exercice 2 :

Dans la suite, f et g désignent des fonctions à valeurs dans $\mathbb{N}^\star$.

1. Montrer que $\forall f, g,\ f \in \Theta(g) \iff g \in \Theta(f)$.
2. Montrer que $\forall f, g,\ \max(f, g) \in \Theta(f + g)$.
3. Qu'en est-il de $\min(f, g)$?

Exercice 3 : classement

Classer les fonctions suivantes en fonction de leur ordre de grandeur dans les classes Θ_1 à Θ_7 en respectant les conditions suivantes :

– deux fonctions appartiennent à la même classe Θ_i si et seulement si elles sont du même ordre de grandeur :

$$\forall f, g, \quad f \in \Theta(g) \iff \exists i,\ f, g \in \Theta_i$$

– les classes Θ_i sont rangées en ordre de grandeur croissant :

$$\forall i,\ \forall f, g, \quad f \in \Theta_i \text{ et } g \in \Theta_{i+1} \implies f \in O(g)$$

Liste des fonctions à traiter :

$$2n^2 + 3n, \quad n^2 + \frac{1}{8}n^3, \quad n^2 + \sqrt{n}, \quad n^2\sqrt{n}, \quad \sqrt{n}, \quad 2^n, \quad 4^n, \quad 2^{n+4}, \quad \log n, \quad \log(n^2)$$

Θ_1	Θ_2	Θ_3	Θ_4	Θ_5	Θ_6	Θ_7

Exercice 4 : complexité des boucles

Exprimer le plus simplement possible l'ordre de grandeur du nombre d'instructions effectuées par les boucles suivantes des morceaux de code suivants :

1.
```
for i in range(n) :
    for j in range(n) :
      for k in range(n) :
        # instruction de coût constant
```

2.
```
for i in range(n) :
    for j in range(i) :
      # instruction de coût constant
```

On considère maintenant des algorithmes prenant un paramètre entier `n`, et deux fonctions de `n` notées f et g.

3. On suppose que l'algorithme `A(n)` fait $\Theta(f(n))$ tours de boucle, chacun ayant une complexité $\Theta(g(n))$. Quelle est l'ordre de grandeur de sa complexité ?

Exercice 5 : un algorithme récursif

On considère l'algorithme suivant :

```
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 $A \in \Omega(2^{n/3})$ 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. Quel est la complexité de cet algorithme ?

5. Existe-t-il un algorithme de complexité inférieure ?