TD5 : Initiation à la complexité

Exercice 1 Classer les fonctions suivantes en fonction de leur ordre de grandeur dans les classes Θ_1 à Θ_7 : les fonctions appartiennent à la même classe Θ_i si et seulement si elles sont du même ordre de grandeur, et les classes Θ_i sont rangées en ordre croissant :

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

Liste des fonctions à traiter :

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

Θ_1	Θ_2	Θ_3	Θ_4	Θ_5	Θ_6	Θ_7

Exercice 2

1. On considère la fonction suivante :

```
1  /** a, n : entiers positifs ou nuls */
2  int puissance(int a, int n){
3    p = 1;
4    for(int i=0; i < n; i = i+1){
5      p = p * a;
6    }
7    return p;
8  }
```

   Combien de multiplications sont faites lors d'un appel à cette fonction avec des paramètres a et n ?

2. Combien y a-t-il d'itérations lors d'un appel à la fonction suivante avec des paramètres a et n ?

```
1  int puissance_rapide(int a, int n){
2    int p = 1;
3    while (n > 0) {
4      if (n%2 == 1) {
5        p = p * a;
6      }
7      a = a * a;
8      n = n / 2;
9    }
10
11   return p;
12 }
```

   En déduire un encadrement du nombre d'opérations (multiplications et divisions).

Exercice 3 On considère la fonction suivante :

```
1  int f(int n){
2    if(n < 2){
3      return n;
4    }
5    return 2 * f(n - 1) + f(n - 2);
6  }
```

On note $M(n)$ le nombre de multiplications effectuées lors d'un appel à `f(n)`.

1. Donner les valeurs de $M(0)$ et $M(1)$.

2. Donner une relation de récurrence sur la suite $(M(n))_{n\geq 0}$.

Exercice 4 Évaluation naïve d'un polynôme – version 1 Dans cet exercice, on suppose que les opérations élémentaires sont les multiplications.
On considère le code suivant :

```
1  /** a : un nombre à virgule flottante
2   * k : un entier
3   * sortie : a puissance k (on s'affranchit des problèmes de précision)
4   */
5  double puissance(double a, int k){
6    double p = 1;
7
8    for(int i = 0; i < k; i = i+1){
9      p = p*a;
10   }
11
12   return p;
13 }
14
15 /** x : point en lequel le polynôme doit être évalué
16  * pol : coefficients d'un polynôme de degré deg
17  * sortie : évaluation du polynôme pol en le point x
18  */
19 double evaluation_naive(double x, double *pol, int deg){
20   double px = 0;
21
22   for(int k = 0; k <= deg; k = k+1){
23     px = px + coeff[k] * puissance(x, k);
24   }
25
26   return px;
27 }
```

On suppose que les fonctions `puissance` et `evaluation_naive` possèdent une correction totale [1].

1. Donner un ordre de grandeur de la complexité en temps de `puissance` en fonction de ses entrées.
2. Donner un ordre de grandeur de la complexité en temps de `evaluation_naive` en fonction de ses entrées.

Exercice 5 Évaluation naïve d'un polynôme – version 2 Dans cet exercice, on suppose que les opérations élémentaires sont les multiplications.

1. Rappeler, sous forme d'une fonction en `C`, l'algorithme d'exponentiation rapide vu en TD.
2. Donner sa complexité en temps dans le pire des cas.
3. Donner la complexité en temps dans le pire des cas d'une fonction `evalutation` identique à la fonction `evalutation_naive` de l'exercice précédent, à ceci près qu'elle ferait appel à votre fonction d'exponentiation rapide plutôt qu'à la fonction `puissance` (ligne 23 dans `evalutation_naive`).

1. C'est un bon exercice à faire chez soi pour s'entraîner