Exercice 1. Évaluation de Horner

Soit $P : \mathbb{R} \to \mathbb{R}$ un polynôme à coefficients réels et $x \in \mathbb{R}$. On souhaite compter le nombre d'additions et de multiplications nécessaires pour évaluer $P(x)$. Le polynôme P sera représenté par une liste de flottants contenant les coefficients de P. Le premier élément de la liste est le coefficient constant et le dernier est le coefficient dominant. Par exemple, $P(X) = X^3 + 2X + 4$ est représenté par `[4.; 2.; 0.; 1.]`.

On s'intéresse dans un premier temps à la fonction `eval` :

```
let eval (p0: float list) (x: float): float =
  let rec aux (y: float) (p: float list): float = match p with
    | [] -> 0.
    | c::q -> c *. y +. aux (y *. x) q in
  aux 1. p0;;
```

1. Montrer la terminaison de la fonction `eval`.
2. Montrer la correction de la fonction `eval`.
3. Déterminer le nombre exact de multiplications et le nombre exact d'additions utilisées par la fonction `eval`.
4. Déterminer le temps d'exécution de la fonction `eval`.

On s'intéresse maintenant à "l'évaluation de Horner". Le principe est d'utiliser l'égalité :

$$\sum_{k=0}^{n-1} p_k x^k = p_0 + x\Big(p_1 + x\big(p_2 + \ldots + x(p_{n-2} + xp_{n-1})\ldots\big)\Big)$$

On écrit donc la fonction :

```
let rec horner (p: float list) (x: float): float = match p with
  | [] -> 0.
  | c::q -> c +. x *. horner q x;;
```

Admettons la terminaison de la fonction `horner`.

5. Montrer la correction de la fonction `horner`.
6. Déterminer le nombre exact de multiplications et le nombre exact d'additions utilisées par la fonction `horner`.

Exercice 2. Fonction 91 de McCarthy

On appelle fonction 91 de McCarthy la fonction récursive suivante :

```
let rec f n = if n > 100 then n-10 else f (f (n+11));;
```

Pour tout `n` $\in \mathbb{Z}$, montrer qu'un appel à « `f n` » termine et déterminer la valeur renvoyée.

Exercice 3. Fonction mystère

```
(* Hypothèse: n >= 0 et k >= 0. *)
let rec f n k = match k with
  | 0 -> 1
  | k -> let res = f (n*n) (k/2) in
         if k mod 2 = 0 then res else n*res;;
```

1. Que renvoie la fonction `f` ?
2. Montrer la terminaison de la fonction `f`.
3. Montrer la correction de la fonction `f`.

Pour $n \in \mathbb{N}$ et $k \in \mathbb{N}$, on note $C(k)$ le temps d'exécution d'un appel à `f` sur n et k. Notez qu'à priori, $C(k)$ dépend de n, mais on peut se convaincre facilement que le fait de modifier n ne modifie pas l'ordre de grandeur du temps d'exécution.

4. Montrer qu'il existe une constante $c_1 > 0$ telle que :
$$\begin{cases} C(0) \leqslant c_1 \\ C(2j) \leqslant c_1 + C(j) & \text{pour tout } j > 0 \\ C(2j+1) \leqslant c_1 + C(j) & \text{pour tout } j \geqslant 0 \end{cases}$$
5. Montrer que pour tout $k \in \mathbb{N}$:
$$C(k) \leqslant \frac{c_1}{\log_2(3/2)} \Big(\log_2(k+1) + 1 \Big)$$
6. Montrer que $C(k) = \Theta(\log k)$.

Exercice 4. Triangle de Pascal

Pour tout $n \in \mathbb{N}$ et $k \in \mathbb{Z}$, le coefficient binomial $\binom{n}{k}$ est défini par :
$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \text{ si } 0 \leqslant k \leqslant n \qquad \binom{n}{k} = 0 \text{ sinon}$$
On rappelle la formule de Stirling :
$$n! = \Theta\left(\frac{n^n}{e^n}\sqrt{n}\right)$$
Afin de calculer les coefficients binomiaux, on propose d'utiliser le triangle de Pascal implémenté avec la fonction `binom1` :

```
(* Hypothese: n >= 0 *)
let rec binom1 n k = match n,k with
  | 0,0 -> 1
  | 0,_ -> 0
  | n,k -> binom1 (n-1) k + binom1 (n-1) (k-1);;
```

1. Montrer la terminaison de la fonction `binom1`.
2. Montrer la correction de la fonction `binom1`.
3. Monter que la complexité de la fonction `binom1` est en $\Theta(2^n)$.

On s'intéresse maintenant à la fonction `binom2` :

```
let rec binom2 n k =
  if k < 0 || n < 0 || n < k then 0
  else if k = 0 || n = k then 1
  else binom2 (n-1) k + binom2 (n-1) (k-1);;
```

On admet que la fonction `binom2` termine et est correcte.

4. Trouver une suite $(k_n)_{n\in\mathbb{N}}$ telle que la complexité de l'appel à « `binom2 n` k_n » soit en $\Theta(1)$.
5. Trouver une suite $(k_n)_{n\in\mathbb{N}}$ telle que la complexité de l'appel à « `binom2 n` k_n » soit en $\Theta(n)$.
6. Trouver une suite $(k_n)_{n\in\mathbb{N}}$ telle que la complexité de l'appel à « `binom2 n` k_n » soit en $\Theta(2^n/\sqrt{n})$ (c'est la complexité maximale qu'on peut obtenir).
   Indication : on pourra calculer le temps d'exécution pour tout couple (n, k) avec $0 \leqslant k \leqslant n$.

Exercice 5. Algorithme d'Euclide

1. Écrire une fonction qui prend en entrée deux entier $n, m \in \mathbb{N}$ et renvoie leur PGCD à l'aide de l'algorithme d'Euclide. On utilisera une fonction récursive.
2. Montrer la terminaison de votre fonction.
3. Montrer la validité de votre fonction.

Remarque. La complexité temporelle pour calculer le PGCD de n et m avec l'algorithme d'Euclide est en $\mathcal{O}(\log(\min(n, m)))$ (*cf.* TP d'informatique commune sur la complexité).