Informatique - MPI

Question 1.

□ **1.1.** Un algorithme naïf applique directement la relation de récurrence de la suite de Fibonacci.

```
(* calcul récursif d'un nombre de Fibonacci *)
let rec fibo n =
  if n = 0 || n = 1 then 1
  else fibo (n-1) + fibo (n-2)
```

□ **1.2.** N_n désigne la complexité de `fibo` en terme de *nombre d'appels à la fonction* et de *nombre d'additions*. Les deux cas de base sont $N_0 = 1$ et $N_1 = 1$ pour lesquels un seul appel à la fonction renvoie directement le résultat. Pour tout entier naturel n au moins égal à 2, deux appels récursifs, de coûts N_{n-1} et N_{n-2}, et une addition sont effectués. Ainsi :

$$\begin{cases} N_0 = 1 \\ N_1 = 1 \\ \forall n \geqslant 2 \quad N_n = N_{n-1} + N_{n-2} + 1 \end{cases}$$

Un raisonnement par récurrence établit alors que :

$$\forall n \in \mathbb{N} \quad N_n = F_n$$

□ **1.3.** La relation de récurrence associée à la suite de Fibonacci (F_i) est linéaire homogène d'ordre 2. Sa résolution mène à l'expression générale suivante de F_n.

$$\forall n \in \mathbb{N} \quad F_n = \lambda\varphi^n + \mu\hat{\varphi}^n$$

avec $\varphi = (1+\sqrt{5})/2$ et $\hat{\varphi} = (1-\sqrt{5})/2$. λ et μ sont des constantes déterminées par les conditions $F_0 = 1$ et $F_1 = 1$. Comme $\varphi > 1$ et $|\hat{\varphi}| < 1$, il vient $F_n = O(\varphi^n)$ puis $N_n = O(\varphi^n)$. La solution récursive est de complexité temporelle exponentielle.

Remarque. On pourrait envisager d'utiliser la connaissance de l'expression *exacte* de F_n pour la faire calculer par un ordinateur. Même si φ et $\hat{\varphi}$ sont des nombres réels, leur combinaison pour former F_n est un nombre entier. Malheureusement, un ordinateur ne peut mener des calculs qui préservent ce caractère entier. Les valeurs de φ et $\hat{\varphi}$ sont définies de manière interne par des *flottants* et tous les calculs qui s'ensuivent sont faits sur les flottants, avec toutes les erreurs susceptibles de se propoager qu'une telle représentation implique. Il est donc impossible d'utiliser l'expression générale de F_n pour calculer les *entiers* associés avec un ordinateur.

Question 2. Un algorithme itératif calcule F_n avec une complexité temporelle linéaire en n. Sa complexité spatiale est en outre constante.

```
(* calcul itératif d'un nombre de Fibonacci *)
let fibo n =
  let u = ref 1 and v = ref 1 in
  for i = 2 to n do
    let w = !v in v := !u + !v; u := w
  done;
  !v
```

La complexité temporelle linéaire est liée à la présence de la boucle `for` qui effectue $n-1$ itérations.

Question 3. En remarquant que :

$$\forall n \in \mathbb{N}^* \begin{cases} F_{n+1} = F_n + F_{n-1} \\ F_n = F_n \end{cases}$$

on peut adopter une formulation matricielle du problème sous la forme :

$$\begin{cases} X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ \forall n \in \mathbb{N}^* \quad X_n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} X_{n-1} \end{cases}$$

Il vient alors :

$$\forall n \in \mathbb{N} \quad X_n = M^n X_0$$

avec $M^0 = I_2$, matrice identité 2×2. On extrait F_n de X_n.

L'analyse précédente montre que le calcul de F_n peut être mené en calculant d'abord M^n. On déplace ainsi le problème initiale vers celui du calcul de la puissance d'une matrice. Dans le cas présent, la matrice M est de taille 2×2. Calculer son carré est de coût $8 = 2^3$. L'algorithme d'*exponentiation rapide* peut être mis en œuvre pour calculer sa puissance n-ième. Il convient de remplacer la multiplication des nombres par celle des matrices, ce qui, dans la situation présente, est de coût constant. Si on se rappelle que l'algorithme d'exponentiation rapide est de complexité temporelle $O(\log n)$, il est donc possible de calculer F_n avec une même complexité temporelle.

```
(* produit de matrices 2 × 2 *)
let prod m1 m2 =
  let a1, b1, c1, d1 = m1
  and a2, b2, c2, d2 = m2
  in
  (a1*a2 + b1*c2, a1*b2 + b1*d2, c1*a2 + d1*c2, c1*b2 + d1*d2)

(* carré d'une matrice 2 × 2 *)
let square m =
  prod m m

(* calcul par exponentiation rapide d'un nombre de Fibonacci *)
let fibo n =
  let m = (1, 1, 1, 0) in
  let rec fast_exp m i =
    if i = 0 || i = 1 then m
    else
      let m2 = square (fast_exp m (i/2)) in
      if i mod 2 = 0 then m2 else prod m m2
  in
  let f, _, _, _ = fast_exp m n in f
```