I Introduction

Exercice 1

```
let rec pgcd a = function
   | 0 -> a
   | b -> pgcd b (a mod b);;
```

Exercice 2

On traite les différents cas de bases (longueur 0, 1 et 2) et on raisonne de la façon suivante. Le mur peut soit se terminer par une brique de taille 3, auquel cas on compte les façons de remplir n - 3, soit se terminer par une brique de taille 2, auquel cas on compte les possibilités.

```
let rec briques = function
   | 0 -> 1
   | 1 -> 0
   | 2 -> 1
   | n -> briques (n - 2) + briques (n - 3);;
```

II Analyse d'algorithmes

II.1 Terminaison

Exercice 3

Les arguments sont des valeurs de $\mathbb{N}^2$. On utilise la relation d'ordre strict suivante :

$$(a,b) \prec (c,d) \Leftrightarrow b < d$$

Dès lors, on constate que les éléments minimaux sont les $(a,0)$ pour $a \in \mathbb{N}^*$, qui sont traités par le cas de base de la fonction. Si $b \neq 0$, alors le calcul de `pgcd a b` lance un appel avec $(b, a \mod b) \prec (a,b)$ par définition du modulo. Par le théorème d'induction, on en déduit le résultat.

Exercice 4

On utilise ici l'ordre lexicographique sur $\mathbb{N}^2$. Les cas de base (qui contiennent plus que l'unique élément minimal) sont les $(0,n)$. Sinon, deux cas sont possibles :

- Si $n = 0$, on lance un appel à $(m-1,1) \prec (m,0)$.
- Sinon, on lance un appel à $(m,n-1) \prec (m,n)$, puis à $(m-1, A(m,n-1)) \prec (m,n)$.

On conclut par le théorème d'induction.

Exercice 5

Distinguons les cas :

- Si $n > 100$, alors $f(n)$ termine.
- Si $n \in [\![90;100]\!]$, alors $f(n) = f(f(n+11)) = f(n+1)$, car $n+11 > 100$. On en déduit que $f(90) = f(91) = \ldots = f(100) = f(101) = 91$ et que ce calcul termine.
- Si $n < 90$, alors soit k l'unique entier tel que $n + k \times 11 \in [\![90;100]\!]$. On a, par une récurrence rapide, que $f(n) = f^{k+1}(n + k \times 11)$. Or, $f(n + k \times 11)$ termine et vaut 91. De plus, $f(91)$ termine

et vaut 91. On en déduit que $f^{k+1}(n + k \times 11)$ termine et vaut 91.

II.2 Correction

Exercice 6

On montre ce résultat par induction sur le couple (n, ℓ), avec la relation d'ordre : $(n, \ell) \prec (m, q) \Leftrightarrow n < m$ et $|\ell| < |q|$:

- Les deux cas de bases sont le cas $n = 0$ ou $|\ell| = 0$. Le résultat est direct de par le traitement de ces cas.
- Sinon, supprimer n éléments de ℓ revient à supprimer son premier élément, puis les $n-1$ suivants. C'est ce que renvoie l'appel à `supprime (n - 1) q`.

On conclut par induction.

II.3 Complexité

Exercice 7

On a, en notant $C(n)$ la complexité de la fonction avec n comme argument :

- $C(0) = O(1)$, $C(1) = O(1)$ et $C(2) = O(1)$.
- Si $n \geq 3$, $C(n) = C(n-2) + C(n-3) + O(1)$.

Ainsi, $C(n)$ est plus grand que le terme général d'une suite récurrente linéaire d'ordre 3. Sans rentrer dans les détails de calcul (très lourds ici), on en déduit que la complexité est exponentielle.

Exercice 8

1. S'il n'y a qu'un seul disque à déplacer, alors on le déplace en un seul mouvement. Sinon, on déplace les $n-1$ premiers disques sur la colonne de libre, puis le n-ème disque sur la colonne d'arrivée, puis les $n-1$ sur la colonne d'arrivé.

```
let hanoi n =
   let rec aux dep arr = function
      | 1 -> prt_tp (dep, arr)
      | n -> let inter = 3 - dep - arr in
             aux dep inter (n - 1);
             prt_tp (dep, arr);
             aux inter arr (n - 1) in
   aux 0 2 n;;
```

2. Si on note $D(n)$ le nombre de déplacements, alors :
   - $D(1) = 1$
   - Si $n > 1$, $D(n) = 2D(n-1) + 1$.

   On reconnait ici une suite arithmético-géométrique. On sait que la solution est de la forme $D(n) = \lambda \times 2^n + \mu$. On utilise les conditions initiales pour déterminer les constantes :
   - $D(1) = 1 = 2\lambda + \mu$.
   - $D(2) = 3 = 4\lambda + \mu$.

   On en déduit : $2\lambda = 2$, soit $\lambda = 1$, puis $\mu = -1$ et donc $D(n) = 2^n - 1$.
3. Cette fois-ci, on distingue selon le nombre de colonne à parcourir pour le déplacement (le cas de base est le même, nécessitant éventuellement 2 déplacements) :

– Si la colonne d'arrivée est la suivante de celle de départ, il faut d'abord mettre les $n-1$ sur l'intermédiaire (donc à distance 2), puis la n-ème sur celle d'arrivée (donc un déplacement nécessaire), puis de l'intermédiaire à celle d'arrivée (à nouveau distance 2).
– Sinon, il faut mettre les $n-1$ sur la colonne d'arrivée, puis la n-ème sur l'intermédiaire, puis les $n-1$ sur la colonne de départ, puis la n-ème sur celle d'arrivée, puis les $n-1$ sur celle d'arrivée.

```
let hanoi n =
   let rec aux dep arr n =
      let inter = 3 - dep - arr in
      match n, (dep + 1 - arr) mod 3 with
         | 1, 0 -> prt_tp (dep, arr)
         | 1, _ -> prt_tp (dep, inter);
                   prt_tp (inter, arr)
         | n, 0 -> aux dep inter (n - 1);
                   prt_tp (dep, arr);
                   aux inter arr (n - 1);
         | n, _ -> aux dep arr (n - 1);
                   prt_tp (dep, inter);
                   aux arr dep (n - 1);
                   prt_tp (inter, arr);
                   aux dep arr (n - 1) in
   aux 0 2 n;;
```

On distingue alors deux calculs : $D_1(n)$ correspond au nombre de déplacements nécessaires pour déplacer n disques vers la colonne suivante et $D_2(n)$ vers une colonne à distance 2. On a les relations :

– $D_1(1) = 1$.
– $D_2(1) = 2$.
– Pour $n > 1$, $D_1(n) = 2D_2(n-1) + 1$.
– Pour $n > 1$, $D_2(n) = 2D_2(n-1) + D_1(n-1) + 2$.

On remarque alors : $D_2(n) = 2D_2(n-1) + 2D_2(n-2) + 3$. Dès lors, en posant $u_n = D_2(n) + 1$, on a $u_n = D_2(n) + 1 = 2D_2(n-1) + 2D_2(n-2) + 4 = 2u_{n-1} + 2u_{n-2}$.
On commence alors par résoudre l'équation caractéristique : $X^2 - 2X - 2$, qui a pour solution $1+\sqrt{3}$ et $1-\sqrt{3}$, toutes deux réelles. Le terme général est donc de la forme :
$u_n = \lambda(1+\sqrt{3})^n + \mu(1-\sqrt{3})^n$. Les conditions initiales permettent de trouver (sauf erreur) $\lambda = \frac{1}{2} + \frac{\sqrt{3}}{3}$ et $\mu = \frac{1}{2} - \frac{\sqrt{3}}{3}$. On en déduit $D_2(n)$ qui est le nombre de déplacements cherchés.

III Utilisation de la mémoire

III.1 Pile d'appels

III.2 Récursivité terminale

Exercice 9

On utilise un accumulateur, initialisé à 1, qui stocke la valeur courante du calcul.

```
let fact n =
   let rec aux acc = function
      | 0 -> acc
      | n -> aux (n * acc) (n - 1) in
   aux 1 n;;
```

III.3 Mémoïsation

Exercice 10

1. On utilise la formule de Pascal :

```
let rec binom k n = match k with
   | 0 -> 1
   | k when k = n -> 1
   | _ -> binom (k - 1) (n - 1) + binom k (n - 1);;
```

2. On remarque, de par sa définition, que $C(k,n) \geq \binom{n}{k}$. On a donc une complexité exponentielle.

3. On s'inspire du calcul de la suite de Fibonacci, mais cette fois-ci avec une matrice :

```
let m = Array.make_matrix 100 100 (-1);;
for i = 0 to 99 do
   m.(0).(i) <- 1;
   m.(i).(i) <- 1
done;;
let rec binom_mem k n =
   if m.(k).(n) = -1 then
      m.(k).(n) <- binom_mem (k - 1) (n - 1) + binom_mem k (n - 1);
   m.(k).(n);;
```

4. Il suffit de déterminer le nombre de cases qui doivent être calculées dans la matrice, c'est-à-dire la moitié de la sous-matrice de taille n (où $k \leq n$). La complexité est donc en $O(n^2)$ dans le pire des cas.