Correction de devoir du Vendredi 28 Mars

Exercice 1

1. (a)

```
let fib1 n =
  let a = ref 0 and b = ref 1 and aux = ref 0 in
    for k = 1 to n do
      aux := !b;
      b := !a + !b;
      a := !aux
    done;
  !a
;;
```

(b)

```
let rec fib2 n = match n with
  | 0 -> 0
  | 1 -> 1
  | _ -> fib2 (n-1) + fib2 (n-2)
;;
```

(c)

```
(*fonction auxiliaire*)

let rec fib3_aux n acc1 acc2 = match n with
  | 0 -> acc1
  | 1 -> acc2
  | _ -> fib3_aux (n-1) acc2 (acc1+acc2)
;;

(*fonction chapeau*)

let fib3 n = fib3_aux n 0 1 ;;
```

(d)

```
let rec fib4_aux n table =
  if (table.(n) = -1) then
     table.(n) <- (fib4_aux (n-1) table) + (fib4_aux (n-2) table);
  table.(n)
;;

let fib4 n=
   let table = Array.make (n+1) -1 in
      table.(0) <- 0;
      table.(1) <- 1;
      fib4_aux n table
;;
```

(e) La taille des données est n et les opérations fondamentales sont les additions. Notons $C(n)$ le nombre d'addiction. On a alors :
— Pour `fib1`, la complexité $C_1(n) = O(n)$ est linéaire.

— Pour `fib2`, la complexité $C_2(n) = O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ est exponentielle.
— Pour `fib3`, la complexité $C_3(n) = O(n)$ est linéaire.
— Pour `fib4`, la complexité $C_4(n) = O(n)$ est linéaire.

2. (a) Soit $p \in \mathbb{N}$. Montrons par récurrence sur $q \in \mathbb{N}^*$ la propriété

$$\mathcal{P}(q) : "F_{p+q} = F_{p+1}F_q + F_pF_{q-1}".$$

Ini. $F_{p+1} = 1 \times F_{p+1} + 0 \times F_p = F_1F_{p+1} + F_0F_p$ donc $\mathcal{P}(1)$ est vraie.
$F_{p+2} = 1 \times F_{p+1} + 1 \times F_p = F_2F_{p+1} + F_1F_p$ donc $\mathcal{P}(2)$ est vraie.
Héré. Soit $q \geqslant 2$. Supposons que $\mathcal{P}(q-1)$ et $\mathcal{P}(q)$ soient vraies.

$$\begin{aligned} F_{p+q+1} &= F_{p+q} + F_{p+q-1} \\ &= F_{p+1}F_q + F_pF_{q-1} + F_{p+1}F_{q-1} + F_pF_{q-2} \\ &= F_{p+1}(F_q + F_{q-1}) + F_p(F_{q-1} + F_{q-2}) \\ &= F_{p+1}F_{q+1} + F_pF_q. \end{aligned}$$

Donc $\mathcal{P}(q+1)$ est vraie.
Ccl. Par récurrence, pour tout $q \in \mathbb{N}^*$, $F_{p+q} = F_{p+1}F_q + F_pF_{q-1}$.

(b) En prenant $p = q = n$, on a :

$$F_{2n} = F_{n+1}F_n + F_nF_{n-1} = F_{n+1}F_n + F_n(F_{n+1} - F_n) = 2F_nF_{n+1} - F_n^2.$$

En prenant $p = n$ et $q = n+1$, on a :

$$F_{2n+1} = F_{n+1}^2 + F_n^2.$$

Enfin, en prenant $p = n$ et $q = n+2$, on a :

$$F_{2n+2} = F_{n+1}F_{n+2} + F_nF_{n+1} = F_{n+1}(F_{n+1} + F_n) + F_nF_{n+1} = F_{n+1}^2 + 2F_nF_{n+1}.$$

(c)

```
let rec fibo5 n = match n with
  | 0 -> 0 , 1
  | _ -> let a,b = fibo5 (n/2) in
    if n mod 2 = 0 then
      2*a*b - a*a , a*a + b*b
    else
      a*a + b*b , 2*a*b + b*b
;;
```

Si on souhaite une fonction donnant simplement la valeur de F_n, on peut créer une fonction chapeau comme ceci :

```
let fibo6 n = fst (fibo5 n) ;;
```

(d) Voici le théorème demandé (à connaitre par coeur !!) :

Soit $q \in \mathbb{N}^*$, $\gamma \in \mathbb{R}_+$ et l'équation de complexité :

$$C(n) = qC(n/2) + O(n^\gamma)$$

— Si $\log_2(q) = \gamma$, alors $C(n) = O(n^\gamma \log_2(n))$.
— Si $\log_2(q) > \gamma$, alors $C(n) = O(n^{\log_2(q)})$.
— Si $\log_2(q) < \gamma$, alors $C(n) = O(n^\gamma)$.

Ici, la taille des données est n et les opérations fondamentales sont les additions et les multiplications. Notons $C(n)$ le nombre d'additions et de multiplications. On a alors :

$$C(n) = C(n/2) + 7 = C(n/2) + O(1).$$

En appliquant le théorème maitre de complexité précédent, avec $q = 1$ et $\gamma = 0$, on obtient $C(n) = O(\log_2(n))$.

Exercice 2

1.
```
let rec appartenance x e = match e with
  | [] -> false
  | t::q -> (t=x) || (appartenance x q)
;;
```
2.
```
let rec test_ensemble l = match l with
  | [x] -> true
  | t::q -> (not (appartenance t q)) && (test_ensemble q)
;;
```
3.
```
let rec union e1 e2 = match e2 with
  | [] -> e1
  | t::q -> if (appartenance t e1) then union e1 q
                                   else t::(union e1 q)
;;
```
4.
```
let rec intersection e1 e2 = match e2 with
  | [] -> []
  | t::q -> if (appartenance t e1) then t::(intersection e1 q)
                                   else (intersection e1 q)
;;
```
5.
```
let rec difference e1 e2 = match e1 with
  | [] -> []
  | t::q -> if (appartenance t e2) then difference q e2
                                   else t::(difference q e2)
;;
```

Exercice 3

1.
```
let compteur1 n x =
  let r = ref n and a = ref 0 in
    while (!r >= x) do
      r := !r-x;
      a := !a+1
    done;
    !a
;;
```
Cet algorithme permet de faire la division euclidienne de n par x.

2.
```
let compteur2 n x p =
  let r = ref n and a = ref 0 in
    while (!r >= x) && (!a < p) do
      r := !r-x;
      a := !a+1
    done;
    !a
;;
```

3.
```
let compteur3 n x p =
  let r = ref n and a = ref 0 in
    while (!r >= x) && (!a < p) do
      r := !r-x;
      a := !a+1
    done;
    if (!r < x) then !a else -1
;;
```

4. (a) — Le problème a toujours une solution. En effet, si une somme de n cts nous est demandée, une solution triviale est de donner n pièces de 1 cts.

— Le problème a également toujours une solution en suivant l'algorithme glouton. Démontrons le par récurrence. Notons $(\mathcal{H}_n)$ la propriété "l'algorithme glouton admet une solution pour une somme de n cts".

• Il est clair que $(\mathcal{H}_n)$ est vraie pour $n = 0, 1, 2$.

• Soit $n \in \mathbb{N}^*$. Supposons que $(\mathcal{H}_0), \dots, (\mathcal{H}_{n-1})$ sont vraies et montrons que $(\mathcal{H}_n)$ est vraie.

Soit x la plus grande valeur de billets ou pièces telle que $x \leqslant n$ et posons $a = \left\lfloor \frac{n}{x} \right\rfloor$ (partie entière de la fraction $\frac{n}{x}$). Alors $a \leqslant \frac{n}{x} < a + 1$ donc $ax \leqslant n < ax + x$ et $0 \leqslant n - ax < x \leqslant n$.

Comme $n - ax < n$, on sait par hypothèse de récurrence qu'il existe une solution du problème en suivant l'algorithme glouton pour la somme $n - ax$ (sans billet ou pièce d'une valeur x car $n - ax < x$). On en déduit une solution du problème en suivant l'algorithme glouton pour la somme n en ajoutant a billets ou pièces d'une valeur x à la monnaie à rendre obtenue par la récurrence. Donc $(\mathcal{H}_n)$ est vraie.

• Par le principe de récurrence, $(\mathcal{H}_n)$ est vraie pour tout $n \in \mathbb{N}$.

(b)

```
let x=[|5000;2000;1000;500;200;100;50;20;10;5;2;1|];;

let gloutonillimite n x =
  let c = Array.make 12 0 and s = ref n and k = ref 0 in
    while (!s <> 0) do
      let aux = compteur1 !s x.(!k) in
        c.(!k) <- aux;
        s := !s - aux * x.(!k);
        k := !k+1
    done;
    c
;;
```

5. (a)

```
let sommemax x p =
  let s = ref 0 in
    for k=0 to 11 do
      s := !s+p.(k)*x.(k)
    done;
    !s
;;
```

Une somme inférieure à cette somme maximale ne peut pas toujours être constituée avec nos pièces et billets disponibles. Par exemple, si nous disposons d'un seul billet de 50 euros, on ne pourra pas constituer une somme de 20 euros.

(b)

```
let gloutonechec x n p =
  let c = Array.make 12 0 and s = ref n and k = ref 0
        and a = ref 0 in
    while (!s <> 0) && (!a <> -1) do
      a := compteur3 !s x.(!k) p.(!k) ;
      if (!a <> -1) then
        begin
          c.(!k) <- !a;
          s := !s - !a * x.(!k);
          k := !k+1
        end
    done;
    if (!a <> -1) then c
                  else failwith "Probleme de monnaie !"
;;
```

(c)

```
let gloutonreel x n p =
  let c = Array.make 12 0 and s = ref n and k = ref 0
        and a = ref 0 in
    while (!s <> 0) && (!k <= 11) do
      a := compteur2 !s x.(!k) p.(!k) ;
      c.(!k) <- !a;
      s := !s - !a * x.(!k);
      k := !k+1
    done;
    if (!s=0) then c else failwith "Probleme de monnaie !"
;;
```

6.

```
let controle x n c p=
  let test = ref true and s = ref 0 in
    for k=0 to 11 do
      test := (!test) && (c.(k) <= p.(k));
      s := !s + c.(k) * x.(k)
    done;
    !test && (!s=n)
;;
```

7. Avec l'ancien système monétaire du Royaume-Uni, l'algorithme glouton n'est pas toujours le plus efficace en termes de nombre de billets et pièces manipulés pour constituer une somme donnée.

   Par exemple, la somme 48 se décompose avec l'algorithme glouton sous la forme $48 = 1 \times 30 + 1 \times 12 + 1 \times 6$ donc avec 3 pièces. Mais $48 = 2 \times 24$ et on peut donc utiliser seulement 2 pièces.
