```
(* exercice 1 *)

let rec flatten = function
  | []        -> []
  | []::q     -> flatten q
  | (a::b)::q -> a::(flatten (b::q)) ;;


(* exercice 2 *)

let rec dernier prop = function
  | []               -> raise Not_found
  | t::q when prop t -> (try dernier prop q with Not_found -> t)
  | _::q             -> dernier prop q ;;


(* exercice 3 *)

let rec prefixe = function
  | [] -> []
  | t::q -> [t]::(map (function l -> t::l) (prefixe q)) ;;


(* exercice 4 *)

let au_moins_deux l =
  let rec aux acc = function
    | []                                      -> acc
    | t::q when mem t q && not (mem t acc) -> aux (t::acc) q
    | t::q                                    -> aux acc q
  in aux [] l ;;


(* exercice 5 *)

let rec scission = function
  | []      -> [], []
  | [a]     -> [], [a]
  | a::b::q -> let l1, l2 = scission q in a::l1, b::l2 ;;

let rec fusion = fun
  | [] l2                       -> l2
  | l1 []                       -> l1
  | (t1::q1) l2 when t1 < hd l2 -> t1::(fusion q1 l2)
  | l1 (t2::q2)                 -> t2::(fusion l1 q2) ;;

let rec merge_sort = function
  | []  -> []
  | [a] -> [a]
  | l   -> let (l1, l2) = scission l in
           fusion (merge_sort l1) (merge_sort l2) ;;


(* exercice 6 *)

let floyd1 a f =
  let rec aux acc = function
    | (x, y) when x = y -> acc
    | (x, y)            -> aux (acc + 1) (f x, f (f y))
  in aux 1 (f a, f (f a)) ;;
```

```
let floyd2 a f =
  let rec aux acc = function
    | (x, y) when x = y -> acc, x
    | (x, y)            -> aux (acc + 1) (f x, f (f y))
  in
  let p, x = aux 1 (f a, f (f a)) in
  let q, _ = aux (p+1) (f x, f (f x)) in q - p ;;

let periode n =
  let f = function x -> 10 * x mod n in
  floyd2 1 f ;;
```