```
let init_matrix n m f = init_vect n (fun i -> init_vect m (f i));;

init_matrix 3 4 (fun i j -> i,j);;

let a  = make_vect 2 [|0;0;0|] in a.(0).(0) <- 42; a;;
let a  = make_matrix 2 3 0 in a.(0).(0) <- 42; a;;

(* ********************************************************************** *)
let random_list longueur max =
  let result = ref [] in
  for i=1 to longueur do
    result:= random__int max :: !result
  done;
  !result;;

let sqrt_int n = int_of_float (ceil (sqrt (float_of_int n)));;

(*extremum min liste renvoie le minimum de liste. extremum max list renvoie le max*)
let extremum ordre = function
  | [] -> failwith "extremum"
  | a::q -> it_list ordre a q;;

let sqrt_plus_petits_naif liste =
  let liste = ref liste in
  let result = ref [] in
  for i=1 to sqrt_int (list_length !liste) do
    let elt = extremum min !liste in
    result := elt :: !result;
    liste := except elt !liste
  done;
  !result;;
(* O(n sqrt n) *)

let liste = random_list 12 10 in
liste,sqrt_plus_petits_naif liste;;

let sqrt_plus_petits_tri liste =
  let rec n_premiers n list = match n,list with
    |0,_ -> []
    |n,a::q -> a:: n_premiers (pred n) q
    |_,[] -> failwith "n_premiers" in
  n_premiers (sqrt_int (list_length liste)) (sort__sort (prefix <) liste)
;;
(* O(n log n) *)

let liste = random_list 12 10 in
liste,sqrt_plus_petits_tri liste;;

let echange vect i j =
  let temp = vect.(i) in
  vect.(i) <- vect.(j);
  vect.(j) <- temp;;

let a = [|0;1;2;3|] in echange a 0 2; a;;

let extremum_vect_indice ordre borne vect =
  let result = ref 0 in
  for i=1 to pred borne do
    if ordre vect.(i) vect.(!result)
    then result := i
  done;
  !result;;

extremum_vect_indice (prefix <) 5 [|4;2;6;4;9|];; (* 1 *)
extremum_vect_indice (prefix >) 5 [|4;2;6;4;9|];; (* 4 *)

let rec pad elt list n =
  if n=0 then list else pad elt (elt::list) (pred n);;

let matrix_of_list n liste =
  let result = make_matrix n n (hd liste) in
  let liste = ref liste in
  for i=0 to pred n do for j=0 to pred n do
    result.(i).(j) <- hd !liste;
    liste := tl !liste;
  done done;
  result;;

matrix_of_list 3 [1;2;3;4;5;6;7;8;9];;

let bubble matrix line =
  let indice_min =
    extremum_vect_indice (prefix <) (vect_length matrix.(line)) matrix.(line) in
  echange matrix.(line) 0 indice_min;;
```

```
let a = [|[|4;2;6;1;5;8|]|] in bubble a 0; a;;

let sqrt_plus_petits liste =
  let len = list_length liste in
  let sqrt = sqrt_int len in
  let plus_grand = extremum max liste in
  let liste = pad plus_grand liste (sqrt * sqrt -len) in
  let matrice = matrix_of_list sqrt liste in
  for i=0 to pred sqrt do
    bubble matrice i
  done;
  let result = ref [] in
  let japd = ref sqrt in
  let jette ligne =
    decr japd;
    echange matrice ligne !japd in
  for i=1 to sqrt do
    let p = extremum_vect_indice (fun v1 v2 -> v1.(0) < v2.(0)) !japd matrice in
    let g = extremum_vect_indice (fun v1 v2 -> v1.(0) > v2.(0)) !japd matrice in
    result := matrice.(p).(0) :: !result;
    matrice.(p).(0) <- matrice.(g).(0);
    bubble matrice p;
    jette g;
  done;
  !result;;
(* Chaque étape prend un temps O(sqrt n), il y a sqrt n étape, donc O(n) au total. *)

sqrt_plus_petits [9; 0; 2; 9; 9; 3; 4; 0; 5; 5; 7; 9];;

let liste = random_list 12 10 in
liste,sqrt_plus_petits liste;;

(* On trouve le (sqrt n)-ieme élément en temps linéaire,
   puis on filtre la liste (toujours en temps linéaire)
   pour ne garder que les éléments plus petits. *)
```