1)
```
let p = Array.make 21 (1,3);;
for i = 1 to 20 do
  let (x, y) = p.(i-1) in
  p.(i) <- x * 17 mod 23, y * 17 mod 19
done;;
```

2)
```
let alpha (a:point) (b:point) (c:point) =
  (fst b - fst a)*(snd c - snd a) -
    (fst c - fst a)*(snd b - snd a)
;;

print_newline ();
for i = 2 to 20 do
  for j = 1 to i-1 do
    for k = 0 to j-1 do
      if alpha p.(i) p.(j) p.(k) = 0
      then begin
          print_string "alignement ";
          print_int k;
          print_string " ";
          print_int j;
          print_string " ";
          print_int i;
          print_newline () end
    done
  done
done;;
```

3)
```
let plus_bas (p : point array) =
  let n = Array.length p in
  let a = ref p.(0) in
  for j = 1 to n-1 do
    if (snd !a > snd p.(j))
       || (snd !a = snd p.(j) && fst !a > fst p.(j))
    then a := p.(j)
  done;
  !a
;;

let plus_haut (p : point array) =
  let n = Array.length p in
  let a = ref p.(0) in
  for j = 1 to n-1 do
    if (snd !a < snd p.(j))
       || (snd !a = snd p.(j) && fst !a < fst p.(j))
    then a := p.(j)
  done;
```

```
    !a
  ;;
```

4)
```
let rec plus_a_droite b h (l : point list) = match l with
  | [] -> b, []
  | t::q ->
     let m, d = plus_a_droite b h q in
     let a = alpha b t h in
     if a <= 0 then m, d
     else match d with
          | [] -> t, [t]
          | _ -> if a > alpha b m h
                 then t, t::d
                 else m, t::d
;;
```

5) On ne met pas le point final pour éviter d'avoir des doublons.

```
let rec quickHull_drte b h (l : point list) =
  match plus_a_droite b h l with
  | _, [] -> [b]
  | m, d -> (quickHull_drte b m d)@(quickHull_drte m h d)
;;
```

6)
```
let quickHull (p : point array) =
  let b = plus_haut p in
  let h = plus_bas p in
  let l = Array.to_list p in
  (quickHull_drte b h l)@(quickHull_drte h b l)
```

7)
```
 let rec nb_inv_tete x l = match l with
    |[] -> 0
    |t :: q when x > t -> 1 + nb_inv_tete x q
    |t :: q -> nb_inv_tete x q;;
```

8)
```
let rec nb_inv liste =
  match liste with
  |[] -> 0
  |h :: tl -> nb_inv_tete h tl + nb_inv tl
;;
```

9) a) Ces deux nombres sont égaux.

b) Le nombre d'inversions dans `l` est la somme de ces trois nombres.

10)
```
let rec scinde_stable l i =
    if i = 0 then [],l
    else let t :: q = l in
         let q1,q2 = scinde_stable q (i-1) in
         t::q1,q2;;
```

11)
```
let  nb_inv_fus liste =
```

```
  let cpt = ref 0 in
  let rec fusion l1 l2 = match l1,l2 with
     |[],_ -> l2
     |_ ,[] -> l1
     |x1::q1,x2::q2 when x1<x2 -> x1::(fusion q1 l2)
     |_ ,x2::q2 ->  cpt := !cpt + List.length l1; x2::(fusion l1 q2)
  in
  let rec tri l = match l with
     |[] -> []
     |[x] -> [x]
     |_ -> let l1,l2 = scinde_stable l (List.length l/2) in
           fusion (tri l1) (tri l2)
  in ignore(tri liste); !cpt;;
```

12)
```
#load "graphics.cma";;
open Graphics;;
open_graph "";;
```

13)
```
let trace_point (a : point) =
  let h = min (size_x () / 23) (size_y () / 19) in
  fill_circle (h * fst a) (h * snd a) 3
;;

let ecrit (a : point) chaine =
  let h = min (size_x () / 23) (size_y () / 19) in
  moveto (h * fst a) (h * snd a);
  draw_string chaine
;;

(* Trace le nuage de points *)
clear_graph ();
for i = 0 to 20 do
  trace_point p.(i);
  moveto (fst p.(i)) (snd p.(i));
  ecrit p.(i) (string_of_int i)
done;;
```

14)
```
let h = min (size_x () / 23) (size_y () / 19) in
    let dilate m = (h * fst m, h * snd m) in
    let qh = quickHull p in
    draw_poly_line (Array.of_list
                      (List.map dilate (qh@[List.hd qh])));;
```