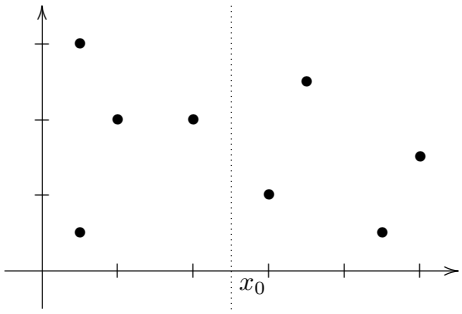
▶ **MPSI – Option Informatique**
Année 2001, Deuxième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Les deux points les plus proches

Lors de cette séance, nous allons nous intéresser au problème suivant : étant donné un ensemble de points du plan, indentifier le couple de points les plus proches au sens de la distance euclidienne. Ce type d'algorithme voit son utilité dans les transports aériens ou maritimes par exemple.
Nous représenterons un point en Caml par un couple de flottants (x, y) représentant ses coordonnées dans un repère orthonormé. L'ensemble des points considérés sera quant à lui stocké dans un tableau t. Nous supposerons toujours que tous les couples stockés dans t sont distincts.



1 Une première solution

Considérons deux points p et p' de coordonnées respectives (x, y) et (x', y'). Leur distance euclidienne est donnée par la formule $\|p-p'\| = \sqrt{(x-x')^2 + (y-y')^2}$.

▶ **Question 1** *Écrivez une fonction dist qui calcule la distance euclidienne séparant deux points.*

Une méthode « naïve » permettant de déterminer les deux points les plus proches dans un tableau t consiste à considérer successivement tous les couples de points possibles, ce qui peut se faire facilement à l'aide de deux boucles **for**.

▶ **Question 2** *Écrivez une fonction* plus_proches *qui prend pour argument un tableau de points t et retourne un couple de points $((x, y), (x', y'))$ situés à une distance minimale l'un de l'autre.*
Vous pourrez
- *Commencer par créer une référence d (initialisée par exemple avec* dist t.(0) t.(1)*) qui contiendra tout au long du déroulement de l'algorithme la plus petite distance trouvée ; ainsi que deux références s_i et s_j contenant les indices des points qui permettent d'atteindre cette distance.*
- *Utiliser deux boucles imbriquées pour considérer tous les couples de points communs.*
- *Mettre à jour les trois références à chaque fois qu'un couple (i, j) tel que $\|t.(i) - t.(j)\| <$!d est trouvé.*

▶ **Question 3** *Évaluez le coût d'un appel à la fonction* plus_proches *en fonction de la taille du tableau t.*

Nous allons maintenant nous intéresser à la mise au point d'un algorithme permettant de résoudre ce même problème en temps $O(n \log n)$. Cet algorithme nécessite de disposer de deux copies du tableau t, l'une triée par abscisses croissantes et l'autre par ordonnées croissantes. Nous allons donc écrire dans un premier temps quelques fonctions permettant de trier un tableau.

2 Tri d'un tableau

Soit $t = [|x_0; x_1; \ldots; x_{n-1}|]$ un tableau. Trier t consiste à permuter le contenu des cases du tableau t de façon à obtenir le tableau $[|x_{\sigma(0)}; x_{\sigma(1)}; \ldots; x_{\sigma(n-1)}|]$ avec $x_{\sigma(0)} \leq x_{\sigma(1)} \leq \ldots \leq x_{\sigma(n-1)}$.

L'algorithme de *tri par sélection* adopte le principe suivant : pour i variant de 0 à $n-2$, chercher l'indice $k \geq i$ de la case de la partie du tableau située à droite de la case i contenant le plus petit élément ; puis permuter le contenu des cases k et i.

Par exemple, pour trier le tableau $[|4;3;1;2|]$, on passe par les étapes suivantes :

$[|\mathbf{4};3;\mathbf{1};2|]$	$i=0$	$k=2$
$[|1;\mathbf{3};4;\mathbf{2}|]$	$i=1$	$k=3$
$[|1;2;\mathbf{4};\mathbf{3}|]$	$i=2$	$k=3$
$[|1;2;3;4|]$		

▶ **Question 4** *Écrivez une fonction* swap *qui prend pour argument un tableau t, deux entiers a et b et permute le contenu des cases d'indices a et b. Votre fonction ne créera pas un nouveau tableau, elle se contentera de modifier* sur place *le tableau t. Elle sera donc de type $'a\ vect \to int \to int \to unit$.*

▶ **Question 5** *Écrivez une fonction* sel *qui prend pour argument un tableau t ; deux entiers a et b et retourne l'indice de la case du sous-tableau $t.(a) \dots t.(b)$ qui contient le plus petit élément pour l'ordre $<$.*

▶ **Question 6** *Déduisez-en une fonction* tri *qui effectue le tri d'un tableau.*

La fonction que vous venez d'écrire vous permet de trier un tableau pour la relation d'ordre $<$. Sur les couples, cet ordre est en *Caml* l'ordre lexicographique. On peut cependant imaginer d'autres relations d'ordre sur des couples, comme par exemple :

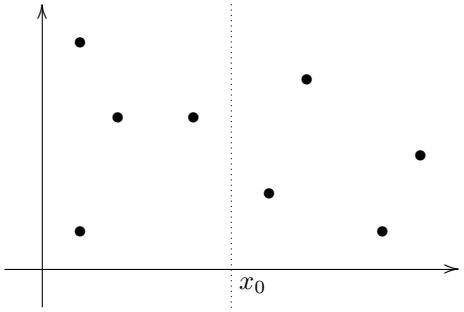
```
let ordre_x (x, y) (x', y') = x < x';;
let ordre_y (x, y) (x', y') = y < y';;
```

▶ **Question 7** *Adaptez les fonctions* sel *et* tri *de sorte qu'elle prennent également en argument une fonction implantant une relation d'ordre (comme* ordre_x *ou* ordre_y*).*

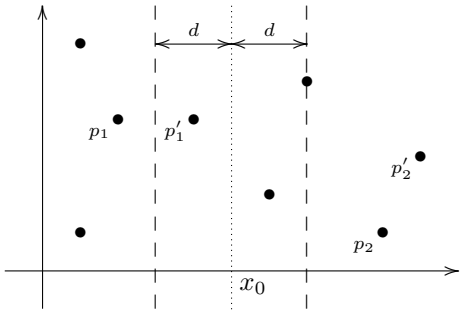
3 Diviser pour régner

Nous revenons maintenant à notre premier problème, déterminer parmi un ensemble de points un couple de points situés à une distance minimale. Nous supposons que nous disposons de deux copies de l'ensemble des n points :
– un tableau t_x, trié par abscisses croissantes,
– un tableau t_y, trié par ordonnées croissantes.

On pose $k = n/2$ et on considère les deux sous-tableaux $t_1 = [|t.(0); \dots ; t.(k)|]$ et $t_2 = [|t.(k+1); \dots ; t.(k+2)|]$. Soit x_0 un flottant plus grand que les abscisses des points de t_1 et plus petit que les abscisses des points de t_2. Le plan est divisé en deux parties :

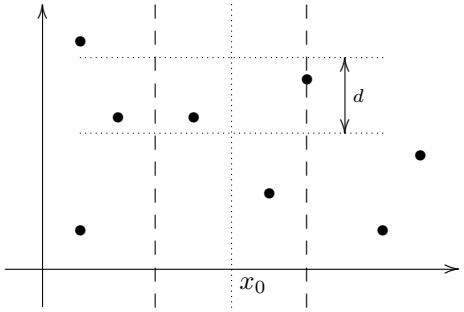


Notons $p_1 = (x_1, y_1)$ et $p'_1 = (x'_1, y'_1)$ les deux points les plus proches dans le tableau t_1 (avec $x_1 \leq x'_1$) ainsi que $p_2 = (x_2, y_2)$ et $p'_2 = (x'_2, y'_2)$ deux points les plus proches dans t_2. On note $d = \min(\|p_1 - p'_1\|, \|p_2 - p'_2\|)$. Si deux points du tableau t sont à une distance strictement inférieure à d alors nécessairement l'un fait partie de t_1 et l'autre de t_2 et ils sont dans la bande d'abscisses $[x_0 - d; x_0 + d]$.



▶ **Question 8** *Écrivez une fonction qui prend pour argument un tableau t_y ainsi que deux flottants x_{min}, x_{max} et qui sélectionne dans ce tableau les points dont l'abscisse est comprise entre x_{min} et x_{max}. Le résultat sera rendu dans un tableau t_z.*

Notons t_z le tableau des points situé dans la bande, obtenu grâce à un appel à la fonction précédente. Dans chaque tranche de hauteur d de cette bande, il ne peut y avoir plus de 7 points. On en déduit que si deux points de t_z sont à une distance inférieure ou égale à d alors ils sont situés à au plus sept cases l'un de l'autre dans le tableau t_z.



▶ **Question 9** *Écrivez une fonction* plus_proches_centre *qui prend pour argument le tableau t_z et retourne le couple de points les plus proches, tels que ces deux points soient situés à au plus 7 cases l'un de l'autre dans le tableau t_z. Vous pourrez vous inspirer de la fonction* plus_proches *précédente.*

On déduit de ce qui précède un algorithme récursif permettant de trouver les deux points les plus proches à partir des deux tableaux t_x et t_y :

1. Découper le tableau t_x en deux tableaux $t_{x,1}$ et $t_{x,2}$ de longueur moitié.
2. Déterminer $p_1 = (x_1, y_1)$, $p'_1 = (x'_1, y'_1)$, $p_2 = (x_2, y_2)$ et $p'_2 = (x'_2, y'_2)$ ainsi que la distance d.
3. Calculer t_z. S'il contient au moins deux points, appliquer la fonction plus_proches_centre à t_z pour obtenir $p = (x, y)$ et $p' = (x', y')$.

4. Retourner le résultat, c'est à dire la paire de points à distance minimale parmi p_1, p'_1 ; p_2, p'_2 et p, p'.

► **Question 10** *Écrivez en appliquant ce procédé une nouvelle fonction* plus_proches. *Évaluez sa complexité en fonction du nombre de points.*

▶ **MPSI – Option Informatique**
Année 2001, Deuxième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Les deux points les plus proches

Un corrigé

▶ **Question 1** On utilise la fonction sqrt de *Caml* qui permet de « calculer » la racine carrée d'un flottant et l'opérateur $**.$ d'exponentiation.

```
let dist (x, y) (x', y') =
  sqrt ((x -. x') **. 2.0 +. (y -. y') **. 2.0)
;;
```

▶ **Question 2** Notre fonction se contente d'envisager tous les couples d'entiers (i, j) avec $0 \leq i < j \leq n-1$ à l'aide de deux boucles **for** imbriquées. La référence d contient à chaque instant la plus petite distance trouvée et les références s_i et s_j les indices des deux points la réalisant.

```
let plus_proches t =
  let n = vect_length t
  and d = ref (dist t.(0) t.(1))
  and si = ref 0
  and sj = ref 1
  in

  for i = 0 to (n - 1) do
    for j = i + 1 to (n - 1) do
      if dist t.(i) t.(j) < !d then
      begin
        d := dist t.(i) t.(j);
        si := i;
        sj := j
      end
    done;
  done;
  (t.(!si), t.(!sj))
;;
```

▶ **Question 4**

```
let swap t a b =
  let x = t.(a) in
  t.(a) <- t.(b);
  t.(b) <- x;
  ()
;;
```

▶ **Question 5** La référence r contient à chaque itération l'indice de la case contenant le plus petit élément parmi $t.(a) \ldots t.(i)$.

```
let sel t a b =
  let r = ref a in
  for i = a + 1 to b do
    if t.(i) < t.(!r) then r := i
  done;
  !r
;;
```

▶ **Question 6**

```
let tri t =
  let n = vect_length t in
  for i = 0 to (n - 1) do
    let k = sel t i (n - 1) in
    swap t i k
  done;
  ()
;;
```

▶ **Question 7**

```
let sel comp t a b =
  let r = ref a in
  for i = a + 1 to b do
    if comp t.(i) t.(!r) then r := i
  done;
  !r
;;

let tri comp t =
  let n = vect_length t in
  for i = 0 to (n - 1) do
    let k = sel comp t i (n - 1) in
    swap t i k
  done;
  ()
;;
```

▶ **Question 8**

```
let selectionne ty x_min x_max =
  let n = vect_length ty in
  let resultat = ref [] in
  for i = n - 1 downto 0 do
    if x_min <= fst ty.(i)
       && fst ty.(i) <= x_max
    then resultat := ty.(i) :: (! resultat)
  done;
  vect_of_list (! resultat)
;;
```

▶ **Question 9** Notre fonction plus_proches_centre est très semblable à la fonction plus_proches. La seule différence réside dans la borne supérieure de la boucle indexée par j : il n'est plus nécessaire de poursuivre jusqu'à $j = n - 1$ mais on peut se limiter à $j = \min(n-1, i+7)$.

```
let plus_proches_centre t =
  let n = vect_length t
  and d = ref (dist t.(0) t.(1))
  and solution_i = ref 0
  and solution_j = ref 1
  in

  for i = 0 to (n - 1) do
    for j = i + 1 to min (n - 1) (i + 7) do
      if dist t.(i) t.(j) < !d then
      begin
        d := dist t.(i) t.(j);
        solution_i := i;
        solution_j := j
      end
    done;
  done;
  (t.(!solution_i), t.(!solution_j))
;;
```

▶ **Question 10**

```
let rec plus_proches' tx ty =
  let n = vect_length tx in
  if n <= 3 then (plus_proches tx)
  else
    begin
      let tx1 = sub_vect tx 0 (n/2)
      and tx2 = sub_vect tx (n/2) (n - n/2)
      and x0 = fst tx.(n/2 - 1) in
      let p1, p1' = plus_proches' tx1 ty
      and p2, p2' = plus_proches' tx2 ty in
      let p12, p12' =
        if (dist p1 p1') < (dist p2 p2')
        then (p1, p1')
        else (p2, p2')
      in
      let d = dist p12 p12' in

      let tz =
        selectionne
          ty
          (max (x0 -. d) (fst tx.(0)))
          (min (x0 +. d) (fst tx.(n - 1)))
      in
      if vect_length tz <= 1 then (p12, p12')
      else
        begin
          let p, p' = plus_proches_centre tz in
          if dist p p' < d then (p, p')
          else (p12, p12')
        end
    end
;;
```