CORRECTION DS 1 INFORMATIQUE

I Approche exhaustive

1 -
```
from math import *          # importation sans préfixe

def distance (i,j) :
    return sqrt((coords_x[j] - coords_x[i])**2
           + (coords_y[j] - coords_y[i])**2)
```

2 -
```
def plus_proche ( ) :
    n = len (coords_x)
    i0 = 0   # on cherche la distance minimale entre Mi et Mj
             avec i<j.
    j0 = 1
    dmin = distance (i0,j0)
    for i in range (n) :
        for j in range (i+1,n) :   # recherche du min
                                     avec i<j
            if distance (i,j) < dmin :
                i0 = i
                j0 = j
                dmin = distance (i0,j0)
    return (i0,j0)
```

II Quelques outils pour s'améliorer.

3 - a) En utilisant le principe du tri à bulles :

```
def tri_liste_abscisses (l) :
    L = copy.deepcopy (l)   # l est une liste de tuples.
    n = len (L)
    for i in range (n-1) :
        for j in range (0,n-i-1) :
            if  L[j][0] > L[j+1][0] :   # tri à bulles
                                          sur les abscisses.
                L[j], L[j+1] = L[j+1], L[j]
    return L
```

b)
```
def listeDict (l) :
    d = {}   # initialisation
    n = len (l)
    for i in range (n) :
        d[l[i]] = i       # création de la clé et de la valeur.
    return d.
```

c)
```
def tri_indice_abscisses (l) :
    n = len (l)
    d = listeDict (l)
    T = tri_liste_abscisses (l)
    L = [d[T[i]] for i in range (n)]   #conversion valeur -> indice
    return L.
```

d) La liste triée par ordonnées croissantes est :

[(1,1), (1,2), (3,2), (5,3), (2,4)]

Donc tri_indice_ordonnees (lex) renvoie :

$$\boxed{[3, 0, 4, 1, 2]}$$

4-
```
def sous_cluster (cl, x_min, x_max) :
    lx = []
    ly = []
    for i in cl[0] :  # sélection des abscisses
        if x_min <= coords_x[i] <= x_max :
            lx.append(i)
    for j in cl[1] :  # sélection des ordonnées
        if x_min <= coords_x[j] <= x_max :
            ly.append(j)
    return [lx, ly]
```

5-
```
def mediane (cl) :
    n = len (cl[0])
    i = cl[0][n//2]  # indice du point d'abscisse médiane
    return coords_x[i]
```


III Méthode sophistiquée

6-
```
def gauche(cl) :
    x_min = coords_x[c1[0][0]]  # abscisse du point le plus à gauche
    x_max = mediane (c1)   # abscisse de la médiane
    return sous_cluster (cl, x_min, x_max).
```

7- Pour l'étape 5, comme on doit avoir $d(M_1, M_2) < d_0$,

on a : $x_2 - x_1 < d_0$, de plus comme $M_1 \in C_1$, $x_1 \leq x_0$

et comme $M_2 \in C_1$, $x_2 \geq x_0$.

Donc $x_1 > x_2 - d_0 \geq x_0 - d_0$ et $x_2 < x_1 + d_0 \leq x_0 + d_0$.

Ainsi $\boxed{x_1, x_2 \in I_0 = [x_0 - d_0, x_0 + d_0]}$

8-
```
def bande_centrale (cl, d0) :
    x0 = mediane (cl)
    return sous_cluster (cl, x0-d0, x0+d0)
```

9-
```
def fusion (cl, d0) :
    n = len (cl[0])
    d = distance (cl[0][0], cl[0][1])  # recherche de la distance minimal
    for i in range (n) :
        for j in range (i+1, n) :
            if distance (cl[0][i], cl[0][j]) < d :
                d = distance (cl[0][i], cl[0][j])
```

```
    if d<=d0:    # renvoi de min(d, d0)
        return d
    else:
        return d0
```

10-
```
def distance_minimal(d):
    n = len(d[0])
    if n==2:    # étape i
        return distance (d[0][0], d[0][1])
    elif n==3:    # étape i
        a= distance (d[0][0], d[0][1])  # 3 distances à comparer
        b = distance ( d [0][0], d[0][2])
        c = distance ( d[0] [1], d[0][2])
        if a<=b and a<=c :
            return a
        elif b<=c and b<=a:
            return b
        else:
            return c
    else:
        dg = distance_minimale( gauche (d)) # étape iv
        dd = distance_minimale ( droite (d))
        d0= dg
        if dd<dg:
            d0= dd    # d0= min (dg, dd)
        c = bande_centrale ( d, d0)    # étape v
        return fusion ( c, d0)    # étape vi
```