```python
#!/usr/bin/env python3
import numpy as np
from timeit import timeit
from sys import argv
from geo.tycat import tycat
from geo.point import Point
from itertools import combinations
from geo.point import Point
from geo.segment import Segment


def load_instance(filename):
    # Entrée : le nom du fichier qu'on veut traiter en format
".pts"
    # Sortie : le seuil à partir duquel 2 points forment une
structure connexe et la liste des points

    with open(filename, "r") as instance_file:
        lines = iter(instance_file)
        distance = float(next(lines))
        points = [Point([float(f) for f in l.split(",")]) for l in
lines]

    return distance, points


def creation_quadrant(distance,points) :
    # Entrée : le seuil à partir duquel 2 points forment une
structure connexe et la liste des points
    # Sortie : quad_point et point_quad
    #
    # Cette fonction créer une liste de quadrant et renvoie deux
dictionnaires :
    #       1) quad_point qui associe à chaque quadrant la liste
de points qu'il contient
    #       2) point_quad qui associe à chaque point le quadrant
auquelle il appartient

    quad=[]
    quad_point={}
    quad_voisin={}
    point_quad={}
    numero_point=-1
    taille_quad= distance
    if int(1/taille_quad) < 1/taille_quad:
        nb_quadrant=int(1/taille_quad)+1
    else:
        nb_quadrant=int(1/taille_quad)
    for i in range(0,nb_quadrant):
        for j in range(0,nb_quadrant):
            p=j+nb_quadrant*i
```

```
            quad.append([(i*taille_quad,(i+1)*taille_quad),
(j*taille_quad,(j+1)*taille_quad)])
            quad_point[j+nb_quadrant*i]=[]
            quad_voisin[p]=[p,p-1,p+1,p-
nb_quadrant,p+nb_quadrant,p+nb_quadrant-1,p+nb_quadrant+1,p-
nb_quadrant+1,p-nb_quadrant-1]
    for point in points:
        numero_point+=1
        x= point.coordinates[0]
        y= point.coordinates[1]
        for k in range( len(quad) ):
            x_min,x_max=quad[k][0]
            y_min,y_max=quad[k][1]
            if x  < x_max and x >= x_min:
                if y  < y_max and y >= y_min:
                    quad_point[k].append(numero_point)
                    point_quad[numero_point]=k
                    break
    return quad_point,point_quad,nb_quadrant,quad_voisin



def
voisin(points,point_quad,quad_point,nb_quadrant,distance,quad_vois
in):
    # Entrée : liste des points, un dicotionnaire qui associe à
chaque quadrant la liste de points qu'il contient,
    #          un autre dictionnaire qui associe à chaque point le
quadrant auquelle il appartient, le seuil à partir duquel 2 points
    #          forment une structure connexe et la liste des
points
    # Sortie : liste des potentiels voisins
    #
    # Cette fonction parcourt chaque points et créer une liste
ens_quad_voisin qui regroupe
    # l'ensemble des potentiels voisins de ce points ( ceux qui se
trouve dans un quadrant adjacent )

    seg=[]
    L = [[] for _ in range(len(points))]
    for i in range (len(points)):
            ens_quad_voisin=[]
            k=point_quad[i]
            for h in quad_voisin[k]:
                try:
                    ens_quad_voisin+=quad_point[h]
                except Exception:
                    pass
            for elem in ens_quad_voisin:
                if distance >
points[i].distance_to(points[elem]) :
                    seg.append(Segment([points[i],points[elem]]))
```

```
                    L[i].append(points[i])
                    L[i].append(points[elem])
    return L


def union(l) :
    # Entrée : Liste des multiples structures connexes trouvées
avec des points partagés
    # Sortie : Création des connexes et suppression des doublons

        out = []
        while len(l)>0:
            first, *rest = l
            first = set(first)
            lf = -1
            while len(first)>lf:
                lf = len(first)

                rest2 = []
                for r in rest:
                    if len(first.intersection(set(r)))>0:
                        first |= set(r)
                    else:
                        rest2.append(r)
                rest = rest2

            out.append(first)
            l = rest
        return out

def print_components_sizes(distance, points):
    # Entrée : le seuil à partir duquel 2 points forment une
structure connexe et la liste des points
    # Sortie : void function
    #
    # Affichage des tailles triees de chaque composante

    pass

quad_point,point_quad,nb_quadrant,quad_voisin=creation_quadrant(di
stance,points)

L=voisin(points,point_quad,quad_point,nb_quadrant,distance,quad_vo
isin)
    LL = union(L)
    LL = [len(LL[i]) for i in range(len(LL))]
    LL.sort()
    LL.reverse()
    print(LL)
    #tycat(points,seg)
```

```
def main():
    """
    ne pas modifier: on charge une instance et on affiche les
tailles
    """
    for instance in argv[1:]:
        distance, points = load_instance(instance)
        print_components_sizes(distance, points)

main()
```