**PseudoCode**

struct **Way**:

ID1<-0

ID2<-0

Dis<-0

end struct

func MERGE(A,p,q,r):void

n1🡨q-p+1

n2🡨r-q

create **Way** vectors L[1..n1+1] and R[1..n1+1]

for i🡨0 to n1-1

do L[i]🡨A[p+i]

for j🡨0 to n2-1

do R[j]🡨A[q+j+1]

L[n1].dis🡨 ∞

R[n2].dis🡨 ∞

i🡨0

j🡨0

for k🡨p to r

do if L[i].dis<R[j].dis || (L[i].dis==R[j].dis&&L[i].ID1<R[j].ID1) ||

(L[i].dis==R[j].dis&&L[i].ID1==R[j].ID1&&L[i].ID2<=R[j].ID2)

then A[k]🡨L[i]

i🡨i+1

else A[k]🡨R[j]

j🡨j+1

end func

func MERGE\_SORT(A,p,r):void

if p<r

then q🡨⌊(p+r)/2⌋

MERGE\_SORT(A,p,q)

MERGE\_SORT(A,q+1,r)

MERGE(A,p,q,r)

end func

func main(argc, \*argv)

input(argv[1])

output(argv[2])

num 🡨input

for i🡨0 to num-1

id[i]🡨input

x[i]🡨input

y[i]🡨input

mode🡨input

disarr🡨a **Way** vector

total🡨num\*(num-1)/2

switch(mode)

case 1:

D🡨input

for i 🡨 0 to num-2

for j🡨 i+1 to num-1

Way temp

temp. dis🡨 sqrt((x[i]-x[j])^2+(y[i]-y[j])^2)

temp.ID1🡨min(id[i],id[j])

temp.ID2🡨max(id[i],id[j])

disarr.push\_back(temp)

MERGE\_SORT(disarr,0,disarr.size()-1)

while disarr[index]<D do

count🡨count+1

index🡨index+1

if(count>=total)

then break;

output🡨count

while(disarr[n]<D) do

output🡨disarr[n].ID1 disarr[n].ID2 setprecision(3)(disarr[n].dis)

n🡨n+1

if(count>=total)

then break;

case 2:

N🡨input

output🡨N

same way as case 1 to build the disarr vector

MERGE\_SORT(disarr,0,disarr.size()-1)

while(n<N) do

output🡨disarr[n].ID1 disarr[n].ID2 setprecision(3)(disarr[n].dis)

n🡨n+1

case 3:

output 🡨1

min🡨∞

for i🡨 0 to num-1

for j🡨 i+1 to num-2

dis🡨 sqrt((x[i]-x[j])^2+(y[i]-y[j])^2)

if min>dis

then min=dis

ID1🡨min(id[i],id[j])

ID2🡨max(id[i],id[j])

else if min==dis

if(ID1>min(id[i],id[j])){

ID1🡨min(id[i],id[j]);

ID2🡨max(id[i],id[j]);

output🡨ID1 ID2 setprecision(3)(min)

end switch

return 0

end func

**Experimental result and Analysis**

When there is 100 person : case 1 : 0.0759s, case 2 : 0.04701s, case 3 : 0.03564s

When there is 1000 person : case 1 : 3.082s, case 2 : 0.5906s, case 3 : 0.04833s

When there is 10000 person : case 1 : 311.6s, case 2 : 59.5s, case 3 : 0.6269s

As the computing of the distance of each individual person is n\*(n-1)/2 = (n^2-n)/2, it’s time complexity will be n^2; moreover, the recursion in MERGE\_SORT will be O(nlgn) per call, therefore, the total time complexity will be, which is (n^2)lg(n^2)=**O(n2lgn)**, and we can find a little clue through the experimental result(bigger case makes the feature more apparent). However, case 1 seemed to grow more in executing time then the other two cases, I think a big portion off the executing time might be based on the time printing out the ID and the distance to the .txt, as in my case, I have way more to print out in case 1 then the other two.