**Quoit Design**

Have you ever played quoit in a playground? Quoit is a game in which flat rings are pitched at some toys, with all the toys encircled awarded.   
In the field of Cyberground, the position of each toy is fixed, and the ring is carefully designed so it can only encircle one toy at a time. On the other hand, to make the game look more attractive, the ring is designed to have the largest radius. Given a configuration of the field, you are supposed to find the radius of such a ring.   
  
Assume that all the toys are points on a plane. A point is encircled by the ring if the distance between the point and the center of the ring is strictly less than the radius of the ring. If two toys are placed at the same point, the radius of the ring is considered to be 0.

Input

The input consists of several test cases. For each case, the first line contains an integer N (2 <= N <= 100,000), the total number of toys in the field. Then N lines follow, each contains a pair of (x, y) which are the coordinates of a toy. The input is terminated by N = 0.

Output

For each test case, print in one line the radius of the ring required by the Cyberground manager, accurate up to 2 decimal places.

Sample Input

2

0 0

1 1

2

1 1

1 1

3

-1.5 0

0 0

0 1.5

0

Sample Output

0.71

0.00

0.75

题意：有N个玩具，给出这些玩具的点的坐标。求出一个半径最大的圆环，使这个圆环至多能套进一个玩具。

思路：圆环至多能套进一个玩具，当且仅当圆环的直径小于等于任意两点的距离。这样就转化成了最近点对问题。

           对于最近点对问题，同样是用分治法进行处理。

           首先，对所有点按照x坐标从小到大进行排序，按照x坐标将点分成两半。

           同样，分治后有下面两种情况：

           1.p,q两点位于左半边或者位于右半边，这种情况可以递归处理。

           2.p,q两点一个位于左半边，一个位于右半边。这是我们重点讨论情况。

           对于第二种情况，虽然我们找到了一个我们待处理的点的性质：跨过中点线，但是这个好像没啥用。

           所以，关于最近点对的分治法，是和前面不同的。前面的几种分治中，第二种情况的处理是不依赖于第一种情况的。但是在最近点对的分治法中，第二种情况的处理要利用第一种情况得到的最小值来约束检查点的范围。

           设在第一种情况下，我们求得的最近距离为d。那对于第二种情况，我们要检查的点的范围x0- d < x < x0 + d ，因为在剩余范围的点的距离必然大于d,不需要再判断。

           同样，我们也可以约束y坐标。对于每个点，我们只检查它下面的的点，即范围 y <= y0 。同样利用距离d,我们可以将检查点的范围缩小到：y0- d <= y <= y0.

           可以证明，在x0- d < x < x0 + d 和y0- d <= y <= y0 内的点至多有5个。

           为了更高效的处理第二种情况，我们要把所有待考虑的点按照y坐标进行排序，因此我们递归处理的时候，按照y坐标进行归并排序。

这道题，类型：求最近点对。

求平面 最近点对的方法，就是分治法。

先将点分成两个区间，假设S1，S2，然后分别求S1内最近点对的点d1，S2内最近点对的点d2

再求S1与S2内最近点对 d=min（d1,d2）

但是，不能忘记，最近点对可能是 一个点在S1一个点在S2。

接下来就是比较精华的部分：

所求的点的位置，一定在于  mid-d,mid+d 之间。

然后，就在这个区间开始找点，并不断更新d值，最后就可以得到d了。

这道题，最终要求半径，所以还要除以2。

#include <stdio.h>

#include <math.h>

#include <algorithm>

#include<iostream>

using namespace std;

#define N 100001

struct Point

{

double x,y;

}p[N];

int arr[N];

double Min(double a,double b)

{

return a<b?a:b;

}

// 求两点之间的距离

double dis(Point a,Point b)

{

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

// 根据点横坐标or纵坐标排序

bool cmp\_y( int a,int b)

{

return p[a].y<p[b].y;

}

bool cmp\_x( Point a,Point b)

{

return a.x<b.x;

}

// 求最近点对

double close\_pair( int l,int r )

{

// 判断两个点和三个点的情况

if(r==l+1)

return dis( p[l],p[r] );

else

if( r==l+2 )

return Min( dis(p[l],p[r]),Min( dis(p[l],p[l+1]),dis(p[r-1],p[r]) ) );

int mid=(l+r)>>1;

double ans=Min(close\_pair(l,mid),close\_pair(mid+1,r));

int i,j,cnt=0;

// 如果当前p[i]点横坐标位于范围（中点横坐标-ans，中点横坐标+ans）位置内，

//则记录点的序号

for(i=l; i<=r; ++i)

if(p[i].x>=p[mid].x-ans && p[i].x<=p[mid].x+ans)

arr[cnt++]=i;

// 按照纵坐标由小到大对于arr数组内点进行排序

sort(arr,arr+cnt,cmp\_y);

for(i=0 ;i<cnt; i++)

for(j=i+1; j<cnt; j++)

{

if(p[arr[j]].y-p[arr[i]].y>=ans)

break;

ans=Min(ans,dis(p[arr[i]],p[arr[j]]));

}

return ans;

}

int main()

{

int i,n;

while( cin>>n && n)

{

for(i=0;i<n;++i)

cin>>p[i].x>>p[i].y;

// 先将所有点按照横坐标由小到大排序

sort(p,p+n,cmp\_x);

printf("%.2lf\n",close\_pair(0,n-1)/2.0); //因为是半径

}

return 0;

}