191 Intersection

You are to write a program that has to decide whether a given line segment intersects a given rectangle.

An example:

line:	start point:	(4,9)
	end point:	(11,2)
rectangle:	left-top:	(1,5)
	right-bottom:	(7,1)

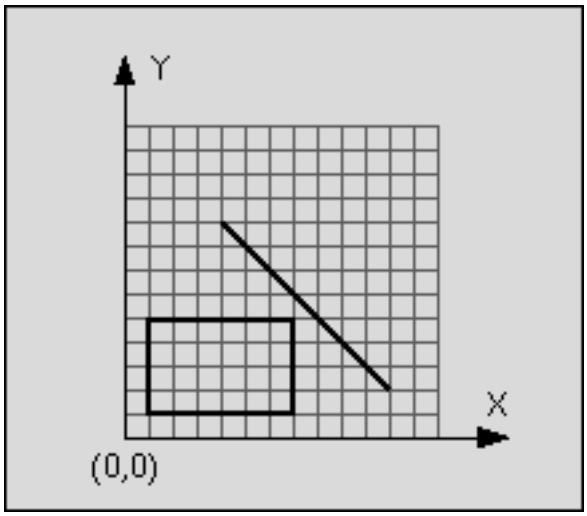


Figure 1: Line segment does not intersect rectangle

The line is said to intersect the rectangle if the line and the rectangle have at least one point in common. The rectangle consists of four straight lines and the area in between. Although all input values are integer numbers, valid intersection points do not have to lay on the integer grid.

Input

The input consists of n test cases. The first line of the input file contains the number n. Each following line contains one test case of the format:

$$xstart\ ystart\ xend\ yend\ xleft\ ytop\ xright\ ybottom$$

where $(xstart, ystart)$ is the start and $(xend, yend)$ the end point of the line and $(xleft, ytop)$ the top left and $(xright, ybottom)$ the bottom right corner of the rectangle. The eight numbers are separated by a blank. The terms *top left* and *bottom right* do not imply any ordering of coordinates.

Output

For each test case in the input file, the output file should contain a line consisting either of the letter "`T`" if the line segment intersects the rectangle or the letter "`F`" if the line segment does not intersect

the rectangle.

Sample Input

```
1
4 9 11 2 1 5 7 1
```

Sample Output

```
F
```

Problem D

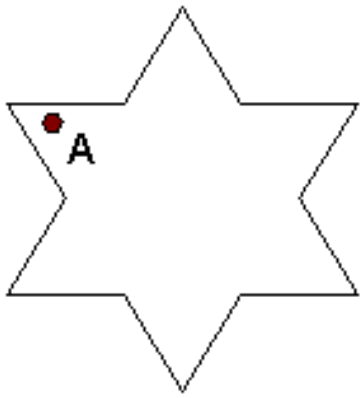
The Art Gallery

Input: standard input

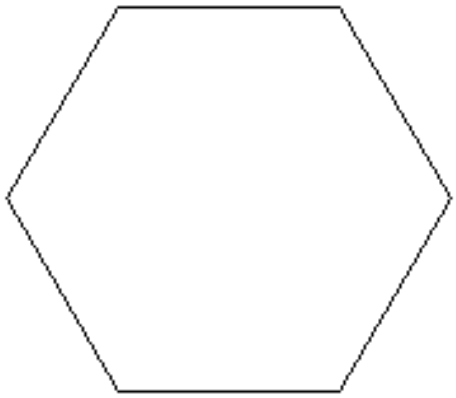
Output: standard output

Century Arts has hundreds of art galleries scattered all around the country and you are hired to write a program that determines whether any of the galleries has a *critical point*. The galleries are polygonal in shape and a *critical point* is a point inside that polygon from where the entire gallery is not visible.

For example, in gallery 1 (drawn below) point *A* is a critical point, but gallery 2 has no critical point at all.



Gallery 1



Gallery 2

The *Century Arts* authorities will provide you with the shape of a gallery as a sequence of (*x*, *y*) co-ordinates (determined using a suitable origin) of the consecutive corner points of that gallery.

Input

The input file consists of several data blocks. Each data block describes one gallery.

The first line of a data block contains an integer *N* (3 £ *N* £ 50) indicating the number of corner points of the gallery. Each of the next *N* lines contains two integers giving the (*x*, *y*) co-ordinates of a corner point where 0 £ *x*, *y* £ 1000. Starting from the first point given in the input the corner points occur in the same order on the boundary of the gallery as they appear in the input. No three consecutive points are co-linear.

The input file terminates with a value of 0 for *N*.

Output

For each gallery in the input output the word "*Yes*" if the gallery contains a critical point, otherwise output the word "*No*". Each output must be on a separate line.

Sample Input

```
4
0 0
3 0
3 3
0 3
4
0 0
3 0
1 1
0 3
0
```

Sample Output

```
No
Yes
```

Rezaul Alam Chowdhury

Problem F

A Hole to Catch a Man

Input : standard input

Output : standard output

How can a manhole be a hole if it is covered? Perhaps, to prove a manhole a hole, most of the manholes of Dhaka are uncovered. So now manhole means *a hole to catch a man*. Anyway, the new Mayor of Dhaka does not like this definition and he has recently been highly acclaimed by general people for ordering corresponding department to cover all the manholes of the city within a month.

Manhole Cover Manufacturing Corporation (*MCMC*) somehow managed to get the order. (Yes, this is a big deal, since a lot of manhole covers are to be made). MCMC makes the cover using steel, and they import polygonal steel sheets of different shapes and thickness from abroad. Then they melt the sheets to make the circular manhole covers, which also differ in size and thickness.

MCMC needs a program which, given dimensions of a number of steel sheets, will calculate how many manhole cover can be made from these sheets. You are to help them by writing the program.

Input

The input file consists of several data blocks.

Each data block starts with an integer *N*, the number of polygonal steel sheets. *i*'th line of the next *N* lines starts with thickness of the *i*'th sheet followed by co-ordinates of the polygons' corner points in some order (clockwise or anti-clockwise). Each line consists of a series of real numbers in following format:

$T_i\ X_0\ Y_0\ X_1\ Y_1\ X_2\ Y_2\ \ldots\ \ldots\ X_n\ Y_n\ X_0\ Y_0$

Where T_i is the thickness of the sheet, and $X_i\ Y_i$ are the coordinates of corner points. The line ends with co-ordinate of the first point. Last line of each data block will have two real numbers, *R* and *T*, radius and thickness of the manhole cover respectively.

Input file ends with a data block with $N = 0$.

Output

For each data block, print the number of manhole cover in separate line.

Sample Input:

```
2
2 0 0 0 10 5 15 12 10 10 0 0 0
5 0 0 5 100 100 0 0 0
5 3
1
2 0 0 10 0 10 10 0 10
5 2
0
```

Sample Output:

```
107
1
```

Rezaul Alam Chowdhury, Suman Kumar Nath, Tarique Mesbaul Islam.

Problem B
Rings and Glue

Input: standard input
Output: standard output
Time Limit: 1 second
Memory Limit: 32 MB

Little John is in big trouble. Playing with his different-sized (and colored!) rings and glue seemed such a good idea. However, the rings now lay on the floor, glued together with something that will definitely not come off with water. Surprisingly enough, it seems like no rings are actually glued to the floor, only to other rings. How about that!

You must help Little John to pick the rings off the floor before his mom comes home from work. Since the glue is dry by now, it seems like an easy enough task. This is not the case. Little John is an irrational kid of numbers, and so he has decided to pick up the largest component (most rings) of glued-together rings first. It is the number of rings in this largest component you are asked to find. Two rings are glued together if and only if they overlap at some point but no rings will ever overlap in only a single point. All rings are of the doughnut kind (with a hole in them). They can however, according to Little John, be considered "infinitely thin".

Input

Input consists of a number **(>0)** of problems. Each problem starts with the number of rings, **n**, where $\mathbf{0 \leq n < 100}$. After that, **n** rows follow, each containing a ring's physical attributes. That is, **3** floating point numbers, with an arbitrary number of spaces between them, describing the **x** coordinate and **y** coordinate for its center and its radius. All floating point numbers fit into the standard floating point type of your favorite programming language **(e.g., `float` for C/C++)**. Input ends with a single row with the integer **-1**.

Output

Output consists of as many grammatically correct answers as there were problems, each answer, on a separate line, being **'The largest component contains X ring(s).'** where **X** is the number of rings in the largest component.

Sample Input

```
4
0.0 0.0 1.0
-1.5 -1.5 0.5
1.5 1.5 0.5
-2.0 2.0 3.5
3
3.0 2.0 2.0
0.0 -0.5 1.0
0.0 0.0 2.0
5
-2.0 0.0 1.0
1.0 -1.0 1.0
```

```
0.0 1.0 0.5
2.0 0.0 1.0
-1.0 1.0 1.0
-1
```

Sample Output

```
The largest component contains 4 rings.
The largest component contains 2 rings.
The largest component contains 3 rings.
```

(Joint Effort Contest, Problem Source: Swedish National Programming Contest, arranged by department of Computer Science at Lund Institute of Technology.)

The Fortified Forest

Once upon a time, in a faraway land, there lived a king. This king owned a small collection of rare and valuable trees, which had been gathered by his ancestors on their travels. To protect his trees from thieves, the king ordered that a high fence be built around them. His wizard was put in charge of the operation.

Alas, the wizard quickly noticed that the only suitable material available to build the fence was the wood from the trees themselves. In other words, it was necessary to cut down some trees in order to build a fence around the remaining trees. Of course, to prevent his head from being chopped off, the wizard wanted to minimize the value of the trees that had to be cut. The wizard went to his tower and stayed there until he had found the best possible solution to the problem. The fence was then built and everyone lived happily ever after.

You are to write a program that solves the problem the wizard faced.

Input

The input contains several test cases, each of which describes a hypothetical forest. Each test case begins with a line containing a single integer n, $2 \leq n \leq 15$, the number of trees in the forest. The trees are identified by consecutive integers 1 to n. Each of the subsequent lines contains 4 integers x_i, y_i, v_i, l_i that describe a single tree. (x_i, y_i) is the position of the tree in the plane, v_i is its value, and l_i is the length of fence that can be built using the wood of the tree. v_i and l_i are between 0 and 10,000.

The input ends with an empty test case ($n = 0$).

Output

For each test case, compute a subset of the trees such that, using the wood from that subset, the remaining trees can be enclosed in a single fence. Find the subset with a minimum value. If more than one such minimum-value subset exists, choose one with the smallest number of trees. For simplicity, regard the trees as having zero diameter.

Display, as shown below, the test case numbers (1, 2, ...), the identity of each tree to be cut, and the length of the excess fencing (accurate to two fractional digits).

Display a blank line between test cases.

Sample Input

```
6
 0  0  8  3
 1  4  3  2
 2  1  7  1
 4  1  2  3
 3  5  4  6
 2  3  9  8
3
 3  0 10  2
 5  5 20 25
 7 -3 30 32
0
```

Sample Output

```
Forest 1
Cut these trees: 2 4 5
Extra wood: 3.16

Forest 2
Cut these trees: 2
Extra wood: 15.00
```

Miguel Revilla 2002-06-25

109 SCUD Busters

Background

Some problems are difficult to solve but have a simplification that is easy to solve. Rather than deal with the difficulties of constructing a model of the Earth (a somewhat oblate spheroid), consider a pre-Columbian flat world that is a 500 kilometer $\times$ 500 kilometer square.

In the model used in this problem, the flat world consists of several warring kingdoms. Though warlike, the people of the world are strict isolationists; each kingdom is surrounded by a high (but thin) wall designed to both protect the kingdom and to isolate it. To avoid fights for power, each kingdom has its own electric power plant.

When the urge to fight becomes too great, the people of a kingdom often launch missiles at other kingdoms. Each SCUD missile (Sanitary Cleansing Universal Destroyer) that lands within the walls of a kingdom destroys that kingdom's power plant (without loss of life).

The Problem

Given coordinate locations of several kingdoms (by specifying the locations of houses and the location of the power plant in a kingdom) and missile landings you are to write a program that determines the total area of all kingdoms that are without power after an exchange of missile fire.

In the simple world of this problem kingdoms do not overlap. Furthermore, the walls surrounding each kingdom are considered to be of zero thickness. The wall surrounding a kingdom is the minimal-perimeter wall that completely surrounds all the houses and the power station that comprise a kingdom; the area of a kingdom is the area enclosed by the minimal-perimeter thin wall.

There is exactly one power station per kingdom.

There may be empty space between kingdoms.

The Input

The input is a sequence of kingdom specifications followed by a sequence of missile landing locations.

A kingdom is specified by a number N ($3 \leq N \leq 100$) on a single line which indicates the number of sites in this kingdom. The next line contains the x and y coordinates of the power station, followed by $N-1$ lines of x, y pairs indicating the locations of homes served by this power station. A value of -1 for N indicates that there are no more kingdoms. There will be at least one kingdom in the data set.

Following the last kingdom specification will be the coordinates of one or more missile attacks, indicating the location of a missile landing. Each missile location is on a line by itself. You are to process missile attacks until you reach the end of the file.

Locations are specified in kilometers using coordinates on a 500 km by 500 km grid. All coordinates will be integers between 0 and 500 inclusive. Coordinates are specified as a pair of integers separated by white-space on a single line. The input file will consist of up to 20 kingdoms, followed by any number of missile attacks.

The Output

The output consists of a single number representing the total area of all kingdoms without electricity after all missile attacks have been processed. The number should be printed with (and correct to) two decimal places.

Sample Input

```
12
3 3
4 6
4 11
4 8
10 6
5 7
6 6
6 3
7 9
10 4
10 9
1 7
5
20 20
20 40
40 20
40 40
30 30
3
10 10
21 10
21 13
-1
5 5
20 12
```

Sample Output

```
70.50
```

A Hint

You may or may not find the following formula useful.

Given a polygon described by the vertices $v_0, v_1, \ldots, v_n$ such that $v_0 = v_n$, the signed area of the polygon is given by

$$a = \frac{1}{2} \sum_{i=1}^{n} (x_{i-1} y_i) - (x_i y_{i-1})$$

where the x, y coordinates of $v_i = (x_i, y_i)$; the edges of the polygon are from v_i to v_{i+1} for $i = 0 \ldots n-1$.

If the points describing the polygon are given in a counterclockwise direction, the value of a will be positive, and if the points of the polygon are listed in a clockwise direction, the value of a will be negative.