The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem A

Balloons in a Box

Input: balloon.in

You must write a program that simulates placing spherical balloons into a rectangular box.

The simulation scenario is as follows. Imagine that you are given a rectangular box and a set of points. Each point represents a position where you might place a balloon. To place a balloon at a point, center it at the point and inflate the balloon until it touches a side of the box or a previously placed balloon. You may not use a point that is outside the box or inside a previously placed balloon. However, you may use the points in any order you like, and you need not use every point. Your objective is to place balloons in the box in an order that maximizes the total volume occupied by the balloons.

You are required to calculate the volume within the box that is not enclosed by the balloons.

Input

The input consists of several test cases. The first line of each test case contains a single integer n that indicates the number of points in the set ($1 \le n \le 6$). The second line contains three integers that represent the (x, y, z) integer coordinates of a corner of the box, and the third line contains the (x, y, z) integer coordinates of the opposite corner of the box. The next n lines of the test case contain three integers each, representing the (x, y, z) coordinates of the points in the set. The box has non-zero length in each dimension and its sides are parallel to the coordinate axes.

The input is terminated by the number zero on a line by itself.

Output

For each test case print one line of output consisting of the test case number followed by the volume of the box not occupied by balloons. Round the volume to the nearest integer. Follow the format in the sample output given below.

Place a blank line after the output of each test case.

Sample Input	Output for the Sample Input
2 0 0 0 10 10 10 3 3 3 7 7 7 0	Box 1: 774

This page is intentionally blank.

The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem B

Undecodable Codes

Input: codes.in

Phil Oracle has a unique ability that makes him indispensable at the National Spying Agency. His colleagues can bring him any new binary code and he can tell them immediately whether the code is uniquely decodable or not. A *code* is the assignment of a unique sequence of characters (a *codeword*) to each character in an *alphabet*. A binary code is one in which the codewords contain only zeroes and ones. For example, here are two possible binary codes for the alphabet {**a**,**c**,**j**,**l**,**p**,**s**,**v**}.

	Code 1	Code 2
a	**1**	**010**
c	**01**	**01**
j	**001**	**001**
l	**0001**	**10**
p	**00001**	**0**
s	**000001**	**1**
v	**0000001**	**101**

The *encoding* of a string of characters from an alphabet (the *cleartext*) is the concatenation of the codewords corresponding to the characters of the cleartext, in order, from left to right. A code is *uniquely decodable* if the encoding of every possible cleartext using that code is unique. In the example above, Code 1 is uniquely decodable, but Code 2 is not. For example, the encodings of the cleartexts "**pascal**" and "**java**" are both **001010101010.** Even shorter encodings that are not uniquely decodable are **01** and **10**.

While the agency is very proud of Phil, he unfortunately gives only "yes" or "no" answers. Some members of the agency would prefer more tangible proof, especially in the case of codes that are not uniquely decodable. For this problem you will deal only with codes that are *not* uniquely decodable. For each of these codes you must determine the single encoding having the minimum length (measured in bits) that is ambiguous because it can result from encoding each of two or more different cleartexts. In the case of a tie, choose the encoding which comes first lexicographically.

Input

One or more codes are to be tested. The input for each code begins with an integer m, $1 \le m \le 20$, on a line by itself, where m is the number of binary codewords in the code. This is followed by m lines each containing one binary codeword string, with optional leading and trailing whitespace. No codeword will contain more than 20 bits.

The input is terminated by the number zero on a line by itself.

Output

For each code, display the sequential code number (starting with 1), the length of the shortest encoding that is not uniquely decodable, and the shortest encoding itself, with ties broken as previously described. The encoding must be displayed with 20 bits on each line except the last, which may contain fewer than 20 bits. Place a blank line after the output for each code. Use the format shown in the samples below.

Sample Input	Output for the Sample Input

3
  0
  01
  10
5
 0110
  00
  111
  001100
  110
5
  1
  001
  0001
  00000000000000000001
  10000000000000000000
0
	Code 1: 3 bits
	010
	
	Code 2: 9 bits
	001100110
	
	Code 3: 21 bits
	10000000000000000000
	1

The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem C

Crossing the Desert

Input: desert.in

In this problem, you will compute how much food you need to purchase for a trip across the desert on foot.

At your starting location, you can purchase food at the general store and you can collect an unlimited amount of free water. The desert may contain oases at various locations. At each oasis, you can collect as much water as you like and you can store food for later use, but you cannot purchase any additional food. You can also store food for later use at the starting location. You will be given the coordinates of the starting location, all the oases, and your destination in a two-dimensional coordinate system where the unit distance is one mile.

For each mile that you walk, you must consume one unit of food and one unit of water. Assume that these supplies are consumed continuously, so if you walk for a partial mile you will consume partial units of food and water. You are not able to walk at all unless you have supplies of both food and water. You must consume the supplies while you are walking, not while you are resting at an oasis. Of course, there is a limit to the total amount of food and water that you can carry. This limit is expressed as a carrying capacity in total units. At no time can the sum of the food units and the water units that you are carrying exceed this capacity.

You must decide how much food you need to purchase at the starting location in order to make it to the destination. You need not have any food or water left when you arrive at the destination. Since the general store sells food only in whole units and has only one million food units available, the amount of food you should buy will be an integer greater than zero and less than or equal to one million.

Input

The first line of input in each trial data set contains n ($2 \leq n \leq 20$), which is the total number of significant locations in the desert, followed by an integer that is your total carrying capacity in units of food and water. The next n lines contain pairs of integers that represent the coordinates of the n significant locations. The first significant location is the starting point, where your food supply must be purchased; the last significant location is the destination; and the intervening significant locations (if any) are oases. You need not visit any oasis unless you find it helpful in reaching your destination, and you need not visit the oases in any particular order.

The input is terminated by a pair of zeroes.

Output

For each trial, print the trial number followed by an integer that represents the number of units of food needed for your journey. Use the format shown in the example. If you cannot make it to the destination under the given conditions, print the trial number followed by the word “Impossible.”

Place a blank line after the output of each test case.

Sample Input	Output for the Sample Input
4 100 10 -20 -10 5 30 15 15 35 2 100 0 0 100 100 0 0	Trial 1: 136 units of food Trial 2: Impossible

Problem D
Ferries
Input: ferries.in

Millions of years ago massive fields of ice carved deep grooves in the mountains of Norway. The sea filled these grooves with water. The Norwegian people call them fjords. This landscape of mountains and water is beautiful, but it makes traveling difficult. The usual scheme is: drive some kilometers, wait for a ferry, cross a fjord with the ferry, drive some more kilometers, and so on until the destination has been reached. To reach a destination as early as possible, most people have the following strategy: drive as fast as allowed (the maximum speed is 80 km/h) to the next ferry, and wait until it goes. Repeat until the destination has been reached.

Since driving fast requires more fuel than driving slow, this strategy is both expensive and harmful to the environment. The new generation of cruise control systems is designed to help. Given the route you want to go, these systems will gather information about the ferries involved, calculate the earliest possible time of arrival at the final destination, and calculate a driving scheme that avoids driving faster than needed. The systems will calculate your road speed so that you board the next ferry the moment it leaves.

Given a route (a sequence of road-pieces and crossings with ferries), you must write a program to calculate the minimal time it takes to complete this route. Moreover, your program must find a driving scheme such that the maximal driving speed at any point during the trip is as small as possible.

Input

The input file contains one or more test cases. Each test case describes a route. A route consists of several sections, each section being either a piece of road or a crossing. The first line in the description contains a single number s ($s > 0$), which is the number of sections in the route. The next s lines contain the descriptions of the sections. Every line describing a section starts with two names: the place of departure and the place of arrival, followed by either the word ‘road’ or the word ‘ferry’ indicating what kind of section it is. If the section is a road, its length (a positive integer) is given in km. For example:

```
Dryna Solholmen road 32
```

Lines describing ferry sections have more information. Following the word “ferry”, the duration of the ferry crossing, in minutes (a positive integer) is given. This is followed by the frequency f ($f > 0$) of the ferry, that is, the number of times the ferry departs in a single hour. The next f integers give the departure times of the ferry, in ascending order. For example:

```
Manhiller Fodnes ferry 20 2 15 35
```

The ferry travels from Manhiller to Fodnes in 20 minutes, and it leaves twice an hour (on 0h15, 0h35, 1h15, 1h35,…). The beginning of the entire trip always starts at a full hour. The sections in a route are consecutive, that is, if a section goes from A to B then the next section starts at B. Every route in the input can be traveled in no more than 10 hours.

The input is terminated by the number zero on a line by itself.

Output

Output for each test case is a single line containing three items. The first item is the test case number. The second is the total travel time for an optimal scheme in the form *hh:mm:ss*. The third item is the maximal road speed in an optimal scheme rounded to two digits to the right of the decimal point.

Place a blank line after the output of each test case.

Sample Input	Output for the Sample Input
1 Bygd Bomvei road 7 2 Ferje Overfarten ferry 20 2 5 25 Overfarten Havneby ferry 30 3 10 30 50 5 Begynnelse Brygge road 30 Brygge Bestemmelse ferry 15 4 10 25 40 55 Bestemmelse Veiskillet road 20 Veiskillet Grusvei road 25 Grusvei Slutt ferry 50 1 10 0	Test Case 1: 00:05:15 80.00 Test Case 2: 01:00:00 0.00 Test Case 3: 03:00:00 45.00

The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

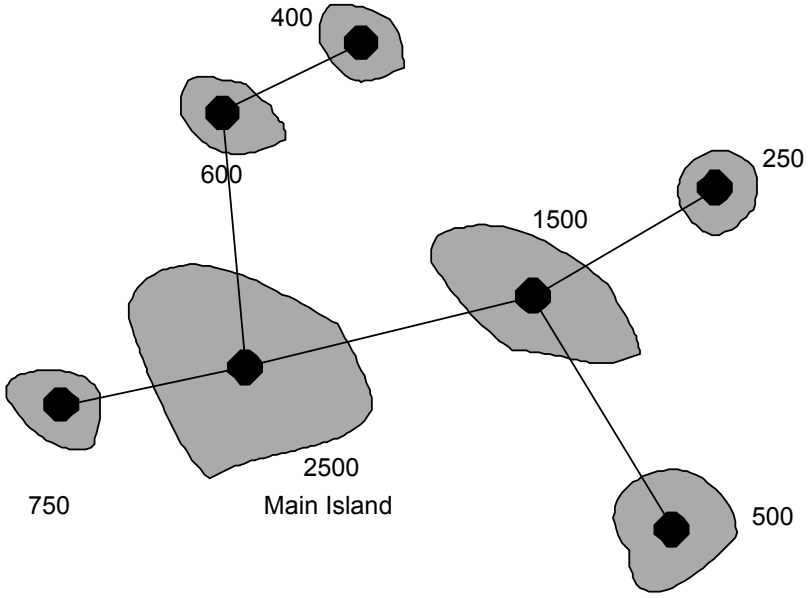
sponsored by IBM

Problem E

Island Hopping

Input: islands.in

The company Pacific Island Net (PIN) has identified several small island groups in the Pacific that do not have a fast internet connection. PIN plans to tap this potential market by offering internet service to the island inhabitants. Each groups of islands already has a deep-sea cable that connects the main island to the closest internet hub on the mainland (be it America, Australia or Asia). All that remains to be done is to connect the islands in a group to each other. You must write a program to help them determine a connection procedure.



For each island, you are given the position of its router and the number of island inhabitants. In the figure, the dark dots are the routers and the numbers are the numbers of inhabitants. PIN will build connections between pairs of routers such that every router has a path to the main island. PIN has decided to build the network such that the total amount of cable used is minimal. Under this restriction, there may be several optimal networks. However, it does not matter to PIN which of the optimal networks is built.

PIN is interested in the average time required for new customers to access the internet, based on the assumption that construction on all cable links in the network begins at the same time. Cable links can be constructed at a rate of one kilometer of cable per day. As a result, shorter cable links are completed before the longer links. An island will have internet access as soon as there is a path from the island to the main island along completed cable links. If m_i is the number of inhabitants of the i^{th} island and t_i is the time when the island is connected to the internet, then the average connection time is:

$$\frac{\sum t_i * m_i}{\sum m_i}$$

Input

The input consists of several descriptions of groups of islands. The first line of each description contains a single positive integer n, the number of islands in the group ($n \leq 50$). Each of the next n lines has three integers x_i, y_i, m_i, giving the position of the router (x_i, y_i) and number of inhabitants m_i ($m_i > 0$) of the islands. Coordinates are measured in kilometers. The first island in this sequence is the main island.

The input is terminated by the number zero on a line by itself.

Output

For each group of islands in the input, output the sequence number of the group and the average number of days until the inhabitants are connected to the internet. The number of days should have two digits to the right of the decimal point. Use the output format in the sample given below.

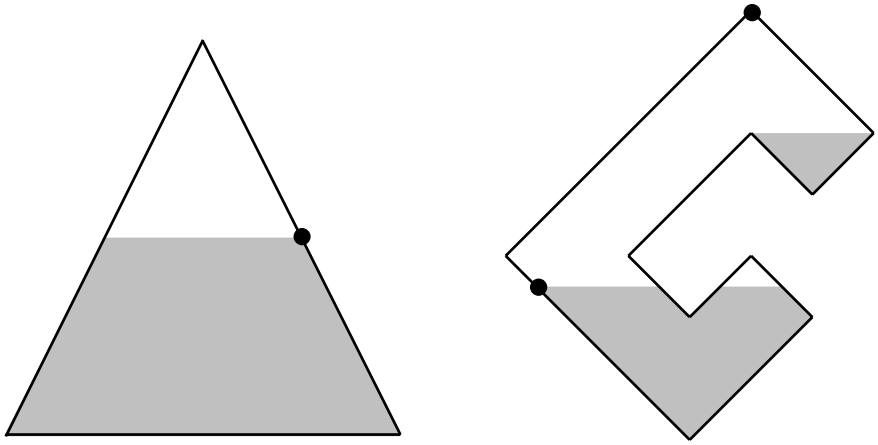
Place a blank line after the output of each test case.

Sample Input	Output for the Sample Input
7 11 12 2500 14 17 1500 9 9 750 7 15 600 19 16 500 8 18 400 15 21 250 0	Island Group: 1 Average 3.20

Problem F
Toil for Oil
Input: oil.in

Prospecting for new sources of oil has become a high-technology industry. With improved drilling technology it has become economically viable to seek out ever smaller and harder to reach deposits of oil. However, using exploratory drilling to locate these deposits is not cost-efficient, so researchers have developed methods to detect oil indirectly.

One such method to detect oil is sonar, which uses reflected sound waves to locate caves in underground rock formations. Determining how much oil can be contained in such a cave is a difficult problem.



In this problem, you will be given some cross-sections of underground caves, represented by polygons such as the ones shown in the figure. Some of the points bounding the polygon may be holes through which oil can seep out into the surrounding rock (represented by black circles in the figure). Given the polygonal shape of the cave and the positions of the holes, you must compute the maximum amount of oil that could be in the cave (shown as gray shaded areas in the figure). This amount is limited by the fact that, in any connected body of oil, the oil level can never be above a hole, since it would drain into the surrounding rock instead.

Input

The input contains several cave descriptions, each in the form of a polygon that specifies a cross-section of a cave. The first line of each description contains a single integer n, representing the number of points on the polygon ($3 \le n \le 100$).

Each of the following n lines contains three integers x_i, y_i, h_i. The values (x_i, y_i) give the positions of the points on the boundary of the polygon in counterclockwise order. The polygon is simple—that is, it does not cross or touch itself. The value of h_i is equal to 1 if the point is a hole through which oil can seep out, and 0 otherwise. The "upward" direction in each case is the positive y-axis.

The input is terminated by a zero on a line by itself.

Output

For each cave description, print its sequence number (starting with 1) followed by its oil capacity. Approximate the oil capacity by the area within the given cross-section that may contain oil, rounded to the nearest integer. Use the format in the example output given below.

Place a blank line after each test case.

Sample Input	Output for the Sample Input
4 10 0 0 5 10 1 0 20 0 -10 0 0 11 0 6 0 1 5 1 6 0 0 10 4 0 8 6 0 6 4 0 4 6 0 8 10 0 10 8 0 12 10 0 8 14 1 0	Cave 1: Oil capacity = 150 Cave 2: Oil capacity = 27

Problem G

Partitions

Input: partitions.in

A *partition* of a rectangle is a subdivision of the rectangle into a set of smaller, non-overlapping sub-rectangles. Figure 1 shows several examples of partitions.

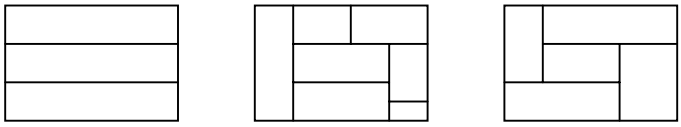


Figure 1

Figure 2 shows three equal sized rectangles, partitioned into sub-rectangles. Partition B is obtained from partition A by partitioning two of the sub-rectangles of A. Generally, if a partition B is obtained from A by partitioning one or more of its sub-rectangles, we say that B is *finer* than A, or that A is *coarser* than B. This relation is partial: partition C is neither coarser nor finer than A or B.

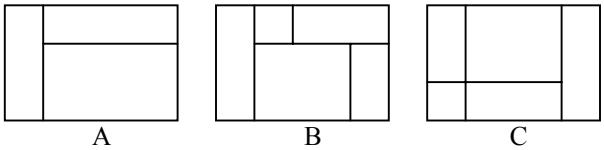


Figure 2

Given two partitions D and E of the same rectangle, infinitely many partitions exist that are finer than both D and E. In Figure 3 both F and G are finer than D and E. Among the partitions that are finer than both D and E, a unique one exists that is *coarsest.* This partition is called the *infimum* of D and E. In Figure 3, partition F is the infimum of D and E.

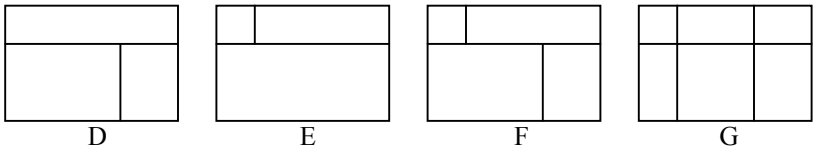


Figure 3

In Figure 4, both H and J are coarser than D and E. Here J is the finest partition that is coarser than D and E. Then J is the *supremum* of D and E.

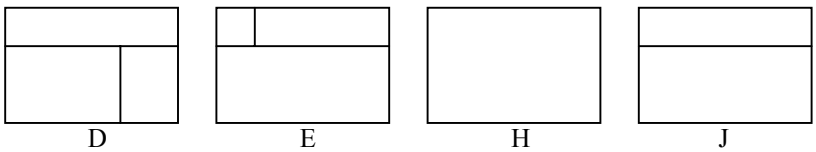


Figure 4

Write a program that, given two partitions of the same rectangle, finds the infimum and the supremum of these partitions.

Input

The input file contains one or more test cases. The first line of each test case gives the width w and height h of the rectangle ($0 < w, h \leq 20$). In the next h+1 lines the two partitions are given, as in the sample. Each of these lines contains 4*w+3 characters. The first 2*w+1 of these belong to the first partition; the last 2*w+1 of these belong to the second partition. A space separates the two partitions. Horizontal lines are created using underscores '_', vertical lines using '|'.

The input is terminated by a pair of zeroes.

Output

For every case in the input file the output contains a single line containing the case number (in the format shown in the sample), followed by the infimum and the supremum of the two partitions, using the same format as the input.

Place a blank line after the output of each test case

Sample Input

```
4 3
 _ _ _ _   _ _ _ _
|_ _ _ _| |_|_ _ _|
|   |   | |       |
|_ _|_ _| |_ _ _ _|
3 4
 _ _ _   _ _ _
| |   | | |   |
| |   | |_|_ _|
|_|_ _| |   | |
|_ _|_| |_ _|_|
0 0
```

Output for the Sample Input

```
Case 1:
 _ _ _ _   _ _ _ _
|_|_ _ _| |_ _ _ _|
|   |   | |       |
|_ _|_ _| |_ _ _ _|

Case 2:
 _ _ _   _ _ _
| |   | | |   |
|_|_ _| |     |
|_|_|_| |     |
|_ _|_| |_ _ _|
```

The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem H

Silly Sort

Input: sillysort.in

Your younger brother has an assignment and needs some help. His teacher gave him a sequence of numbers to be sorted in ascending order. During the sorting process, the places of two numbers can be interchanged. Each interchange has a cost, which is the sum of the two numbers involved.

You must write a program that determines the minimal cost to sort the sequence of numbers.

Input

The input file contains several test cases. Each test case consists of two lines. The first line contains a single integer n (n >1), representing the number of items to be sorted. The second line contains n different integers (each positive and less than 1000), which are the numbers to be sorted.

The input is terminated by a zero on a line by itself.

Output

For each test case, the output is a single line containing the test case number and the minimal cost of sorting the numbers in the test case.

Place a blank line after the output of each test case

Sample Input	Output for the Sample Input
3 3 2 1 4 8 1 2 4 5 1 8 9 7 6 6 8 4 5 3 2 7 0	Case 1: 4 Case 2: 17 Case 3: 41 Case 4: 34

This page is intentionally blank.

The 2002 26th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem I

Merrily, We Roll Along!

Input: wheel.in

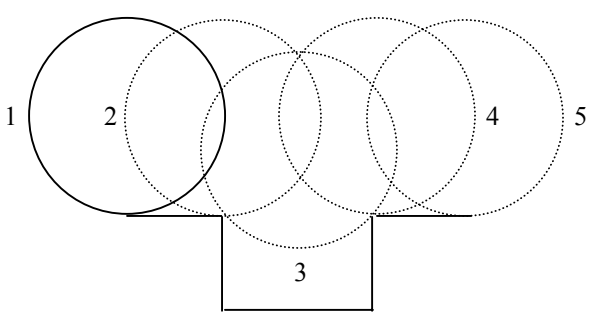
One method used to measure the length of a path is to roll a wheel (similar to a bicycle wheel) along the path. If we know the radius of the wheel and the number of revolutions it makes as it travels along the path, the length of the path can be computed.

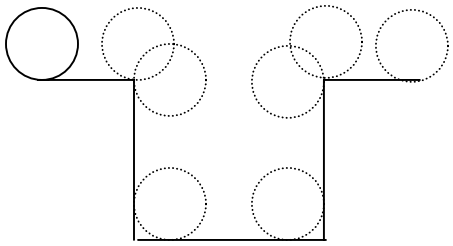
This method works well if the path is smooth. But when there are curbs or other abrupt elevation changes in the path, the path distance may not be accurately determined, because the wheel may rotate around a point (like the edge of a curb), or the wheel may roll along a vertical surface. In this problem you are to determine the distance moved by the center of such a wheel as it travels along a path that includes only horizontal and vertical surfaces.

To measure a path, the wheel is placed with its center directly above the origin of the path. The wheel is then moved forward over the path as far as possible, always remaining in contact with the surface, ending with its center directly above the end of the path.

Consider the path shown in the illustration on the left below, and assume the wheel has a radius of 2. The path begins and ends with horizontal segments of length 2 at the same elevation. Between these there is a horizontal segment of length 2.828427 at 2 units below the elevation of the other two horizontal segments. To measure this path, the wheel is placed at position 1. It then moves horizontally to position 2, rotates 45 degrees to position 3, rotates another 45 degrees to position 4, and finally rolls horizontally to position 5. The center of the wheel moved a distance of 7.1416, not 6.8284.

In the illustration on the right below, the path begins and ends with horizontal segments of length 3, separated by a 7-unit wide region placed 7 units below the surface. If the wheel has a radius of 1, then it will move 26.142 units before reaching the end of the path.





Input

For this problem there are multiple input cases. Each case begins with a positive real number specifying the radius of the wheel and an integer *n*, which is at least 1 but not greater than 50. There then follow *n* pairs of real numbers. The first number in each pair gives the horizontal distance along the path to the next vertical surface. The second number in each pair gives the signed change in the elevation of the path at the vertical surface, with positive numbers representing an increase in elevation. The vertical surfaces are always perpendicular to the horizontal surfaces. The elevation change in the *n*th pair will always be 0.

The input is terminated by a pair of zeroes.

Output

For each case, display the case number and the distance moved by the center of the wheel with 3 digits to the right of the decimal point.

Place a blank line after the output of each test case

Sample Input	Output for the Sample Input
2.0 3 2.0 -2.0 2.828427 2.0 2.0 0.0 1.0 3 3.0 -7.0 7.0 7.0 3.0 0.0 1.0 3 1.0 -4.0 2.0 4.0 1.0 0.0 0 0	Case 1: Distance = 7.142 Case 2: Distance = 26.142 Case 3: Distance = 5.142