University of Manitoba
Open Programming Contest
September 25, 2010

General Instructions:

1. Submit solutions using the PC^2 software.
2. The questions are **not** listed in order of difficulty. Some questions will be easier than others, and these are not necessarily the problems listed first.
3. All input should be read from standard input.

Problem 1 – BOGO

Jim works at a clerk at a store that perpetually has a "buy one get one free" (BOGO) sale. Technically, the store's policy is "buy one item, get another item of equal or lesser value for free". Unfortunately, Jim's customers rarely understand which items they should pay for and which they should get for free. They always buy an even number of items, though.

Write a program that takes an even number of prices of items, and calculates the minimum total price the customer can pay for all the items with they "buy one, get one cheaper item free" rule.

Input

The input consists of several lines of input. Each line should be processed as a separate customer. Each line consists of an unknown (but even) number of floating-point numbers, each separated by a space. These represent the prices of the items.

Output

For each line, output "Customer pays $x" where x is the minimum amount that the customer can pay with the BOGO rule.

Sample Input	Sample Output
13 15 17.7 19.1 13.37 31.41 14.14 9001.01 1 4.1 4.1 4.1	Customer pays $34.1 Customer pays $9015.15 Customer pays $8.2

Problem 2 – Multi-tap Input

In non-smart cell phones, text input can be handled by a technique called *multi-tap*. With multi-tap, you use the letters on each key, and press a key multiple times to access different letters. For instance, to get the letter 'a', you press the 2 key once, but to get 'n', you press the 6 key twice (the first press gives you 'm', the second switches 'm' to 'n'). A slight pause by the user will indicate the end of inputting one character and a beginning of the input of the next character.

The letters 'q' and 'z', not originally part of the letters on a phone, force the keys 7 and 9 to have four characters each. All other keys have 3 characters.

Write a program which, given a word (a string of lower case letters with no spaces), outputs the sequence of keystrokes using multi-tap. If a key is pressed multiple times, it should appear that many times in the input. A pause is indicated by a single space.

Input

Each line of input contains a single word (all lower case, no spaces). Process each line until the end of the file.

Output

For each line, output the correct multi-tap output.

Sample Input	Sample Output
`parasitic` `quizzical`	`7 2 777 2 7777 444 8 444 222` `77 88 444 9999 9999 444 222 2 555`

Problem 3 – Sliding Square Puzzle

Noyes Chapman, the Postmaster of Canastota, New York invented the well-known 15 Puzzle, in 1880. The puzzle consists of 15 squares, numbered from 1 to 15, that are placed in 4x4 grid leaving one grid cell empty. The player makes a move by sliding squares to the adjacent empty cell, which will reveal another empty cell. The goal is to reorder the squares until the numbers are in increasing order leaving the bottom-right hand square empty.

In this problem, you will be asked to write a program that finds the minimum number of moves required to solve the simpler 8 Puzzle which consists of a 3x3 grid with 8 squares numbered from 1 to 8. For example, the initial configuration may be:

1	**3**	**5**
4	**8**	**2**
7	**6**	

The puzzle is solved when the following configuration is reached:

1	**2**	**3**
4	**5**	**6**
7	**8**	

If no solution exists within 20 moves, the configuration is considered to be unsolvable.

Input

The first input line contains a single integer c, which is the number of board configuration. Each board configuration is represented by 9 integers, which corresponds to the 9 squares starting from the upper-left hand corner to the bottom-right hand corner. The empty square is represented by 0.

Output

For each board configuration, output the minimum number of moves required to solve it followed by the newline character. If the configuration is unsolvable, then output “Unsolvable.” followed by the newline character.

Sample Input	Sample Output
3 1 2 3 4 5 6 0 7 8 1 3 5 4 8 2 7 6 0 0 8 7 6 5 4 3 2 1	2 8 Unsolvable.

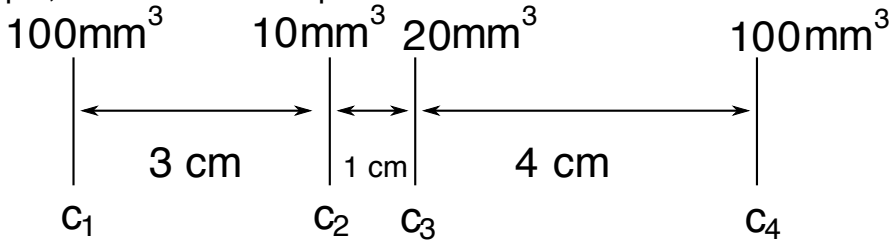
Problem 4 – Thin it to win it

Mike has been given the job of thinning the carrots in his garden. According to the directions on the packet of seeds, you should plant seeds close to each other but after the seeds sprout they should be "thinned" to allow 5 cm between each carrot.

Several factors affect the carrots: some seeds don't sprout and not all carrots grow at the same rate. So, by the time Mike is ready to thin the plants, there are a variety of distances between the carrots and a variety of sizes of carrots. Mike also plants different varieties of carrots, which should be thinned to different distances.

Mike wants to determine which carrots to pull. He gives you two arrays of integers: the distance between the plants (in cm) and the sizes of the carrot (an estimate of the volume of the carrot, in mm^3). He would like to know the biggest volume of carrots that can remain in the ground where each carrot is at least the minimum distance from their neighbour in the row.

For example, consider an example where there are four carrots in a row:



If the minimum distance is 5 cm, then Mike would pull out carrots c_2 and c_3, leaving carrots c_1 and c_4 with a total volume of 200 mm^3.

Input

The first line of input gives the number of rows of carrots to process. For each row of carrots, the first two numbers are on a single line; they give the minimum distance `D` between carrots and the number of carrots `n`. The next two lines of input are

- the volumes of carrots: this line has `n` integers, specifying the volumes of the `n` carrots (in order).
- the distance between carrots: this line has `n-1` integers, specifying the distance between the `n` carrots (in the same order as the volumes).

Output

For each line, give the maximum volume of carrots that can be left in the row after thinning.

Sample Input	Sample Output
3 5 4 100 10 20 10 3 1 4 5 3 10 100 10 4 4 4 6 90 40 30 20 10 55 2 1 5 3 2	110 100 165

Problem 5 – Magnets: how do they work??

A set of magnetic objects are placed on a flat surface. After being released, some of these magnets will be forced towards others. When the system stabilizes, some magnets will have grouped together. How many such groups are there in the final configuration?

Our model of the attraction between the objects is:

- each object has a magnitude, given by a positive integer. The larger the magnitude, the stronger the attracting power the object has.
- all objects attract other objects in all possible directions (i.e., there are no poles on our magnetic objects, and nothing repels anything else).
- we calculate the force between two objects of magnitude m_1 and m_2 which are r units apart by m_1m_2/r^2.
- all objects need the same force applied to them to move. This is given as the "moving force" F_M for the system. (This may vary due to the type of surface being used.) So if two objects are to move towards each other, their force must exceed F_M. Once two objects begin moving, they continue to do so until they meet. We assume the same force is required regardless of how many objects are in a group of objects.
- when two objects with magnitude m_1 and m_2 are attracted to each other, their new magnitude is the sum of their individual magnitudes. Their position is the midpoint of the line segment between their two positions.
- when multiple pairs of objects can be attracted to each other, we consider those with the greatest total force between them to be attracted first. No tiebreaking which will impact the final outcome will be required.

An initial configuration is given by the size of the square surface, the moving force, the number of objects and their positions and magnitudes. From this, calculate the number of distinct groups of objects after all movement has concluded.

Input

The first line of the input consists of a single integer, which gives the number of surfaces to test.

The first line of each surface is two integers: `F` (the minimum force to move to objects) and `r` (the number of objects on the surface). The next `r` lines each consist of three integers: the (x,y)-coordinates of the object, followed by its magnitude.

Output

For each surface, output: "`Surface #x: YY groups`." where `x` is the number of the test (starting from 1) and `YY` is the final number of groups of objects on the surface.

Sample Input	Sample Output
3 5 3 1 1 4 3 1 2 3 2 3 2 4 0 0 1 0 1 1 1 0 1 1 1 1 2 3 0 0 2 0 1 2 3 1 1	Surface #1: 2 groups. Surface #2: 4 groups. Surface #3: 2 groups.

Problem 6 – Diagonally

Imagine an n-by-n square grid of numbers written by starting at the upper-left hand corner and writing the numbers 1 through n^2 clockwise, spiralling in to the centre. So for n=5, the square would look like this:

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Imagine the diagonal that goes from the upper-left hand corner (always labelled 1) to the centre of the square (always labelled n^2). What is the sum of the entries on this diagonal? In the case of n=5 it is 1+17+25 = 43.

Input

The first input line contains a single integer c, which is the number of test cases. The next c lines each containing a single 16-bit odd number. This is the size of the grid to be tested.

Output

For each input line (except the first), output the sum of the diagonal from the upper-left hand corner to the centre of an n-by-n grid labelled as above.

Sample Input	Sample Output
4 5 9 1001 12345	43 245 334334501 627121989933