**Problem A. Magic Trick**

1. Problem

Recently you went to a magic show. You were very impressed by one of the tricks, so you decided to try to figure out the secret behind it!

The magician starts by arranging 16 cards in a square grid: 4 rows of cards, with 4 cards in each row. Each card has a different number from 1 to 16 written on the side that is showing. Next, the magician asks a volunteer to choose a card, and to tell him which row that card is in.

Finally, the magician arranges the 16 cards in a square grid again, possibly in a different order. Once again, he asks the volunteer which row her card is in. With only the answers to these two questions, the magician then correctly determines which card the volunteer chose. Amazing, right?

You decide to write a program to help you understand the magician's technique. The program will be given the two arrangements of the cards, and the volunteer's answers to the two questions: the row number of the selected card in the first arrangement, and the row number of the selected card in the second arrangement. The rows are numbered 1 to 4 from top to bottom.

Your program should determine which card the volunteer chose; or if there is more than one card the volunteer might have chosen (the magician did a bad job); or if there's no card consistent with the volunteer's answers (the volunteer cheated).

1. Solving this problem

Usually, Google Code Jam problems have 1 Small input and 1 Large input. This problem has only **1 Small input**. Once you have solved the Small input, you have finished solving this problem.

1. Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing an integer: the answer to the first question. The next 4 lines represent the first arrangement of the cards: each contains 4 integers, separated by a single space. The next line contains the answer to the second question, and the following four lines contain the second arrangement in the same format.

1. Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1).

If there is a single card the volunteer could have chosen, y should be the number on the card. If there are multiple cards the volunteer could have chosen, y should be "Bad magician!", without the quotes. If there are no cards consistent with the volunteer's answers, y should be "Volunteer cheated!", without the quotes. The text needs to be exactly right, so consider copying/pasting it from here.

1. Limits

1 ≤ **T** ≤ 100.  
1 ≤ both answers ≤ 4.  
Each number from 1 to 16 will appear exactly once in each arrangement.

1. Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  2  1 2 3 4  5 6 7 8  9 10 11 12  13 14 15 16  3  1 2 5 4  3 11 6 15  9 10 7 12  13 14 8 16  2  1 2 3 4  5 6 7 8  9 10 11 12  13 14 15 16  2  1 2 3 4  5 6 7 8  9 10 11 12  13 14 15 16  2  1 2 3 4  5 6 7 8  9 10 11 12  13 14 15 16  3  1 2 3 4  5 6 7 8  9 10 11 12  13 14 15 16 | Case #1: 7  Case #2: Bad magician!  Case #3: Volunteer cheated! |

**Problem B. Cookie Clicker Alpha**

1. Introduction

*Cookie Clicker* is a Javascript game by Orteil, where players click on a picture of a giant cookie. Clicking on the giant cookie gives them cookies. They can spend those cookies to buy buildings. Those buildings help them get even more cookies. Like this problem, the game is very cookie-focused. This problem has a similar idea, but it does not assume you have played *Cookie Clicker*. Please don't go play it now: it might be a long time before you come back.

1. Problem

In this problem, you start with 0 cookies. You gain cookies at a rate of 2 cookies per second, by clicking on a giant cookie. Any time you have at least **C** cookies, you can buy a cookie farm. Every time you buy a cookie farm, it costs you **C** cookies and gives you an extra **F** cookies per second.

Once you have **X** cookies that you haven't spent on farms, you win! Figure out how long it will take you to win if you use the best possible strategy.

1. Example

Suppose **C**=500.0, **F**=4.0 and **X**=2000.0. Here's how the best possible strategy plays out:

1. You start with 0 cookies, but producing 2 cookies per second.
2. After **250** seconds, you will have **C**=500 cookies and can buy a farm that produces**F**=4 cookies per second.
3. After buying the farm, you have 0 cookies, and your total cookie production is 6 cookies per second.
4. The next farm will cost 500 cookies, which you can buy after about **83.3333333**seconds.
5. After buying your second farm, you have 0 cookies, and your total cookie production is 10 cookies per second.
6. Another farm will cost 500 cookies, which you can buy after **50** seconds.
7. After buying your third farm, you have 0 cookies, and your total cookie production is 14 cookies per second.
8. Another farm would cost 500 cookies, but it actually makes sense not to buy it: instead you can just wait until you have **X**=2000 cookies, which takes about**142.8571429** seconds.

Total time: 250 + 83.3333333 + 50 + 142.8571429 = 526.1904762 seconds.

Notice that you get cookies continuously: so 0.1 seconds after the game starts you'll have 0.2 cookies, and π seconds after the game starts you'll have 2π cookies.

1. Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains three space-separated real-valued numbers: **C**, **F** and **X**, whose meanings are described earlier in the problem statement.

**C**, **F** and **X** will each consist of at least 1 digit followed by 1 decimal point followed by from 1 to 5 digits. There will be no leading zeroes.

1. Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of seconds it takes before you can have **X** delicious cookies.

We recommend outputting y to 7 decimal places, but it is not required. y will be considered correct if it is close enough to the correct number: within an absolute or relative error of 10-6. See the [FAQ](https://code.google.com/codejam/faq.html#floating_point) for an explanation of what that means, and what formats of real numbers we accept.

1. Limits

1 ≤ **T** ≤ 100.

1. Small dataset

1 ≤ **C** ≤ 500.  
1 ≤ **F** ≤ 4.  
1 ≤ **X** ≤ 2000.

1. Large dataset

1 ≤ **C** ≤ 10000.  
1 ≤ **F** ≤ 100.  
1 ≤ **X** ≤ 100000.

1. Sample

|  |  |
| --- | --- |
| Input | Output |
| 4  30.0 1.0 2.0  30.0 2.0 100.0  30.50000 3.14159 1999.19990  500.0 4.0 2000.0 | Case #1: 1.0000000  Case #2: 39.1666667  Case #3: 63.9680013  Case #4: 526.1904762 |

1. Note

*Cookie Clicker* was created by Orteil. Orteil does not endorse and has no involvement with Google Code Jam.

**Problem C. Minesweeper Master**

1. Problem

*Minesweeper* is a computer game that became popular in the 1980s, and is still included in some versions of the *Microsoft Windows* operating system. This problem has a similar idea, but it does not assume you have played *Minesweeper*.

In this problem, you are playing a game on a grid of identical cells. The content of each cell is initially hidden. There are **M** mines hidden in **M** different cells of the grid. No other cells contain mines. You may click on any cell to reveal it. If the revealed cell contains a mine, then the game is over, and you lose. Otherwise, the revealed cell will contain a digit between 0 and 8, inclusive, which corresponds to the number of neighboring cells that contain mines. Two cells are neighbors if they share a corner or an edge. Additionally, if the revealed cell contains a 0, then all of the neighbors of the revealed cell are automatically revealed as well, recursively. When all the cells that don't contain mines have been revealed, the game ends, and you win.

For example, an initial configuration of the board may look like this ('\*' denotes a mine, and 'c' is the first clicked cell):

\*..\*...\*\*.

....\*.....

..c..\*....

........\*.

..........

There are no mines adjacent to the clicked cell, so when it is revealed, it becomes a 0, and its 8 adjacent cells are revealed as well. This process continues, resulting in the following board:

\*..\*...\*\*.

1112\*.....

00012\*....

00001111\*.

00000001..

At this point, there are still un-revealed cells that do not contain mines (denoted by '.' characters), so the player has to click again in order to continue the game.

You want to win the game as quickly as possible. There is nothing quicker than winning in one click. Given the size of the board (**R** x **C**) and the number of hidden mines **M**, is it possible (however unlikely) to win in one click? You may choose where you click. If it is possible, then print any valid mine configuration and the coordinates of your click, following the specifications in the *Output* section. Otherwise, print "Impossible".

1. Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains three space-separated integers: **R**, **C**, and **M**.

1. Output

For each test case, output a line containing "Case #x:", where x is the test case number (starting from 1). On the following **R** lines, output the board configuration with **C** characters per line, using '.' to represent an empty cell, '\*' to represent a cell that contains a mine, and 'c' to represent the clicked cell.

If there is no possible configuration, then instead of the grid, output a line with"Impossible" instead. If there are multiple possible configurations, output any one of them.

1. Limits

0 ≤ **M** < **R** \* **C**.

1. Small dataset

1 ≤ **T** ≤ 230.  
1 ≤ **R**, **C** ≤ 5.

1. Large dataset

1 ≤ **T** ≤ 140.  
1 ≤ **R**, **C** ≤ 50.

1. Sample

|  |  |
| --- | --- |
| Input | Output |
| 5  5 5 23  3 1 1  2 2 1  4 7 3  10 10 82 | Case #1:  Impossible  Case #2:  c  .  \*  Case #3:  Impossible  Case #4:  ......\*  .c....\*  .......  ..\*....  Case #5:  \*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*  \*\*\*\*....\*\*  \*\*\*.....\*\*  \*\*\*.c...\*\*  \*\*\*....\*\*\*  \*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\* |

**Problem D. Deceitful War**

1. Problem

Naomi and Ken sometimes play games together. Before they play, each of them gets **N**identical-looking blocks of wood with masses between 0.0kg and 1.0kg (exclusive). All of the blocks have different weights. There are lots of games they could play with those blocks, but they usually play something they call War. Here is how War works:

1. Each player weighs each of his or her own blocks, so each player knows the weights of all of his or her own blocks, but not the weights of the other player's blocks.
2. They repeat the following process **N** times:
   1. Naomi chooses one of her own blocks, with mass ChosenNaomi.
   2. Naomi tells Ken the mass of the block she chose.
   3. Ken chooses one of his own blocks, with mass ChosenKen.
   4. They each put their block on one side of a [balance scale](https://www.google.com/search?q=balance+scale&tbm=isch), and the person whose block is heavier gets one point.
   5. Both blocks are destroyed in a fire.

Naomi has realized three things about War. First, she has realized that she loses a lot. Second, she has realized that there is a unique strategy that Ken can follow to maximize his points without assuming anything about Naomi's strategy, and that Ken always uses it. Third, she has realized that she hates to lose. Naomi has decided that instead of playing War, she will play a game she calls Deceitful War. The great thing about Deceitful War is that Ken will think they're playing War!

Here is how Deceitful War works, with differences between Deceitful War and War in bold:

1. Each player weighs each of his or her own blocks. **Naomi also weighs Ken's blocks while he isn't looking, so Naomi knows the weights of all blocks** and Ken only knows the weights of his own blocks.
2. They repeat the following process **N** times:
   1. Naomi chooses one of her own blocks, with mass ChosenNaomi.
   2. Naomi tells Ken **a number, ToldNaomi, between 0.0kg and 1.0kg exclusive.** Ken, who thinks they're playing War, thinks the number Naomi just told him is ChosenNaomi.
   3. Ken chooses one of his own blocks, with mass ChosenKen.
   4. They each put their block on one side of a [balance scale](https://www.google.com/search?q=balance+scale&tbm=isch), and the person whose block is heavier gets one point.
   5. Both blocks are destroyed in a fire.

Naomi doesn't want Ken to know that she isn't playing War; so when she is choosing which block to play, and what mass to tell Ken, she must make sure that the balance scale won't reveal that ChosenNaomi ≠ ToldNaomi. In other words, she must make decisions so that:

* ChosenNaomi > ChosenKen if, and only if, ToldNaomi > ChosenKen, and
* ToldNaomi is not equal to the mass of any of Ken's blocks, because he knows that isn't possible.

It might seem like Naomi won't win any extra points by being deceitful, because Ken might discover that she wasn't playing War; but Naomi knows Ken thinks both players are playing War, and she knows what he knows, and she knows Ken will always follow his unique optimal strategy for War, so she can always predict what he will play.

You'll be given the masses of the blocks Naomi and Ken started with. Naomi will play Deceitful War optimally to gain the maximum number of points. Ken will play War optimally to gain the maximum number of points *assuming that both players are playing War*. What will Naomi's score be? What would it have been if she had played War optimally instead?

1. Examples

If each player has a single block left, where Naomi has 0.5kg and Ken has 0.6kg, then Ken is guaranteed to score the point. Naomi can't say her number is ≥ 0.6kg, or Ken will know she isn't playing War when the balance scale shows his block was heavier.

If each player has two blocks left, where Naomi has [0.7kg, 0.2kg] and Ken has [0.8kg, 0.3kg], then Naomi could choose her 0.2kg block, and deceive Ken by telling him that she chose a block that was 0.6kg. Ken assumes Naomi is telling the truth (as in how the War game works) and will play his 0.8kg block to score a point. Ken was just deceived, but he will never realize it because the balance scale shows that his 0.8kg block is, like he expected, heavier than the block Naomi played. Now Naomi can play her 0.7kg block, tell Ken it is 0.7kg, and score a point. If Naomi had played War instead of Deceitful War, then Ken would have scored two points and Naomi would have scored zero.

1. Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing a single integer **N**, the number of blocks each player has. Next follows a line containing **N** space-separated real numbers: the masses of Naomi's blocks, in kg. Finally there will be a line containing **N** space-separated real numbers: the masses of Ken's blocks, in kg.

Each of the masses given to Ken and Naomi will be represented as a 0, followed by a decimal point, followed by 1-5 digits. Even though all the numbers in the input have 1-5 digits after the decimal point, Ken and Naomi don't know that; so Naomi can still tell Ken that she played a block with mass 0.5000001kg, and Ken has no reason not to believe her.

1. Output

For each test case, output one line containing "Case #**x**: **y** **z**", where **x** is the test case number (starting from 1), **y** is the number of points Naomi will score if she plays Deceitful War optimally, and **z** is the number of points Naomi will score if she plays War optimally.

1. Limits

1 ≤ **T** ≤ 50.  
All the masses given to Ken and Naomi are distinct, and between 0.0 and 1.0 exclusive.

1. Small dataset

1 ≤ **N** ≤ 10.

1. Large dataset

1 ≤ **N** ≤ 1000.

1. Sample

|  |  |
| --- | --- |
| Input |  |
| 4  1  0.5  0.6  2  0.7 0.2  0.8 0.3  3  0.5 0.1 0.9  0.6 0.4 0.3  9  0.186 0.389 0.907 0.832 0.959 0.557 0.300 0.992 0.899  0.916 0.728 0.271 0.520 0.700 0.521 0.215 0.341 0.458 |  |
|  |  |
| Output |  |
| Case #1: 0 0  Case #2: 1 0  Case #3: 2 1  Case #4: 8 4 |  |