## 11605 - Lights inside a 3d Grid

Time limit: 5.000 seconds

You are given a 3D grid, which have dimensions **N**, **M** and **P**. Each of the **M x N x P** cells has a light. Initially all lights are off. You will have **K** turns. In each of the **K** turns,

- You will select a cell A randomly from the grid

- You will select a cell B randomly from the grid

- Toggle the states of all the bulbs bounded by cell A and cell B, ie make all the ON lights OFF and make all the OFF lights ON which are bounded by A and B. To be more clear, consider cell A is (x1, y1, z1) and cell B is (x2, y2, z2). Then you have to toggle all the bulbs in grid cell (x, y, z) where min(x1,x2)<=x<=max(x1,x2) , min(y1,y2)<=y<=max(y1,y2) and min(z1, z2)<=z<=max(z1, z2).

How many bulbs are expected to be ON after K turns?

Note:

- A and B can be the same cell.

**Input**

First line of the input is an integer **T(T<101)** which denotes the number of test cases. Each of the next **T** lines represents one test case by 4 integers **N, M, P (0 < M, N, P < 101)** and **K (0<=K<=10000)** separated by spaces.

**Output**

Output one line for each test cases giving the expected number of ON lights. Up to 1E-6 error in your output will be acceptable. Print the case number followed by the output. Look at the sample output for exact format.

| **Sample Input** | **Output for Sample Input** |
|---|---|
| 2<br><br>2 3 4 1<br><br>2 3 4 2 | Case 1: 6.3750000000<br><br>Case 2: 9.0976562500 |

## 11606 - Atoms

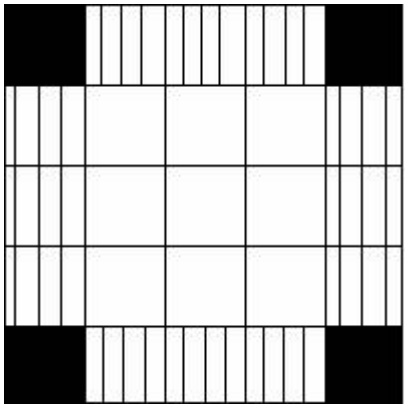
Time limit: 2.000 seconds

*Atoms* is a two player game consisting of an **N*N** grid. The top-left cell has a coordinate of (**0, 0**) and the bottom-right has a coordinate of (**N-1,N-1**). Each player has an infinite number of *atoms* with them. Every *atom* of player one has the same color. The *atoms* of player two also have same color but they are different from those of player one. The two players take turns to place *atoms* on the grid. On each turn, a player selects a *stable* cell to place one *atom* from his collection. The rule for *stability* is simple; the cell must either be empty or it must already contain an *atom* of that player's color. There is however an upper limit to the number of *atoms* a cell may contain and they are described below:

i) The four corner cells can contain at most 1 *atom*.

ii) The other outer cells can contain at most 2 *atoms*.

iii) The remaining cells can contain at most 3 *atoms*.



As an example, the figure above shows a grid of 5 x 5. The filled cells can contain at most 1 *atom* each. The stripped ones can contain at most 2 *atoms* each and the blank ones can contain at most 3 *atoms* each.

When selecting a *stable* cell, the two players follow rather simple rules:

i) First, player one places an *atom* on a randomly selected cell.

ii) Player two looks for a cell whose minimum distance from any cell occupied by *atoms* of player one is maximized. Here distances between cells are measured as Manhattan distance. The Manhattan distance between two cells is the sum of the absolute row difference and column difference. For example, the Manhattan distance between (1, 5) and (4, 2) is equal to (4-1) + (5-2) = 3 + 3 = 6.

iii) In case of multiple cells fulfilling the criteria of rule (**ii**), the cell with lowest row number is selected. If there is still a tie, then the cell with lowest column number is selected.

iv) Player one then follows a similar strategy to that described in (**ii**) to place another *atom* and the game continues likewise.

If a player is unable to make any valid move, then he is considered the loser and the game stops there.

Your task in this problem is simple. Given a state of the grid, you are to determine if this state is achievable if the players follow the rules mentioned earlier.

**Input**

The first line of input will start with a positive integer **T<50**, where **T** denotes the number of test cases. Each case will begin with a number **N(3<=N<=7**) which denotes the size of the grid for that case. **N** lines will follow with **N** integers on each line. The absolute value of these integers will be less than $\mathbf{10^7}$. A positive integer means player one has placed *atoms* in the corresponding cell and negative integer means player two placed *atoms* in the cell. The number of *atoms* in a cell is denoted by the absolute value of the cell.

**Output**

For each case, output the case number first. If the given configuration is valid output “valid”, otherwise output “invalid”. Look at the sample in/out for exact format.

| **Sample Input** | **Output for Sample Input** |
|---|---|
| 2<br>3<br>1 0 0<br>0 0 0<br>0 0 -1<br>3<br>2 0 0<br>0 0 0<br>0 0 0 | Case 1: valid<br>Case 2: invalid |

## 11608 - No Problem

Time limit: 1.000 seconds

Programming contests are being arranged so frequently these days. While this might be a good news for the contestants, the scenario is completely opposite for the problemsetters. So far, the problemsetters somehow managed to produce some sorts of a set & say “No problem!”. But it is doubtful how long will it be possible if the trend of arranging contests in a short notice continues.

You are given the number of problems created in every month of a year and number of problems required in each month. If N problems are required in a month & there are not enough problems at that time, all contests of that month is canceled. Write a program to determine if there are enough problems for the contests. Please keep in mind that, if a problem is created in month X, it can only be used in month X + 1 & the later months.

### Input

The first line of every test case has an integer **S (0<=S<=100)**. Number of problems that is ready at the beginning of the year. The 2nd line has 12 space separated integers, denoting the number of problems created in each of the 12 months of that year. The months are in the same order as they appear in a year. The 3rd line has another 12 space separated integers, the number of problems required to use in contests in those 12 months (With the same order as above). These integers will be between 0 & 20 (inclusive). The end of input will be denoted by a negative integer.

### Output

For each test case, print a line of the form, “**Case X:** “, where **X** is the case number. Then print 12 lines. If there are enough problems to meet the requirements in month **i (1<=i<=12)**, print **“No problem! :D”** in the **i**'th line, otherwise print **“No problem. :(”**

| **Sample Input** | **Output for Sample Input** |
|---|---|
| 5<br>3 0 3 5 8 2 1 0 3 5 6 9<br>0 0 10 2 6 4 1 0 1 1 2 2<br>-1 | Case 1:<br>No problem! :D<br>No problem! :D<br>No problem. :(<br>No problem! :D<br>No problem! :D<br>No problem! :D<br>No problem! :D<br>No problem! :D<br>No problem! :D<br>No problem! :D<br>No problem! :D<br>No problem! :D |

## 11609 - Teams

Time limit: 1.000 seconds

In a galaxy far far away there is an ancient game played among the planets. The specialty of the game is that there is no limitation on the number of players in each team, as long as there is a captain in the team. (The game is totally strategic, so sometimes less player increases the chance to win). So the coaches who have a total of **N** players to play, selects **K (1 ≤ K ≤ N)** players and make one of them as the captain for each phase of the game. Your task is simple, just find in how many ways a coach can select a team from his **N** players. Remember that, teams with same players but having different captain are considered as different team.

### Input

The first line of input contains the number of test cases **T ≤ 500**. Then each of the next T lines contains the value of **N (1 ≤ N ≤ 10^9)**, the number of players the coach has.

### Output

For each line of input output the case number, then the number of ways teams can be selected. You should output the result modulo 1000000007.

For exact formatting, see the sample input and output.

**Sample Input** **Output for Sample Input**

| Sample Input | Output for Sample Input |
|---|---|
| 3<br>1<br>2<br>3 | Case #1: 1<br>Case #2: 4<br>Case #3: 12 |

## 11610 - Reverse Prime

Time limit: 1.000 seconds

There are a few 7 digit positive numbers whose reverse number is a prime number and less than **10^6**. For example: 1000070, 1000090 and 1000240 are first few reverse prime numbers because all of the numbers are 7 digit numbers whose reverse number is a prime number and less than 10^6. You have to find out all the 7 digit reverse prime numbers and also it's number of prime factors. Prime factors of a positive integer are the prime numbers that divide into that integer exactly, without leaving a remainder. For example, prime factors of 24 are 2, 2, 2 and 3.

In this problem, you'll encounter 2 types of input –

Query:

This type of input will be formatted like this – **"q *i*"**. For this input, you have to calculate the cumulative summation of the number of prime factors of reverse prime numbers from 0-th to ***i***-th index.

Deletion:

This type of input will be formatted like this – **"d *reverse_prime*"**. For this input, you have to delete *reverse_prime* from the set and update your summation. No output is required in such cases.

It is guaranteed that *i* will be a valid index and *reverse_prime* will be a valid 7 digit reverse prime number. It is also guaranteed that no two *reverse_prime* will be same.

There will be at most **71000** query lines and **3500** deletion lines in the data set. The program will terminated by EOF.

| **Sample Input** | **Output for Sample Input** |
|---|---|
| q 0<br>q 1<br>q 2<br>d 1000070<br>d 1000090<br>q 0<br>d 1000240<br>q 0<br>q 1 | 4<br>10<br>16<br>6<br>3<br>7 |

## 11612 - Sultan and Khairun Shundori

Time limit: 1.000 seconds

Flowerland is in festive mood. Their princess Khairun Shundori will choose her life-partner today. Princes and handsome young men from distant and neighboring countries have gathered in the capital. Princess will interview them one after one. Among these young men, there's Sultan from Codeland.

The interviews start. One after one, failed young men get out of King's castle in great disappointment. Then comes our hero Sultan. He impresses the Princess with his amazing grace and smartness. He passes all the tests the Princess poses before him. Almost impressed, the Princess gives him one last task – he have to collect roses from all the gardens in the city for her.

Seems like an easy task, doesn't it? Especially when Sultan has already pin-pointed every garden in the city using Google Maps. But there are some catches – as soon as Sultan gets out of the castle, the losers will try to kill him out of jealousy. Sultan will be followed and followed in great number. So he can't afford to take the risk of going through a point he previously visited i.e. he can't intersect his previous path. Then again, he wants to finish this task as soon as possible. So he will always run from one garden to another in a straight line.

### Input

There will be at most ***50*** test cases. Each test case starts with an integer ***n***, number of points of interest (one of them is the castle and the rest are gardens) ***[3≤n≤1000]***. Each point of interest will be specified by the integer ***(x,y)*** co-ordinate of that point ***[-10000≤x,y≤10000]***. No two points of interest will be located in the same co-ordinate. The castle will always be the first point in input. Sultan has to start and end his journey in the Castle. Input is terminated by a case where ***n*** equals 0.

### Output

For each test case, you have to output a sequence of indices of points of interest in one line that will ensure the task. Any sequence that doesn't violate the conditions will be accepted. Points will be indexed from 0 in the output. If there is no valid sequence for the input, you have output "***no solution***".

| Sample Input | Output for Sample Input |
|---|---|
| 4<br>0 0<br>1 0<br>1 1<br>0 1<br>5<br>5 5<br>10 0<br>10 10<br>0 10<br>0 0<br>0 | 0 1 2 3 0<br>0 1 2 3 4 0 |