1. Benford's Law: Who's Number One?

Program Name: Benford.java   Input File: benford.dat

Benford's Law, also known as the first digit law, states that given a set of values based on real data or measurements the distribution of the leading digits of those values is **not** expected to be uniform. Instead it is often the case that there are more values that start with a 1 than any other. Values that start with a 2 will be the next most frequent and so forth with values starting with 9 expected to be the least frequent. Write a program that given a data set of integers determines if the data confirms Benford's Law or not. For this problem a data set confirms Benford's Law if the number of values that start with a 1 is greater than or equal to each of the number of values that start with 2 through 9. Likewise the number of values that start with 2 is greater than or equal to each of the number of values that start with 3 through 9. And so forth for the number of values that start with 3 through 8.

Input

- The first line will contain a single integer `n` that indicates the number of data sets that follow.
- The first line of each data set will contain a single integer `m` that indicates how many values are in the data set.
- The values in the data set will be on the following lines, 5 values per line, with a space between each value. The last line in the data set could have fewer than 5 values.
- All values in the data set will be integers greater than 0 and less than 1,000,000. Values will not contain leading 0s.

Output

For each data set print out `CONFIRMS` if the leading digits of the values in the data set confirm Benford's Law. Print out `DOES NOT CONFIRM` if the leading digits of the values in the data set do not follow Benford's Law.

Example Input File

```
3
36
661 59 1452 1338 1438
2923 3796 4796 1 2781
1742 20 626 2230 7553
42 2113 7450 5034 3516
5381 31 5665 101 19
11100 4028 2 6336 4683
35 8975 2908 3177 49
389111
40
76 4319 6859 286 172811
8442 32530 29 4170 2890
61 1044 4478 28310 1619
5093 2500 68 5939 5
55 6105 2841 1207 5
4207 12 2248 1967 124715
13 1613 7391 45 4855
60 4949 28 5488 1755
9
1 22 333 4444 55555
678 713 80200 9000
```

Example Output To Screen

```
CONFIRMS
DOES NOT CONFIRM
CONFIRMS
```

2. Code

Program Name: Code.java   Input File: code.dat

Sally and Nancy have decided to write to each other in a special code. They decide to use the smallest square matrix that will hold all of the characters in their message. They will place their message, one character per cell, into the matrix in column-row order and fill any remaining matrix cells with an asterisk (*). For example, the coded message:

```
Ntlnofu*oiltttn*wmgohht*ieoceer*sfooaiy*todmir.*hrmedc**eaetoo**
```

would have been placed in an 8x8 matrix `as`:

```
Ntlnofu*
oiltttn*
wmgohht*
ieoceer*
sfooaiy*
todmir.*
hrmedc**
eaetoo**
```

so it could be decoded by reading column one, then column two, and so forth and finally getting the message:

```
Nowisthetimeforallgoodmentocometotheaidoftheircountry.**********
```

Input
The first line of input will contain a single integer `n` that indicates the number of coded messages to follow. Each of the following `n` lines will contain a single coded message. Each message will contain a perfect square number of characters. There will be no spaces in the input.

Output
For each line of input, you will print the decoded message with no spaces and with the filler asterisks at the end.

Example Input File
```
3
Ntlnofu*oiltttn*wmgohht*ieoceer*sfooaiy*todmir.*hrmedc**eaetoo**
SnydhradadohlNrloelaeelaynmsadacicis
Malfaa*alalss*rimews*ytbehn*htIcio*altetw*deswe**
```

Example Output to Screen
```
Nowisthetimeforallgoodmentocometotheaidoftheircountry.**********
SallyandNancyaremiddleschoolairheads
MaryhadalittlelambItsfleecewaswhiteassnow********
```

3. Diamond

Program Name: Diamond.java   Input File: none

David wants to create a diamond design for a deck of cards. You are to write a program that will print this design for him.

Input
There is no input for this problem.

Output
Print this diamond design exactly as it appears below.

Example Input File
There is no input for this problem.

Example Output to Screen
```
*************************
***********   ***********
**********  *  **********
*********  ***  *********
********  *****  ********
*******  *******  *******
******  *********  ******
*****  ***********  *****
****  *************  ****
***  ***************  ***
**  *****************  **
*  *******************  *
**  *****************  **
***  ***************  ***
****  *************  ****
*****  ***********  *****
******  *********  ******
*******  *******  *******
********  *****  ********
*********  ***  *********
**********  *  **********
***********   ***********
*************************
```

4. IP Addresses

Program Name: IP.java   **Input File: ip.dat**

IP addresses are formed from 32 character data streams of 1s and 0s. The decimal format of an IP address is formed by grouping 8 bits at a time and converting each binary group to a decimal number. Decimal IP addresses are written in the form of four integers separated by periods. An example of an IP address is `10.103.45.230`.

Input

The first line of input will contain a single integer `n` that indicates the number of IP addresses to follow. Each of the following `n` lines will contain a 32 character binary number.

Output

For each binary IP address input, you will print the decimal format of the address.

Example Input File

```
4
11111111111111111111111111111111
00000011100000001111111111111111
11110000101010100101010100000111
10101010101010101010101010101010
```

Example Output to Screen

```
255.255.255.255
3.128.255.255
240.170.85.7
170.170.170.170
```

5. Max-out

Program Name: Max.java   Input File: max.dat

Max-out is a game that is played on a board consisting of a 6x6 matrix of squares and with an unlimited number of discs. In each square, there is a positive two digit random integer. A sample Max-out board is displayed to the right.

78	78	11	**55**	20	11
98	54	81	43	39	**97**
12	15	79	**99**	58	10
13	**79**	83	65	34	**17**
85	59	61	12	58	97
40	**63**	97	**85**	66	**90**

The object of the game is to cover with a single disc the integer in as many squares as you wish so that the sum of all of the integers covered is as large as possible while abiding by the following rules:

- A single disc can cover exactly one square.
- An individual square can be covered by at most one disc.
- No disc can be placed in a square adjacent to a square containing a disc.
- Squares are considered adjacent only if they touch horizontally, vertically or diagonally.

You are to write a program that will determine the placement of discs to maximize the sum of the integers covered. On the board above, the shaded squares are the squares on which a disc was placed to obtain the maximum score of 683.

Input

The first line of input will contain a single integer `n` that indicates the number of games to follow. Each of the following games will consist of 6 lines with six two-digit integers on each line. The integers on each line will be separated by a space.

Output

For each game and on a separate line, you will print the largest sum possible when covering the integers with discs as outlined by the rules above.

Example Input File

```
1
78 78 11 55 20 11
98 54 81 43 39 97
12 15 79 99 58 10
13 79 83 65 34 17
85 59 61 12 58 97
40 63 97 85 66 90
```

Example Output to Screen

```
683
```

6. Semester Average

Program Name: Semester.java Input File: semester.dat

Ms. Appleworth is a teacher and needs you to write a program to compute the minimum grade that a student must make on the final exam to make a specific grade in her class. To compute the semester average, the average of the three six-weeks grades will count 80% and the final exam will count 20%.

Input

The first line of input will contain a single integer `n` that indicates the number of students in her class. Each of the following `n` lines will contain a student's six-digit ID number, three six-weeks' grades and the semester average the student wishes to have. Each of items will be separated by a space.

Output

In ID number order, you will print the student's ID number, a space, and the minimum grade the student can make to achieve the desired average. To keep the student from having false expectations, the minimum grade should be rounded up to the nearest integer. For example, 87.2 would become 88.

Note: Do not round any computation except the final minimum grade, which will be the smallest integer that is greater than or equal to the grade needed to obtain the desired grade.

Example Input File

```
4
123456 90 90 90 90
543216 70 80 80 70
345326 65 65 65 70
435366 80 80 80 90
```

Example Output to Screen

```
123456 90
345326 90
435366 130
543216 44
```