The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

General Remarks

All languages

- Input must be read from a file called '`x.in`', where `x` stands for the letter of the problem you are solving. That is, input for problem A should be read from file '`a.in`', while input for problem E should be read from file '`e.in`'.
- Output must be written to a file called '`x.out`', where x stands for the letter of the problem you are solving. That is, output for problem A should be written to file '`a.out`', while output for problem C should be written to file '`c.out`'.
- Submitted program files must have as name the single letter of the problem for which they are submitted. For extension naming see below.
- Printing under DOS is not supported. Use for example notepad for printing under Windows.
- If you encounter any print-errors, please notify the runners instead of submitting a clarification request.
- Emacs and WinVi have been installed, you will find shortcuts on your desk top.

Pascal submissions
- Solutions submitted must have file extension '`.pas`'. That is submissions for problem E should be named '`e.pas`'.
- You can use the following command to compile from a DOS box: `bpc`
- Your problem will be compiled with the following compiler switches: `-$M65000,0,650000`

C submissions
- Solutions submitted must have file extension '`.c`'. That is submissions for problem E should be named '`e.c`'.
- You can use the following command to compile from a DOS box: `icc`
- Your problem will be compiled with the following compiler switches: `/Sa`

C++ submissions
- Solutions submitted must have file extension '`.cpp`'. That is submissions for problem E should be named '`e.cpp`'.
- You can use the following command to compile from a DOS box: `icc`
- Your problem will be compiled with no compiler switches.

Java submissions
- Solutions submitted must have **no** file extension (!). That is submissions for problem E should be named '`e`'.
- From a Java solution an object from the class '`solution`' will be executed in order to verify the correctness of your submission. Therefore your submission must always contain this class.
- You can use the following command to compile from a DOS box: `javac`
- You can execute your java program with: `java solution`
- Your problem will be compiled with no compiler switches.

The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

Problem A

Space Management

The new general manager Gill Bates has been to a seminar on time management. Ever since she has been bothering all of her staff to manage their own time. Now she plans to take it one step further: she wants to start managing her space too. And since she has also learned to start small, she wants to start by managing her desk space. Work tends to pile up on her desk, especially since the time management training where she was taught to order all documents in neat piles and organise her work accordingly. She now needs to calculate how much desk space is occupied. For this she comes up with a cunning plan. First, she orders her staff to measure the size and exact location of each document on the desk, in tenths of millemetres accuracy. As Gill is a very tidy person, all her document are completely on her desk, and are all perfectly aligned to the edges of the desk (i.e. the edges of all documents are parallel to an edge of the desk). For each document, the staff writes down the position of the lower left corner of the document and its size.

What you need to do now is to write a program that, given the input from the staff, calculates the occupied space on the desk.

The maximum size of the desk is 2 * 1 metres. There is a maximum of 500 documents on the desk.

Input
The first line of the input consists of the number N of desks to be considered.
Next follows for each desk the number of documents. Then follows for each document a seperate line with 4 numbers: the smallest X and Y co-ordinates (i.e. the lower left corner), and the width and height of the document respectively.

Output
The output consists of N lines containing the amount of occupied space on each desk in squared tenths of millimetres.

Sample input
```
2
2
0 0 1000 1000
0 0 1000 1000
3
0 0 2 2
2 2 2 2
1 1 2 2
```

Output for the sample input
```
1000000
10
```

The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

Problem B

Plane Spotting

Craig is fond of planes. Making photographs of planes forms a major part of his daily life. Since he tries to stimulate his social life, and since it's quite a drive from his home to the airport, Craig tries to be very efficient by investigating what the optimal times are for his plane spotting.
Together with some friends he has collected statistics of the number of passing planes in consecutive periods of fifteen minutes (which for obvious reasons we shall call 'quarters'). In order to plan his trips as efficiently as possible, he is interested in the average number of planes over a certain time period. This way he will get the best return for the time invested. Furthermore, in order to plan his trips with his other activities, he wants to have a list of possible time periods to choose from. These time periods must be ordered such that the most preferable time period is at the top, followed by the next preferable time period, etc. etc. The following rules define which is the order between time periods:

1. A period has to consist of at least a certain number of quarters, since Craig will not drive three hours to be there for just one measly quarter.
2. A period P_1 is better than another period P_2 if:
   - the number of planes per quarter in P_1 is higher than in P_2;
   - the numbers are equal but P_1 is a longer period (more quarters);
   - the numbers are equal and they are equally long, but period P_1 ends earlier.

Now Craig is not a clever programmer, so he needs someone who will write the good stuff: that means you. So, given input consisting of the number of planes per quarter and the requested number of periods, you will calculate the requested list of optimal periods. If not enough time periods exist which meet requirement 1, you should give only the allowed time periods.

Input
The input starts with a line containing the number of runs N. Next follows two lines for each run. The first line contains three numbers: the number of quarters (1–300), the number of requested best periods (1–100) and the minimum number of quarters Craig wants to spend spotting planes (1–300). The second line contains one number per quarter, describing for each quarter the observed number of planes. The airport can handle a maximum of 200 planes per quarter.

Output
The output contains the following results for every run:
- A line containing the text "Result for run <N>:" where <N> is the index of the run.
- One line for every requested period: "<F>-<L>" where <F> is first quarter and <L> is the last quarter of the period. The numbering of quarters starts at 1. The output must be ordered such that the most preferable period is at the top.

Sample input

```
3
10 5 5
1 5 0 2 1 4 2 5 0 2
10 3 5
10 3 1 4 2 6 3 0 8 0
5 5 5
1 2 3 4 5
```

Output for the sample input

```
Result for run 1:
4-8
2-8
6-10
1-8
2-6
Result for run 2:
1-6
1-7
1-9
Result for run 3:
1-5
```

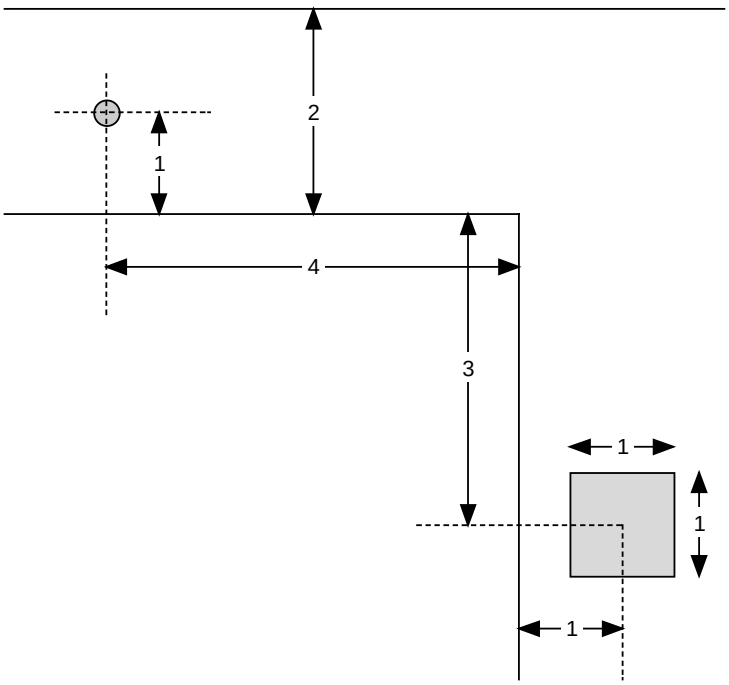
The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

Problem C

Kroket

It is well-known throughout the world that the Dutch wear wooden shoes. Less well-known is the fact that one of their favourite games is played with this peculiar footwear: the ancient game of Kroket. Kroket is a mixture of football and golf. The objective is to kick a wooden ball (this is where the wooden shoes come in handy) with a diameter of 10–20 cm from a one square meter kick-off patch into a hole in the ground. But to make things more interesting, the playing ground is L-shaped and lined by walls 50 cm high. The ball will only fall into the hole if the centre of the ball passes exactly over the centre of the hole.

See the diagram below for a layout of the playing field, together with the dimensions (in metres) of the playing field.



Note that there is no wall to the right of the kick-off patch, so it is not wise to kick ball that way. There is also no wall left of the hole. The only way to kick the ball into the hole is to let it bounce against the top and bottom walls a couple of times. But how many times will it bounce?

This is where you come in again. Write a program to calculate the minimum number of bounces required to get the ball into the hole, and the distance the ball travels in that case. You may assume that

the ball is perfectly round, rolls perfectly straight, and bounces perfectly off both walls with angles of entry and exit being exactly equal (known as Snell's Law, named after Snellius, a famous Dutchman!).

Input

The input consists of a line containing the number of configurations N ($1 \leq N \leq 10000$). It is followed by N lines, one for each configuration. This line consists of:

- The diameter of the ball in centimetres: a real number ranging from 10 to 20;
- The kick-off position of the centre of the ball relative to the lower left corner of the kick-off area, in metres: two real co-ordinates (horizontal and vertical) ranging from 0 to 1.

Output

The output consists of one line per configuration containing the minimum number of bounces (an integer number), one space, and the distance the ball travels (a real number in decimal notation with three decimal digits).

Sample input

```
15
10.0 1.0 0.5
10.0 0.9 0.5
10.0 0.8 0.5
10.0 0.7 0.5
10.0 0.6 0.5
10.0 0.5 0.5
12.0 0.5 0.5
14.0 0.5 0.5
16.0 0.5 0.5
18.0 0.5 0.5
20.0 0.5 0.5
20.0 0.4 0.5
20.0 0.3 0.5
20.0 0.2 0.5
20.0 0.0 0.5
```

Output for the sample input

```
4 12.838
5 14.540
5 14.503
6 16.254
6 16.223
7 18.008
7 17.874
7 17.739
7 17.605
7 17.471
8 19.067
9 20.786
10 22.518
11 24.260
17 34.891
```

The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

Problem D

Psychic Yahtzee

Yahtzee is a popular dice game, which can be played with any number of people. Each player has a score card such as the one below, and all players use the same set of 5 fair dice.

Ones	
Twos	
Threes	
Fours	
Fives	
Sixes	
3-Of-A-Kind	
4-Of-A-Kind	
Yahtzee	
Chance	
Total	

The object for each player is to get as many points on his scorecard as possible. In each turn, each player gets to throw with the 5 dice. After the first throw, the player can decide to pick up any set of dice on the table to throw again for the second time. After this second throw, the player can again decide to pick any set of dice on the table, and throw these for the third time. Note that the player may even throw a dice again that was left on the table after the first throw. If the player decides, after the first or second throw, not to throw any dice on the table again, this ends the turn for the player.

Anyway, after a maximum of three throws there are 5 dice on the table. Now comes the task of writing down this turn in the scorecard. The following rules must be followed:

- After every turn, exactly one empty line on the score card must be filled in.
- Once a line has been filled in, it cannot be replaced with another score in a later turn.
- This means that after exactly 10 turns the entire score card will be completely filled in.
- Every line on the score card has special rules governing when a score may be filled in on that line, and how to calculate the score for that line given the 5 final dice:
    - Lines ‘Ones’ till ‘Sixes’ can always be filled in; the score is the total of the dice which show the corresponding number (i.e. in line ‘Sixes’ you can fill in the sum of the 6’s thrown, in line ‘Fours’ the sum of the 4’s thrown).
    - Lines ‘3-Of-A-Kind’, ‘4-Of-A-Kind’ can only be filled in if at least 3 and 4 (respectively) of the same faces of the dice are on the table; the score is the total of **all** the dice on the table.
    - Line ‘Yahtzee’ can only be filled in if all the faces of the dice are the same; the score is always 50 (no matter what the number on the dice is).
    - Line ‘Chance’ may always be filled in; the score is the total of **all** the dice on the table.

- In addition to the rules given above it is also always allowed to fill in a score of 0 on a line. For example, if the only open line for a player is the 4-Of-A-Kind line and the player does not throw at least 4 the same dice, he must fill in 0 on that line.

Consider the following dice on the table at the end of the turn: 2 3 3 3 5. This turn can be filled in on line 'Twos' (score is 2), line 'Threes' (score is 9), line 'Fives' (score is 5), line '3-Of-A-Kind' (score is 16) and line 'Chance' (score is 16). For all other lines the score is 0.
Consider the following dice on the table at the end of the turn: 1 1 1 1 1. This turn can be filled in on line 'Ones' (score is 5), line '3-Of-A-Kind' (score is 5), line '4-Of-A-Kind' (score is 5), line 'Yahtzee' (score is 50) and on line 'Chance' (score is 5). For all other lines the score is 0.

Each player gets to take 10 turns, and afterwards the totals on everybody's scorecard are compared and the winner is decided (usually the one with the most number of points).

One of your friends likes to play Yahtzee very much. Lately he thinks that he is psychic: he is convinced that he can predict the dice he will throw in advance. He likes to cheat a bit on his friends by taking this foresight into account in his games of Yahtzee, but he does not know how. This is where you come in:
Your task is to write a program which, given the dice which will be thrown, determines the optimal score that can be reached with those dice.

The dice for a given game are described by 10 lines of 15 digits between 1 and 6 (inclusive). Each line represents one turn. The first 5 digits on the line give the first throw in a turn. The next 5 digits represent the outcome of the dice that can be thrown in the second throw. Only as many as are actually thrown are used (starting at the left-most), those remaining are then skipped. The last 5 digits on the line represent the outcome of the dice that can be thrown in the throw. Again, only as many as are actually thrown are used (again, starting at the left-most; numbers may not be skipped in favour of the following number).

For instance, if the input line is '1 1 3 4 5 1 3 1 4 6 1 6 6 1 3', a final configuration of '1 1 1 1 1' on the table after the third throw can be obtained as follows:
- After the first throw, keep the two 1's and throw the dice with 3,4 and 5 again.
- This second throw will result in the dice 1,3 and 1 appearing on the table too. Keep the two 1's again and throw the dice 3 again.
- This throw will result in the last 1 appearing on the table.

You can also obtain 3 sixes as follows:
- After the first throw, keep none of the dice and throw them all again.
- Of the second throw, keep the 6 and the 4. Both 1's and the 3 are thrown again.
- In the third throw the final configuration will become: '6 6 6 4 1'.

The output is the score table which the maximum score (total) that can be reached. The input is such, that the score table with the maximum score is uniquely defined.

Input

The input consists of a line containing the number of games N ($0 \leq N \leq 1000$). Following this line there are descriptions for N games. Each game is described in 10 lines, each with 15 digits (1,2,3,4,5 or 6).

Output

The output for each game is the score table leading to the maximum score possible for that game. A score table consists of 11 lines. Each line in the score table starts with the string identifying the line (see the example score table above), followed by a colon, a single space and the score filled in on this row. Before each colon as many spaces as needed are added to make sure that all the colons are neatly in the same column. There are no spaces between the colon and the longest identification strings (that's either '3-Of-A-Kind' or '4-Of-A-Kind'). Between two score cards there is an empty line, a line containing 12 dashes ('-') and an empty line. See also the example output given on the next page.

Sample input

```
2
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```

Output for the sample input

```
Ones       : 3
Twos       : 6
Threes     : 9
Fours      : 8
Fives      : 5
Sixes      : 0
3-Of-A-Kind: 12
4-Of-A-Kind: 0
Yahtzee    : 0
Chance     : 15
Total      : 58

------------

Ones       : 0
Twos       : 0
Threes     : 0
Fours      : 0
Fives      : 0
Sixes      : 30
3-Of-A-Kind: 30
4-Of-A-Kind: 30
Yahtzee    : 50
Chance     : 30
Total      : 170
```

The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

Problem E

Super Snooker

On one of my many interplanetary travels I landed on a beautiful little planet called Crucible. It was inhabited by an extremely peace-loving people who called themselves Snooks. They received me very gracefully and told me I had arrived at precisely the right time, as the biggest event of the year was just then taking place: the Super Snooker Championship. Of course I couldn't decline an invitation to go and watch.

That year the Super Snooker Championship was contested by two experienced Snooks universally acclaimed as the best players on the planet: Stephanie McHendry and Joanna McHiggins. The game involved an immense rectangular table covered with green cloth and lined by edges two inches high, except in the four corners and in the middle of the longer sides where there were holes. On it were put a number of balls (from 6 up to as many as 25), each representing a value or certain number of points (anywhere from 2 to 1000, but numbered consecutively). Each player in turn tried to nudge the lowest valued ball left on the table into one of the holes on the edges of the table using a strange limb called a "kew". If one succeeded, she was said to have "podded" the ball and the value of the podded ball was added to her score.

But here is the strange thing: the object of the game was not to finish with more points than the opponent. No, being a people who loved peace above all else, the object for both players was to end up with an equal number of points. This presented a bit of a problem. It was very important to them to know if it was possible to finish equal given the score of both players and the values of the balls left on the table. For instance, with a score-line of 56–34 and three balls left with values 13, 14 and 15, it is impossible to reach equal end-scores. If there are five balls left with values 20–24, it is possible: 56 + 20 + 24 = 34 + 21 + 22 + 23 = 100.

You are asked to write a program that helps the Snooks by calculating whether it is possible for two Super Snooker players to win their game by finishing equal, given a score-line and the range of values of the range of the remaining balls.

Input

The input consists of a line containing the number of configurations N ($0 \leq N \leq 10000$) to be calculated. It is followed by N lines each containing two scores and the lowest and highest values of the remaining balls.

Output

The output consists of one line for each configuration with the string '`possible`' or the string '`not possible`', depending on whether it is possible or not to finish equal from the given configuration.

Sample input

```
5
56 34 13 15
56 34 20 24
0 0 500 519
0 0 500 520
0 0 500 521
```

Output for the sample input

```
not possible
possible
possible
not possible
not possible
```

The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

Problem F

Inverso

The game of 'Inverso' is played on a 3x3 grid of coloured fields (each field is either black or white). Each field is numbered as follows:

1	2	3
4	5	6
7	8	9

The player can click on a field, which will result in the inversion of that field and all its direct neighbouring fields (i.e. the colour changes from black to white or vice-versa). Hence,

- Clicking field 1 inverts fields 1, 2, 4, 5
- Clicking field 2 inverts fields 1, 2, 3, 4, 5, 6
- Clicking field 3 inverts fields 2, 3, 5, 6
- Clicking 4 inverts fields 1, 2, 4, 5, 7 , 8
- Clicking 5 inverts fields all fields
- Clicking 6 inverts fields 2, 3, 5, 6, 8, 9
- Clicking 7 inverts fields 4, 5, 7, 8
- Clicking 8 inverts fields 4, 5, 6, 7, 8, 9
- Clicking 9 inverts fields 5, 6, 8, 9

The aim of the game is to find the shortest sequence of clicks to make all fields white from a given start colouring of the grid.

Input
The first line contains a number N (0≤N≤10000) of runs.
The following N lines each contain a string of nine letters '`b`' (black) or '`w`' (white) for the colour of the fields 1 to 9. This is the initial colouring of the grid.

Output
For each input string the shortest word in the letters '1', …, '9' (for clicking field one, …, nine) which makes all fields white. If there is more than one shortest word then the lexicographically smallest one must be printed ('`1234`' is smaller than '`1342`').

Sample input

```
3
bbwbwbwbw
bwwwbwbwb
bbbbwbbbw
```

Output for the sample input

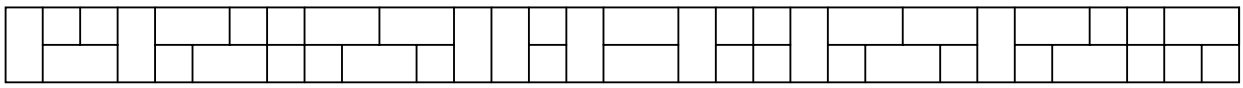
```
2459
267
356789
```

The ACM International Collegiate Programming Contest 1998–1999
North-western Europe Regional Contest
Sponsored by IBM

Problem G

Mondriaan

Squares and rectangles fascinated the famous Dutch painter Piet Mondriaan. One day, while working on his latest project, he was intrigued by the number of different ways in which he could order several objects to fill an arbitrary region. Expert as he was in this material, he saw at a glance that this was going to be too hard, for there seemed to be innumerable ways to do this. To make his task a little easier, he decided to start with only two kinds of objects: squares with width 1 and height 1, and rectangles with width 2 and height 1. After working on it for half an hour, he knew that even this was too much, for all of his paper was filled with pages like this. The only paper left was his toilet paper, and strange as it now seems, he continued with his task. Fortunately the width of the toilet paper equalled the width of the rectangle, which simplified things a lot. This seemed to do just fine, for in a few minutes time, he produced the following drawing:



Mondriaan decided to make several of these drawings, each on a piece of toilet paper with a different length. He wanted to give the drawings in his 'toilet series' names according to the last digit of the number of ways to fill a piece of toilet paper of a particular length with squares and rectangles. Computers might come in handy in cases like this, so your task is to calculate the name of the drawing, given the length of the toilet paper. The length will be measured in the same dimension as the squares and rectangles.

Input
The input consists of a line containing the number N ($1 \leq N \leq 100$) of drawings in the series. Each consecutive line consists of a number L ($1 \leq L \leq 1000000$) which is the length of the piece of toilet paper used for the drawing.

Output
The output consists of the number that is the name for the corresponding drawing.

Sample input
```
5
0
1
2
3
4
```

Output for the sample input

```
1
2
7
2
1
```

Problem H

Numbers & Letters

In the early 80's, a popular TV show on Dutch television was 'Cijfers en Letters' (Numbers and Letters). This game consisted of two game elements, in which the main goal was to outclass your opponent. Letters is a game in which you are given a number of letters with which you should form the longest Dutch word possible. Since Dutch is a very hard language to learn we will postpone implementation of this game element until after the contest.
For the second game element 'Numbers' 5 different numbers are chosen, together with a target number. The aim is to use some arithmetic on (some of) the five numbers to form the target number. Each number can be used only once. It might not be possible to form the target number given the input numbers, in that case the largest number smaller than the target number that can be calculated should be given. The only mathematical operations allowed are: +, -, *, /. All intermediate results should be integers, so division is not always allowed (e.g. (2*2)/4 is OK, but 2*(2/4) is not).
Examples:

- If the 5 numbers are 1, 2, 3, 7 and 100 and the target number is 573, the target number can be reached as follows: (((100-1)*2)-7)*3.
- If the 5 numbers are 3, 26, 78, 12 and 17, and the target number is 30, the target number can be reached as follows: (78*3)-(12*17).
- If the 5 numbers are 67, 69, 58, 22, 2, and the target number is 929, the target number cannot be reached, but the largest number smaller than the target number that can be reached is 923 = (22-(67-58))*(69+2).

Your assignment is to write a program that calculates the best approximation from below of the target number using arithmetic on the 5 given numbers. Note that if it is not possible to reach the exact number, you should give the largest reachable number below the target number.

Input
The first line contains the number of runs, N. The next N lines consist of six numbers separated by a space. The first 5 numbers Mi, $1 \leq Mi \leq 100$, are the numbers you can use to calculate the target number. The sixth number is the target number T, $1 \leq T \leq 1000$.

Output
The output consists of N rows, each containing the best approximation of the target number using the 5 given numbers.

Sample input

```
3
1 2 3 7 100 573
3 26 78 12 17 30
67 69 58 22 2 929
```

Output for the sample input

```
573
30
923
```