Paradox With Averages

One well-known joke goes as follows: If a bad Computer Science student drops out of college and goes to a different college to study Economics instead, he will increase the average intelligence on both colleges.

In this problem we will investigate the maths behind this joke.

Problem specification

Given the list of student IQs for each of the two colleges, find the number of students of Computer Science that can make the joke true – that is, compute the size of the set

$\{S \mid S$ is a Computer Science student, and if he went to study Economics, both average IQs would increase$\}$

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case looks as follows: The first line two positive integers N_{CS} and N_E – the number of students of Computer Science and Economics, respectively. The number of Computer Science students will be at least 2.

The following lines contain a total of $N_{CS} + N_E$ whitespace-separated positive integers giving the IQs of all the students. The first N_{CS} students mentioned in the input are Computer Science students, the remaining ones study Economics.

Output specification

For each test case output a single line with a single integer – the number of Computer Science students that would cause the funny event to happen.

Example

input

```
1

5 5
100 101 102 103 104
98 100 102 99 101
```

output

```
1
```

The average Computer Science IQ increases only if the leaving student is #1 or #2. Student #1 is too dumb to raise the average IQ for Economics, thus only student #2 remains and the answer is 1.

QuackQuack

Our most famous programming language QUACK is back to make this practice session an even more pleasant experience. For those who are new to IPSC (and didn't care to browse the archives before the contest) we will give an overview of the language.

A queue is a data structure that supports two operations: Put (insert an element into the queue) and Get (remove and return the element that has been in the queue for the longest time). Just like a queue in a shop – new customers are "inserted into the queue" at the end while the customer at the beginning of the queue is served and leaves.

We will define a simple programming language QUACK. QUACK uses a single queue to store (almost) all the data. At the start of the program, this queue is empty. The only data type in QUACK is **number**: an integer from the range 0-65535. All mathematical operations are computed modulo 65536. (For example, $65530 + 10 = 4$.) In addition to the queue, QUACK has 26 registers denoted by lowercase letters (i.e. $a - z$). Each register holds one **number**.

A program is a sequence of commands separated by whitespaces. The first character determines the command, optionally followed by parameters. The commands are defined below:

Arithmetic operations

operation	description
$+$	Addition: get x, get y, put $(x + y)$ mod 65536.
$-$	Subtraction: get x, get y, put $(x - y)$ mod 65536.
$*$	Multiplication: get x, get y, put $(x * y)$ mod 65536.
$/$	Integer division: get x, get y, put x div y. (y mustn't be zero)
%	modulo: get x, get y, put x mod y. (y mustn't be zero)

Operations with the queue and the registers

command	description
>[register]	Get into register: get x, set the value of [register] to x.
<[register]	Put from register: put the value of [register].
P	Print: get x, print x to the standard output, print a newline.
P[register]	Print register: print the value of [register] to the standard output, print a newline.
C	Print as char: get x, print the character with ASCII code x mod 256 to the standard output.
C[register]	Print register as char: print the character with ASCII code x mod 256 (where x is the value of [register]) to the standard output.

Jumps

command	description
:[label]	Label: this line of the program has the label [label].
J[label]	Jump (unconditional): the execution of the program continues on the line with the label [label].
Z[register][label]	Jump if zero: if the value of [register] is zero, the execution of the program continues on the line with the label [label].
E[register1][register2][label]	Jump if equal: if the values of [register1] and [register2] are equal, the execution of the program continues on the line with the label [label].
G[register1][register2][label]	Jump if greater: if the value of [register1] is greater than the value of [register2], the execution of the program continues on the line with the label [label].

Other

command	description
Q	Quit: terminate the execution of the program. The program also terminates if the instruction pointer reaches the end of the program.
[number]	Put a number: If the first letter of a command doesn't match any of the previous lines, try to convert the command to a number and put in in the queue.

Example

```
20
0
:start
>a
Zaend
<a
<a
1
+
-
>b
<b
Jstart
:end
P
```

The program on the left computes the sum $1 + 2 + \ldots + 20$.

Every time the execution of this program reaches the third line (the line with the label "start"), the queue contains exactly two numbers: the first is N, the second is the sum $(N+1)+\cdots+20$. The number N is loaded into the register a. If it is already zero, the program jumps to the line with the label "end", prints the computed sum, and quits. Otherwise, we want to add N to the partial sum and decrease it. After the eighth instruction (put 1) the queue contains the values: $PartialSum$, N, N, 1. The following two instructions compute $PartialSum + N$ and $N - 1$. Finally, we place them in the correct order and repeat the loop.

Interpreter

For your convenience we provide you with an interpreter of this programming language. You may either download it at `http://foja.dcs.fmph.uniba.sk/~misof/quack/interpreter.cc` and compile it (you need a C++ compiler and STL to compile it) or use the web form at `http://foja.dcs.fmph.uniba.sk/~misof/quack/` Please don't overuse the web form so that everyone gets a chance to use it.

Problem specification – Easy

Write a program to factorize in Quack.

The input is a **positive number** N (between 2 and 65 535, inclusive) which is placed in the queue before your program is executed. Your program should output several lines, each containing a single number. Each of these numbers must be a prime, the product of the numbers must be N, and the numbers have to be printed in non-decreasing order.

For example, for $N = 12$ your program is supposed to print three lines containing the numbers 2, 2, and 3, respectively.

Your program will be tested on various inputs. To be accepted it has to solve all of them correctly.

(Note that you may test your program by simply writing the input number as the first line of the code.)

Problem specification – Hard

Write a program that duplicates the contents of the queue.

The input is a sequence containing at least one and at most 400 **positive numbers**. This sequence is placed in the queue before your program is executed. Your program should **not** output anything. After your program terminates the queue must contain exactly two copies of the input sequence, stored one after another.

For example, for the input sequence 1 2 3 4 your program is supposed to transform the contents of the queue to 1 2 3 4 1 2 3 4.

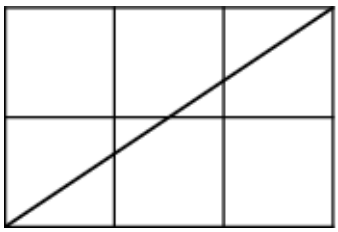
Limits

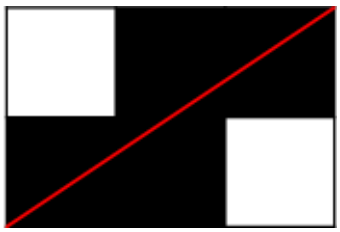
Execution of your program may not take more than **one hundred thousand** (100 000) steps to finish. If this limit is exceeded, the program will be terminated. In such case you will receive a message "Too many steps." and your submission will be rejected.

Rasterized Lines

Tomas is a computer graphics student. He has a homework which is very easy for him. He has to make a program that draws a line from point $(0, 0)$ to (a, b), where integers a, b ($a > 0$, $b > 0$) are the input of the program.

He uses the following algorithm. He divides the plane into squares 1×1 – these squares are pixels. When the line from $(0, 0)$ to (a, b) intersects a square **in more than one point**, the square (pixel) will be black. Otherwise it will be white. Look at the example:





Problem specification

Tomas did his homework in 30 minutes and now he is interested in a slightly different problem. Given an integer N, for how many different inputs does his algorithm produce exactly N black pixels?

More precisely, he is only interested in lines beginning in $(0, 0)$ and ending in (a, b), where both a and b are positive integers. Given N, find out how many of these lines will produce exactly N black pixels.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case looks as follows: The one and only line contains a positive integer N. You can assume that number N has at most 47 divisors.

Output specification

For each test case output one line with one integer – the number of lines that use exactly N black pixels.

Example

input

```
2

2

6
```

output

```
3
11
```

In the first test case, the three good lines are those ending in $(1, 2)$, $(2, 1)$, *and in* $(2, 2)$.