Code Jam Round 1C 2008

Text Messaging Outrage

The story

Professor Loony, a dear friend of mine, stormed into my office. His face was red and he looked very angry. The first thing that came out of his mouth was "Damn those phone manufacturers. I was trying to send a text message, and it took me more than ten minutes to type a one-line message." I tried to calm him down. "But what is wrong? Why did it take you so long?" He continued, "Don't you see?! Their placement of the letters is so messed up? Why is 's' the 4th letter on its key? and 'e'? Why is it not the first letter on its key? I have to press '7' FOUR times to type an 's'? This is lunacy!"

"Calm down, my friend," I said, "This scheme has been in use for so long, even before text messaging was invented. They had to keep it that way."

"That's not an excuse," his face growing redder and redder. "It is time to change all this. It was a stupid idea to start with. And while we are at it, how come they only put letters on 8 keys? Why not use all 12? And why do they have to be consecutive?"

"Umm... I... don't... know," I replied.

"Ok, that's it. Those people are clearly incompetent. I am sure someone can come up with a better scheme."

He was one of *those* people, I could see. People who complain about the problem, but never actually try to solve it.

In this problem, you are required to come up with the best letter placement of keys to minimize the number of key presses required to type a message. You will be given the number of keys, the maximum number of letters we can put on every key, the total number of letters in the alphabet, and the frequency of every letter in the message. Letters can be placed anywhere on the keys and in any order. Each letter can only appear on one key. Also, the alphabet can have more than 26 letters (it is not English).

For reference, the current phone keypad looks like this

```
key 2: abc
key 3: def
key 4: ghi
key 5: jkl
key 6: mno
key 7: pqrs
key 8: tuv
key 9: wxyz
```

The first press of a key types the first letter. Each subsequent press advances to the next letter. For example, to type the word "snow", you need to press "7" four times, followed by "6" twice, followed by "6" three times, followed by "9" once. The total number of key presses is 10.

Input

The first line in the input file contains the number of test cases **N**. This is followed by **N** cases. Each case consists of two lines. On the first line we have the maximum number of letters to place on a key (**P**), the number of keys available (**K**) and the number of letters in our alphabet (**L**) all separated by single spaces. The second line has **L** non-negative integers. Each number represents the frequency of a certain letter. The first number is how many times the first letter is used, the second number is how many times the second letter is used, and so on.

Output

For each case, you should output the following

```
Case #x: [minimum number of keypad presses]
```

indicating the number of keypad presses to type the message for the optimal layout.

Limits

P * K ≥ **L**
0 ≤ The frequency of each letter ≤ 1 000 000

Small dataset

1 ≤ **N** ≤ 10
1 ≤ **P** ≤ 10
1 ≤ **K** ≤ 12
1 ≤ **L** ≤ 100

Large dataset

1 ≤ **N** ≤ 100
1 ≤ **P** ≤ 1 000
1 ≤ **K** ≤ 1 000
1 ≤ **L** ≤ 1 000

Sample

Input

```
2
3 2 6
8 2 5 2 4 9
3 9 26
1 1 1 100 100 1 1 1 1 1 1 1 1 1 1 1 10 11 11 11 11 1 1 1 100
```

Output

```
Case #1: 47
Case #2: 397
```

Ugly Numbers

Problem

Once upon a time in a strange situation, people called a number *ugly* if it was divisible by any of the one-digit primes (2, 3, 5 or 7). Thus, 14 is ugly, but 13 is fine. 39 is ugly, but 121 is not. Note that 0 is ugly. Also note that negative numbers can also be ugly; -14 and -39 are examples of such numbers.

One day on your free time, you are gazing at a string of digits, something like:

```
123456
```

You are amused by how many possibilities there are if you are allowed to insert *plus* or *minus* signs between the digits. For example you can make

```
1 + 234 - 5 + 6 = 236
```

which is ugly. Or

```
123 + 4 - 56 = 71
```

which is not ugly.

It is easy to count the number of different ways you can play with the digits: Between each two adjacent digits you may choose put a plus sign, a minus sign, or nothing. Therefore, if you start with D digits there are 3^{D-1} expressions you can make.

Note that it is fine to have leading zeros for a number. If the string is "01023", then "01023", "0+1-02+3" and "01-023" are legal expressions.

Your task is simple: Among the 3^{D-1} expressions, count how many of them evaluate to an ugly number.

Input

The first line of the input file contains the number of cases, **N**. Each test case will be a single line containing a non-empty string of decimal digits.

Output

For each test case, you should output a line

```
Case #X: Y
```

where **X** is the case number, starting from 1, and **Y** is the number of expressions that evaluate to an ugly number.

Limits

0 <= **N** <= 100.

The string in each test case will be non-empty and will contain only characters '0' through '9'.

Small dataset

Each string is no more than 13 characters long.

Large dataset

Each string is no more than 40 characters long.

Sample

Input	Output
4 1 9 011 12345	Case #1: 0 Case #2: 1 Case #3: 6 Case #4: 64

Increasing Speed Limits

Problem

You were driving along a highway when you got caught by the road police for speeding. It turns out that they've been following you, and they were amazed by the fact that you were accelerating the whole time without using the brakes! And now you desperately need an excuse to explain that.

You've decided that it would be reasonable to say "all the speed limit signs I saw were in increasing order, that's why I've been accelerating". The police officer laughs in reply, and tells you all the signs that are placed along the segment of highway you drove, and says that's unlikely that you

were so lucky just to see some part of these signs that were in increasing order.

Now you need to estimate that likelihood, or, in other words, find out how many different subsequences of the given sequence are strictly increasing. The empty subsequence does not count since that would imply you didn't look at any speed limits signs at all!

For example, (1, 2, 5) is an increasing subsequence of (1, 4, 2, 3, 5, 5), and we count it twice because there are two ways to select (1, 2, 5) from the list.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. The first line of each case contains **n**, **m**, **X**, **Y** and **Z** each separated by a space. **n** will be the length of the sequence of speed limits. **m** will be the length of the generating array A. The next **m** lines will contain the **m** elements of A, one integer per line (from A[0] to A[**m**-1]).

Using A, **X**, **Y** and **Z**, the following pseudocode will *print* the speed limit sequence in order. mod indicates the remainder operation.

```
for i = 0 to n-1
  print A[i mod m]
  A[i mod m] = (X * A[i mod m] + Y * (i + 1)) mod Z
```

Note: The way that the input is generated has nothing to do with the intended solution and exists solely to keep the size of the input files low.

Output

For each test case you should output one line containing "Case #**T**: **S**" (quotes for clarity) where **T** is the number of the test case and **S** is the number of non-empty increasing subsequences mod 1 000 000 007.

Limits

$1 \leq \mathbf{N} \leq 20$
$1 \leq \mathbf{m} \leq 100$
$0 \leq \mathbf{X} \leq 10^9$
$0 \leq \mathbf{Y} \leq 10^9$
$1 \leq \mathbf{Z} \leq 10^9$
$0 \leq \mathbf{A[i]} < \mathbf{Z}$

Small dataset

$1 \leq \mathbf{m} \leq \mathbf{n} \leq 1000$

Large dataset

$1 \leq \mathbf{m} \leq \mathbf{n} \leq 500\,000$

Sample

Input	Output
2 5 5 0 0 5 1 2 1 2 3	Case #1: 15 Case #2: 13

```
6 2 2 1000000000 6
1
2
```
The sequence of speed limit signs for case 2 should be 1, 2, 0, 0, 0, 4.