Greetings From Globussoft

- Given below are 5 Programming questions, you have to solve any 3 out of 5 questions.

- These 5 questions you can attempt in any technology like C/C++, java, .Net, PHP

- To solve these 3 questions you've max. 3 hours.

- While Solving these questions you are not allowed to use any **Search Engine** like Google, Yahoo, Bing …

All the best for your test

Globussoft

QUESTION – 1

In the two-player game "Two Ends", an even number of cards is laid out in a row. On each card, face up, is written a positive integer. Players take turns removing a card from either end of the row and placing the card in their pile. The player whose cards add up to the highest number wins the game. Now one strategy is to simply pick the card at the end that is the largest — we'll call this the greedy strategy. However, this is not always optimal, as the following example shows: (The first player would win if she would first pick the 3 instead of the 4.)

3 2 10 4

You are to determine exactly how bad the greedy strategy is for different games when the second player uses it but the first player is free to use any strategy she wishes.

Input

There will be multiple test cases. Each test case will be contained on one line. Each line will start with an even integer n followed by n positive integers. A value of $n = 0$ indicates end of input. You may assume that n is no more than 1000. Furthermore, you may assume that the sum of the numbers in the list does not exceed 1,000,000.

Output

For each test case you should print one line of output of the form:

In game m, the greedy strategy might lose by as many as p points.

where m is the number of the game (starting at game 1) and p is the maximum possible difference between the first player's score and second player's score when the second player uses the greedy strategy. When employing the greedy strategy, always take the larger end. If there is a tie, remove the left end.

Example

Input:

```
4 3 2 10 4
8 1 2 3 4 5 6 7 8
8 2 2 1 5 3 8 7 3
0
```

Output:

```
In game 1, the greedy strategy might lose by as many as 7 points.
In game 2, the greedy strategy might lose by as many as 4 points.
In game 3, the greedy strategy might lose by as many as 5 points.
```

QUESTION – 2

Your task consists of evaluate a polynomial of degree **n** (**0** <= **n** <= **999**) represented by its **n+1** coefficients of the form:

$$p_n(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_2 x^2 + c_1 x + c_0$$

in each one of the **k** (**1** <= **k** <= **100**) points x_1, x_2, …, x_k. The coefficients of the polynomial and the values where they will be evaluated are integers in the interval **[-100, 100]** that guarantees that the polynomial's evaluation is at the most $2^{63} - 1$.

Input

There will be multiple test cases, each one with **4** lines that are described below
n: degree of polynomial.
$c_n\ c_{n-1} \ldots c_2\ c_1\ c_0$: coefficients of the polynomial separated by a single space.
k: number of points to evaluate the polynomial.
$x_1\ x_2 \ldots x_{k-1}\ x_k$: points to evaluate the polynomial separated by a single space.

The final test case is a single line where **n = -1** and this case should not be processed.

Output

For each test case you should print **k + 1** lines of output, the very first line containing the case number and the following **k** lines with the result of the polynomial's evaluation in each one of the **k** given points. See the sample.

Example

Input:

```
2
1 -2 -1
5
0 1 -1 2 -2
3
2 1 -2 -1
4
0 -1 2 -2
-1
```

Output:

```
Case 1:
-1
-2
2
-1
7
Case 2:
-1
```

```
0
15
-9
```

QUESTION – 3

Arithmetic expressions are usually written with the operators in between the two operands (which is called infix notation). For example, *(x+y)*(z-w)* is an arithmetic expression in infix notation. However, it is easier to write a program to evaluate an expression if the expression is written in postfix notation (also known as reverse polish notation). In postfix notation, an operator is written behind its two operands, which may be expressions themselves. For example, *x y + z w - ** is a postfix notation of the arithmetic expression given above. Note that in this case parentheses are not required.

To evaluate an expression written in postfix notation, an algorithm operating on a stack can be used. A stack is a data structure which supports two operations:

1. **push**: a number is inserted at the top of the stack.
2. **pop**: the number from the top of the stack is taken out.

During the evaluation, we process the expression from left to right. If we encounter a number, we push it onto the stack. If we encounter an operator, we pop the first two numbers from the stack, apply the operator on them, and push the result back onto the stack. More specifically, the following pseudocode shows how to handle the case when we encounter an operator O:

```
a := pop();
b := pop();
push(b O a);
```

The result of the expression will be left as the only number on the stack.

Now imagine that we use a queue instead of the stack. A queue also has a push and pop operation, but their meaning is different:

1. **push**: a number is inserted at the end of the queue.
2. **pop**: the number from the front of the queue is taken out of the queue.

Can you rewrite the given expression such that the result of the algorithm using the queue is the same as the result of the original expression evaluated using the algorithm with the stack?

Input Specification

The first line of the input contains a number **T** (*T* ≤ *200*). The following **T** lines each contain one expression in postfix notation. Arithmetic operators are represented by uppercase letters, numbers are represented by lowercase letters. You may assume that the length of each expression is less than *10000* characters.

Output Specification

For each given expression, print the expression with the equivalent result when using the algorithm with the queue instead of the stack. To make the solution unique, you are not allowed to assume that the operators are associative or commutative.

Sample Input

```
2
xyPzwIM
abcABdefgCDEF
```

Sample Output

```
wzyxIPM
gfCecbDdAaEBF
```

QUESTION – 4

Flavius Josephus and 40 fellow rebels were trapped by the Romans. His companions preferred suicide to surrender, so they decided to form a circle and to kill every third person and to proceed around the circle until no one was left. Josephus was not excited by the idea of killing himself, so he calculated the position to be the last man standing (and then he did not commit suicide since nobody could watch).

We will consider a variant of this "game" where every second person leaves. And of course there will be more than 41 persons, for we now have computers. You have to calculate the safe position. Be careful because we might apply your program to calculate the winner of this contest!

Input Specification

The input contains several test cases. Each specifies a number `n`, denoting the number of persons participating in the game. To make things more difficult, it always has the format `"xyez"` with the following semantics: when `n` is written down in decimal notation, its first digit is `x`, its second digit is `y`, and then follow `z` zeros. Whereas `0<=x,y<=9`, the number of zeros is `0<=z<=6`. You may assume that `n>0`. The last test case is followed by the string `00e0`.

Output Specification

For each test case generate a line containing the position of the person who survives. Assume that the participants have serial numbers from `1` to `n` and that the counting starts with person 1, i.e., the first person leaving is the one with number 2. For example, if there are 5 persons in the circle, counting proceeds as `2, 4, 1, 5` and person `3` is staying alive.

Sample Input

```
05e0
01e1
42e0
66e6
00e0
```

Sample Output

```
3
5
21
64891137
```

QUESTION – 5

Read the statement of problem E: Edge to understand how to fold a sheet of paper and how to interpret the input. We define a "stripe" to be a maximally large part of the sheet that has no folding line going through. Since the turns occur at equidistant places, all stripes are congruent.

In this problem you are given the description of the result of performing several folding steps as in problem E: Edge, i.e., in the unfolded state. Additionally, you know that the length of the sheet in its folded state is exactly the length of 1 stripe (again, we ignore thickness).

Find the minimum number of folding steps necessary to generate the described sheet from an initially flat sheet of paper. Note that performing a folding step may create more than one turn in the result because parts of the sheet already overlay due to previous folding steps. When a step is carried out, however, all overlaying parts of the sheet are affected, i.e., it is not allowed to fold, say, only the top three layers.

Finally, note that every result can be obtained by iterating through the turns in a fixed direction and performing a folding step at each turn, thereby accumulating a 1 stripe long stack of all stripes. If `n` is the number of turns in the input description, this procedure in fact requires `n` folding steps, which is not necessarily minimal as can be observed in the sample output.

Input Specification

The input contains several test cases, each on a separate line. Each line contains a nonempty string of characters `A` and `V` describing the longer edge of the sheet. You may assume that the length of the string is less than 200. The input file terminates immediately after the last test case.

Output Specification

For each test case print on a line the minimum number of folding steps required to produce the described sheet of paper.

Sample Input

```
V
AVV
AAVAAVVVAAV
```

Sample Output

```
1
2
4
```