Painting Squares (squares)

Mike is playing a game with Peter. There are n squares drawn on the ground in a single row, numbered 0 to $n-1$ from left to right. At the start of the game, Peter is allowed to paint each of these squares either **black** or **white**. He will then give Mike a single positive integer k ($1 \leq k \leq n$).

This game lasts a total of q rounds. In each round, Mike will randomly pick a square x ($0 \leq x < n$), and tell Peter the colours of the squares from positions x to $x+k-1$ inclusive. If any of these positions are out of range, Mike will inform Peter accordingly as well. Peter will then need to correctly deduce x based purely on this information alone.

Peter wishes to impress Mike, and thus wants to pick a value of k that is as low as possible. Help Peter devise a strategy to win this game with the minimum possible value of k.

Implementation details

You should implement the following procedures:

```
int[] paint(int n)
```

- n: number of squares.
- This procedure should return a single array of size $n+1$. The first n elements of the array will be the colours of the n squares. The i-th element of the array should be set to 1 if the i-th square is to be painted white, or 0 if it is to be painted black. The last element of the array will be the value of k.
- This procedure will be called exactly once for each scenario, before any calls to `find_location`.

```
int find_location(int n, int c[])
```

- n: number of squares.
- c: an array of size k. The i-th element of the array is set to 1 if the $(i+x)$-th square is painted white, or 0 if it is painted black. If the $(i+x)$-th square does not exist, the i-th element of the array will be set to -1.
- This procedure should return the deduced value of x.
- This procedure will be called exactly q times for each scenario, once for each round.

Each test case may involve multiple independent scenarios (i.e., different values of n). For a test case involving r scenarios, a program that calls the above procedures is run exactly two times, as

follows.

During the first run of the program:

- `paint` procedure is called r times,
- the returned colours and value of k are stored by the grading system, and
- `find_location` is not called.

During the second run of the program:

- `find_location` may be called multiple times,
- the value of n and colours given to each call to `find_location` are those produced by a call to `paint` for an **arbitrarily chosen** scenario from the first run,
- `paint` is not called.

Example

Consider the following call:

```
paint(5)
```

There are a total of 5 squares. Peter may choose to paint the squares black, black, white, black, white in that order, and decides that $k = 3$ would be sufficient for him to deduce the value of x. In that case, the procedure should return [0, 0, 1, 0, 1, 3].

Several calls would then be made to `find_location`.

Consider a possible call:

```
find_location(5, [0, 1, 0])
```

This means that the colour of the x-th, $(x + 1)$-th and $(x + 2)$-th squares are black, white and black respectively. Peter could deduce from this that $x = 1$. Therefore, the procedure should return 1.

Consider another possible call:

```
find_location(5, [1, 0, 1])
```

This means that the colour of the x-th, $(x + 1)$-th and $(x + 2)$-th squares are white, black and white respectively. Peter could deduce from this that $x = 2$. Therefore, the procedure should return 2.

Constraints

- $1 \leq r \leq 10$
- $2 \leq n, q \leq 1000$
- $-1 \leq c[i] \leq 1$ $(0 \leq i < k)$

Subtasks

1. (10 points) The value of k returned by `paint` can be no greater than 1000.
2. (15 points) The value of k returned by `paint` can be no greater than 100.
3. (20 points) The value of k returned by `paint` can be no greater than 70.
4. (55 points) The value of k returned by `paint` can be no greater than 30.

In subtask 4 you can obtain a partial score. Let m be the maximum value of k returned by `paint` across all scenarios. Your score for this subtask is calculated according to the following table:

Maximum value of k	Score
$m \geq 30$	0
$10 < m < 30$	$\frac{30-m}{20} \cdot 55$
$m \leq 10$	55

Sample grader

The sample grader reads the input in the following format:

- line 1: r

r blocks follow, each describing a single scenario. The format of each block is as follows:

- line 1: n q
- line $2 + i$ ($0 \leq i \leq q - 1$): the value of x for the i-th call to `find_location`.

The sample grader prints in the following format:

- line 1: m

r blocks corresponding to the consecutive scenarios in the input follow. The format of each block is as follows:

- line $1 + i$ ($0 \leq i \leq q - 1$): the deduced value of x returned by the i-th call to `find_location`.

Note that each run of the sample grader calls both `paint` and `find_location`.