Transmitter

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You have a simple transmitter device that can be used to send a sequence of bits over the network. The transmitter can be loaded with n bits $b_0, \ldots, b_{n-1}$ by the sender, where $0 \leq b_i \leq 1$ for each i. The receiver will receive all values $b_0, \ldots, b_{n-1}$ in the same order.

Sadly, the transmitter is broken. Some of the positions on the transmitter are malfunctioning and they will not send their value. Thus, the receiver will only receive the values of bits on positions that are not broken.

You immediately noticed that the device is broken, but you don't know which exact positions are. You can transmit any sequence of bits $b_0, \ldots, b_{n-1}$ to yourself. Then you will receive some values $c_0, \ldots, c_{m-1}$ from the broken transmitter.

You can transmit multiple such sequences and observe the results. Find all the malfunctioning positions using up to 10 transmissions.

Input

YOUR SUBMISSION MUST NOT READ FROM THE STANDARD INPUT, PRINT TO THE STANDARD OUTPUT OR INTERACT WITH ANY OTHER FILE.

Your task is to implement the following function: `int[] guess(int n)`

- n: the number of bits the transmitter can send.
- The function is called exactly one time for each test.
- The function has to return a list of all broken positions, **in increasing order**.

Your function can call the following function: `int[] transmit(int[] b)`

- b: A sequence of n bits $b_0, \ldots, b_{n-1}$ that you want to transmit.
- Each value b_i has to be either 0 or 1.
- The function returns some list of m numbers $c_0, \ldots, c_{m-1}$: values of all bits that were successfully transmitted, in the same order.

You can call the `transmit` function no more than 10 times in total for each test. If any of the above conditions are violated, your program will get a **Wrong Answer** verdict.

Scoring

- $2 \leq n \leq 1000$.
- There is at least one broken position.
- There is at least one not broken position.

In this task, the grader is **not adaptive**. It means that broken positions are fixed at the beginning of execution and do not depend on calls from your program.

This problem contains 4 subtasks.

Subtask	Additional Constraints	Points
0	Examples	0
1	$n \leq 10$	13
2	There is exactly one broken position.	21
3	There are no adjacent broken positions.	20
4	—	46

Note

Suppose the transmitter has a capacity of $N = 4$ bits and positions 1 and 3 are broken. The grader starts your program by calling `guess(4)`. Example of further interaction is below.

Call	Result	Comments
`transmit([0, 1, 1, 0])`	`[0, 1]`	Positions 1 and 3 are not transmitted.
`transmit([1, 1, 1, 1])`	`[1, 1]`	—
`transmit([0, 0, 1, 1])`	`[0, 1]`	—

After the interaction, you are confident that the broken positions are 1 and 3. So you can return a list `[1, 3]`. Please note that returning `[3, 1]` is **not correct** because the list has to be sorted.

The sample grader reads the input in the following format:

- Line 1: n and k: transmitter's bit capacity and the number of broken positions.
- Line 2: $p_0, \ldots, p_{k-1}$: all broken positions in increasing order ($0 \leq p_i \leq n-1$, $p_i < p_{i+1}$).

You can download an archive `transmitter.zip` in the system that contains sample test cases, solutions and graders for all supported languages (C++17, Java 8 and Python 3).

For solutions in Java, file and class name have to be named `transmitter.java` and `transmitter` respectively. Please note that both have to be lowercase. Also, if you want to call the `transmit` function, you should use `grader.transmit()`. You can refer to the sample Java code `transmitter.java` from the archive.

For solutions in Python, the `guess` function additionally accepts `transmit` function as a second argument. You can refer to the sample Python code `transmitter.py` from the archive.