Ancient Machine

The ancient Egyptian computer scientists were the first to explore the problem of random number generation. Based on their findings, they invented a machine that shuffles an array of integers of length n at random. However, due to the limitations of technology at the time, the machine operates in a very predictable manner. The machine has a built-in permutation p of numbers from 0 to $n-1$ that it uses to shuffle the array.

Thousand of years later, the modern archaeologists discovered the machine and examined it. They found that, because of corrosion, the machine now not only shuffles the numbers but also takes their XOR with some unknown number x. More formally, the machine takes an array a of length n and returns another array b such that $b[i] = a[p[i]] \oplus x$ (for all $0 \leq i \leq n-1$) where $\oplus$ denotes the bitwise XOR operator. Note that x is a fixed number, it does not change when you use the machine.

The archaeologists are intersted in the built-in permutation p. That's why they are asking for your help to find p in a limited number of machine uses.

Implementation details

You should implement the following procedure:

```
int[] guess_permutation(int n)
```

- n: the length of the permutation.
- This procedure is called exactly once per scenario.
- This procedure should return the built-in permutation p.

The above procedure can make calls to the following procedure:

```
int[] use_machine(int[] a)
```

- a: an array of length n.
- The elements of a must be integers from 0 to 32768.
- This procedure returns an array b such that $b[i] = a[p[i]] \oplus x$ (for all $0 \leq i \leq n-1$).
- This procedure can be called at most 129 times.

There is a total of T scenarios. For each scenario, the grader calls the `guess_permutation` procedure exactly once.

Examples

Example 1

Consider a scenario in which $p = [0, 1, 2, 3, 4]$ and $x = 3$. The procedure `guess_permutation` is called in the following way:

```
guess_permutation(5)
```

This procedure may call `use_machine([6, 6, 2, 9, 5])`, which returns $[5, 5, 1, 10, 6]$. It may then call `use_machine([1, 8, 4, 0, 4])`, which returns $[2, 11, 7, 3, 7]$. To guess the built-in permutation, this procedure should return $[0, 1, 2, 3, 4]$.

In this example, 2 calls are made to the `use_machine` procedure, and 9 is the maximum number sent in an array to the machine.

Example 2

Consider a scenario in which $p = [0, 4, 3, 1, 2]$ and $x = 8$. The procedure `guess_permutation` is called in the following way:

```
guess_permutation(5)
```

This procedure may call `use_machine([0, 5, 1, 1, 2])`, which returns $[8, 10, 9, 13, 9]$. To guess the built-in permutation, this procedure should return $[0, 4, 3, 1, 2]$.

In this example, only 1 call is made to the `use_machine` procedure, and 5 is the maximum number sent in the array to the machine.

Constraints

- $1 \leq T \leq 100$
- $3 \leq n \leq 128$
- $0 \leq x \leq 255$
- $0 \leq p[i] \leq n - 1$
- $p[i] \neq p[j]$ (for all $0 \leq i < j \leq n - 1$)

Subtasks

If in any of the test cases, the calls to the procedure `use_machine` do not conform to the rules mentioned above, or the return value of `guess_permutation` is incorrect, the score of your solution will

be 0. Otherwise, let Q be the maximum number of calls to the procedure `use_machine` and M be the maximum number among all the arrays you sent to the machine.

Subtask	Points	Condition	Additional Constraints
1	7	$Q \leq 129, M \leq 32678$	
2	12	$Q \leq 2, M \leq 32678$	
3	19	$Q \leq 1, M \leq 32678$	
4	21	$Q \leq 1, M \leq 2 \cdot n$	
5	10	$Q \leq 1, M \leq n + 2$	n is odd
6	31	$Q \leq 1, M \leq n + 2$	

Sample grader

The sample grader reads the input in the following format:

- line 1: T
- block i (for $0 \leq i \leq T - 1$): a block of 2 lines representing scenario i.
  - line 1: $n\ x$
  - line 2: $p[0]\ p[1]\ \ldots\ p[n-1]$

For each scenario, if you made an invalid query or guessed the permutation incorrectly, the grader will terminate printing a line containing `WA: msg` where `msg` is the cause of the termination. Otherwise, the grader will print a single line containing `OK: q m` where `q` is the number of calls made to `use_machine`, and `m` is the maximum number among all the arrays sent to the machine.