Coreputer

Coreputer, the brand new computing machine has N **core**s numbered from 0 to $N-1$. Recent maintenance revealed that some of the cores are malfunctioning. It is unknown which specific cores are malfunctioning, but there is *at least one* malfunctioning core.

To find the malfunctioning cores, Coreputer can run **diagnostic scan**s. In each scan, the user tags a (possibly empty) group of *distinct* cores $T[0], \ldots, T[l-1]$ for some $0 \leq l \leq N$. The rest of the cores are untagged. Coreputer then benchmarks the tagged cores and the untagged ones. Finally, it reports which of the two groups contains a greater number of malfunctioning cores, or that the two groups contain an equal number of malfunctioning cores. Note that an empty group contains 0 malfunctioning cores.

Your task is to find the malfunctioning cores by running diagnostic scans on Coreputer.

Implementation Details

You should implement the following procedure.

```
int[] malfunctioning_cores(int N)
```

- N: the number of cores.
- This procedure should return an array $c = [c[0], c[1], \ldots, c[N-1]]$, where for each i from 0 to $N-1$, inclusive, $c[i] = 1$ if core i is malfunctioning, and $c[i] = 0$ otherwise.
- This procedure is called exactly once for each test case.

The above procedure can make calls to the following procedure:

```
int run_diagnostic(int[] T)
```

- T: an array of distinct cores.
- This procedure returns
  - 1 if there are more tagged cores than untagged cores which are malfunctioning;
  - 0 if the number of tagged and untagged malfunctioning cores are equal;
  - -1 if there are fewer tagged cores than untagged cores which are malfunctioning.
- This procedure can be called at most 32 times in each test case.

The grader is **not adaptive**, meaning that the set of malfunctioning cores is fixed before a call to `malfunctioning_cores` is made.

Example

Consider a scenario when there are $N = 4$ cores, and only core 2 is malfunctioning.

Procedure `malfunctioning_cores` is called the following way:

```
malfunctioning_cores(4)
```

The procedure may call `run_diagnostic` as follows.

Call	Tagged cores	Untagged cores	Return value
`run_diagnostic([0])`	0	$1,2,3$	-1
`run_diagnostic([1, 2])`	$1, 2$	$0, 3$	1
`run_diagnostic([2])`	2	$0, 1, 3$	1

In the first call, there are no malfunctioning tagged cores, and one untagged core is malfunctioning, so the procedure returns -1.

After the third call returns 1, it is clear that at least one tagged core (that is, core 2) is malfunctioning. But then, the number of untagged malfunctioning cores must be zero, so we conclude that no other core is malfunctioning.

Therefore, the procedure `malfunctioning_cores` should return $[0, 0, 1, 0]$.

Constraints

- $2 \leq N \leq 16$

Subtasks

1. (20 points) $N = 2$
2. (40 points) The number of malfunctioning cores is even.
3. (40 points) No additional constraints.

If, in any of the test cases, the calls to the procedure `run_diagnostic` do not conform to the constraints described in Implementation Details, or the return value of `malfunctioning_cores` is incorrect, the score of your solution for that subtask will be 0.

In each subtask, you can obtain a partial score. Let q be the maximum number of calls to the procedure `run_diagnostic` among all test cases. Then, you will get a percentage of the subtask's score according to the following table:

Condition	Percentage
$24 < q \leq 32$	50%
$18 < q \leq 24$	75%
$q \leq 18$	100%

Sample Grader

The sample grader reads the input in the following format:

- line 1: N
- line 2: $M[0]\ M[1]\ \ldots\ M[N-1]$

where for each i from 0 to $N-1$, inclusive, $M[i] = 1$ if core i is malfunctioning, and $M[i] = 0$ otherwise.

Before calling `malfunctioning_cores`, the sample grader checks whether there is at least one malfunctioning core. If this condition is not met, it prints the message `No Malfunctioning Cores` and terminates.

If the sample grader detects a protocol violation, the output of the sample grader is `Protocol Violation: <MSG>`, where `<MSG>` is one of the error messages:

- `invalid array`: in a call to `run_diagnostic`, array T
  - has more than N elements, or
  - contains an element that is not an integer between 0 and $N-1$, inclusive, or
  - contains the same element at least twice.
- `too many calls`: the number of calls made to `run_diagnostic` exceeds 32.

Otherwise, let the elements of the array returned by `malfunctioning_cores` be $c[0], c[1], \ldots, c[n-1]$ for some nonnegative n. The output of the sample grader is in the following format:

- line 1: $c[0]\ c[1]\ \ldots\ c[n-1]$
- line 2: the number of calls to `run_diagnostic`