Operating Machine

This is an interactive task. There is a hidden permutation P of size N containing all integers in the range $[0, N)$, and an integer variable X which is initially 0.

There is a machine with N buttons. Your code will interact by pressing those buttons. When you press the i-th button, the machine will add $2^{P[i]}$ to X, and then return the **popcount** of X (the number of 1's in the binary representation of X).

Your task is to determine the hidden permutation P.

Note: The hidden permutation is fixed and does not change based on your queries.

Implementation Details

- Include the header file `machine.h`.
- Implement the following function: `vector < int > find_permutation(int N);` this function should return a vector of length N representing the hidden permutation P.
- You can use the following function: `int press_button(int position);` this function takes an integer `position` in the range $[0, N)$, representing the index of the button you are pressing. It returns a positive integer, which is the popcount of X after pressing the button.

Scoring Details

Let T be the number of calls to `press_button`:

- If $T \leq Q$, you will get all the points for that subtask.
- Otherwise, it is treated as Wrong Answer.

Constraints

Subtasks

Subtask	Points	Additional Constraints
1	8	$N = 2, Q = 400\,000$
2	15	$N \leq 10, Q = 400\,000$
3	32	$N \leq 500, Q = 400\,000$
4	25	$N \leq 3\,000, Q = 400\,000$
5	20	$N \leq 3\,000, Q = 250\,000$

Sample Grader

The sample grader reads the input in the following format:

Line 1: N
Line 2: $P[0]$, $P[1]$, ..., $P[N-1]$

Here, P is a permutation of size N, describing the hidden permutation your program needs to guess.

Before calling `find_permutation`, the sample grader will check whether the vector P is a permutation. If this condition is not met, it prints the message "Vector `P` is not a permutation" 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 following error messages:

- **Invalid guess**: in a call to `press_button`, the index is not valid, i.e., if you try to press button L, and L is less than 0 or greater than or equal to N (the size of the permutation).
- **Invalid size**: if the size of the answer vector is not equal to the size of permutation P.
- **Not a permutation**: if the answer vector is not a permutation of the numbers in the interval $[0, N)$.

Otherwise, let A be the permutation your code returns, then you receive:

- **Line 1**: **Wrong Answer**, if A is not equal to P, otherwise **Accepted**.
- **Line 2**: $A[0]$, $A[1]$, ..., $A[N-1]$
- **Line 3**: the number of calls to `press_button`.

Example

Consider a scenario where $N = 5$ and the hidden permutation is $P = [4, 2, 0, 1, 3]$. The function find_permutation is called in the following way:

```
find_permutation(5)
```

The function may make calls to `press_button` as follows:

Call	Value of X	Return value
`press_button(3)`	2	1
`press_button(2)`	3	2
`press_button(0)`	19	3
`press_button(2)`	20	2

The value of X during the interaction is unknown to us; we can only use the return value for each call, which is the **popcount** of X.

After determining the hidden permutation (after some, possibly zero, calls to `press_button`), you need to return the permutation.

In the previous example, you will only get points if you return $[4, 2, 0, 1, 3]$, depending on the number of calls to `press_button`.