Monster Game

A new video game is now on sale. In the world of this game, there are N monsters numbered from 0 to $N - 1$. Each monster has an integer called its **strength**. The strength of the monster i ($0 \le i \le N - 1$) is S_i. It is known that the strengths of the monsters satisfy the following conditions.

- The strength of each monster is an integer between 0 and $N - 1$, inclusive.
- No two different monsters have the same strength.

You can choose two monsters and make them fight each other. If the monster a and the monster b ($0 \le a \le N - 1,\ 0 \le b \le N - 1,\ a \ne b$) fight each other, the result of the fight is determined in the following way.

- If $|S_a - S_b| = 1$, the monster with the smaller strength wins.
- If $|S_a - S_b| > 1$, the monster with the larger strength wins.

Regardless of the result of the fight, you can make the same monster fight as many times as you want.

You do not know the strengths of the monsters in the beginning. You want to know the strength of every monster. For this purpose, you can make the monsters fight at most 25 000 times, and you know the results of the fights. Moreover, you want to minimize the number of fights.

Write a program which, given the number of the monsters, calculates the strength of every monster by making the monsters fight each other several times.

Implementation Details

You need to submit one file.

The file is `monster.cpp`. It should implement the following function. The program should include `monster.h` using the preprocessing directive `#include`.

- `std::vector<int> Solve(int N)`
  For each test case, this function is called exactly once.
  - The parameter `N` is the number of monsters N.
  - This function returns an array which describes the strength of every monster. In the following, let `T` be the array returned by this function.
  - The length of `T` should be N. If this condition is not satisfied, your program is judged as **Wrong Answer [1]**.
  - Each element of `T` should be between 0 and $N - 1$, inclusive. If this condition is not satisfied, your

program is judged as **Wrong Answer [2]**.
- ◦ For every i ($0 \le i \le N-1$), the equality T[i] $= S_i$ should hold. If this condition is not satisfied, your program is judged as **Wrong Answer [3]**.

Your program can call the following function.

★ `bool Query(int a, int b)`
Using this function, you make the monsters fight each other.
- ⋄ The parameters a and b are the indices a, b of the monsters that will fight each other.
- ⋄ This function returns the result of the fight. If the monster a wins, it returns `true`. If the monster b wins, it returns `false`.
- ⋄ The inequalities $0 \le a \le N-1$ and $0 \le b \le N-1$ should hold. If this condition is not satisfied, your program is judged as **Wrong Answer [4]**.
- ⋄ The condition $a \ne b$ should hold. If this condition is not satisfied, your program is judged as **Wrong Answer [5]**.
- ⋄ The function `Query` should not be called more than 25 000 times. If it is called more than 25 000 times, your program is judged as **Wrong Answer [6]**.

Important Notices

- Your program can implement other functions for internal use, or use global variables.
- Your program must not use the standard input and the standard output. Your program must not communicate with other files by any methods. However, your program may output debugging information to the standard error.

Compilation and Test Run

You can download an archive file from the contest webpage which contains the sample grader to test your program. The archive file also contains a sample source file of your program.

The sample grader is the file `grader.cpp`. In order to test your program, put `grader.cpp`, `monster.cpp`, `monster.h` in the same directory, and run the following command to compile your programs.

```
g++ -std=gnu++17 -O2 -o grader grader.cpp monster.cpp
```

When the compilation succeeds, the executable file `grader` is generated.

Note that the actual grader is different from the sample grader. The sample grader will be executed as a single process, which will read input data from the standard input and write the results to the standard output.

Input for the Sample Grader

The sample grader reads the following data from the standard input.

N
$S_0 \cdots S_{N-1}$

Output of the Sample Grader

When the program terminates successfully, the sample grader writes the following information to the standard output (quotes for clarity).

- If your program is judged as correct, it writes the number of calls to the function `Query` as “`Accepted: 100`”.
- If your program is judged as incorrect, it writes its type as “`Wrong Answer [1]`”.

If your program is judged as several types of Wrong Answer, the sample grader reports only one of them.

Notices for the Grader

For some of the test cases, the actual grader is adaptive. This means the actual grader does not have a fixed answer in the beginning, and it responds depending on the previous calls to the function `Query`. Here, it is guaranteed that there exists at least one answer which is consistent with all the responses.

Constraints

- $4 \leq N \leq 1\,000$.
- $0 \leq S_i \leq N - 1$ $(0 \leq i \leq N - 1)$.
- $S_i \neq S_j$ $(0 \leq i < j \leq N - 1)$.

Subtasks

1. (10 points) $N \leq 200$.
2. (15 points) The actual grader is not adaptive.
3. (75 points) No additional constraints. In this subtask, if your program answers all the test cases correctly,

your score is calculated as follows.

- Let X be the maximum of the number of calls to the function Query among all test cases in this subtask.
- Your score of this subtask is calculated as follows.
  - If $10\,000 < X \le 25\,000$, your score is $\left\lfloor 75 \times \frac{25\,000 - X}{15\,000} \right\rfloor$ points.
  - If $X \le 10\,000$, your score is 75 points.

Sample Communication

Here is a sample input for the sample grader and corresponding function calls.

Sample Input 1	Sample Function Calls			
	Call	Return	Call	Return
`5`	`Solve(5)`			
`3 1 4 2 0`			`Query(1, 0)`	`false`
			`Query(4, 0)`	`false`
			`Query(1, 3)`	`true`
		`[3, 1, 4, 2, 0]`		