The Collection Game (swaps)

Phew! You only narrowly escaped from prison after your disastrous debut as an art thief. A legal route into the art business seems to be a better idea after all. So, you decided to work as an art critic at the same museum where you were caught before.

This means that you visit the museum several times and produce an art review for each visit. An art review covers several pairs of rooms of the museum. For each pair, you compare the art of the two rooms during your visit and determine which room displays the art collection with the higher aesthetic value.* In the end, however, you also want to survey the art in the entire museum. That is, using all your reviews, you want to rank all rooms of the museum by decreasing aesthetic value of the art collections displayed.

Because planning is the key, you decide in advance for each visit which pairs of rooms you will compare. Also, for the sake of diversity, no room should appear more than once in a single art review.

Unfortunately, your newly gained reputation in the art community proves to be an obstacle. Whenever you announce that you will compare a particular pair of rooms during your next visit, the museum might spontaneously swap the art collections of these two rooms. Your final survey and room ranking is supposed to be based on what is displayed in each room during your last visit.

Write a program that schedules no more than V visits to the museum and, based on the resulting art reviews, computes a list of all rooms of the museum ordered by decreasing aesthetic value of the art collections displayed in each room at the time of your last visit.

Communication

This is a communication task. You must implement the function **void** *solve*(**int** N, **int** V) where N is the number of rooms of the museum, numbered 1 to N, and V is the maximal number of visits you are allowed to make. For each testcase, this function is called exactly once. In the function, you can use the following other functions provided by the grader:

- **void** *schedule*(**int** i, **int** j) schedules a comparison of rooms i and j during your next visit ($1 \leq i, j \leq N, i \neq j$). Immediately after a call to this function, the museum can decide to swap the art collections of rooms i and j.
- **vector⟨int⟩** *visit*() visits the museum and performs all scheduled comparisons. This function returns an array with exactly one entry for each scheduled comparison since your last visit to the museum (that is, for each call to *schedule* since the last call to *visit* or the beginning of the program). The entry at index k is 1 if the art in room i has a higher aesthetic value than in room j, and 0 otherwise. Here, i and j are the rooms from the $(k+1)$-th scheduled comparison.
- **void** *answer*(**vector⟨int⟩** r) publishes your list of all the rooms of the museum ordered by decreasing aesthetic value. r must be an array of length N; its i-th entry is the room with the $(i+1)$-th most aesthetic art collection during your last visit to the museum. You must call *answer* exactly once; your program will be automatically terminated afterwards.

If any of your function calls does not match the above format, if a room is passed more than once as a parameter to *schedule* between two calls to *visit*, or if you call *visit* more than V times, your program will be immediately terminated and judged as **Not correct** for the respective testcase. You must not write anything to standard output, otherwise you may receive the verdict **Security violation!**.

* Of course, any judgement of art can only be relative. That's why, after your visit, you will only know the relative order for each pair of rooms you compared, but no order between visited rooms that were in different pairs.

If you use C++, you must include the file `swaps.h` in your source code. To test your program locally, you can link your program with `sample_grader.cpp` which can be found in the attachment for this task in CMS (see below for a description of the sample grader). The attachment also contains a sample implementation with additional explanations as `swaps_sample.cpp`.

If you use Python, you can find a sample implementation as `swaps_sample.py` in the attachment which also describes the interface for Python submissions.

Constraints

We always have $1 \le N \le 500$ and $50 \le V \le 5\,000$.

Subtask 1 (5 points). $V = 5\,000$ and the museum never swaps art collections.

Subtask 2 (10 points). $V \ge 1\,000$ and the museum never swaps art collections.

Subtask 3 (5 points). $N \le 100$, $V = 5\,000$

Subtask 4 (15 points). $V = 5\,000$

Subtask 5 (15 points). $V \ge 500$

Subtask 6 (35 points). $V \ge 100$

Subtask 7 (15 points). $V \ge 50$

Moreover, the following holds: In each of the subtasks 3 to 7 you get 60% of the points awarded for the respective subtask if you solve all testcases in which the museum always puts the art collection with the higher aesthetic value into room i for each call to *schedule*. Inside CMS this is shown as “Group 1” of the corresponding subtask.

Sample Interaction

Consider a testcase with $N = 4$ and $V = 50$ where initially the rooms are ordered $1, 2, 3, 4$ by decreasing aesthetic value. At the beginning, the grader calls your function *solve* as *solve*$(4, 50)$. Then, one possible interaction between your program and the grader could look as follows:

Your program	Return value	Explanation
schedule$(1, 2)$		schedules to compare the art in rooms 1 and 2
schedule$(3, 4)$		schedules to compare the art in rooms 3 and 4
		the museum swaps the art of rooms 3 and 4
visit$()$	$\{1, 0\}$	visits the museum and performs all comparisons; the art in room 1 and 4 has a higher aesthetic value than in room 2 and 3 respectively
schedule$(2, 4)$		schedules to compare the art in rooms 2 and 4
visit$()$	$\{1\}$	visits the museum; the art in room 2 has a higher aesthetic value than in room 4
answer$(\{1, 2, 4, 3\})$		you are convinced that the rooms are ordered 1, 2, 4, 3 by decreasing aesthetic value
		the solution is correct and is accepted

Note that the above queries are of course not sufficient to determine the order of the rooms with certainty: for example, the order 2, 1, 4, 3 is also consistent with all answers to *visit*. This order is obtained when starting from $4, 1, 2, 3$ and swapping the art of rooms 2 and 4 after the last call to *schedule*.

Grader

The sample grader expects on standard input the numbers N and V as well as a list of N integers, the rooms ordered by decreasing aesthetic value at the beginning of *solve*. Then, the grader writes to standard output a protocol of all grader functions called by your program. Whenever *schedule*(i, j) is called, the grader expects on standard input the number 1 if the art collections of rooms i and j should be swapped at that point in time, or 0 otherwise. At the end, the grader writes one of the following messages to standard output:

Invalid input. The input to the grader via standard input was not of the above format.

Invalid schedule. The function *schedule* was called with invalid parameters.

Out of visits. The function *visit* was called more than V times.

Invalid answer. The function *answer* was called with invalid parameters.

Wrong answer. The function *answer* was called with a wrong list of rooms.

No answer. The function *solve* terminated without calling *answer*.

Correct: *v* visit(s) used. None of the above cases occurred and the function *visit* was called v times.

In contrast, the grader which is used for the evaluation of your submission will only output **Not correct** (for any of the above errors) or **Correct**. Both the sample grader and the grader used for evaluation will terminate your program automatically whenever one of the above errors occurs or if your program calls *answer*.

Limits

Time: 1.5 s
Memory: 512 MiB