Task 3: `Sorting`

Mr. Panda has managed to infiltrate into the NUS servers and plans to get information about the N questions in the upcoming NOI. Unfortunately, Mr. Panda's skills are not good enough to get the questions themselves. However, he is able to get information about the difficulty of each question. He is also able to change the order the questions will come out for the actual NOI. Currently, the questions are not sorted in order of difficulty. To make his life easier, he would like to sort the questions in increasing order of difficulty from easiest to hardest.

Implementation Details

In this task, you are asked to implement the following function: (It is different from the normal task, which requires the program to read from standard input and output to standard output.)

```
void sort_questions(int N, int Q, int A[])
```

This function will be called exactly once with two input parameters and one output parameter. The input parameters are N and Q, the number of questions in NOI and the maximum number of times you can call the function below. The output parameter is A. You are required to determine the order of difficulty of the questions and return them in array A where $A[0]$ is the easiest question, $A[1]$ is the second easiest question and so on until $A[N-1]$ is the hardest question. The questions are labeled from 1 to N so A should contain each number from 1 to N exactly once.

You are allowed to call the following grader function to complete the task:

```
bool compare_difficulty(int X, int Y)
```

This function will return true if task X is easier than task Y and false otherwise. No two tasks will be of the same difficulty. If you call this function more than Q times or X or Y is not a valid problem number, the program will terminate immediately and you will be given a *Wrong Answer* verdict.

Consider an NOI with 3 problems where problem, 1, 2, 3 are rated with difficulty 3, 1, 6 respectively. A possible interaction between your program *sort_questions* and the grader function *compare_difficulty* could be as follows:

- `compare_difficulty(1,2)=false`

  Problem 1 is harder than problem 2

- `compare_difficulty(2,3)=true`

  Problem 2 is easier than problem 3

- `compare_difficulty(1,3)=true`

  Problem 1 is easier than problem 3

At this point, the contestant's *sort_questions* function decides it has enough information to deduce the order of the difficulty of the questions. It sets $A = [2, 1, 3]$ and then terminates. This would yield a correct answer and the contestant would pass that test case.

Subtasks

The maximum execution time on each instance is 1.0s.

Your program will be tested on sets of input instances that satisfy the following restrictions:

Subtask	Marks	Q	N
1	23	1	$N = 2$
2	33	500000	$2 \leq N \leq 1000$
3	44	12000	$2 \leq N \leq 1000$

Testing

You may download the grader file, *grader.cpp*, the header file, *sorting.h*, a solution template, *sorting.cpp*, and the compilation command, *compile.sh* under *Attachments*. Edit the solution template *sorting.cpp* directly when implementing your solution, and run *compile.sh* to compile the executable. A small and large test case for each subtask are also provided for your reference. The grader input format is provided below.

Grader Input Format

- one line with a two integers N and Q, the number of questions and the maximum number of times the function can be called
- N lines with one signed 32-bit integer each where the integer on the i-th line represents the difficulty of the i-th problem