Art Collections (art)

While your days as an art thief are long past, this does not mean that you lost interest in contemporary art. Unfortunately, you've been pretty busy lately with BOI preparations. That's why you have lost track of how the N hottest contemporary art collections (conveniently numbered from 1 to N) rank according to value. Since simply asking someone would be quite embarrassing, you will have to resort to different means: *anonymous online rankings*.

That is, you will repeatedly do the following: You first guess a ranking of the N art collections (based on their value, most expensive first), then publish this ranking on some website, and finally wait for the collection owners' complaints in the comments section. As you don't want to read each individual comment, you will only keep track of the total number of complaints you receive. Fortunately, the owners' behaviour is very reliable: Each of them will complain exactly once for each collection that ranks higher than their own in your guessed ranking although it doesn't in the true ranking, but none will complain about collections you erroneously guessed to rank lower than theirs. You can assume that the values of all collections are distinct.

"Readers also liked: 13 SHOCKING applications of Dijkstra's algorithm computer scientists don't want you to know about!"

However, as publishing a ranking puts your anonymity at risk,* you only want to publish up to 4 000 guessed rankings before finding the correct ranking of the collections. Write a program that helps you to decide which rankings to publish!

Communication

This is a communication task. You must implement the function **void** *solve*(**int** N) where N is as described above. For each testcase, this function is called exactly once. Inside *solve*, you can use the following other functions provided by the grader:

- **int** *publish*(**std::vector**⟨**int**⟩ R) publishes a ranking R of the collections on the website. R must be a permutation of the numbers 1 to N with the collections you guess to be more expensive first. The function returns the number of complaints you receive after publishing this ranking. You can call this function at most 4 000 times per testcase.
- **void** *answer*(**std::vector**⟨**int**⟩ R) states that you have found the correct ranking R of the collections, in the same format as for *publish*. You must call *answer* exactly once; your program will be automatically terminated afterwards.

If any of your function calls does not satisfy the above constraints, your program will be immediately terminated and judged as **Not correct** for the respective testcase. You must not write to standard output or read from standard input; otherwise you may receive the verdict **Security violation!**.

You must include the file `art.h` in your source code. To test your program locally, you can link it 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, and see `sample_grader.cpp` for instructions on how to run it with your program). The attachment also contains a sample implementation as `art_sample.cpp`.

* Definitely because of your distinctive writing style and not because you have a tendency to accidentally sign them with your name.

Constraints

We always have $2 \le N \le 4\,000$.

Subtask 1 (5 points). $N \le 6$

Subtask 2 (15 points). $N \le 40$

Subtask 3 (15 points). $N \le 250$

Subtask 4 (15 points). $N \le 444$

Subtask 5 (20 points). $N \le 2\,000$

Subtask 6 (30 points). No further constraints.

Example Interaction

Consider a testcase with $N = 3$ where collection 1 is the most expensive, followed by 3, and then collection 2 being the least expensive. First, the grader calls your function *solve* as *solve*(3). Then, a possible interaction between your program and the grader could look as follows:

Your program	Return value	Explanation
publish({1, 2, 3})	1	you get a single complaint from the owner of collection 3
publish({2, 3, 1})	3	you get two complaints from the owner of collection 1 and one from the owner of collection 3
answer({1, 3, 2})	—	you are convinced that you have found the correct ranking your solution is correct and is accepted

Grader

The sample grader first expects on standard input two lines. The first line should contain the integer N. The second line should contain a list of N space-separated integers, the correct ranking of the collections in the same format as for *publish* and *answer*. Then, the grader calls *solve*(N) and writes to standard output a protocol of all grader functions called by your program. Upon termination, it 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 published ranking. You called *publish* with invalid parameters.

Too many published rankings. You called *publish* more than 4 000 times.

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

Wrong answer. You called *answer* with an incorrect ranking.

Correct: *p* published ranking(s). You called *answer* with the correct ranking and there were p calls to *publish*.

In contrast, the grader actually used to judge your program will only output **Not correct** (for any of the above errors) or **Correct**. Both the sample grader and the grader used to judge your program will terminate your program automatically whenever one of the above errors occurs or after your program calls *answer*.

Limits

Time: 3 s
Memory: 512 MiB

Event Hopping (events)

What a strange coincidence! After having determined the most valuable contemporary art collection, you noticed that it is apparently located somewhere near Lübeck. Since you don't know its exact location, you want to gather more information. Luckily, on the day you arrive for this year's BOI, the local art community hosts N events about contemporary art. This seems to be just the opportunity you were waiting for.

To plan your visit of these events, you numbered them from 1 to N with the i-th event starting at time S_i and ending at time E_i. You want to start your visit by attending some event s and finish your visit at some event e. As long as you are not attending event e, you will always attend your current event until the end* and then immediately switch to a different event that is currently running. This means that you can switch from event i to event j if and only if $S_j \leq E_i \leq E_j$.

Obviously, switching events too frequently would make you look suspicious. Thus, you want to determine the minimum number of event switches necessary if you start at event s and finish at e. Moreover, since you do not yet know when you will arrive in Lübeck and when you will have to leave for the BOI registration in the evening, you want to determine this for Q different pairs of starting and ending events s and e.

Input

The first line of input contains two integers, the number of events N and the number of pairs of events Q for which you want to determine the minimum number of event switches.

Then, N lines follow describing the events. The i-th of these lines contains two integers S_i and E_i, the starting and ending time of event i.

Then, Q lines follow describing the queries. The i-th of these lines contains two integers s_i and e_i, meaning that you want to determine the minimum number of event switches necessary if you want to start at event s_i and end your visit at event e_i.

Output

Your program should output Q lines. The i-th of these lines should consist of an integer, the minimum number of event switches necessary if you start at event s_i and end your visit at event e_i, or the string `impossible` if there is no way to achieve this.

Constraints

We always have $1 \leq N, Q \leq 100\,000$, $1 \leq S_i < E_i \leq 10^9$, and $1 \leq s_i, e_i \leq N$.

Subtask 1 (10 points). For every event, you can switch to at most one other event.

Subtask 2 (10 points). $N \leq 1\,000$ and $Q \leq 100$

Subtask 3 (15 points). $N \leq 5\,000$

* It would be rude to leave earlier—though nobody will complain about you being late as you are obviously an important and busy art critic.

Subtask 4 (15 points). $Q \leq 100$

Subtask 5 (20 points). No event is completely contained in another event, i.e. there are no two events $i \neq j$ with $S_i \leq S_j < E_j \leq E_i$.

Subtask 6 (30 points). No further constraints.

Examples

Input	Output
5 2 1 3 2 4 4 7 7 9 3 7 1 4 3 2	2 impossible
8 5 1 2 3 4 1 5 6 7 5 10 10 20 15 20 999999999 1000000000 1 6 1 7 2 4 3 3 5 8	3 4 impossible 0 impossible

In the first example, it is possible to start at event 1 and end at event 4 by switching from event 1 to event 5 and then to event 4, resulting in two event switches. However, there is no way to start at event 3 and end at event 2 because event 2 ends before event 3.

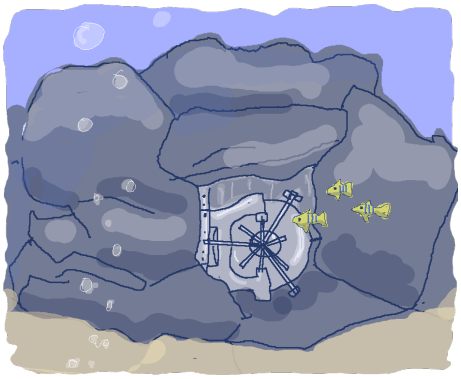
Limits

Time: 1 s
Memory: 512 MiB

Uplifting Excursion (vault)

Those events you attended on your arrival day were an exciting opportunity to become familiar again with the state of the (contemporary) art. And even more: The rumors you heard there revealed that the art collection you are interested in is stored in a secret underwater vault in the nearby Baltic Sea, owned by an old Lübeck grain merchant family! In memory of your past as an art thief, you decided to plan to break into this vault as a relaxing afternoon activity.*

To break into the vault, you want to use your newly acquired submarine. Unfortunately, your submarine will need a very specific amount of uplift L when you try to escape from the crime scene. After all, you don't want your submarine to crash into the bottom of the sea or float on the water surface where the police could catch you easily!



There surely is some pun about phishing hidden here, but to be honest we're out of our depth

In order to plan your break-in accordingly, you need to know about the uplift of the art pieces in the vault. Skilled as you are, you were able to obtain relevant information.† For every possible uplift ℓ you now know how many art pieces A_ℓ with that uplift are stored in the vault.

Write a program that uses this information to either calculate the maximum number of art pieces you can steal such that their total uplift (obtained by summing the individual uplift of every stolen art piece) is exactly L or to decide that this is impossible.

Input

The first line of input contains two integers M and L, meaning that the uplift of every art piece in the vault is between $-M$ and M inclusive and that the total required uplift is L.

The next line contains $2M + 1$ integers $A_{-M}, \ldots, A_M$ where A_ℓ describes the number of art pieces with uplift ℓ in the vault.

Output

Your program should output a single line. This line should consist of an integer, the maximum number of art pieces you can steal such that their total uplift is exactly L, or the string `impossible` if there is no way to achieve this.

Constraints

We always have $1 \le M \le 300$, $-10^{18} \le L \le 10^{18}$, and $0 \le A_\ell \le 10^{12}$.

Subtask 1 (5 points). $M, A_\ell \le 50$

Subtask 2 (15 points). $M, A_\ell \le 100$

Subtask 3 (20 points). $M \le 30$

* A purely hypothetical heist, of course.

† It's not your fault that their security system uses a weak hash algorithm, is it?

Subtask 4 (20 points). $M \leq 50$

Subtask 5 (20 points). $M \leq 100$

Subtask 6 (20 points). No further constraints.

Moreover, the following holds: In each of the subtasks 3 to 6 you get 50% of the points awarded for the respective subtask if you solve all testcases in which $A_\ell = 0$ for all $\ell < 0$. Inside CMS, this is shown as "Group 1" of the corresponding subtask.

Examples

Input	Output
2 5 2 3 1 1 4	9
3 5 3 1 0 2 0 0 2	impossible
1 5 0 0 6	5

In the first example, you can steal one art piece each with uplift –2, 0 and 1 respectively, two art pieces with uplift –1, and four art pieces with uplift 2. This results in a total of $1 + 1 + 1 + 2 + 4 = 9$ stolen art pieces with a total uplift of $1 \cdot (-2) + 1 \cdot 0 + 1 \cdot 1 + 2 \cdot (-1) + 4 \cdot 2 = 5$ as required.

In the second example, it is impossible to steal art pieces such that their total uplift is 5.

Limits

Time: 4 s
Memory: 512 MiB