PROBLEM 1 – ICC

Astro the astronaut just started his internship on the International Space Station (ISS). His job is to study the Terran civilization from the planet Mar Sara. The Terrans already built `N` cities, numered from `1` to `N`, but they did not build any roads between them.

Just as Astro started his internship, the Terrans began building roads on Mar Sara. It's known that the Terrans want to connect the cities as fast as possible, so they won't build a road between 2 cities that are already connected by a path composed of one or more roads. Each time the Terrans build a new road, Astro wants to know what cities it connects before the next road is built.

Luckily for Astro, he has access to SETI Masterpieces, a new satellite that can look at Mar Sara and see what roads have been built. Given two disjoint sets of cities, SETI is able to tell whether there is at least one **direct** road that connects a city from the first set with another city from the second set.

The way the interaction works is the following:

1. The Terrans build a road.
2. Astro uses SETI to try to discover the newly built road.
3. He goes to the ISS Scientific Committee and shows them his answer for the newly created road.
4. If there are less than `N - 1` roads build, repeat from step 1.

Your job is to write a program that will solve Astro's problem.

Interaction

You should implement a function `run()` which will be called once, in the beginning of the program. From this function, you should call `query()` every time you use the SETI Masterpieces. Every time you determine the newest built road, you should call `setRoad()`. You can call `query()` multiple times before calling `setRoad()`.

Grader function: `query()`

- C/C++: `int query(int size_a, int size_b, int a[], int b[]);`
- Pascal: `function query(size_a, size_b : longint; a, b : array of longint) : longint;`

Call this function to make a query to the SETI Masterpieces. Arguments `size_a` and `size_b` should represent the sizes of the two sets that are queried. Arrays `a[]` and `b[]` should contain the indices of the cities from the first and second set, respectively. Note that these sets should be disjoint! The function returns `1` if there is at least one direct road between a city from the first set and a city from the second set, and `0` otherwise.

Grader procedure: `setRoad()`

- C/C++: `void setRoad(int a, int b);`
- Pascal: `procedure setRoad(a : longint, b : longint);`

Call this procedure to indicate that you discovered that the newest road connects cities `a` and `b`. If the road is incorrectly discovered, you will receive 0 points for the current test case and the program terminates. If this is the `(N-1)`th call to this procedure, the program terminates and returns the score for the current test case. Otherwise, a new road is built by the Terrans and the interaction continues. Any other subsequent call to `query()` will take into account the road built after the call to `setRoad()`.

Your procedure: `run()`

- C/C++: `void run(int N);`
- Pascal: `procedure run(N : longint);`

Your submission must implement this procedure. You must alternately call `query()` and `setRoad()` from this procedure. The received argument `N` represents the number of cities built by the Terrans. If this procedure finishes its execution before guessing all the roads, you will not receive credit for that test case.

Language notes

- C/C++: you must `#include "icc.h"`
- Pascal: you must define the `unit icc`, and you must also import the grader routines with the statement `uses graderhelplib`

Scoring

The test cases for this task are grouped into 6 batches. The tests belonging to the same batch will have the same value for `N`. You will get the score for a batch, if, for each test from the batch, you guess all the roads correctly using at most `M` calls to the function `query()`. The values for `N` and `M`, as well as the score for each batch, are listed in the table below.

Batch number	N = Number of cities	M = Maximum calls to *query()*	Score
1	15	1500	7
2	50	2500	11
3	100	2250	22
4	100	2000	21
5	100	1775	29
6	100	1625	10

Sample execution

Contestant action	Grader action	Note
-	run(4)	- The grader starts a run for N = 4 cities. The first road is built between cities 2 and 4, unknown to the contestant.
query(1, 3, {1}, {2, 3, 4})	return 0	- Query for sets {1} and {2, 3, 4}. The answer is "false": there is no road from 1 to any of the other cities.
query(1, 2, {2}, {3, 4})	return 1	- Query for sets {2} and {3, 4}. The answer is "true": there is a road from city 2 to city 4.
query(1, 1, {2}, {3})	return 0	
setRoad(2, 4)	-	- The contestant guesses correctly road (2, 4). The grader generates a new road between 1 and 3.
query(2, 2, {2, 4}, {1, 3})	return 0	- Query for sets {2, 4} and {1, 3}. The answer is "false".

`setRoad(1, 3)`	-	- The contestant guesses correctly road (1, 3). The grader generates a new road between 1 and 4.
`query(2, 2, {2, 4}, {1, 3})`	return 1	- Same query as the last one. The answer is now "true", because of the newly generated road.
`query(1, 2, {2}, {1, 3})`	return 0	
`query(1, 1, {4}, {3})`	return 0	
`setRoad(4, 1)`	exit	- The contestant guesses correctly the last road (4, 1). The grader accepts the last guess and gives full score for the test.

PROBLEM 2 – Kangaroo

A garden is composed of a row of `N` cells numbered from `1` to `N`. Initially, all cells contain plants. A kangaroo arrived in the garden in cell numbered `cs`. Then he jumps from cell to cell, eating the plants as he goes. He will always finish in cell numbered `cf`, after visiting each of the `N` cells exactly once, including `cs` and `cf`. Obviously, the kangaroo will make `N-1` jumps.

The kangaroo doesn't want to be caught, so after each jump he changes the direction in which he jumps next: if he is currently in cell numbered `current` after he jumped there from a cell numbered `prev`, and will jump from `current` to cell numbered `next`, then:

- if `prev < current`, then `next < current`; else,
- if `current < prev`, then `current < next`.

Knowing the number `N` of cells in the garden, the starting cell `cs` from where the kangaroo starts to eat plants and the final cell `cf` where the kangaroo finishes, you should calculate the number of distinct routes the kangaroo can take while jumping through the garden.

Input format

The input file `kangaroo.in` will contain three space separated positive integers `N`, `cs`, `cf`.

Output format

In the output file `kangaroo.out` you should write a single integer, the number of distinct routes the kangaroo can take modulo `1000000007` ($10^9 + 7$).

Notes and constraints

- $2 \le N \le 2000$
- $1 \le cs \le N$
- $1 \le cf \le N$
- $cs \ne cf$
- For tests worth `6` points, $N \le 8$.
- For tests worth `36` points, $N \le 40$.
- For tests worth `51` points, $N \le 200$.
- Any route is uniquely determined by the order in which cells are visited.
- We guarantee that for each test there is at least one route which follow the rules.
- The kangaroo can start jumping in any direction from `cs`.

Example

kangaroo.in	kangaroo.out	Note
4 2 3	2	The kangaroo starts from cell 2 and finishes in cell 3. The two correct routes are 2 -> 1 -> 4 -> 3 and 2 -> 4 -> 1 -> 3

PROBLEM 3 – Trick

A group of tourists that was visiting Bran castle has been captured by Count Dracula! Among the tourists there is a magician and he made the following deal with the Count: he will perform an exquisite magic trick, and if he succeeds, Dracula will free all the tourists.

To perform the trick, the magician needs two assistants. After the trick begins, he is no longer allowed to communicate with the assistants, and neither the assistants with each other. The magician will give the Count a deck of $2N + 1$ cards on which there are written each of the numbers from 0 to $2N$. The Count will pick a card which he will keep hidden. Then, from the remaining $2N$ cards, the Count will choose N cards which will be given to the first assistant. The remaining N cards will be given to the second assistant. Then, each assistant will choose two cards from the received ones and will show them, in order, to the magician (but not to the other assistant). Knowing only those four cards, the magician will guess Dracula's hidden card!

Help the magician and don't let the tourists become the vampire's dinner!

Your program will be run three times for each test. On the first run, it will play the role of the first assistant. On the second run, it will play the role of the second assistant. On the third run, it will play the role of the magician.

Input format

The first line of the input file `trick.in` will contain an integer T representing the number of test cases (the number of times the trick will be performed for this test). The second line will contain an integer R belonging to the set $\{1, 2, 3\}$, representing the role your program will play for all the test cases in this file.

If $R = 1$, then your program will play the role of the first assistant. If $R = 2$, then your program will play the role of the second assistant. In these two cases, line number $2i + 1$ $(1 \le i \le T)$ will contain an integer N_i, meaning that, for the i-th test case, the deck will contain $2N_i + 1$ cards. Line number $2i + 2$ $(1 \le i \le T)$ will contain N_i integers representing the cards received by the current assistant in the i-th test case.

If $R = 3$, then your program will play the role of the magician. Line number $2i + 1$ $(1 \le i \le T)$ will contain an integer N_i with the same meaning as in the previous roles. Line number $2i + 2$ $(1 \le i \le T)$ will contain four integers representing the two cards printed by your program for the i-th test case in the run with $R = 1$ followed by the two cards printed by your program for the i-th test case in the run with $R = 2$. The cards in both pairs will be given in the same order as they were printed by your program in the corresponding run.

Output format

If $R = 1$ or $R = 2$, then your program must print on the i-th line $(1 \le i \le T)$ of the output file `trick.out` two integers separated by a single space representing the two cards you will show the magician in the i-th test case. These two numbers must be distinct and must exist among the N_i cards given in the input file.

If $R = 3$, then your program must print on the i-th line $(1 \le i \le T)$ a single integer representing Dracula's hidden card for the i-th test case.

Constraints

- $6 \le N_i \le 1\ 234\ 567\ (1 \le i \le T)$
- $S_N \le 1\ 234\ 567$, where $S_N = N_1 + N_2 + \ldots + N_T$
- For tests worth 29 points, $N_i = 6\ (1 \le i \le T)$
- For tests worth another 19 points, $6 \le N_i \le 30\ (1 \le i \le T)$ and $S_N \le 123\ 456$
- For tests worth another 30 points, $6 \le N_i \le 500\ (1 \le i \le T)$, $S_N \le 123\ 456$ and there will be at most 10 test cases with $N_i > 50$

Example

trick.in	trick.out	Note

2
1
6
6 1 2 5 7 10
6
9 8 2 0 4 6
	1 2
	8 4
		The three samples represent the three runs.
		
		The trick is performed twice. For the first test case, the first assistant receives the cards 1, 2, 5, 6, 7 and 10 and shows the magician, in order, the cards 1 and 2. The second assistant receives the cards 0, 3, 4, 8, 9 and 12 and shows the magician, in order, the cards 4 and 3. After he saw the four cards, the magician uses his magical abilities and guesses that Dracula's hidden card is 11.

2
2
6
3 0 4 9 12 8
6
7 1 11 10 3 5
	4 3
	1 3	

2
3
6
1 2 4 3
6
8 4 1 3
	11
	12	