Task 1: `AskOneGetOneFree`

You are playing a guessing game with Bob. First, Bob announces a positive integer N. Next, Bob chooses two secret numbers, say x and y, where both are integers within the range $0 \leq x < N$ and $0 \leq y < N$. It is possible that x is the same as y. You can ask Bob a series of questions regarding the value of x and y. Your goal is to correctly determine the values of x and y using as little number of questions as possible. You can't ask a question like "*What is the value of x and y?*". Your questions must be of a certain format. In each question, you choose a number r, and the question is of the form:

Are the numbers greater or equal to r?

Bob must honestly tells you whether $x \geq r$, and whether $y \geq r$. After you have received the answer, you can choose to ask the next question, or make a guess of the value of x and y. Below is an example of the interactions with Bob.

Bob:	Let's choose N to be 4.
You:	Are the numbers greater or equal to 3?
Bob:	Yes for x; no for y.
You:	Are the numbers greater or equal to 1?
Bob:	Yes for x; yes for y.
You:	Are the numbers greater or equal to 2?
Bob:	Yes for x; yes for y.
You:	$x = 3$ and $y = 2$.
Bob:	You are correct. You have asked 3 questions.

Bob always answers the questions honestly. However, at any point during the game, he may modify the secrets x, y as long as the modified secrets do not contradict his previous answers. For example, in the above game, Bob initially could had chosen $x = 3$ and $y = 0$, but after seen the second question, modified the secrets to $x = 3$ and $y = 2$. Note that the modified secrets do not contradict the first answer (i.e. "Yes for x; no for y,") and thus is allowed. With this flexibility, Bob can adopt different strategies. For instance, he can be very simple-minded: chooses two secrets randomly initially, and does not modify them. Bob can also be very cunning: to respond to a question, he modifies the secrets so as to force the opponent to ask many questions.

Interactions

Bob has written a program that uses the cunning strategy. You have to submit a program that interacts with Bob's program. The last few sections give details on how to submit your program. In this contest, we have invited a well-known programmer Ada, and she will also submit a program.

Subtasks

1. (30 marks) Bob always chooses $N = 12$. Let Q be the number of questions your program has asked. If your program has made a wrong guess, no mark will be awarded. Otherwise, the number of marks awarded is calculated as follow:

$8 < Q < 12$,	10 marks.
$6 < Q \leq 8$,	20 marks.
$Q \leq 6$,	30 marks.

   (Hints: Note that a simple strategy that asks many questions can already obtain some marks.)

2. (30 marks) Bob chooses a $N \leq 2000$. Bob also chooses the same value of N when playing with Ada. Let Q be the number of questions your program has asked, and X be the number of questions Ada's program has asked. If your program has made a wrong guess, no mark will be awarded. Otherwise,

$25 < (Q - X) < 2000$	10 marks
$2 < (Q - X) \leq 25$	15 marks
$(Q - X) = 2$	20 marks
$(Q - X) = 1$	25 marks
$(Q - X) \leq 0$	30 marks

3. (40 marks) Bob chooses a $N \leq 10000$. Similarly, let Q and X as defined in the previous subtask.

$(Q - X) = 2$	10 marks
$(Q - X) = 1$	20 marks
$(Q - X) \leq 0$	40 marks

Each subtask will be evaluated multiple times. The final number of marks awarded for that subtask is the minimum among the different evaluations. For example, suppose during the first evaluation of subtask 2, Bob chose N=1000 and your program was awarded 30 marks. However, during the second evaluation, Bob chose N=2000 and your program was awarded 25 marks. Without further evaluation, the final number of marks awarded would be $\min(25, 30) = 25$.

How to submit your solution

You can obtain a set of programs from the contest web page.

In this task, you only have to submit a subroutine (either c, c++ or Pascal). The following files are examples that you have to submit:

- `askonegetonefree.c`
- `askonegetonefree.cpp`
- `askonegetonefree.pas`

The program you submitted must implement the subroutine `guess(N)` where the parameter `N` gives the positive integer that Bob has chosen. Within the subroutine, you can call a function `query(r)` (which is written by Bob) multiple times. Calling the function `query(r)` corresponds to asking Bob the question: *are the numbers greater or equal to r?* The function `query(r)` returns an object `Answer`, containing two integers p and q (see the given programs on the datatype of `Answer`). If Bob's answer for the secret x is yes (i.e. $x \geq r$), then the returned value p is 1; otherwise it is 0. Likewise, If Bob's answer for the secret y is yes (i.e. $y \geq r$), then the returned value q is 1; otherwise it is 0.

When you (or rather your subroutine) believe that you have found the secrets, exit the routine and return the guess as an `Answer` object. For example, if you believe that the secrets are 3 and 2, store them in the `Answer` object and return it.

For debugging and testing, you may want to write a program for Bob. The programs `bob.pas`, `bob.c`, `bob.cpp` are examples of Bob's programs (of course, the programs by the cunning Bob are not provided). You can see that these programs call your subroutine `guess(N)`. The given Bob's programs use a silly strategy. It always chooses $N = 4$, $x = 3$ and $y = 2$. We stress that during evaluation, another program that uses the cunning strategy is used. To compile your program together with Bob's program, use the following command:

- For C: `gcc -O2 bob.c askonegetonefree.c -o game`
- For C++: `g++ -O2 bob.cpp askonegetonefree.cpp -o game`
- For Pascal: `fpc -O2 bob.pas -ogame`

Now, by running the compiled executable `./game`, a game between Bob and your subroutine/function will be played.

Important notes

Your routine `query(r)` must not perform input/output operations, like reading from the standard input (the keyboard), or writing to the standard output (the display).

Additional note for Pascal

Your program must include the library `askonegetonefreelib`, similar to the sample pascal program. In addition, you must not declare a variable/function with the following name:

`askonegetonefreecheckanswer`

Task 2: Sudoku

Sullivan Dolkan Kuralski likes puzzle challenges. One of them is Sudoku. The task is to place digits in a square with 9×9 fields such that each field has one digit from $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and furthermore, in any row, column or quadrant (which is one of the nine 3×3 subsquares obtained by dividing the field evenly into 3×3 subsquares) the digits 1 to 9 occur exactly once. Below is an example of a Sudoku and also an example of a field a in a Sudoku with its row r, its column c and another field b with its quadrant q; so the entries marked with a and r are the row of a, the entries marked with a and c are the column of a and the entries marked with b and q are the quadrant of b.

```
1 2 3  4 5 6  7 8 9      . . .  . . .  . c .
7 8 9  1 2 3  4 5 6      r r r  r r r  r a r
4 5 6  7 8 9  1 2 3      . . .  . . .  . c .

9 1 2  3 4 5  6 7 8      . . .  q q q  . c .
6 7 8  9 1 2  3 4 5      . . .  q q b  . c .
3 4 5  6 7 8  9 1 2      . . .  q q q  . c .

8 9 1  2 3 4  5 6 7      . . .  . . .  . c .
5 6 7  8 9 1  2 3 4      . . .  . . .  . c .
2 3 4  5 6 7  8 9 1      . . .  . . .  . c .
```

The Sudoku game provides a 9×9 square with some empty fields. The player needs to recover the digits in those empty fields. As Sullivan became a well-known expert for Sudokus, an international newspaper offered him a job: When he creates Sudoku tasks for each of its editions, they would pay him a honorarium. However, the Sudoku tasks which he makes should satisfy certain conditions:

- They should have a minimum number of empty fields (here indicated as a 0 and in the printed newspaper just left blank);
- They should have only a single solution (so that the solution of the newspaper published the next day will match what the readers did in the case that they did it right);
- They should be human-solvable, that is, at every time there should be at least one free field which can be filled in by human with a unique digit.

The third condition is a bit fuzzy and when Sullivan enquired on what it meant precisely, the newspaper hesitated a bit but eventually came up with the following answer:

- At every stage of solving the Sudoku task, there must be a still free field so that every digit except the correct one is already filled-in at some other field in the column, the row or the quadrant of this free field; here "filled-in" includes the given digits and the digits filled-in in previous steps of making the solution.

Below gives an example that satisfies the human-solvable condition. In the example, all fields with zero are empty fields. The field "a" is forced to be "1" since the entries in the column, the row and the quadrant contain all possible digits except "1":

```
. . 0  . . .  . . .
. . 2  . . .  . . .
. . 0  . . .  . . .

4 5 a  0 0 0  6 8 9
6 7 0  . . .  . . .
0 0 0  . . .  . . .

. . 0  . . .  . . .
. . 3  . . .  . . .
. . 0  . . .  . . .
```

Sullivan knew that he could make such Sudokus, but it would need some time; so he did not accept the job right away. Instead he asked his friend Claude Pastel Procal for help and Claude answered: "Why don't you use computers? Sometimes simple algorithms can solve difficult problems." So your task is now to write a program which makes the Sudoku tasks for Sullivan according to the conditions above. Furthermore, his boss at the newspaper does not want that all Sudokus have the same solution; therefore he will provide Sullivan with ideas where some of the 1s have to be in the final Sudoku. So Sullivan and his boss agreed on the following protocol:

- The boss provides Sullivan with 82 numbers: the first number u is the minimum number of fields that should be free (to be indicated as 0); Sullivan is permitted to make u or more fields to be free, but not less than u free fields. The next 81 numbers are either 0 or 1 and contain suggestions of the boss where should be a 1; note that the boss might only suggest some and not all of the positions of 1s.
- Sullivan should deliver 81 numbers representing the entries of the Sudoku task: a 0 indicates a free field and an entry from 1 up to 9 indicates a prefilled field. The Sudoku task should be human solvable, should have at least u free fields and the solution of the Sudoku should have a 1 at every place where the boss had suggested that there is a 1.

The boss will ask his IT department to make a computer program which checks the requirements he has put on Sullivan and he will only pay for those Sudoku tasks which meet the requirements. So your computer program should produce Sudoku tasks which meet the above specifications.

Input Format

Your program should read the input from standard input. The input consists of 82 numbers. These numbers are first the number of free fields required followed by 81 numbers 0 or 1; these are arranged row by row by the format of a Sudoku and those which are 1 should have a 1 in the final solution of the Sudoku. An example for an **input** this one:

```
3

1 0 0  0 0 0  0 0 0
0 0 0  1 0 0  0 0 0
0 0 0  0 0 0  1 0 0

0 1 0  0 0 0  0 0 0
0 0 0  0 1 0  0 0 0
0 0 0  0 0 0  0 1 0

0 0 1  0 0 0  0 0 0
0 0 0  0 0 1  0 0 0
0 0 0  0 0 0  0 0 1
```

Output Format

Your program should write 81 numbers onto standard output satisfying the specifications given above. These 81 numbers are a valid Sudoku task which is human solvable and which has the required number (or more) of entries which are 0. It has to meet the positions of the 1s given by the Boss. The entries 1 to 9 in the 81 numbers correspond to the prefilled positions with the corresponding numbers in the Sudoku task. Here a possible **output** which matches the above input, note that the output is not unique.

```
0 0 3  4 5 6  7 8 9
0 8 9  1 2 3  4 5 6
0 5 6  7 8 9  1 2 3

9 1 2  3 4 5  6 7 8
6 7 8  9 1 2  3 4 5
3 4 5  6 7 8  9 1 2

8 9 1  2 3 4  0 6 7
5 6 7  8 9 1  2 3 4
0 3 4  5 6 7  8 9 1
```

The output leaves more than 3 entries blank, that is permitted. It is also permitted that some of the 1s of the boss are left blanc; however, when the solution is made, there must be a 1 which goes into this position.

Subtasks

20 marks should be awarded for each subtask:

- Making a Sudoku matching the given 1s with at least 5 free fields;

- Making a Sudoku matching the given 1s with at least 15 free fields;
- Making a Sudoku matching the given 1s with at least 25 free fields;
- Making a Sudoku matching the given 1s with at least 35 free fields;
- Making a Sudoku matching the given 1s with at least 45 free fields.

Task 3: RADIOACTIVE

In the Cow Village, every cow owns exactly one house. These houses are arranged in a straight line, ending at the inaptly named Village Centre. The N houses are labelled from 1 to N in order of their distance from the Village Centre, starting from the house nearest to the Village Centre.

Now Kraw the Krow has lots and lots of bananas and is feeling very generous today. He decides to give the cows some of his bananas. He has so many bananas, anyway, that he will not run out of bananas no matter how many he gives the cows.

However, as we all know, bananas are radioactive. What's worse, Kraw's bananas are particularly radioactive because he lives close to the Equator where the average cosmic ray flux is the highest. If too many of Kraw's bananas were to be placed in close proximity of one another, they could trigger a nuclear explosion which would certainly tar Kraw's environmentally friendly corporate image. Kraw stores his bananas in small lead boxes, but he'll have to take them out if he wants to give them to the cows. To prevent a nuclear explosion, Kraw must distribute his bananas according to the following rules:

- No cow gets more than one banana.
- No more than C cows with consecutive house numbers can all get bananas.

So, for example, if $N = 4$ and $C = 2$, Kraw can give the cows in houses 1, 2 and 4 a banana each, but he can't give the cows in houses 2, 3 and 4 a banana each because houses 2, 3 and 4 are all next to each other and a nuclear explosion would occur.

Kraw wants to find the number of ways he can distribute his bananas to the cows, mod 1,000,000,007. Note that there is no restriction on the number of bananas that Kraw can distribute to the cows – in particular, Kraw can give the cows no bananas at all.

Input format

Your program must read from standard input. The first and only line of the input are the two positive numbers N, the number of houses, and C, the number of consecutive houses that can all get bananas without causing a nuclear explosion. For the above example, the input is as follow:

```
4 2
```

Output format

You program must write to the standard output a number, which is the number of ways Kraw can distribute his bananas to the cows, mod 1,000,000,007. For the input above, the output is:

```
13
```

Explanation

When N = 4 and C = 2, there are 13 sets of house numbers that satisfy the above rules: {1, 2, 4}, {1, 3, 4}, {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}, {1}, {2}, {3}, {4}, {} (the empty set).

More samples

Here are a few more samples.

Input

```
3 1
```

Output

```
5
```

Remark

When N = 3 and C = 1, there are 5 sets of house numbers that satisfy the above rules: {1, 3}, {1}, {2}, {3}, {} (the empty set). Note that {1, 3} is allowed because houses 1 and 3 are not next to each other, so a nuclear explosion would not occur.

Input

```
3 5
```

Output

```
8
```

Remark

When N = 3 and C = 5, Kraw doesn't need to worry about the second rule because there are no C = 5 consecutive house numbers to begin with. So all 8 sets of house numbers satisfy the above rules: {1, 2, 3}, {1, 2}, {1, 3}, {2, 3}, {1}, {2}, {3}, {} (the empty set).

Subtasks

The maximum execution time on each instance is 1.0s. Your program will be tested on sets of input instances as follows:

1. (11 marks) Each instance satisfies $0 < N < C < 200$;
2. (4 marks) Each instance satisfies $0 < N < 10^6, C = 1$;
3. (7 marks) Each instance satisfies $0 < N < 16, 0 < C < 16$;
4. (9 marks) Each instance satisfies $0 < N < 10^5, 0 < C < 20$;
5. (27 marks) Each instance satisfies $0 < N < 10^6, 0 < C < 200$;
6. (14 marks) Each instance satisfies $0 < N < 10^{18}, C = 1$;
7. (15 marks) Each instance satisfies $0 < N < 10^{18}, 0 < C < 50$;
8. (13 marks) Each instance satisfies $0 < N < 10^{18}, 0 < C < 200$;

Task 4: `BananaFarm`

After Kraw the Krow has distributed radioactive bananas to cows in Cow Village, Moosa the cow is now addicted to eating bananas. In fact, he is not the only cow that appreciates bananas. Kraw the Krow's radioactive bananas has influenced the entire cow species such that every cow now eats nothing but bananas. Although Kraw the Krow claims to have a lot of bananas, he still has a limited supply of bananas. Thus, at the present rate of cows consuming bananas, it is predicted that those bananas will run out in the near future.

In order to replenish the stockpile of bananas, the cows have decided to setup banana plantations. The banana plantation consists of N trees planted on a line from left to right. They are numbered from 1 to N from the leftmost tree through the rightmost tree. Each tree has a number of bananas growing on them and the i^{th} tree would have B_i bananas. Radiation has caused these bananas to be mutated. As such, these mutated bananas can have up to 10^9 bananas per tree when planted in the banana plantation. Of course, there cannot be a negative number of bananas on the banana tree. Seems like mutation is not always a bad thing after all!

The harvest season is approaching and the cows have just one problem. Bananas grow on trees and they are not tall enough to reach it. Moosa's parents, Cow Pei and Cow Bu have come up with an ingenious plan to co-operate with Rar the Cat and his feline friends. The plan involves cats climbing banana trees to harvest the fruit. However, Rar the Cat and his feline friends would like something in return for their efforts.

After much deliberation, Rar the Cat has shortlisted P plans. The i^{th} plan involves C_i cats harvesting trees labelled S_i to E_i inclusive. However, each cat can only harvest one tree at one time. Hence, only C_i trees would be harvested between the trees labelled S_i to E_i. Obviously, the cats would harvest the C_i trees within the range that would yield the largest total number of harvested bananas. Do note that since the harvesting is still in the planning stage, no bananas are actually harvested and the i^{th} banana tree will always have B_i bananas for every single plan.

As part of the agreement, the cats would be rewarded some bananas. This amount depends on which trees the cats decide to climb, and is equal to the minimum number of bananas found in one of those trees they decide to climb. For example, consider 3 cats that are part of a plan to harvest a group of four trees, consisting of 1, 2, 3, and 5 bananas, respectively. If the three cats decide to climb the trees containing 2, 3, and 5 bananas, their reward would be 2 bananas. Otherwise, if they climb the tree containing a 1 banana and two other trees, their reward would only be 1 banana. The cats are smart, thought, and they will always decide to climb the trees that would maximize their reward. Thus, for the example above they will always climb the trees with 2, 3, and 5 bananas, and gets a reward of two bananas.

Since Rar the Cat is smart, he guarantees that the number of cats in each plan, C_i, will be positive and not exceed the number of trees in the range to be harvested. Also, all S_i will be less than or equal to E_i. However, he is not smart enough to calculate how many bananas each plan will earn the cats. As such, your task is to tell Rar the Cat the amount of bananas the cats will be rewarded with for each plan Rar the Cat has.

Input

Your program must read from standard input.
The first line of input will contain 2 integers, N and P.
The next line of input will contain N integers. The i^{th} integer would be B_i, the bananas available for harvesting on the i^{th} tree.
The subsequent P lines of input will contain 3 integers each, S_i, E_i followed by C_i, describing the i^{th} plan whereby C_i cats would be harvesting banana trees with labels S_i to E_i inclusive.

Output

Your program must output to standard output only.
For each plan, you are to output how many bananas the cats would be rewarded with, one on each line.

Subtasks

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

1. (13 marks) $0 < N, P \leq 100,000$. However, all $S_i = 1$ and all $E_i = N$. In other words, all plans will involve harvesting all the trees in the plantation.
2. (12 marks) $0 < N, P \leq 100,000$. However, all $S_i = 1$ and $C_i = 1$. In other words, all plans will only involve one cat and trees 1 to E_i inclusive.
3. (21 marks) $0 < N, P \leq 100,000$. However, all $C_i = 1$. In other words, all plans will only involve one cat.
4. (21 marks) $0 < N, P \leq 100,000$. However, all $C_i \leq 2$. In other words, all plans will only involve one or two cats.
5. (33 marks) $0 < N, P \leq 100,000$. No other restrictions apply.

Sample Testcase 1

This testcase is only valid for subtasks 1 and 5 only.

Input	Output
10 5 1 4 2 8 5 2 3 5 3 4 1 10 1 1 10 4 1 10 3 1 10 9 1 10 10	8 4 5 2 1

All the plans described will be harvesting all the banana trees in the plantation.

The first plan has only 1 cat, hence only tree 4 (with 8 bananas) would be harvested. Hence, the cats will get 8 bananas as their reward.
The second plan involves 4 cats. Either trees 4, 5, 8 and 2 or trees 4, 5, 8 and 10 will be harvested. In both scenarios, 8, 5, 5, 4 bananas will be harvested from the respective trees. Hence, the cats will get the lowest number of bananas, 4 as their reward.
The third plan involves 3 cats. Trees 4, 5 and 8 with bananas 8, 5 and 5 respectively would be harvested. The cats will hence be rewarded with 5 bananas.
The fourth plan involves 9 cats. All the trees except tree 1 will be harvested. Hence, the cats will be rewarded with 2 bananas.
The last plan involves 10 cats. All the banana trees will be harvested. Their reward will thus be equivalent to the minimum number of bananas on all the trees, which is 1.

Sample Testcase 2

This testcase is only valid for subtasks 2, 3, 4 and 5 only.

Input	Output
10 5 1 4 2 8 5 2 3 5 3 4 1 1 1 1 2 1 1 3 1 1 4 1 1 5 1	1 4 4 8 8

Sample Testcase 3

This testcase is only valid for subtasks 3, 4 and 5 only.

Input	Output
10 5 1 4 2 8 5 2 3 5 3 4 1 2 1 1 7 1 5 8 1 9 10 1 7 10 1	4 8 5 4 5

Sample Testcase 4

This testcase is only valid for subtasks 4 and 5 only.

Input	Output
10 5 1 4 2 8 5 2 3 5 3 4 1 2 2 1 3 2 5 8 2 9 10 2 7 10 2	1 2 5 3 4

Sample Testcase 5

This testcase is only valid for subtask 5 only.

Input	Output
10 5 1 4 2 8 5 2 3 5 3 4 1 1 1 2 7 4 5 10 3 2 10 8 1 10 10	1 3 4 2 1