A. Game of Life

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

William really likes the cellular automaton called "Game of Life" so he decided to make his own version. For simplicity, William decided to define his cellular automaton on an array containing n cells, with each cell either being alive or dead.

Evolution of the array in William's cellular automaton occurs iteratively in the following way:

- If the element is dead and it has **exactly** 1 alive neighbor **in the current state of the array**, then on the next iteration it will become alive. For an element at index i the neighbors would be elements with indices $i-1$ and $i+1$. If there is no element at that index, it is considered to be a dead neighbor.
- William is a humane person so all alive elements stay alive.

Check the note section for examples of the evolution.

You are given some initial state of all elements and you need to help William find the state of the array after m iterations of evolution.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^3$). Description of the test cases follows.

The first line of each test case contains two integers n and m ($2 \le n \le 10^3, 1 \le m \le 10^9$), which are the total number of cells in the array and the number of iterations.

The second line of each test case contains a string of length n made up of characters "0" and "1" and defines the initial state of the array. "1" means a cell is alive and "0" means it is dead.

It is guaranteed that the sum of n over all test cases does not exceed 10^4.

Output

In each test case output a string of length n, made up of characters "0" and "1" — the state of the array after m iterations of evolution.

Example

input

```
4
11 3
01000000001
10 2
0110100101
5 2
10101
3 100
000
```

output

```
11111001111
1110111101
10101
000
```

Note

Sequence of iterations of evolution for the first test case

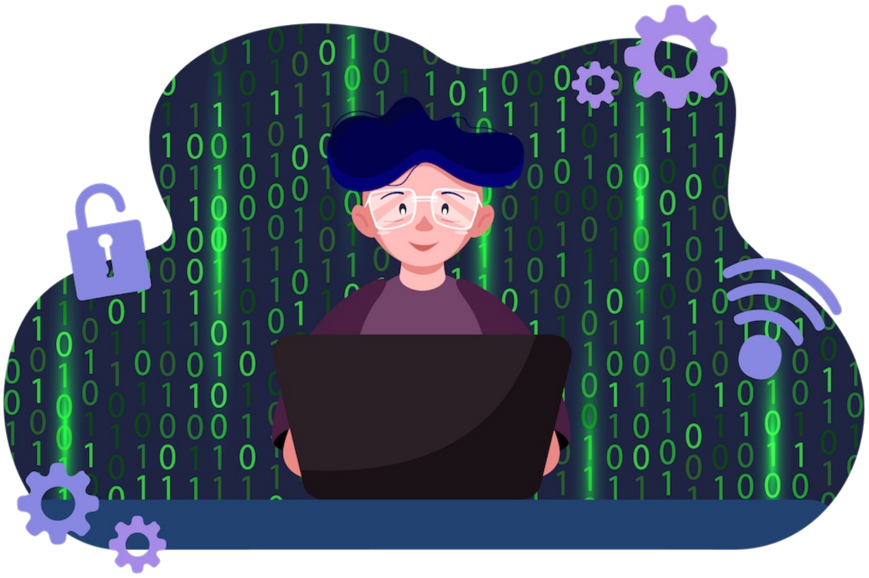
- `01000000001` — initial state
- `11100000011` — first iteration of evolution
- `11110000111` — second iteration of evolution
- `11111001111` — third iteration of evolution

Sequence of iterations of evolution for the second test case

- `0110100101` — initial state
- `1110111101` — first iteration of evolution
- `1110111101` — second iteration of evolution

B. Lord of the Values

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output



While trading on his favorite exchange trader William realized that he found a vulnerability. Using this vulnerability he could change the values of certain internal variables to his advantage. To play around he decided to change the values of all internal variables from $a_1, a_2, \ldots, a_n$ to $-a_1, -a_2, \ldots, -a_n$. For some unknown reason, the number of service variables is always an even number.

William understands that with his every action he attracts more and more attention from the exchange's security team, so the number of his actions must not exceed $5\,000$ and after every operation no variable can have an absolute value greater than 10^{18}. William can perform actions of two types for two chosen variables with indices i and j, where $i < j$:

1. Perform assignment $a_i = a_i + a_j$
2. Perform assignment $a_j = a_j - a_i$

William wants you to develop a strategy that will get all the internal variables to the desired values.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 20$). Description of the test cases follows.

The first line of each test case contains a single **even** integer n ($2 \le n \le 10^3$), which is the number of internal variables.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le 10^9)$, which are initial values of internal variables.

Output

For each test case print the answer in the following format:

The first line of output must contain the total number of actions k, which the strategy will perform. Note that you do not have to minimize k. The inequality $k \le 5\,000$ must be satisfied.

Each of the next k lines must contain actions formatted as "`type i j`", where "`type`" is equal to "`1`" if the strategy needs to perform an assignment of the first type and "`2`" if the strategy needs to perform an assignment of the second type. Note that $i < j$

should hold.

We can show that an answer always exists.

Example

input

```
2
4
1 1 1 1
4
4 3 1 2
```

output

```
8
2 1 2
2 1 2
2 1 3
2 1 3
2 1 4
2 1 4
1 1 2
1 1 2
8
2 1 4
1 2 4
1 2 4
1 2 4
1 3 4
1 1 2
1 1 2
1 1 4
```

Note

For the first sample test case one possible sequence of operations is as follows:

1. "2 1 2". Values of variables after performing the operation: [1, **0**, 1, 1]
2. "2 1 2". Values of variables after performing the operation: [1, **-1**, 1, 1]
3. "2 1 3". Values of variables after performing the operation: [1, -1, **0**, 1]
4. "2 1 3". Values of variables after performing the operation: [1, -1, **-1**, 1]
5. "2 1 4". Values of variables after performing the operation: [1, -1, -1, **0**]
6. "2 1 4". Values of variables after performing the operation: [1, -1, -1, **-1**]
7. "1 1 2". Values of variables after performing the operation: [**0**, -1, -1, -1]
8. "1 1 2". Values of variables after performing the operation: [**-1**, -1, -1, -1]

For the second sample test case one possible sequence of operations is as follows:

1. "2 1 4". Values of variables after performing the operation: [4, 3, 1, **-2**]
2. "1 2 4". Values of variables after performing the operation: [4, **1**, 1, -2]
3. "1 2 4". Values of variables after performing the operation: [4, **-1**, 1, -2]
4. "1 2 4". Values of variables after performing the operation: [4, **-3**, 1, -2]
5. "1 3 4". Values of variables after performing the operation: [4, -3, **-1**, -2]
6. "1 1 2". Values of variables after performing the operation: [**1**, -3, -1, -2]
7. "1 1 2". Values of variables after performing the operation: [**-2**, -3, -1, -2]
8. "1 1 4". Values of variables after performing the operation: [**-4**, -3, -1, -2]

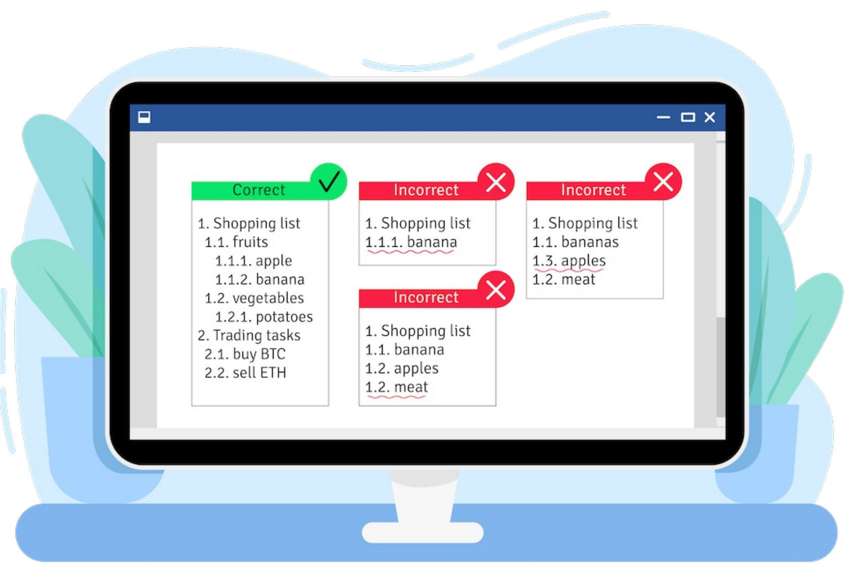
C. Compression and Expansion

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output



William is a huge fan of planning ahead. That is why he starts his morning routine by creating a nested list of upcoming errands.

A valid nested list is any list which can be created from a list with one item "1" by applying some operations. Each operation inserts a new item into the list, **on a new line**, just after one of existing items $a_1 \,.\, a_2 \,.\, a_3 \,.\, \cdots \,.\, a_k$ and can be one of two types:

1. Add an item $a_1 \,.\, a_2 \,.\, a_3 \,.\, \cdots \,.\, a_k \,.\, 1$ (starting a list of a deeper level), or
2. Add an item $a_1 \,.\, a_2 \,.\, a_3 \,.\, \cdots \,.\, (a_k + 1)$ (continuing the current level).

Operation can only be applied if the list does not contain two identical items afterwards. And also, if we consider every item as a sequence of numbers, then the sequence of items should always remain increasing in lexicographical order. Examples of valid and invalid lists that are shown in the picture can found in the "Notes" section.
When William decided to save a Word document with the list of his errands he accidentally hit a completely different keyboard shortcut from the "Ctrl-S" he wanted to hit. It's not known exactly what shortcut he pressed but after triggering it all items in the list were replaced by a single number: the last number originally written in the item number.

William wants you to help him restore a fitting original nested list.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10$). Description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 10^3$), which is the number of lines in the list.

Each of the next n lines contains a single integer a_i ($1 \le a_i \le n$), which is what remains of William's nested list.

It is guaranteed that in each test case at least one fitting list exists.

It is guaranteed that the sum of values n across all test cases does not exceed 10^3.

Output

For each test case output n lines which represent a valid nested list, which could become the data provided to you by William.

If there are multiple answers, print any.

Example

input

```
2
4
1
1
2
3
9
1
1
1
2
2
1
2
1
2
```

output

```
1
1.1
1.2
1.3
1
1.1
1.1.1
```

```
1.1.2
1.2
1.2.1
2
2.1
2.2
```

Note

In the second example test case one example of a fitting list is:

1

1.**1**

1.1.**1**

1.1.**2**

1.**2**

1.2.**1**

2

2.**1**

2.**2**

This list can be produced by using the sequence of operations shown below:

1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1
	2	1.1	1.1	1.1	1.1	1.1	1.1	1.1
		2	1.2	1.1.1	1.1.1	1.1.1	1.1.1	1.1.1
			2	1.2	1.1.2	1.1.2	1.1.2	1.1.2
				2	1.2	1.2	1.2	1.2
					2	1.2.1	1.2.1	1.2.1
						2	2	2
							2.1	2.1
								2.2

1. Original list with a single item 1.
2. Insert item 2 by using the insertion operation of the second type after item 1.
3. Insert item 1.1 by using the insertion operation of the first type after item 1.
4. Insert item 1.2 by using the insertion operation of the second type after item 1.1.
5. Insert item $1.1.1$ by using the insertion operation of the first type after item 1.1.
6. Insert item $1.1.2$ by using the insertion operation of the second type after item $1.1.1$.
7. Insert item $1.2.1$ by using the insertion operation of the first type after item 1.2.
8. Insert item 2.1 by using the insertion operation of the first type after item 2.
9. Insert item 2.2 by using the insertion operation of the second type after item 2.1.

D. Love-Hate

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

William is hosting a party for n of his trader friends. They started a discussion on various currencies they trade, but there's an issue: not all of his trader friends like every currency. They like some currencies, but not others.

For each William's friend i it is known whether he likes currency j. There are m currencies in total. It is also known that a trader may not like more than p currencies.

Because friends need to have some common topic for discussions they need to find the largest by cardinality (possibly empty) subset of currencies, such that there are at least $\lceil \frac{n}{2} \rceil$ friends (rounded up) who like each currency in this subset.

Input

The first line contains three integers n, m and p $(1 \le n \le 2 \cdot 10^5, 1 \le p \le m \le 60, 1 \le p \le 15)$, which is the number of trader friends, the number of currencies, the maximum number of currencies each friend can like.

Each of the next n lines contain m characters. The j-th character of i-th line is 1 if friend i likes the currency j and 0 otherwise. It is guaranteed that the number of ones in each line does not exceed p.

Output

Print a string of length m, which defines the subset of currencies of the maximum size, which are liked by at least half of all friends. Currencies belonging to this subset must be signified by the character 1.

If there are multiple answers, print any.

Examples

input

```
3 4 3
1000
0110
1001
```

output

```
1000
```

input

```
5 5 4
11001
10101
10010
01110
11011
```

output

```
10001
```

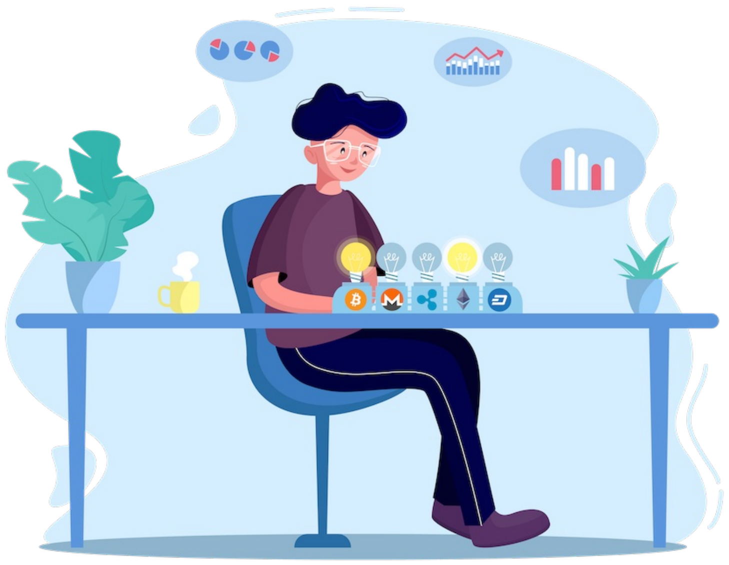
Note

In the first sample test case only the first currency is liked by at least $\lceil \frac{3}{2} \rceil = 2$ friends, therefore it's easy to demonstrate that a better answer cannot be found.

In the second sample test case the answer includes 2 currencies and will be liked by friends 1, 2, and 5. For this test case there are other currencies that are liked by at least half of the friends, but using them we cannot achieve a larger subset size.

E. Crypto Lights

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output



To monitor cryptocurrency exchange rates trader William invented a wonderful device consisting of n lights arranged in a row. The device functions in the following way:

Initially, all lights on William's device are turned off. At the beginning of a new iteration the device randomly, with a uniform distribution, picks a light that is turned off and turns it on, telling William which cryptocurrency he should invest in. After this iteration if any k consecutive lights contain more than one turned on light, then the device finishes working.

William doesn't like uncertainty, so he wants you to calculate the expected value of the number of lights that are turned on in the device after it finishes working.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10$). Description of the test cases follows.

The only line for each test case contains two integers n and k ($2 \le k \le n \le 10^5$), which are the total number of lights and the length of subsegment of lights that are being checked, respectively.

Output

For each test case print the answer, modulo 10^9+7.

Formally, let $M = 10^9+7$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p \cdot q^{-1} \bmod M$. In other words, output such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

Example

input

```
3
3 2
15 2
40 15
```

output

```
333333338
141946947
329622137
```

Note

Explanation of the first sample test case:

Let's write out all possible sequences of light toggles, which will make the device complete its operation:

1. $(1,2)$ — 2 lights are turned on
2. $(1,3,2)$ — 3 lights are turned on
3. $(2,1)$ — 2 lights are turned on
4. $(2,3)$ — 2 lights are turned on
5. $(3,2)$ — 2 lights are turned on
6. $(3,1,2)$ — 3 lights are turned on

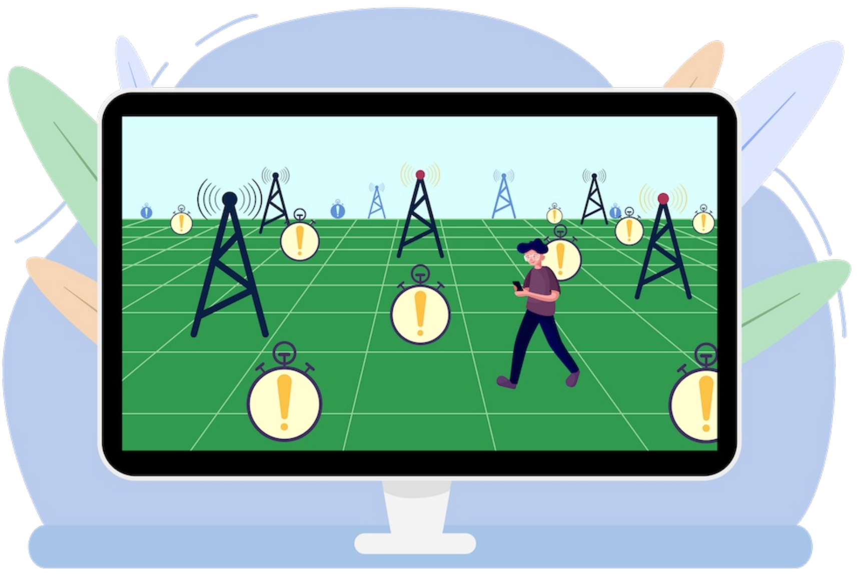
Then the final expected value will be equal to $\frac{2}{6}$ + $\frac{3}{6}$ + $\frac{2}{6}$ + $\frac{2}{6}$ + $\frac{2}{6}$ + $\frac{3}{6}$ = $\frac{14}{6}$ = $\frac{7}{3}$.

Then the required output will be 333333338, since $333333338 \cdot 3 \equiv 7 \pmod{10^9+7}$.

F. Favorite Game

time limit per test: 2 seconds

memory limit per test: 256 megabytes
input: standard input
output: standard output



After William is done with work for the day, he enjoys playing his favorite video game.

The game happens in a 2D world, starting at turn 0. William can pick any cell in the game world and spawn in it. Then, each turn, William may remain at his current location or move from the current location (x, y) to one of the following locations: (x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1).

To accelerate movement the game has n fast travel towers. i-th tower is located at location (xa_i, ya_i). To be able to instantly travel to the tower from any location in the game world it must first be activated. Activation of tower i happens at the moment when the player is in cell (xa_i, ya_i) after this the tower remains active throughout the entire game.

William also knows that the game has m quests. i-th quest can be completed instantly by being at location (xb_i, yb_i) on turn t_i.

William wants to find out the maximal number of quests he will be able to complete by optimally traversing the game world.

Input

The first line contains two integers n and m ($0 \le n \le 14, 1 \le m \le 100$), which are the number of towers and the number of quests, respectively.

Each of the next n lines contains two integers xa_i, ya_i ($1 \le xa_i, ya_i \le 10^6$), which are the coordinates of fast travel towers.

Each of the next m lines contains two integers xb_i, yb_i and t_i ($1 \le xb_i, yb_i \le 10^6$, $1 \le t_i \le 10^9$), which are the coordinates of quests and the turn at which it may be completed.

It is guaranteed that all locations in a test are different.

Output

Print a single number — the maximal number of quests William will be able to complete.

Example

input

```
3 4
1 1
2 3
5 2
2 2 12
5 1 4
6 2 11
3 5 10
```

output

```
3
```

Note

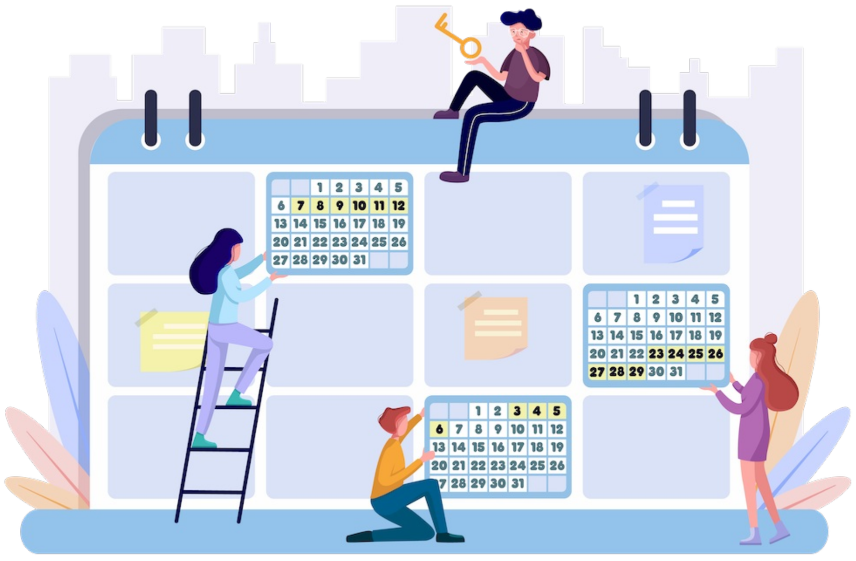
In the first sample test one of the possible sequences of William's actions is as follows:

- Spawn at $(3, 2)$
- On turn 1 move to $(4, 2)$
- On turn 2 move to $(5, 2)$. By visiting this cell William activates tower number 3.
- On turn 3 move to $(5, 1)$, where he waits for 1 turn to complete the 2nd quest
- On turn 5 move to $(5, 2)$
- On turn 6 move to $(5, 3)$
- On turn 7 move to $(5, 4)$
- On turn 8 move to $(5, 5)$

- On turn 9 move to $(4, 5)$
- On turn 10 move to $(3, 5)$. By moving to this location William will complete the 4th quest
- On turn 10 instantly move to an already activated fast travel tower at $(5, 2)$
- On turn 11 move to $(6, 2)$. By moving to this location William will complete the 3rd quest
- William will not be able to complete the quest number 1, because the tower at $(2, 3)$ was not activated

G. Try Booking

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output



William owns a flat in central London. He decided to rent his flat out for the next n days to earn some money.

Since his flat is in the center of the city, he instantly got m offers in the form (l_i, r_i), which means that someone wants to book the flat from day l_i until day r_i inclusive. To avoid spending a lot of time figuring out whether it's profitable for him to accept an offer, William decided to develop an algorithm. The algorithm processes all offers as they arrive and will only accept offer i if the following two conditions are satisfied:

- $r_i - l_i + 1 \geq x$.
- None of the days between l_i and r_i are occupied by a previously accepted offer

William isn't sure what value x should have and he asks you for help. For all x from 1 to n he wants you to calculate the total number of days for which the flat would be occupied if the corresponding value will be assigned to x.

Input

The first line contains two integers n and m $(1 \leq n \leq 5 \cdot 10^4, 1 \leq m \leq 10^5)$, which are the number of days and the number of offers, respectively.

Each of the next m lines contains two integers l_i and r_i $(1 \leq l_i \leq r_i \leq n)$, which describe the i-th renting offer. All offers are given in chronological order.

Output

Print n integers. The number in i-th line must be equal to the number of days the flat would be occupied if the algorithm will use the value of x equal to i.

Example

input

```
6 5
2 3
3 5
1 1
1 5
1 6
```

output

```
3
2
3
5
5
6
```

Note

The description of segments from the first sample test for each x:

- $x = 1$ — algorithm will approve offers: 1 (2..3), 3 (1..1). The total number of days for which William's flat will be rented out is 3
- $x = 2$ — algorithm will approve offers: 1 (2..3). The total number of days for which William's flat will be rented out is 2
- $x = 3$ — algorithm will approve offers: 2 (3..5). The total number of days for which William's flat will be rented out is 3
- $x = 4$ — algorithm will approve offers: 4 (1..5). The total number of days for which William's flat will be rented out is 5
- $x = 5$ — algorithm will approve offers: 4 (1..5). The total number of days for which William's flat will be rented out is 5
- $x = 6$ — algorithm will approve offers: 5 (1..6). The total number of days for which William's flat will be rented out is 6

H. Hopping Around the Array

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

William really wants to get a pet. Since his childhood he dreamt about getting a pet grasshopper. William is being very responsible about choosing his pet, so he wants to set up a trial for the grasshopper!

The trial takes place on an array a of length n, which defines lengths of hops for each of n cells. A grasshopper can hop around the sells according to the following rule: from a cell with index i it can jump to any cell with indices from i to $i + a_i$ inclusive.

Let's call the k-*grasshopper value* of some array the smallest number of hops it would take a grasshopper to hop from the first cell to the last, but before starting you can select no more than k cells and remove them from the array. When a cell is removed all other cells are renumbered but the values of a_i for each cell remains the same. **During this the first and the last cells may not be removed.**

It is required to process q queries of the following format: you are given three numbers l, r, k. You are required to find the k-grasshopper value for an array, which is a subarray of the array a with elements from l to r inclusive.

Input

The first line contains two integers n and q ($1 \le n, q \le 20000$), the length of the array and the number of queries respectively.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) – the elements of the array.

The following q lines contain queries: each line contains three integers l, r and k ($1 \le l \le r \le n$, $0 \le k \le min(30, r - l)$), which are the edges of the subarray and the number of the grasshopper value respectively.

Output

For each query print a single number in a new line — the response to a query.

Example

input

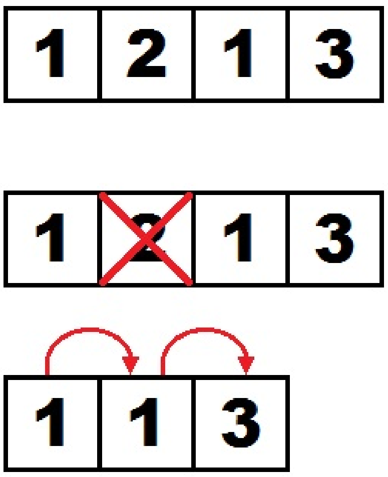
```
9 5
1 1 2 1 3 1 2 1 1
1 1 0
2 5 1
5 9 1
2 8 2
1 9 4
```

output

```
0
2
1
2
2
```

Note

For the second query the process occurs like this:



For the third query the process occurs like this: