ICPC SouthWestern Europe Regional Contest 2025

Lisbon, Lyon, Pisa, Nov 23rd, 2025

Judges and Problem Setters

- Dmitry Akulov (École polytechnique)
- Guillaume Aubian (Université Paris-Panthéon-Assas)
- Alessandro Bortolin (Università degli Studi di Milano)
- Fabrizio Brioni (Università degli Studi di Milano)
- Thomas Deniau
- Tommaso Dossi (Università di Pisa)
- Vincent Jugé (Université Gustave Eiffel)
- Raphaël Marinier (Google), *deputy chief judge*
- Henrique Navas (Técnico, ULisbon)
- Kevin Pucci (UPorto)
- Jaime Ramos (Técnico, ULisbon)
- Valerio Stancanelli (Università di Pisa), *chief judge*

This problem set consists of 12 problems, on 28 pages.

A Adjusting Drones

You are managing a fleet of n drones, each with an energy level $a_1, \ldots, a_n$. You are also given a positive integer k, which represents the maximum number of drones allowed to share the same energy level.

To prevent overloads, the drones automatically perform energy balancing operations as follows. While there exists an energy level that appears strictly more than k times, they execute the following steps:

- first, every drone i whose energy a_i has already appeared before (that is, there exists $j < i$ with $a_j = a_i$) is marked;
- then, for each marked drone, its energy is increased by 1 unit;
- then, the marks are removed.

The process stops once no energy level appears more than k times. Determine how many energy balancing operations will be performed.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains two integers n, k ($1 \le k \le n \le 2 \cdot 10^5$) — the number of drones and the maximum allowed number of drones with the same energy level.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 2n$) — the initial energy levels of the drones.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single line containing an integer: the number of energy balancing operations performed.

Samples

Sample input 1	Sample output 1
5 6 3 1 1 1 1 1 1 5 1 1 3 2 1 4 16 2 6 3 7 5 13 6 7 6 3 5 13 7 6 4 3 6 16 3 6 3 7 5 13 6 7 6 3 5 13 7 6 4 3 6 16 4 6 3 7 5 13 6 7 6 3 5 13 7 6 4 3 6	3 4 14 5 4

Explanation of sample 1.
In the first test case, the drones' energy levels evolve as follows:

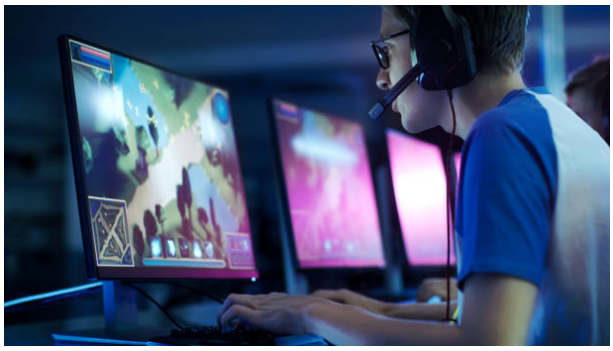
- at the beginning, $[1, 1, 1, 1, 1, 1]$;
- after 1 operation, $[1, 2, 2, 2, 2, 2]$;
- after 2 operations, $[1, 2, 3, 3, 3, 3]$;
- after 3 operations, $[1, 2, 3, 4, 4, 4]$.

After 3 operations, every energy level appears at most 3 times, so the process stops.

In the second test case, the energy levels change as follows: $[1, 3, 2, 1, 4] \to [1, 3, 2, 2, 4] \to [1, 3, 2, 3, 4] \to [1, 3, 2, 4, 4] \to [1, 3, 2, 4, 5]$. The process stops after 4 operations.

Image credits: Adonyi Gábor, public domain.

B Billion Players Game



You are following the world championship of the Billion Players Game. There are 10^9 players competing, and you want to predict the final ranking p of Godflex, your favorite streamer. After analyzing recent matches, you are sure that $l \le p \le r$, but you don't know anything else.

There are n offers made by the in-game bookmaker. In the i-th offer, the bookmaker suggests an estimation a_i for Godflex's final ranking. For each offer, you must choose exactly one of the following actions:

- Ignore the offer.
- Accept the offer by claiming that $p \le a_i$. If you are right, you earn $|p - a_i|$ coins; otherwise you lose $|p - a_i|$ coins.
- Accept the offer by claiming that $p \ge a_i$. If you are right, you earn $|p - a_i|$ coins; otherwise you lose $|p - a_i|$ coins.

After you decide how to deal with all the offers, the actual Billion Players Game is played. Godflex gets an integer position p in $[l, r]$, and then all the offers are settled up.

Your total score is the number of coins you are guaranteed to earn, that is, the minimum number of coins you earn over all possible values of p in $[l, r]$. Find the maximum possible score you can guarantee.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains three integers n, l, r ($1 \le n \le 2 \cdot 10^5$, $1 \le l \le r \le 10^9$) — the number of offers and the possible range of Godflex's final ranking.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the bookmaker's estimations of Godflex's ranking in each offer.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single line containing an integer: the maximum possible score you can guarantee.

Samples

Sample input 1	Sample output 1
4 1 1 5 3 2 100 100 50 200 5 1 10 5 7 3 9 1 5 6 10 9 3 1 7 5	0 150 12 13

Explanation of sample 1.
In the first test case, there is only one offer.

- If you ignore the offer, your score is 0;
- if you accept the offer claiming that $p \leq 3$, your score is -2 (reached when $p = 5$, which makes you lose $|5 - 3| = 2$ coins);
- if you accept the offer claiming that $p \geq 3$, your score is -2 (reached when $p = 1$, which makes you lose $|1 - 3| = 2$ coins).

So the maximum possible score is 0.

In the second test case, it is optimal to accept the offers claiming that $p \geq 50$ and $p \leq 200$. Since p must be 100, you gain $|100 - 50| + |100 - 200| = 150$ coins.

C Chamber of Secrets 2

You are playing the game Henry Spotter and the Chamber of Secrets 2.

You want to unlock the next level, the Chamber of Secrets. The entry door contains n panels, each displaying a sequence of m symbols. The product nm is even. The system generates these sequences from a **secret permutation** using the following four-step process:

- first, it starts from a secret permutation $[p_1, p_2, \ldots, p_{nm/2}]$;
- then, it repeats the secret permutation by concatenating it with itself, forming the array $[b_1, b_2, \ldots, b_{nm}]$;
- then, it splits this array into n consecutive blocks, i.e., disjoint subarrays of length m;
- then, it shuffles these blocks in arbitrary order across the panels.

You are given the final n panel sequences produced by the system. The i-th panel shows $[a_{i,1}, a_{i,2}, \ldots, a_{i,m}]$. Your task is to recover one possible original secret permutation $[p_1, p_2, \ldots, p_{nm/2}]$. For the given input, at least one solution exists. If multiple secret permutations are valid, output any one of them.

The concatenation of two arrays $[x_1, x_2, \ldots, x_{k_1}]$, $[y_1, y_2, \ldots, y_{k_2}]$ is the array $[x_1, x_2, \ldots, x_{k_1}, y_1, y_2, \ldots, y_{k_2}]$ of length $k_1 + k_2$.

A permutation of length l is an array consisting of l distinct integers from 1 to l in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($l = 3$ but there is 4 in the array).

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 100$). The description of the test cases follows.

The first line of each test case contains two integers n, m ($1 \le n \le 70$, $1 \le m \le 70$) — the number of panels, and the length of each displayed sequence.

The i-th of the next n lines contains m integers $a_{i,1}, a_{i,2}, \ldots, a_{i,m}$ ($1 \le a_{i,j} \le nm/2$), representing the sequence shown on the i-th panel.

Note that there are no constraints on the sum of n and m over all test cases.

Output

For each test case, output a single line containing a secret permutation $[p_1, p_2, \ldots, p_{nm/2}]$ such that the process described above can produce the n panel sequences $[a_{i,1}, a_{i,2}, \ldots, a_{i,m}]$. For the given input, at least one solution exists.

Samples

Sample input 1	Sample output 1
5 6 2 1 2 3 4 5 6 5 6 3 4 1 2 5 2 1 3 4 1 2 4 5 2 3 5 5 4 4 1 3 2 6 9 5 10 5 10 4 1 3 2 7 8 7 8 6 9 4 3 3 5 2 1 4 6 1 4 6 3 5 2 1 8 3 1 2 4 3 1 2 4	5 6 3 4 1 2 2 4 1 3 5 3 2 7 8 6 9 5 10 4 1 3 5 2 1 4 6 3 1 2 4

Explanation of sample 1.
In the first test case, one valid secret permutation is $[p_1, p_2, \ldots, p_{nm/2}] = [5, 6, 3, 4, 1, 2]$:

- the array $[b_1, b_2, \ldots, b_{nm}]$ is the concatenation of two copies of $[p_1, p_2, \ldots, p_{nm/2}] = [5, 6, 3, 4, 1, 2]$;
- $[b_1, b_2, \ldots, b_{nm}]$ splits into the blocks $[5, 6]$, $[3, 4]$, $[1, 2]$, $[5, 6]$, $[3, 4]$, $[1, 2]$;
- after shuffling, the final panel sequences $[a_{i,1}, a_{i,2}, \ldots, a_{i,m}]$ can coincide with these blocks.

D Dungeon Equilibrium

You are developing a new RPG game where each monster species has a peculiar rule about how many times it should appear on a level.

A level is represented by an array of integers $a_1, a_2, \ldots, a_n$, where each integer represents the type of a monster.

A level is considered balanced if, for every monster type x that appears, there are exactly x monsters of that type. For example, a balanced level might have three monsters of type 3, five of type 5, and so on. An empty level (no monsters at all) is also considered balanced.

Unfortunately, your current level design is not necessarily balanced. You can delete some monsters (i.e., remove elements from the array) to make it balanced.

Given the array $a_1, a_2, \ldots, a_n$, find the minimum number of monsters you need to remove to make the level balanced.

Input

The first line contains an integer n ($1 \leq n \leq 1000$) — the current number of monsters in the level.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \leq a_i \leq n$) — the monster types.

Output

Output a single line containing an integer: the minimum number of monsters you should remove from the level to make it balanced.

Samples

Sample input 1	Sample output 1
5 1 1 2 2 3	2

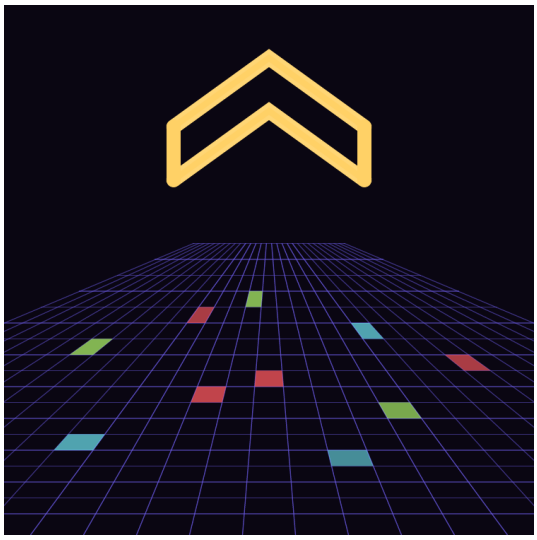
Explanation of sample 1.
In the first example, you can remove one monster of type 1 and one of type 3. The level then becomes $[1, 2, 2]$, which is balanced (it has one monster of type 1 and two of type 2). You cannot make the level balanced with fewer than 2 removals, so the answer is 2.

Sample input 2	Sample output 2
10 1 2 3 2 4 4 4 4 5 2	3

Explanation of sample 2.
In the second example, you can make the level $[1, 2, 4, 4, 4, 4, 2]$.

Image credits: LadyOfHats, Kobolds, public domain, via Wikimedia Commons.

E Expansion Plan 2



You are dealing with side quests in the video game Expansion Plan 2.

There is an infinite grid of **bonus levels**, with coordinates (x, y) (specifically, the cell immediately above $(0, 0)$ is $(0, 1)$, and the cell immediately on the right of $(0, 0)$ is $(1, 0)$). Initially, only the bonus level at $(0, 0)$ is **unlocked**.

Given a string $a_1 a_2 \ldots a_l$ of length l consisting of characters `"4"` and `"8"`, you play l times in a row; at the i-th play you obtain a **score** equal to $s_i \in \{\texttt{"4"}, \texttt{"8"}\}$. For each i from 1 to l:

- if $s_i =$ `"4"`: for each bonus level, if it is orthogonally adjacent (i.e., it shares a side) to a level which was already **unlocked** before the i-th play, it becomes unlocked; otherwise, its state remains the same;
- if $s_i =$ `"8"`: for each bonus level, if it is orthogonally or diagonally adjacent (i.e, it shares a side or a corner) to a level which was already **unlocked** before the i-th play, it becomes unlocked; otherwise, its state remains the same.

You are given a string s of length n consisting of characters `"4"` and `"8"`.

You have to answer q queries. In each query, you start with an infinite grid where only the bonus level $(0, 0)$ is unlocked, and you are given four integers l, r, x, y. You have to determine whether the bonus level (x, y) is unlocked after getting the scores in the substring $[l, r]$ of s.

Input

The first line contains two integers n, q ($1 \le n, q \le 2 \cdot 10^5$) — the length of the string and the number of queries, respectively.

The second line contains a string s of length n consisting of characters `"4"` and `"8"`.

Each of the next q lines contains four integers l, r, x, y ($1 \le l \le r \le n$, $-10^9 \le x, y \le 10^9$), representing a query on the substring $[l, r]$ and the bonus level (x, y).

Output

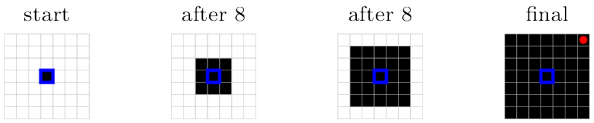
For each query, output `YES` if the bonus level (x, y) is **unlocked** after getting the scores in the substring $[l, r]$ of s, and `NO` otherwise.

The judge is case-insensitive (for example, `YES`, `Yes`, `yes`, `yEs` will all be recognized as positive answers).

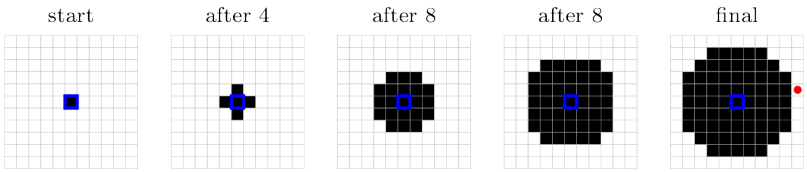
Samples

Sample input 1	Sample output 1
10 6 4884884888 8 10 3 3 4 7 5 1 4 7 3 -3 1 7 -7 -5 1 10 0 0 1 1 1 1	YES NO YES NO YES NO

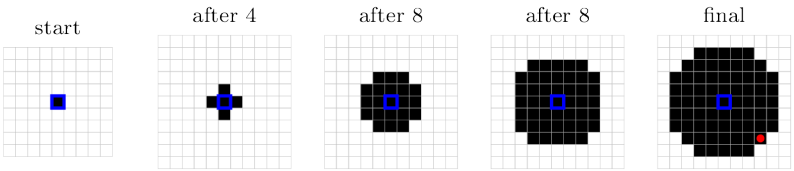
Explanation of sample 1.
The first three queries are illustrated below:



Query 1: substring `"888"`, target $(3, 3)$ (final `YES`)



Query 2: substring `"4884"`, target $(5, 1)$ (final `NO`)

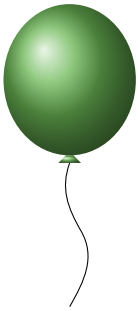


Query 3: substring `"4884"`, target $(3, -3)$ (final `YES`)

In the first query, $[l, r] = [8, 10]$, and $(x, y) = (3, 3)$. The substring $[8, 10]$ of s is `"888"`. After getting the scores in this substring, the bonus level $(3, 3)$ is unlocked, so the answer is `YES`.

In the second query, the bonus level $(5, 1)$ is not unlocked after getting the scores in the substring `"4884"`.

F Factory Table



×	**1**	**2**	**3**	**4**
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

You are playing the sandbox game Mathcraft. Each crafting table of level k can produce all possible products obtained by multiplying two numbers between 1 and k.

If you unroll the k-th crafting table, you get the array $[1\cdot1, 1\cdot2, \ldots, 1\cdot k, 2\cdot1, 2\cdot2, \ldots, 2\cdot k, \ldots, k\cdot1, k\cdot2, \ldots, k\cdot k]$. For example, for $k = 4$, the unrolled table is $[1, 2, 3, 4, 2, 4, 6, 8, 3, 6, 9, 12, 4, 8, 12, 16]$.

Your friend Bob crafted a (contiguous) subarray of one unrolled crafting table. This subarray is $a_1, a_2, \ldots, a_n$.

You want to know how skilled Bob is, so you want to find the minimum possible level of his crafting table. Specifically, you want to determine the smallest k such that $a_1, a_2, \ldots, a_n$ appears as a subarray of the k-th unrolled table.

An array b is a subarray of an array c if b can be obtained from c by deleting several (possibly zero or all) elements from the beginning and several (possibly zero or all) elements from the end.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 500$). The description of the test cases follows.

The first line of each test case contains a single integer n ($2 \le n \le 100$) — the length of the array $a_1, a_2, \ldots, a_n$.

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

Note that there are no constraints on the sum of n over all test cases.

Output

For each test case, output a single line containing an integer: the smallest crafting table level k such that the array $a_1, a_2, \ldots, a_n$ appears as a contiguous subarray of the k-th unrolled table. For the given input, such a k always exists.

Samples

Sample input 1	Sample output 1
4 4 4 6 8 10 6 8 3 6 9 12 4 5 30 40 50 60 70 4 1 2 2 4	5 4 10 2

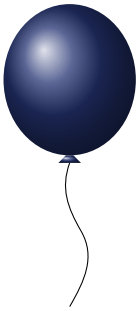
Explanation of sample 1.

In the first test case, the array $[4, 6, 8, 10]$ is a subarray of the 5-th unrolled table, which is $[1, 2, 3, 4, 5, 2, 4, 6, 8, 10, \ldots, 5, 10, 15, 20, 25]$. There is no smaller valid k, so the answer is 5.

In the second test case, the array $[8, 3, 6, 9, 12, 4]$ is a subarray of the 4-th unrolled table, which is $[1, 2, 3, 4, 2, 4, 6, 8, 3, 6, 9, 12, 4, 8, 12, 16]$.

Image credits: Bernard Ladenthin, CC0, via Wikimedia Commons.

G Git Gud



You are an adventurer working for the futuristic corporation RoboCorp. Your initial skill level is an integer in the range $[1, n]$, but you don't know its exact value. Your goal is to reach a skill level of at least n by completing missions for RoboCorp, but each mission costs precious robocoins.

You can choose missions of any difficulty and any positive integer duration (in hours). However, the cost depends both on the new mission's length, and on how the new mission's difficulty compares to your most recent one.

Let x be the difficulty of your most recent mission. If you want to undertake a new mission of difficulty y and duration l:

- If it is your first mission, or $y \leq x$, the cost is l robocoins.
- If $y > x$, the cost is $1000 + l$ robocoins.

Your skill improves only when you take on a mission whose difficulty y exactly matches your current skill s:

- If $y = s$, then your skill increases by l.
- Otherwise, your skill does not change.

After each mission, you still don't know your actual skill value.

You start with 10^6 robocoins. Find a strategy (a sequence of missions) that does not exceed your budget and guarantees your skill becomes at least n, no matter what your initial skill was.

Input

The input contains a single line with an integer n ($1 \leq n \leq 250\,000$) — the target skill level (that is, by the end, your skill must be greater or equal to n).

Output

In the first line, print a single integer k ($0 \leq k \leq 10^6$) — the number of missions you plan to complete.

Then, print k lines. The i-th line must contain two integers y and l ($1 \leq y, l \leq 10^6$) — the difficulty and duration (in hours) of the i-th mission, respectively.

Samples

Sample input 1	Sample output 1
4	4 1 4 3 1 2 1 3 1

Explanation of sample 1.
In the example, your target skill is $n = 4$. You can use the following strategy:

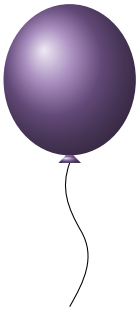
- Take a mission of difficulty $y = 1$ and duration $l = 4$. It's your first mission, so you pay $l = 4$ robocoins.
- Take a mission of difficulty $y = 3$ and duration $l = 1$. The previous mission had difficulty $x = 1$, so since $y > x$, you pay $l + 1000 = 1001$ robocoins.
- Take a mission of difficulty $y = 2$ and duration $l = 1$. Now $x = 3$, and since $y \leq x$, you pay $l = 1$ robocoin.
- Take a mission of difficulty $y = 3$ and duration $l = 1$. Now $x = 2$, and since $y > x$, you pay $l + 1000 = 1001$ robocoins.

In total, you spend 2007 robocoins, which is within your 10^6-robocoin budget.

Now we verify that this strategy always works:

- If your initial skill was 1, after the first mission it becomes 5.
- If your initial skill was 2, after the third mission it becomes 3, and after the fourth mission it becomes 4.
- If your initial skill was 3, after the second mission it becomes 4.
- If your initial skill was 4, it stays 4.

So no matter what your starting skill is, you end with skill $\geq n$.



H Hyper Smawk Bros

You and Bob are playing Hyper Smawk Bros against each other, facing a single boss with health n.

You and Bob act alternately, and you start. On your turn, you may use an attack that deals an integer amount of damage x in $[1, m]$, replacing n with $n - x$. However, you cannot use the same x that your opponent just used on the previous turn (on the very first move of the game, any x in $[1, m]$ is allowed).

The winner is the first player to reduce the boss's health to $n \leq 0$. Determine whether you can force a win if Bob plays optimally.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). The description of the test cases follows.

The only line of each test case contains two integers n, m ($1 \leq n \leq 10^6$, $2 \leq m \leq 10^6$) — the starting health n and the maximum damage per attack m.

Note that there are no constraints on the sum of n over all test cases, and there are no constraints on the sum of m over all test cases.

Output

For each test case, output `YES` if you can force a win against Bob, and `NO` otherwise.

The judge is case-insensitive (for example, `YES`, `Yes`, `yes`, `yEs` will all be recognized as positive answers).

Samples

Sample input 1	Sample output 1
8 6 9 20 10 69 2 42 9 42 10 44 9 44 10 400000 400000	YES YES NO NO YES YES NO YES

Explanation of sample 1.
In the first test case, you can win immediately by dealing damage 8, so that n becomes $6 - 8 = -2 \leq 0$.

In the second test case,

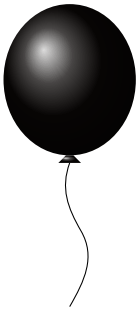
- you choose to deal damage 10;
- Bob can choose to deal any damage in $[1, 10]$ different from 10;
- then you can choose to deal damage 10 and win.

In the third test case,

- either you start by dealing damage 1, then Bob must deal damage 2, then you must deal damage 1, etc.;
- or you start by dealing damage 2, then Bob must deal damage 1, then you must deal damage 2, etc.

In both cases, you lose.

I Isaac's Queries



You have reached the final level of the popular roguelike game "Isaac's Keybindings". Instead of a boss, you encounter a shopkeeper who holds a hidden array of integers $a_1, a_2, \ldots, a_n$, where $0 \le a_i < 2^{30}$ for each i in $[1, n]$.

It is guaranteed that **the array is generated randomly**, i.e., a_i is a uniformly random integer in $[0, 2^{30})$ for each i in $[1, n]$, in all the tests excluding the example.

Let $f(u, v) = a_u \oplus a_{u+1} \oplus \ldots \oplus a_v$, where $\oplus$ is the bitwise `XOR`.

You can ask queries of the following form: `? u v`, with $1 \le u \le v \le n$.

The answer to the query is:

- -1, if $f(u, v) = 0$;
- $\lfloor \log_2(f(u, v)) \rfloor$ otherwise.

Each query has a cost of $\frac{1}{v-u+1}$ robocoins. You initially have 35 robocoins and if your balance ever becomes negative you lose. Note that your robocoin balance does not need to be an integer at any moment.

Find the answer to all possible $\frac{n(n+1)}{2}$ queries without losing.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 30$). The description of the test cases follows.

The first line of each test case contains one integer n ($1 \le n \le 100$) — the length of the array $a_1, a_2, \ldots, a_n$. It is guaranteed that the array is generated randomly in all the tests excluding the example.

There are exactly 50 tests in this problem (including the example). The example has $t = 1$ and $n = 3$, and all the other tests have $t = 30$ and $n = 100$.

Interaction

For each test case, first read a single integer n. If the integer you read is -2, it means that the answer to the previous test case was wrong, and you should exit immediately.

To ask a query, print a line in the format `? u v` with $1 \le u \le v \le n$.

As a response to the query, you will receive -2 if you made an invalid query (i.e., the format is invalid, or you would reach a negative amount of robocoins). In that case, you should exit immediately. Otherwise, you will receive the answer to the query.

When you determine the answers to all the $\frac{n(n+1)}{2}$ queries, print them in the following format.

Print $n+1$ lines. The first line must contain a single character `!`. The i-th of the next n lines must contain $n - i + 1$ integers: the answers to queries `? i i`, `? i i+1`, ..., `? i n`, respectively.

After printing a query, do not forget to output the end of line and flush the output. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `stdout.flush()` in Python;
- see the documentation for other languages.

Samples

Sample input 1	Sample output 1
1 3 2 -1 1	 ? 1 2 ? 1 3 ? 2 3 ! 1 2 -1 2 1 2

Explanation of sample 1.
In the example, the hidden array is $[a_1, a_2, a_3] = [2, 4, 6]$.

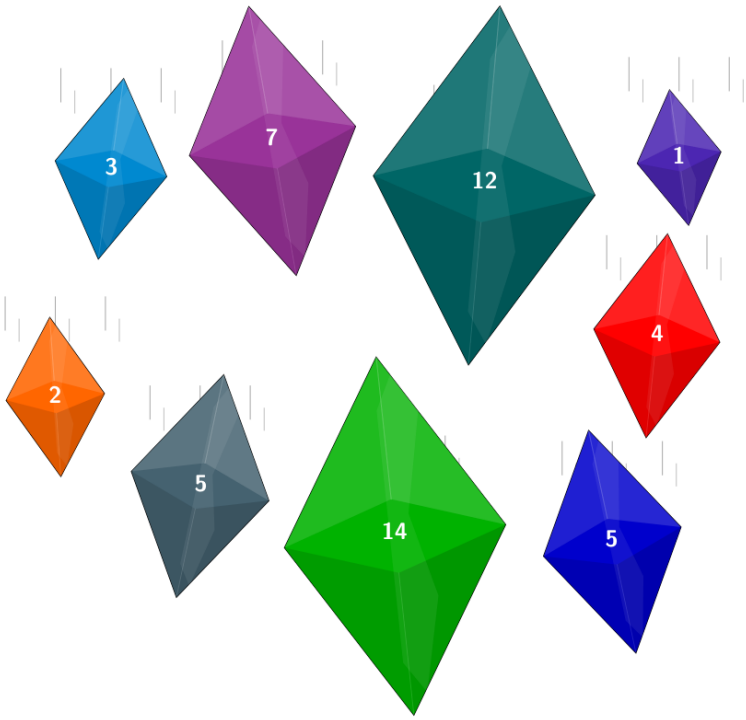
- First, you ask `? 1 2`. Since $f(1,2) = a_1 \oplus a_2 = 6$, the answer is $\lfloor \log_2(6) \rfloor = 2$.
- Then, you ask `? 1 3`. Since $f(1,3) = a_1 \oplus a_2 \oplus a_3 = 0$, the answer is -1.
- Then, you ask `? 2 3`. Since $f(2,3) = a_2 \oplus a_3 = 2$, the answer is $\lfloor \log_2(2) \rfloor = 1$.

Now, even without knowing the hidden array, you claim the answer to all the possible 6 queries. For example, you claim that the answer to `? 1 1` is 1.

You have spent $\frac{1}{2} + \frac{1}{3} + \frac{1}{2} = \frac{4}{3}$ robocoins, which is less than the allowed 35 robocoins.

Image credits: Photo by Michael Ocampo, CC BY 2.0, modified to remove background, via Wikimedia Commons

J Jewels Building



You are playing a fantasy game where you start with a row of n power crystals. The i-th crystal has energy level a_i.

You can perform the following operation any number of times:

- choose a consecutive group of identical crystals, that is, choose l and r ($1 \le l \le r \le |a|$) such that $a_l = a_{l+1} = \ldots = a_r$;
- fuse them into a single crystal whose energy becomes $r - l + 1$, obtaining the new sequence $[a_1, \ldots, a_{l-1}, r - l + 1, a_{r+1}, \ldots, a_{|a|}]$.

Note that you may also choose $l = r$.

You want to craft a specific configuration of crystals with energy levels $b_1, \ldots, b_m$. Determine whether it is possible.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 500$). The description of the test cases follows.

The first line of each test case contains two integers n, m ($1 \le m \le n \le 4000$) — the number of crystals in the initial and target configurations, respectively.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the energy levels of the initial crystals.

The third line of each test case contains m integers $b_1, b_2, \ldots, b_m$ ($1 \le b_i \le 10^9$) — the desired energy levels of the target crystals.

It is guaranteed that the sum of n over all test cases does not exceed 4000.

Output

For each test case, output `YES` if you can transform the initial configuration into the target one, and `NO` otherwise.

The judge is case-insensitive (for example, `YES`, `Yes`, `yes`, `yEs` will all be recognized as positive answers).

Samples

Sample input 1	Sample output 1
3 5 1 2 4 4 2 3 2 5 2 2 4 4 2 3 4 4 1 1 2 1	YES NO YES

Explanation of sample 1.
In the first test case:

- the initial configuration is $[2, 4, 4, 2, 3]$;
- after fusing the two crystals in the subarray $[l, r] = [2, 3]$, the configuration becomes $[2, 2, 2, 3]$;
- after fusing crystals in the subarray $[l, r] = [1, 3]$, the configuration becomes $[3, 3]$;
- after fusing crystals in the subarray $[l, r] = [1, 2]$, the configuration becomes $[2] = [b_1]$. So the answer is `YES`.

In the second test case, it is not possible to obtain $[4, 4]$ starting from $[2, 4, 4, 2, 3]$, so the answer is `NO`.

K Keygen 3

Luke has just discovered a new 5D kart videogame. However, the software is not free: in order to activate it, you need to provide a license key.

Luke has found out that, in order to be valid, a license key must be a permutation of length n with both of the following properties:

- it is bitonic;
- it has m cycles.

Let k denote the number of license keys (i.e., the number of permutations satisfying the above conditions). Luke is an altruistic hacker, so he wants to provide distinct license keys for all his 2000 friends. However, k might be less than 2000.

Help Luke by printing $r = \min(k, 2000)$ distinct valid license keys.

A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

A permutation $p_1, p_2, \ldots, p_n$ is bitonic if there exists an index i ($1 \leq i \leq n$) such that

- $p_{j-1} \leq p_j$ for $2 \leq j \leq i$;
- $p_j \geq p_{j+1}$ for $i \leq j \leq n - 1$.

A cycle is a subset $C \subseteq \{1, 2, \ldots, n\}$ such that

- C is non-empty;
- if $x \in C$, then $p_x \in C$;
- C is minimal, i.e., there does not exist a cycle C' such that $C' \subsetneq C$.

Input

The input consists of a single line containing two integers n, m ($1 \leq m \leq n \leq 100$) — the length of the permutations and the target number of cycles.

Output

In the first line, print a single integer r: the number of permutations you are going to print. Note that r must be $\min(k, 2000)$ as defined in the statement.

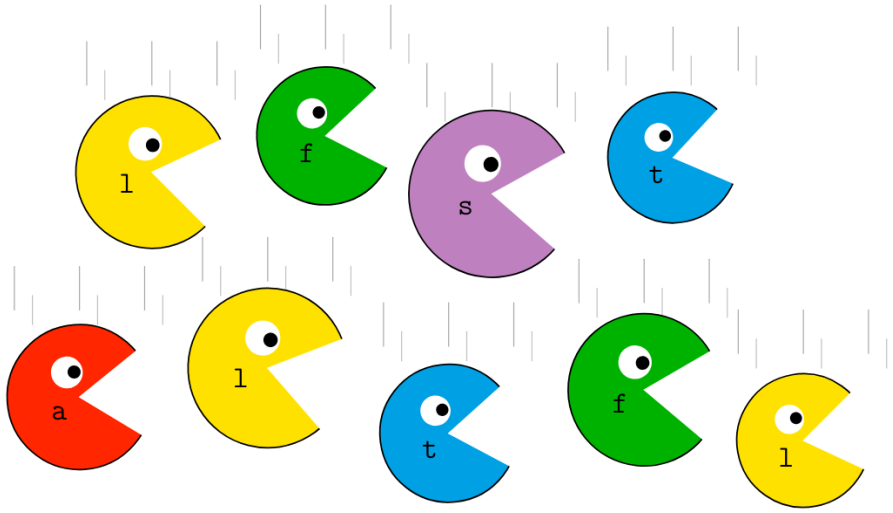
Then, print r lines. Each line must contain a bitonic permutation of length n, with m cycles. The permutations must be distinct.

Samples

Sample input 1	Sample output 1
6 3	9 1 4 5 6 3 2 6 5 4 3 2 1 1 2 4 5 6 3 1 2 5 6 4 3 1 3 4 6 5 2 1 5 6 4 3 2 3 5 6 4 2 1 1 3 6 5 4 2 2 6 5 4 3 1

Explanation of sample 1.
In the example, there are 9 valid license keys (i.e., bitonic permutations of length 6, with 3 cycles). For example, $[3, 5, 6, 4, 2, 1]$ is bitonic (in the above definition, $i = 3$), and it has 3 cycles: $\{1, 3, 6\}$, $\{2, 5\}$, $\{4\}$. So you have to print $r = \min(9, 2000) = 9$ such permutations.

L LFS



You are designing the new videogame Live Fight Simulator. A level is defined by a string s of length n consisting of lowercase English letters, where each letter represents a type of enemy that appears in sequence.

To analyze the gameplay balance, you need to measure how repetitive different parts of the level are. You will consider q specific contiguous segments $s[l, r]$ of the level, with $1 \leq l \leq r \leq n$.

For each of these queries, you want to compute the length of the LFS (Longest Frequent Substring). Formally, for any string t:

- let $f(t)$ be the maximum frequency of any substring in t;
- let $|\text{LFS}(t)|$ be the maximum length of a substring of t that appears exactly $f(t)$ times.

For each query (l, r), you must compute $|\text{LFS}(s[l, r])|$, which represents the maximum length among the most repetitive patterns of enemy spawns in that part of the level.

A string x is a substring of a string y if x can be obtained from y by the deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

Input

The first line contains two integers n, q ($1 \leq n \leq 5 \cdot 10^5$, $1 \leq q \leq 5 \cdot 10^5$) — the length of the sequence and the number of level segments to analyze.

The second line contains a string s of length n consisting of lowercase English letters.

Each of the next q lines contains two integers l, r ($1 \leq l \leq r \leq n$), representing a query on the substring $s[l, r]$.

Output

Print q lines. The i-th line must contain a single integer: the value of $|\text{LFS}(s[l, r])|$ for the pair (l, r).

Samples

Sample input 1	Sample output 1
5 4 ababa 1 1 1 5 1 4 2 5	1 1 2 2

Explanation of sample 1.
In the first example:

- In the first query, the substring is $t = s[1, 1] =$ "a". The maximum frequency of a substring inside "a" is 1, reached by "a" itself, whose length is 1. Therefore, the answer is 1.
- In the second query, the substring is $t = s[1, 5] =$ "ababa". The maximum frequency of a substring inside "ababa" is 3, reached by "a", whose length is 1. Therefore, the answer is 1.
- In the third query, the substring is $t = s[1, 4] =$ "abab". The maximum frequency of a substring inside "abab" is 2, reached by "a", "b" and "ab". Among these strings, the one with maximum length is "ab", whose length is 2. Therefore, the answer is 2.
- In the fourth query, the substring is $t = s[2, 5] =$ "baba". The maximum frequency of a substring inside "baba" is 2, reached by "a", "b" and "ba". Among these strings, the one with maximum length is "ba", whose length is 2. Therefore, the answer is 2.

Sample input 2	Sample output 2
10 1 aaaaaaaaaa 1 10	1

Sample input 3	Sample output 3
17 5 deabcabcabdeadede 1 8 1 10 4 9 2 16 1 17	3 2 3 1 2