Educational Codeforces Round 97 (Rated for Div. 2)

A. Marketing Scheme

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

You got a job as a marketer in a pet shop, and your current task is to boost sales of cat food. One of the strategies is to sell cans of food in packs with discounts.

Suppose you decided to sell packs with a cans in a pack with a discount and some customer wants to buy x cans of cat food. Then he follows a greedy strategy:

- he buys $\left\lfloor \frac{x}{a} \right\rfloor$ packs with a discount;
- then he wants to buy the remaining $(x \bmod a)$ cans one by one.

$\left\lfloor \frac{x}{a} \right\rfloor$ *is* x *divided by* a *rounded down,* $x \bmod a$ *is the remainer of* x *divided by* a.

But customers are greedy in general, so if the customer wants to buy $(x \bmod a)$ cans one by one and it happens that $(x \bmod a) \geq \frac{a}{2}$ he decides to buy the whole pack of a cans (instead of buying $(x \bmod a)$ cans). It makes you, as a marketer, happy since the customer bought more than he wanted initially.

You know that each of the customers that come to your shop can buy any number of cans from l to r inclusive. Can you choose such size of pack a that each customer buys more cans than they wanted initially?

Input

The first line contains a single integer t ($1 \leq t \leq 1000$) — the number of test cases.

The first and only line of each test case contains two integers l and r ($1 \leq l \leq r \leq 10^9$) — the range of the number of cans customers can buy.

Output

For each test case, print YES if you can choose such size of pack a that each customer buys more cans than they wanted initially. Otherwise, print NO.

You can print each character in any case.

Example

input
3 3 4 1 2 120 150
output
YES NO YES

Note

In the first test case, you can take, for example, $a = 5$ as the size of the pack. Then if a customer wants to buy 3 cans, he'll buy 5 instead ($3 \bmod 5 = 3$, $\frac{5}{2} = 2.5$). The one who wants 4 cans will also buy 5 cans.

In the second test case, there is no way to choose a.

In the third test case, you can take, for example, $a = 80$.

B. Reverse Binary Strings

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

You are given a string s of even length n. String s is binary, in other words, consists only of 0's and 1's.

String s has exactly $\frac{n}{2}$ zeroes and $\frac{n}{2}$ ones (n is even).

In one operation you can reverse any substring of s. A substring of a string is a contiguous subsequence of that string.

What is the minimum number of operations you need to make string s *alternating*? A string is alternating if $s_i \neq s_{i+1}$ for all i. There are two types of alternating strings in general: `01010101...` or `10101010...`

Input
The first line contains a single integer t ($1 \leq t \leq 1000$) — the number of test cases.

The first line of each test case contains a single integer n ($2 \leq n \leq 10^5$; n is even) — the length of string s.

The second line of each test case contains a binary string s of length n ($s_i \in$ {`0`, `1`}). String s has exactly $\frac{n}{2}$ zeroes and $\frac{n}{2}$ ones.

It's guaranteed that the total sum of n over test cases doesn't exceed 10^5.

Output
For each test case, print the minimum number of operations to make s alternating.

Example

input
3 2 10 4 0110 8 11101000
output
0 1 2

Note
In the first test case, string `10` is already alternating.

In the second test case, we can, for example, reverse the last two elements of s and get: `01`<u>`10`</u> $\rightarrow$ `0101`.

In the third test case, we can, for example, make the following two operations:

1. `1`<u>`1101`</u>`000` $\rightarrow$ `10101100`;
2. `10101`<u>`10`</u>`0` $\rightarrow$ `10101010`.

C. Chef Monocarp

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

Chef Monocarp has just put n dishes into an oven. He knows that the i-th dish has its optimal cooking time equal to t_i minutes.

At any **positive integer** minute T Monocarp can put **no more than one** dish out of the oven. If the i-th dish is put out at some minute T, then its unpleasant value is $|T - t_i|$ — the absolute difference between T and t_i. Once the dish is out of the oven, it can't go back in.

Monocarp should put all the dishes out of the oven. What is the minimum total unpleasant value Monocarp can obtain?

Input
The first line contains a single integer q ($1 \leq q \leq 200$) — the number of testcases.

Then q testcases follow.

The first line of the testcase contains a single integer n ($1 \leq n \leq 200$) — the number of dishes in the oven.

The second line of the testcase contains n integers $t_1, t_2, \ldots, t_n$ ($1 \leq t_i \leq n$) — the optimal cooking time for each dish.

The sum of n over all q testcases doesn't exceed 200.

Output
Print a single integer for each testcase — the minimum total unpleasant value Monocarp can obtain when he puts out all the dishes out of the oven. Remember that Monocarp can only put the dishes out at positive integer minutes and no more than one dish at any minute.

Example

input
6 6 4 2 4 4 5 2 7 7 7 7 7 7 7 7 1

```
1
5
5 1 2 4 3
4
1 4 4 4
21
21 8 1 4 1 5 21 1 8 21 11 21 11 3 12 8 19 15 9 11 13
```

output

```
4
12
0
0
2
21
```

Note

In the first example Monocarp can put out the dishes at minutes $3, 1, 5, 4, 6, 2$. That way the total unpleasant value will be $|4 - 3| + |2 - 1| + |4 - 5| + |4 - 4| + |6 - 5| + |2 - 2| = 4$.

In the second example Monocarp can put out the dishes at minutes $4, 5, 6, 7, 8, 9, 10$.

In the third example Monocarp can put out the dish at minute 1.

In the fourth example Monocarp can put out the dishes at minutes $5, 1, 2, 4, 3$.

In the fifth example Monocarp can put out the dishes at minutes $1, 3, 4, 5$.

D. Minimal Height Tree

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

Monocarp had a tree which consisted of n vertices and was rooted at vertex 1. He decided to study BFS (Breadth-first search), so he ran BFS on his tree, starting from the root. BFS can be described by the following pseudocode:

```
a = [] # the order in which vertices were processed
q = Queue()
q.put(1) # place the root at the end of the queue
while not q.empty():
    k = q.pop() # retrieve the first vertex from the queue
    a.append(k) # append k to the end of the sequence in which vertices were visited
    for y in g[k]: # g[k] is the list of all children of vertex k, sorted in ascending order
        q.put(y)
```

Monocarp was fascinated by BFS so much that, in the end, he lost his tree. Fortunately, he still has a sequence of vertices, in which order vertices were visited by the BFS algorithm (the array a from the pseudocode). Monocarp knows that each vertex was visited exactly once (since they were put and taken from the queue exactly once). Also, he knows that all children of each vertex were viewed *in ascending order*.

Monocarp knows that there are many trees (in the general case) with the same visiting order a, so he doesn't hope to restore his tree. Monocarp is okay with any tree that **has minimum height**.

The height of a tree is the maximum depth of the tree's vertices, and the depth of a vertex is the number of edges in the path from the root to it. For example, the depth of vertex 1 is 0, since it's the root, and the depth of all root's children are 1.

Help Monocarp to find any tree with given visiting order a and minimum height.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains a single integer n ($2 \le n \le 2 \cdot 10^5$) — the number of vertices in the tree.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$; $a_i \neq a_j$; $a_1 = 1$) — the order in which the vertices were visited by the BFS algorithm.

It's guaranteed that the total sum of n over test cases doesn't exceed $2 \cdot 10^5$.

Output

For each test case print the minimum possible height of a tree with the given visiting order a.

Example

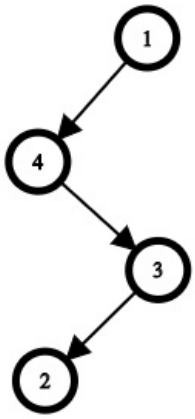
input

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

3 1 2 3
output
3 1 1

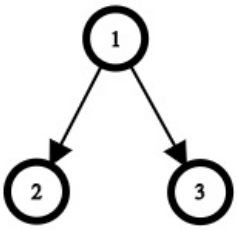
Note

In the first test case, there is only one tree with the given visiting order:



In the second test case, there is only one tree with the given visiting order as well:

In the third test case, an optimal tree with the given visiting order is shown below:



E. Make It Increasing

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

You are given an array of n integers a_1, a_2, ..., a_n, and a set b of k distinct integers from 1 to n.

In one operation, you may choose two integers i and x ($1 \le i \le n$, x can be any integer) and assign $a_i := x$. This operation can be done only if i does not belong to the set b.

Calculate the minimum number of operations you should perform so the array a is increasing (that is, $a_1 < a_2 < a_3 < \cdots < a_n$), or report that it is impossible.

Input

The first line contains two integers n and k ($1 \le n \le 5 \cdot 10^5$, $0 \le k \le n$) — the size of the array a and the set b, respectively.

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

Then, if $k \neq 0$, the third line follows, containing k integers b_1, b_2, ..., b_k ($1 \le b_1 < b_2 < \cdots < b_k \le n$). **If $k = 0$, this line is skipped**.

Output

If it is impossible to make the array a increasing using the given operations, print -1.

Otherwise, print one integer — the minimum number of operations you have to perform.

Examples

input
7 2 1 2 1 1 3 5 1 3 5
output
4

input
3 3 1 3 2 1 2 3
output
-1

input
5 0 4 3 1 2 3
output
2

input
10 3 1 3 5 6 12 9 8 10 13 15 2 4 9
output
3

F. Emotional Fishermen

time limit per test: 4 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

n fishermen have just returned from a fishing vacation. The i-th fisherman has caught a fish of weight a_i.

Fishermen are going to show off the fish they caught to each other. To do so, they firstly choose an order in which they show their fish (each fisherman shows his fish exactly once, so, formally, the order of showing fish is a permutation of integers from 1 to n). Then they show the fish they caught according to the chosen order. When a fisherman shows his fish, he might either become happy, become sad, or stay content.

Suppose a fisherman shows a fish of weight x, and the maximum weight of a previously shown fish is y ($y = 0$ if that fisherman is the first to show his fish). Then:

- if $x \geq 2y$, the fisherman becomes happy;
- if $2x \leq y$, the fisherman becomes sad;
- if none of these two conditions is met, the fisherman stays content.

Let's call an order in which the fishermen show their fish *emotional* if, after all fishermen show their fish according to this order, each fisherman becomes either happy or sad. Calculate the number of *emotional* orders modulo 998244353.

Input

The first line contains one integer n ($2 \leq n \leq 5000$).

The second line contains n integers a_1, a_2, ..., a_n ($1 \leq a_i \leq 10^9$).

Output

Print one integer — the number of *emotional* orders, taken modulo 998244353.

Examples

input
4 1 1 4 9
output
20

input
4 4 3 2 1

output
0

input
3 4 2 1
output
6

input
8 42 1337 13 37 420 666 616 97
output
19200

G. Death DBMS

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

For the simplicity, let's say that the "Death Note" is a notebook that kills a person when their name is written in it.

It's easy to kill with it, but it's pretty hard to keep track of people you haven't killed and still plan to. You decided to make a "Death Database Management System" — a computer program that provides the easy access to the database of possible victims. Let me describe its specifications to you.

Let's define a victim entity: a victim has a name (not necessarily unique) that consists only of lowercase Latin letters and an integer suspicion value.

At the start of the program the user enters a list of n victim names into a database, each suspicion value is set to 0.

Then the user makes queries of two types:

- $1\ i\ x$ — set the suspicion value of the i-th victim to x;
- $2\ q$ — given a string q find the maximum suspicion value of a victim whose name is a contiguous substring of q.

Just to remind you, this program doesn't kill people, it only helps to search for the names to write down in an actual notebook. Thus, the list of the victims in the database doesn't change throughout the queries.

What are you waiting for? Write that program now!

Input

The first line contains two integers n and m ($1 \le n, m \le 3 \cdot 10^5$) — the number of victims and the number of queries, respectively.

Each of the next n lines contains a single string s_i — the name of the i-th victim. Each name consists only of lowercase Latin letters.

Each of the next m lines contains a query of one of two types:

- $1\ i\ x$ ($1 \le i \le n$, $0 \le x \le 10^9$) — change the suspicion value of the i-th victim to x;
- $2\ q$ — given a string q consisting only of lowercase Latin letters find the maximum suspicion value of a victim whose name is a contiguous substring of q.

There is at least one query of the second type. The total length of the strings s_i doesn't exceed $3 \cdot 10^5$. The total length of the strings q doesn't exceed $3 \cdot 10^5$.

Output

For each query of the second type print an integer value. If there is no victim name that is a contiguous substring of q, then print -1. Otherwise, print the maximum suspicion value of a victim whose name is a contiguous substring of q.

Examples

input
5 8 kurou takuo takeshi naomi shingo 2 nakiraomi 2 abanaomicaba 1 3 943 2 takuotakeshishingo 1 5 135832 2 shingotakeshi

1 5 0
2 shingotakeshi

output

-1
0
943
135832
943

input

6 15
a
ab
ba
b
a
ba
2 aa
1 4 4
2 bbb
1 2 1
1 2 18
2 b
2 c
1 6 10
2 aba
2 abbbba
1 2 12
2 bbaaab
1 1 11
1 5 5
2 baa

output

0
4
4
-1
18
18
12
11