A. Short Substrings

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

Alice guesses the strings that Bob made for her.

At first, Bob came up with the secret string a consisting of lowercase English letters. The string a has a length of 2 or more characters. Then, from string a he builds a new string b and offers Alice the string b so that she can guess the string a.

Bob builds b from a as follows: he writes all the substrings of length 2 of the string a in the order from left to right, and then joins them in the same order into the string b.

For example, if Bob came up with the string a="abac", then all the substrings of length 2 of the string a are: "ab", "ba", "ac". Therefore, the string b="abbaac".

You are given the string b. Help Alice to guess the string a that Bob came up with. It is guaranteed that b was built according to the algorithm given above. It can be proved that the answer to the problem is unique.

Input

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

Each test case consists of one line in which the string b is written, consisting of lowercase English letters ($2 \le |b| \le 100$) — the string Bob came up with, where $|b|$ is the length of the string b. It is guaranteed that b was built according to the algorithm given above.

Output

Output t answers to test cases. Each answer is the secret string a, consisting of lowercase English letters, that Bob came up with.

Example

input

```
4
abbaac
ac
bccddaaf
zzzzzzzzzz
```

output

```
abac
ac
bcdaf
zzzzzz
```

Note

The first test case is explained in the statement.

In the second test case, Bob came up with the string a="ac", the string a has a length 2, so the string b is equal to the string a.

In the third test case, Bob came up with the string a="bcdaf", substrings of length 2 of string a are: "bc", "cd", "da", "af", so the string b="bccddaaf".

B. Even Array

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

You are given an array $a[0 \dots n-1]$ of length n which consists of non-negative integers. **Note that array indices start from zero.**

An array is called *good* if the parity of each index matches the parity of the element at that index. More formally, an array is good if for all i ($0 \le i \le n-1$) the equality $i \bmod 2 = a[i] \bmod 2$ holds, where $x \bmod 2$ is the remainder of dividing x by 2.

For example, the arrays $[0, 5, 2, 1]$ and $[0, 17, 0, 3]$ are good, and the array $[2, 4, 6, 7]$ is bad, because for $i = 1$, the parities of i and $a[i]$ are different: $i \bmod 2 = 1 \bmod 2 = 1$, but $a[i] \bmod 2 = 4 \bmod 2 = 0$.

In one move, you can take **any** two elements of the array and swap them (these elements are not necessarily adjacent).

Find the minimum number of moves in which you can make the array a good, or say that this is not possible.

Input

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

Each test case starts with a line containing an integer n ($1 \le n \le 40$) — the length of the array a.

The next line contains n integers $a_0, a_1, \ldots, a_{n-1}$ ($0 \le a_i \le 1000$) — the initial array.

Output

For each test case, output a single integer — the minimum number of moves to make the given array a good, or -1 if this is not possible.

Example

input

```
4
4
3 2 7 6
3
3 2 6
1
7
7
4 9 2 1 18 3 0
```

output

```
2
1
-1
0
```

Note

In the first test case, in the first move, you can swap the elements with indices 0 and 1, and in the second move, you can swap the elements with indices 2 and 3.

In the second test case, in the first move, you need to swap the elements with indices 0 and 1.

In the third test case, you cannot make the array good.

C. Social Distance

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

Polycarp and his friends want to visit a new restaurant. The restaurant has n tables arranged along a straight line. People are already sitting at some tables. The tables are numbered from 1 to n in the order from left to right. The state of the restaurant is described by a string of length n which contains characters "1" (the table is occupied) and "0" (the table is empty).

Restaurant rules prohibit people to sit at a distance of k or less from each other. That is, if a person sits at the table number i, then all tables with numbers from $i-k$ to $i+k$ (except for the i-th) should be free. In other words, the absolute difference of the numbers of any two occupied tables must be strictly greater than k.

For example, if $n=8$ and $k=2$, then:

- strings "10010001", "10000010", "00000000", "00100000" satisfy the rules of the restaurant;
- strings "10100100", "10011001", "11111111" do not satisfy to the rules of the restaurant, since each of them has a pair of "1" with a distance less than or equal to $k=2$.

In particular, if the state of the restaurant is described by a string without "1" or a string with one "1", then the requirement of the restaurant is satisfied.

You are given a binary string s that describes the current state of the restaurant. It is guaranteed that the rules of the restaurant are satisfied for the string s.

Find the maximum number of free tables that you can occupy so as not to violate the rules of the restaurant. Formally, what is the maximum number of "0" that can be replaced by "1" such that the requirement will still be satisfied?

For example, if $n=6$, $k=1$, $s=$ "100010", then the answer to the problem will be 1, since only the table at position 3 can be occupied such that the rules are still satisfied.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases in the test. Then t test cases follow.

Each test case starts with a line containing two integers n and k ($1 \le k \le n \le 2 \cdot 10^5$) — the number of tables in the restaurant and the minimum allowed distance between two people.

The second line of each test case contains a binary string s of length n consisting of "0" and "1" — a description of the free and occupied tables in the restaurant. The given string satisfy to the rules of the restaurant — the difference between indices of any two "1" is more than k.

The sum of n for all test cases in one test does not exceed $2 \cdot 10^5$.

Output

For each test case output one integer — the number of tables that you can occupy so as not to violate the rules of the restaurant. If additional tables cannot be taken, then, obviously, you need to output 0.

Example

input

```
6
6 1
100010
6 2
000000
5 1
10101
3 1
001
2 2
00
1 1
0
```

output

```
1
2
0
1
1
1
```

Note

The first test case is explained in the statement.

In the second test case, the answer is 2, since you can choose the first and the sixth table.

In the third test case, you cannot take any free table without violating the rules of the restaurant.

D. Task On The Board

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

Polycarp wrote on the board a string s containing only lowercase Latin letters ('a'-'z'). This string is known for you and given in the input.

After that, he erased some letters from the string s, and he rewrote the remaining letters in **any** order. As a result, he got some new string t. You have to find it with some additional information.

Suppose that the string t has length m and the characters are numbered from left to right from 1 to m. You are given a sequence of m integers: $b_1, b_2, \ldots, b_m$, where b_i is the sum of the distances $|i-j|$ from the index i to all such indices j that $t_j > t_i$ (consider that 'a'<'b'<...<'z'). In other words, to calculate b_i, Polycarp finds all such indices j that the index j contains a letter that is later in the alphabet than t_i and sums all the values $|i-j|$.

For example, if t = "abzb", then:

- since t_1='a', all other indices contain letters which are later in the alphabet, that is: $b_1 = |1-2| + |1-3| + |1-4| = 1+2+3 = 6$;
- since t_2='b', only the index $j=3$ contains the letter, which is later in the alphabet, that is: $b_2 = |2-3| = 1$;
- since t_3='z', then there are no indexes j such that $t_j > t_i$, thus $b_3 = 0$;
- since t_4='b', only the index $j=3$ contains the letter, which is later in the alphabet, that is: $b_4 = |4-3| = 1$.

Thus, if t = "abzb", then $b = [6, 1, 0, 1]$.

Given the string s and the array b, find any possible string t for which the following two requirements are fulfilled simultaneously:

- t is obtained from s by erasing some letters (possibly zero) and then writing the rest in **any** order;
- the array, constructed from the string t according to the rules above, equals to the array b specified in the input data.

Input

The first line contains an integer q ($1 \le q \le 100$) — the number of test cases in the test. Then q test cases follow.

Each test case consists of three lines:

- the first line contains string s, which has a length from 1 to 50 and consists of lowercase English letters;
- the second line contains positive integer m ($1 \le m \le |s|$), where $|s|$ is the length of the string s, and m is the length of the array b;
- the third line contains the integers $b_1, b_2, \ldots, b_m$ ($0 \le b_i \le 1225$).

It is guaranteed that in each test case an answer exists.

Output

Output q lines: the k-th of them should contain the answer (string t) to the k-th test case. It is guaranteed that an answer to each test case exists. If there are several answers, output any.

Example

input

```
4
abac
3
2 1 0
abc
1
0
abba
3
1 0 1
ecoosdcefr
10
38 13 24 14 11 5 3 24 17 0
```

output

```
aac
b
aba
codeforces
```

Note

In the first test case, such strings t are suitable: "aac', "aab".

In the second test case, such trings t are suitable: "a", "b", "c".

In the third test case, only the string t equals to "aba" is suitable, but the character 'b' can be from the second or third position.

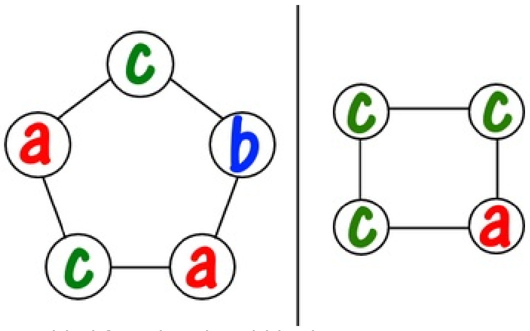
E. Necklace Assembly

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

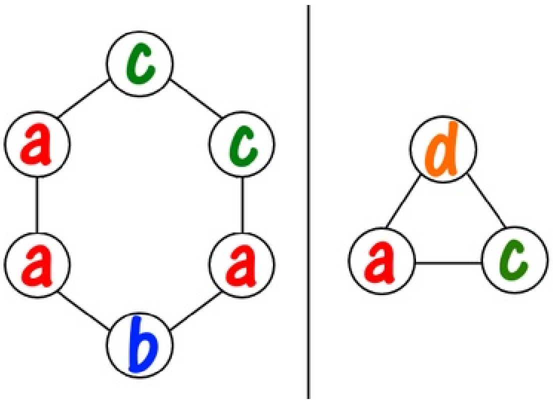
The store sells n beads. The color of each bead is described by a lowercase letter of the English alphabet ("a"–"z"). You want to buy some beads to assemble a necklace from them.

A necklace is a set of beads connected in a circle.

For example, if the store sells beads "a", "b", "c", "a", "c", "c", then you can assemble the following necklaces (these are not all possible options):

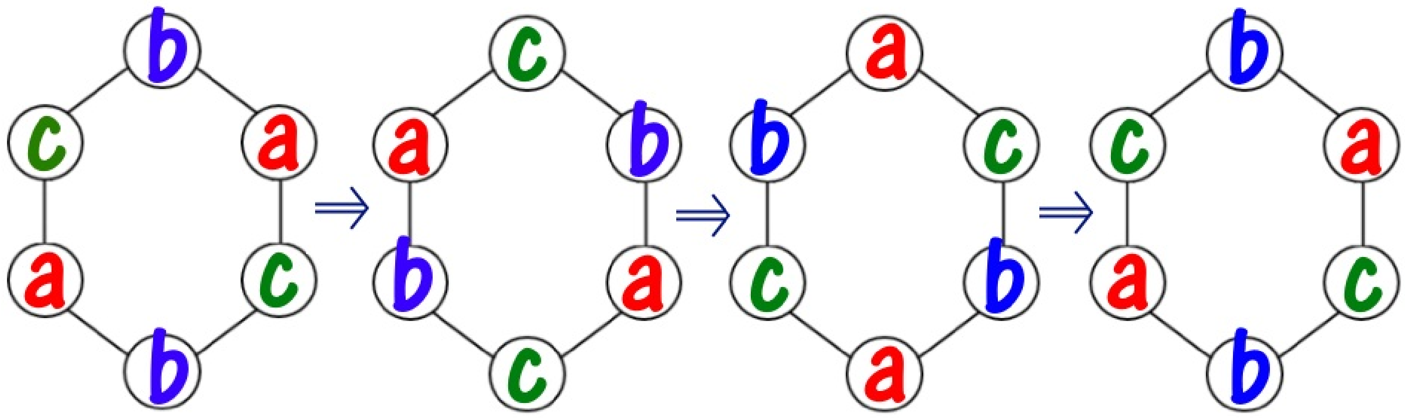


And the following necklaces cannot be assembled from beads sold in the store:



The first necklace cannot be assembled because it has three beads "a" (of the two available). The second necklace cannot be assembled because it contains a bead "d", which is not sold in the store.

We call a necklace k-beautiful if, when it is turned clockwise by k beads, the necklace remains unchanged. For example, here is a sequence of three turns of a necklace.



As you can see, this necklace is, for example, 3-beautiful, 6-beautiful, 9-beautiful, and so on, but it is not 1-beautiful or 2-beautiful. In particular, a necklace of length 1 is k-beautiful for any integer k. A necklace that consists of beads of the same color is also beautiful for any k.

You are given the integers n and k, and also the string s containing n lowercase letters of the English alphabet — each letter defines a bead in the store. You can buy any subset of beads and connect them in any order. Find the maximum length of a k-beautiful necklace you can assemble.

Input

The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases in the test. Then t test cases follow.

The first line of each test case contains two integers n and k ($1 \le n, k \le 2000$).

The second line of each test case contains the string s containing n lowercase English letters — the beads in the store.

It is guaranteed that the sum of n for all test cases does not exceed 2000.

Output

Output t answers to the test cases. Each answer is a positive integer — the maximum length of the k-beautiful necklace you can assemble.

Example

input

```
6
6 3
abcbac
3 6
aaa
7 1000
abczgyo
5 4
ababa
20 10
aaebdbabdbbddaadaadc
20 5
ecbedececacbcbccbdec
```

output

```
6
3
5
4
15
10
```

Note

The first test case is explained in the statement.

In the second test case, a 6-beautiful necklace can be assembled from all the letters.

In the third test case, a 1000-beautiful necklace can be assembled, for example, from beads "`abzyo`".

F1. Flying Sort (Easy Version)

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

This is an easy version of the problem. In this version, all numbers in the given array are distinct and the constraints on n are less than in the hard version of the problem.

You are given an array a of n integers **(there are no equals elements in the array)**. You can perform the following operations on array elements:

1. choose any index i ($1 \le i \le n$) and move the element $a[i]$ to the **begin** of the array;

2. choose any index i ($1 \le i \le n$) and move the element $a[i]$ to the **end** of the array.

For example, if $n = 5$, $a = [4, 7, 2, 3, 9]$, then the following sequence of operations can be performed:

- after performing the operation of the first type to the second element, the array a will become $[7, 4, 2, 3, 9]$;
- after performing the operation of the second type to the second element, the array a will become $[7, 2, 3, 9, 4]$.

You can perform operations of any type any number of times in any order.

Find the minimum total number of operations of the first and second type that will make the a array sorted in non-decreasing order. In other words, what is the minimum number of operations that must be performed so the array satisfies the inequalities $a[1] \le a[2] \le \ldots \le a[n]$.

Input

The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases in the test. Then t test cases follow.

Each test case starts with a line containing an integer n ($1 \le n \le 3000$) — length of the array a.

Then follow n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — an array that needs to be sorted by the given operations. **All numbers in the given array are distinct**.

The sum of n for all test cases in one test does not exceed 3000.

Output

For each test case output one integer — the minimum total number of operations of the first and second type, which will make the array sorted in non-decreasing order.

Example

input

```
4
5
4 7 2 3 9
5
3 5 8 1 7
5
1 4 5 7 12
4
0 2 1 3
```

output

```
2
2
0
2
```

Note

In the first test case, you first need to move 3, and then 2 to the beginning of the array. Therefore, the desired sequence of operations: $[4, 7, 2, 3, 9] \rightarrow [3, 4, 7, 2, 9] \rightarrow [2, 3, 4, 7, 9]$.

In the second test case, you need to move the 1 to the beginning of the array, and the 8 — to the end. Therefore, the desired sequence of operations: $[3, 5, 8, 1, 7] \rightarrow [1, 3, 5, 8, 7] \rightarrow [1, 3, 5, 7, 8]$.

In the third test case, the array is already sorted.

F2. Flying Sort (Hard Version)

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

This is a hard version of the problem. In this version, the given array can contain equal elements and the constraints on n are greater than in the easy version of the problem.

You are given an array a of n integers **(the given array can contain equal elements)**. You can perform the following operations on array elements:

1. choose any index i ($1 \le i \le n$) and move the element $a[i]$ to the **begin** of the array;
2. choose any index i ($1 \le i \le n$) and move the element $a[i]$ to the **end** of the array.

For example, if $n = 5$, $a = [4, 7, 2, 2, 9]$, then the following sequence of operations can be performed:

- after performing the operation of the first type to the second element, the array a will become $[7, 4, 2, 2, 9]$;
- after performing the operation of the second type to the second element, the array a will become $[7, 2, 2, 9, 4]$.

You can perform operations of any type any number of times in any order.

Find the minimum total number of operations of the first and second type that will make the a array sorted in non-decreasing order. In other words, what is the minimum number of operations must be performed so the array satisfies the inequalities $a[1] \le a[2] \le \ldots \le a[n]$.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases in the test. Then t test cases follow.

Each test case starts with a line containing an integer n ($1 \le n \le 2 \cdot 10^5$) — the size of the array a.

Then follow n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — an array that needs to be sorted by the given operations. **The given array can contain equal elements**.

The sum of n for all test cases in one test does not exceed $2 \cdot 10^5$.

Output

For each test case output one integer — the minimum total number of operations of the first and second type, which will make the array sorted in non-decreasing order.

Example

input

```
9
5
4 7 2 2 9
5
3 5 8 1 7
5
1 2 2 4 5
2
0 1
3
0 1 0
4
0 1 0 0
4
0 1 0 1
4
0 1 0 2
20
16 15 1 10 0 14 0 10 3 9 2 5 4 5 17 9 10 20 0 9
```

output

```
2
2
0
0
1
1
1
1
16
```

Note

In the first test case, you first need to move two 2, to the beginning of the array. Therefore, the desired sequence of operations: $[4, 7, 2, 2, 9] \rightarrow [2, 4, 7, 2, 9] \rightarrow [2, 2, 4, 7, 9]$.

In the second test case, you need to move the 1 to the beginning of the array, and the 8 — to the end. Therefore, the desired sequence of operations: $[3, 5, 8, 1, 7] \rightarrow [1, 3, 5, 8, 7] \rightarrow [1, 3, 5, 7, 8]$.

In the third test case, the array is already sorted.