A. Brogramming Contest

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

One day after waking up, your friend challenged you to a brogramming contest. In a brogramming contest, you are given a binary string* s of length n and an initially empty binary string t. During a brogramming contest, you can make either of the following moves any number of times:

- remove some suffix† from s and place it at the end of t, or
- remove some suffix from t and place it at the end of s.

To win the brogramming contest, you must make the *minimum* number of moves required to make s contain only the character `0` and t contain only the character `1`. Find the minimum number of moves required.

*A *binary string* is a string consisting of characters `0` and `1`.

†A string a is a suffix of a string b if a can be obtained from deletion of several (possibly, zero or all) elements from the beginning of b.

Input

The first line contains an integer t ($1 \leq t \leq 100$) — the number of test cases.

The first line of each test case is an integer n ($1 \leq n \leq 1000$) — the length of the string s.

The second line of each test case contains the binary string s.

The sum of n across all test cases does not exceed 1000.

Output

For each testcase, output the minimum number of moves required.

Standard Input	Standard Output
5 5 00110 4 1111 3 001 5 00000 3 101	2 1 1 0 3

Note

An optimal solution to the first test case is as follows:

- $s = 00110$, $t =$ empty string.
- $s = 00$, $t = 110$.
- $s = 000$, $t = 11$.

It can be proven that there is no solution using less than 2 moves.

In the second test case, you have to move the whole string from s to t in one move.

In the fourth test case, you don't have to do any moves.

,

B. Variety is Discouraged

```
Input file:     standard input
Output file:    standard output
Time limit:     1.5 seconds
Memory limit:   256 megabytes
```

Define the score of an arbitrary array b to be the length of b minus the number of distinct elements in b. For example:

- The score of $[1, 2, 2, 4]$ is 1, as it has length 4 and only 3 distinct elements $(1, 2, 4)$.
- The score of $[1, 1, 1]$ is 2, as it has length 3 and only 1 distinct element (1).
- The empty array has a score of 0.

You have an array a. You need to remove some **non-empty** contiguous subarray from a **at most** once.

More formally, you can do the following **at most** once:

- pick two integers l, r where $1 \leq l \leq r \leq n$, and
- delete the contiguous subarray $[a_l, \ldots, a_r]$ from a (that is, replace a with $[a_1, \ldots, a_{l-1}, a_{r+1}, \ldots, a_n]$).

Output an operation such that the score of a is **maximum**; if there are multiple answers, output one that **minimises** the final length of a after the operation. If there are still multiple answers, you may output any of them.

Input

The first line contains an integer t $(1 \leq t \leq 10^4)$ — the number of testcases.

The first line of each testcase contains an integer n $(1 \leq n \leq 2 \cdot 10^5)$ — the length of the array a.

The second line of each testcase contains n integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq n)$.

The sum of n across all testcases does not exceed $2 \cdot 10^5$.

Output

For each testcase, if you wish to not make a move, output 0.

Otherwise, output two integers l and r $(1 \leq l \leq r \leq n)$, representing the left and right bound of the removed subarray.

The removed subarray should be chosen such that the score is maximized, and over all such answers choose any of them that minimises the final length of the array.

Standard Input	Standard Output
3 1 1 5 1 1 1 1 1 4 2 1 3 2	1 1 0 2 3

Note

In the first testcase, we have two options:

- do nothing: the score of $[1]$ is $1 - 1 = 0$.
- remove the subarray with $l = 1, r = 1$: we remove the only element, and we get an empty array with score 0.

Therefore, the maximum score possible is 0. However, since we need to additionally minimise the final length of the array, we **must** output the second option with $l = r = 1$. Note that the first option of doing nothing is **incorrect**, since it has a longer final length.

In the second testcase, no subarray is selected, so after which a is still $[1, 1, 1, 1, 1]$. This has length 5 and 1 distinct element, so it has a score of $5 - 1 = 4$. This can be proven to be a shortest array which maximises the score.

In the third testcase, the subarray selected is $[2, 1, 3, 2]$, after which a becomes $[2, 2]$. This has length 2 and 1 distinct element, so it has a score of $2 - 1 = 1$. This can be proven to be a shortest array which maximises the score.

,

C. Remove the Ends

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 256 megabytes

You have an array a of length n consisting of **non-zero** integers. Initially, you have 0 coins, and you will do the following until a is empty:

- Let m be the current size of a. Select an integer i where $1 \le i \le m$, gain $|a_i|$* coins, and then:
  - if $a_i < 0$, then replace a with $[a_1, a_2, \ldots, a_{i-1}]$ (that is, delete the suffix beginning with a_i);
  - otherwise, replace a with $[a_{i+1}, a_{i+2}, \ldots, a_m]$ (that is, delete the prefix ending with a_i).

Find the maximum number of coins you can have at the end of the process.

*Here $|a_i|$ represents the absolute value of a_i: it equals a_i when $a_i > 0$ and $-a_i$ when $a_i < 0$.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of testcases.

The first line of each testcase contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the length of a.

The second line of each testcase contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$, $a_i \neq 0$).

The sum of n across all testcases does not exceed $2 \cdot 10^5$.

Output

For each test case, output the maximum number of coins you can have at the end of the process.

Standard Input	Standard Output
3 6 3 1 4 -1 -5 -9 6 -10 -3 -17 1 19 20 1 1	23 40 1

Note

An example of how to get 23 coins in the first testcase is as follows:

- $a = [3, 1, 4, -1, -5, -9] \xrightarrow{i=6} a = [3, 1, 4, -1, -5]$, and get 9 coins.
- $a = [3, 1, 4, -1, -5] \xrightarrow{i=1} a = [1, 4, -1, -5]$, and get 3 coins.
- $a = [1, 4, -1, -5] \xrightarrow{i=1} a = [4, -1, -5]$, and get 1 coin.
- $a = [4, -1, -5] \xrightarrow{i=3} a = [4, -1]$, and get 5 coins.
- $a = [4, -1] \xrightarrow{i=2} a = [4]$, and get 1 coin.
- $a = [4] \xrightarrow{i=1} a = []$, and get 4 coins.

After all the operations, you have 23 coins.

An example of how to get 40 coins in the second testcase is as follows:

- $a = [-10, -3, -17, 1, 19, 20] \xrightarrow{i=4} a = [19, 20]$, and get 1 coin.
- $a = [19, 20] \xrightarrow{i=1} a = [20]$, and get 19 coins.
- $a = [20] \xrightarrow{i=1} a = []$, and get 20 coins.

After all the operations, you have 40 coins.

,

D. Eating

Input file: standard input
Output file: standard output
Time limit: 5 seconds
Memory limit: 512 megabytes

There are n slimes on a line, the i-th of which has weight w_i. Slime i is able to eat another slime j if $w_i \geq w_j$; afterwards, slime j disappears and the weight of slime i becomes $w_i \oplus w_j$*.

The King of Slimes wants to run an *experiment with parameter* x as follows:

- Add a new slime with weight x to the right end of the line (after the n-th slime).
- This new slime eats the slime to its left if it is able to, and then takes its place (moves one place to the left). It will continue to do this until there is either no slime to its left or the weight of the slime to its left is greater than its own weight. (No other slimes are eaten during this process.)
- The *score* of this experiment is the total number of slimes eaten.

The King of Slimes is going to ask you q queries. In each query, you will be given an integer x, and you need to determine the score of the experiment with parameter x.

Note that the King does not want you to actually perform each experiment; his slimes would die, which is not ideal. He is only asking what the hypothetical score is; in other words, the queries are **not** persistent.

*Here $\oplus$ denotes the bitwise XOR operation.

Input

The first line contains an integer t ($1 \leq t \leq 10^4$) — the number of test cases.

The first line of each test case contains integers n and q ($1 \leq n, q \leq 2 \cdot 10^5$) — the number of slimes and the number of queries, respectively.

The following line contains n integers $w_1, w_2, \ldots, w_n$ ($1 \leq w_i < 2^{30}$) — the weights of the slimes.

The following q lines contain a single integer x ($1 \leq x < 2^{30}$) — the parameter for the experiment.

The sum of n does not exceed $2 \cdot 10^5$ and the sum of q does not exceed $2 \cdot 10^5$ across all test cases.

Output

For each query, output a single integer — the score of the experiment.

Standard Input	Standard Output
3 1 1 5 6 4 4 1 5 4 11 8 13 16 15 10 9	1 0 2 4 2 0 1 1 1 3 3 1 0 1

10 4 3 9 7 4 6 1 9 4 2 6 5 6 9 8 6 2 7	

Note

For the first query of the *second* testcase:

- A slime of weight 8 would be added to the end, so $w = [1, 5, 4, 11, 8]$.
- The added slime has smaller weight than the slime to its left so it cannot eat it, and thus ends the process after eating no slimes with score 0.

For the second query of the second testcase:

- A slime of weight 13 would be added to the end, so $w = [1, 5, 4, 11, 13]$.
- The added slime has bigger weight than the slime to its left, and so it will eat it. Its weight will become $13 \oplus 11 = 6$. Now $w = [1, 5, 4, 6]$.
- The added slime will now eat the slime to its left, and its weight becomes $6 \oplus 4 = 2$. Now $w = [1, 5, 2]$.
- The added slime is no longer able to eat the slime to its left, so it ends the process with a score of 2.

,

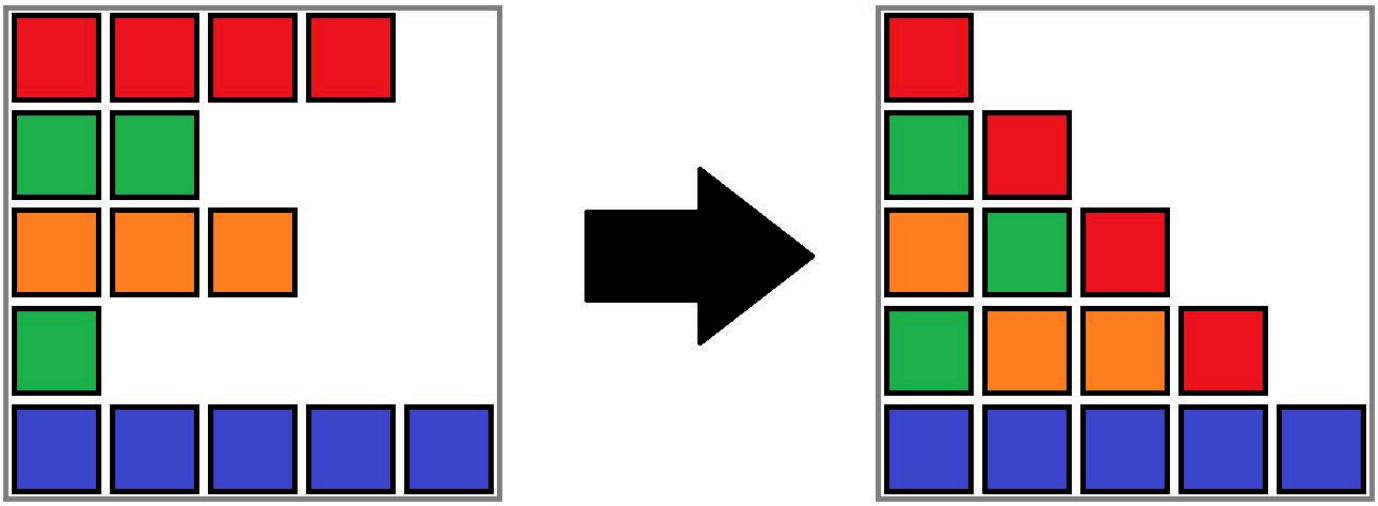
E. Mycraft Sand Sort

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   256 megabytes
```

Steve has a permutation* p and an array c, both of length n. Steve wishes to sort the permutation p.

Steve has an infinite supply of coloured sand blocks, and using them he discovered a physics-based way to sort an array of numbers called gravity sort. Namely, to perform gravity sort on p, Steve will do the following:

- For all i such that $1 \le i \le n$, place a sand block of color c_i in position (i, j) for all j where $1 \le j \le p_i$. Here, position (x, y) denotes a cell in the x-th row from the top and y-th column from the left.
- Apply downwards gravity to the array, so that all sand blocks fall as long as they can fall.



An example of gravity sort for the third testcase. $p = [4, 2, 3, 1, 5]$, $c = [2, 1, 4, 1, 5]$

Alex looks at Steve's sand blocks after performing gravity sort and wonders how many pairs of arrays (p', c') where p' is a permutation would have resulted in the same layout of sand. Note that the original pair of arrays (p, c) will always be counted.

Please calculate this for Alex. As this number could be large, output it modulo $998\,244\,353$.

---

*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 a 4 in the array).

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each testcase contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the lengths of the arrays.

The second line of each testcase contains n **distinct** integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$) — the elements of p.

The following line contains n integers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le n$) — the elements of c.

The sum of n across all testcases does not exceed $2 \cdot 10^5$.

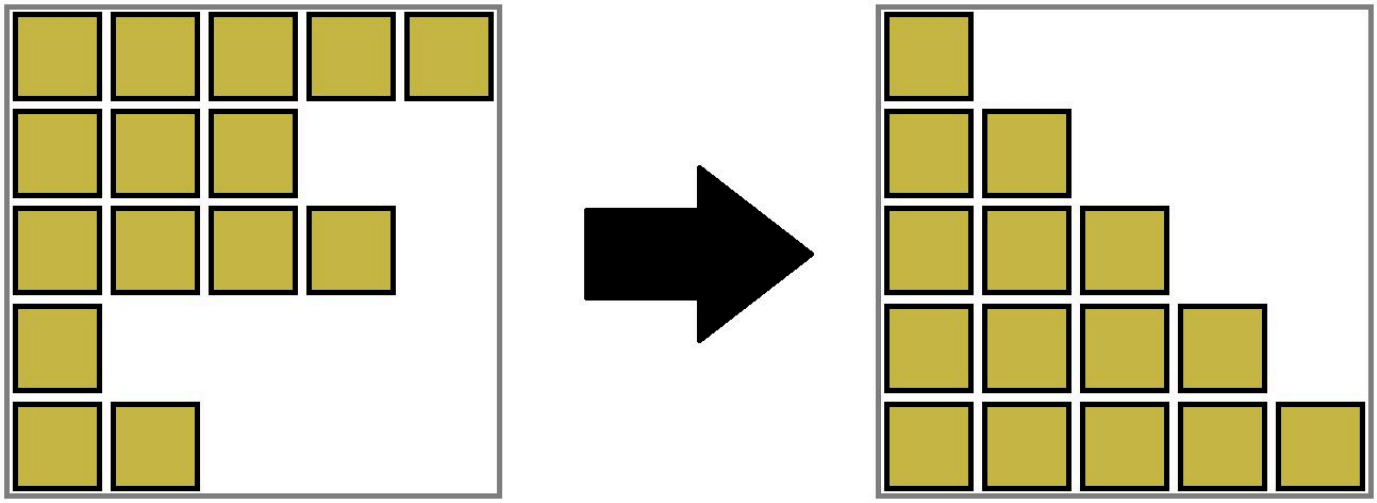
Output

For each testcase, output the number of pairs of arrays (p', c') Steve could have started with, modulo $998\,244\,353$.

Standard Input	Standard Output
4 1 1 1 5 5 3 4 1 2 1 1 1 1 1 5 4 2 3 1 5 2 1 4 1 5 40 29 15 20 35 37 31 27 1 32 36 38 25 22 8 16 7 3 28 11 12 23 4 14 9 39 13 10 30 6 2 24 17 19 5 34 18 33 26 40 21 3 1 2 2 1 2 3 1 1 1 1 2 1 3 1 1 3 1 1 1 2 2 1 3 3 3 2 3 2 2 2 2 1 3 2 1 1 2 2 2	1 120 1 143654893

Note

The second test case is shown below.



Gravity sort of the second testcase.

It can be shown that permutations of p will yield the same result, and that c must equal $[1, 1, 1, 1, 1]$ (since all sand must be the same color), so the answer is $5! = 120$.

The third test case is shown in the statement above. It can be proven that no other arrays p and c will produce the same final result, so the answer is 1.

,

F. We Be Summing

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   256 megabytes
```

You are given an array a of length n and an integer k.

Call a non-empty array b of length m *epic* if there exists an integer i where $1 \leq i < m$ and $\min(b_1, \ldots, b_i) + \max(b_{i+1}, \ldots, b_m) = k$.

Count the number of *epic* subarrays* of a.

---

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

Input

The first line contains an integer t ($1 \leq t \leq 10^4$) — the number of testcases.

The first line of each testcase contains integers n and k ($2 \leq n \leq 2 \cdot 10^5$; $n < k < 2 \cdot n$) — the length of the array a and k.

The second line of each testcase contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq n$).

The sum of n across all testcases does not exceed $2 \cdot 10^5$.

Output

For each testcase, output the number of epic contiguous subarrays of a.

Standard Input	Standard Output
6 5 7 1 2 3 4 5 7 13 6 6 6 6 7 7 7 6 9 4 5 6 6 5 1 5 9 5 5 4 5 5 5 6 3 3 3 3 3 6 8 4 5 4 5 4 5	2 12 3 8 10 4

Note

These are all the epic subarrays in the first testcase:

- $[2, 3, 4, 5]$, because $\min(2, 3) + \max(4, 5) = 2 + 5 = 7$.
- $[3, 4]$, because $\min(3) + \max(4) = 3 + 4 = 7$.

In the second test case, every subarray that contains at least one 6 and at least one 7 is epic.