Codeforces Round 1051 (Div. 2)

A. All Lengths Subtraction

1 second, 256 megabytes

You are given a permutation* p of length n.

You must perform exactly one operation for each integer k from 1 up to n in that order:

- Choose a subarray† of p of length exactly k, and subtract 1 from every element in that subarray.

After completing all n operations, your goal is to have all elements of the array equal to zero.

Determine whether it is possible to achieve this.

* 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).

† 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

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 contains the value n $(1 \le n \le 100)$ — the length of the permutation.

The second line contains $p_1, p_2, \ldots p_n$ $(1 \le p_i \le n)$ — the permutation itself.

Output

For each test case, output `YES` if it is possible to make all elements of the array p equal to 0 after performing all the operations; otherwise, output `NO`.

You can output the answer in any case (upper or lower). For example, the strings "`yEs`", "`yes`", "`Yes`", and "`YES`" will be recognized as positive responses.

input

```
4
4
1 3 4 2
5
1 5 2 4 3
5
2 4 5 3 1
3
3 1 2
```

output

```
YES
NO
YES
NO
```

For the first test case, we can proceed as follows:

- $k=1$: Choose the subarray $[3,3]$. The array $[1,3,4,2]$ becomes $[1,3,3,2]$.
- $k=2$: Choose the subarray $[2,3]$. The array $[1,3,3,2]$ becomes $[1,2,2,2]$.
- $k=3$: Choose the subarray $[2,4]$. The array $[1,2,2,2]$ becomes $[1,1,1,1]$.
- $k=4$: Choose the subarray $[1,4]$. The array $[1,1,1,1]$ becomes $[0,0,0,0]$.

Thus, we have successfully reduced the entire array to zeros, so the answer is `YES`.

For the second test case, it can be shown that it is impossible to reduce all elements to zero.

For the third test case, the process is as follows:

$$[2,4,\mathbf{5},3,1] \to [2,\mathbf{4},\mathbf{4},3,1] \to [2,\mathbf{3},\mathbf{3},\mathbf{3},1] \to [\mathbf{2},\mathbf{2},\mathbf{2},\mathbf{2},1] \to [\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1}] \to [0,0,0,0,0].$$

The bolded values indicate the subarrays from which we subtract at each step.

For the fourth test case, it can also be proven that it is impossible to make all values zero.

The problem statement has recently been changed. View the changes. ×

B. Discounts

1 second, 256 megabytes

You want to buy n products with prices $a_1, a_2, \dots, a_n$. You can either:

- buy product i individually, paying a_i coins, or
- use a discount voucher to buy it as part of a group purchase.

You have k discount vouchers with values $b_1, b_2, \dots, b_k$. A voucher of value x allows you to select exactly x products and pay only for the $x - 1$ most expensive ones, as such, you can consider that the cheapest product in the group is free. Each product can be included in **at most one** discount group, even if it is not the free one. Also, any single voucher can be used at most one single time.

What is the **minimum total cost** required to purchase all n products?

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 contains two integers n and k $(1 \le n, k \le 2 \cdot 10^5)$ —the number of products and the number of available discount vouchers.

The second line contains n integers $a_1, a_2, \dots, a_n$ $(1 \le a_i \le 10^9)$ — the prices of the products.

The third line contains k integers $b_1, b_2, \dots, b_k$ $(1 \le b_i \le n)$ — the values of the discount vouchers.

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

Output

Print t lines. The i-th line should contain the answer for the i-th test case — the minimum total cost required to purchase all products in that test case.

input

```
5
5 3
18 3 7 2 9
3 1 1
6 1
1 2 6 3 3 4
5
2 3
1 1
2 2 2
1 1
10
1
5 3
99 99 999 999 123
2 1 4
```

output

```
10
17
1
0
1197
```

In the first test case, you can apply the first discount to products 2, 3, and 4. You will pay for the two most expensive ones (3 and 7 coins) and get the cheapest one for free, resulting in a cost of $3 + 7 = 10$ coins. Then, apply the second and third discounts to products 1 and 5, getting both for free. The total cost is 10 coins.

In the second test case, you can use the single discount on products 2, 3, 4, 5, and 6. Among them, the cheapest is product 2 (costing 2 coins), which you get for free. You pay for the remaining products: $1 + 6 + 3 + 3 + 4 = 17$ coins in total.

In the third test case, only one discount is available. You can use it on both products, getting the cheaper one for free and paying 1 coin for the other.

The problem statement has recently been changed. View the changes. ×

C. Max Tree

2 seconds, 256 megabytes

You are given a tree consisting of n vertices, numbered from 1 to n. Each of the $n - 1$ edges is associated with two non-negative integers x and y.

Consider a permutation* p of the integers 1 through n, where p_i represents the value assigned to vertex i.

For an edge (u, v), such that $u < v$ with associated values x and y, its **contribution** is defined as follows:

$$\begin{cases} x & \text{if } p_u > p_v, \\ y & \text{otherwise.} \end{cases}$$

The **value** of the permutation is the sum of the contributions from all edges.

Your task is to find any permutation p that **maximizes** this total value.

[*] 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).

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 contains a single integer n ($2 \le n \le 2 \cdot 10^5$) — the number of vertices in the tree.

Each of the next $n - 1$ lines contains four integers u, v, x, and y ($1 \le u < v \le n$, $1 \le x, y \le 10^9$) — describing an edge between vertices u and v with associated values x and y. It is guarranteed that the given edges form a tree.

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

Output

For each test case, you must output a permutation p of the integers 1 through n that maximizes the total value as defined in the problem. If there are multiple answers, you can print any of them.

input

```
3
3
1 2 2 1
2 3 3 2
5
1 2 1 3
1 5 2 1
2 4 5 7
2 3 1 100
5
2 5 5 2
3 5 4 6
4 5 1 5
1 5 3 5
```

output

```
3 2 1
2 3 5 4 1
1 5 2 3 4
```

In the first test case:

Consider the permutation $p = [3, 2, 1]$. Its value is $5 = 2 + 3$. Here's why:

- Since $p_1 > p_2$, the first edge contributes $+2$.
- Since $p_2 > p_3$, the second edge contributes $+3$.

The values of some other permutations are as follows:

- $p = [1, 2, 3]$ has value $1 + 2 = 3$.
- $p = [1, 3, 2]$ has value $1 + 3 = 4$.
- $p = [3, 1, 2]$ has value $2 + 2 = 4$.

In the second test case:

The value of $p = [2, 3, 5, 4, 1]$ is $3 + 2 + 7 + 100 = 112$. Another possible correct answer is $p = [2, 3, 4, 5, 1]$.

D1. Inversion Graph Coloring (Easy Version)

3 seconds, 512 megabytes

This is the easy version of the problem. The difference between the versions is that in this version, $n \le 300$. You can hack only if you solved all versions of this problem.

A sequence $b_1, b_2, \ldots, b_k$ is called **good** if there exists a coloring of each index i in red or blue such that for every pair of indices $i < j$ with $b_i > b_j$, the colors assigned to i and j are different.

You are given a sequence $a_1, a_2, \ldots, a_n$. Compute the number of good subsequences of the sequence, including the empty subsequence[*]. Since the answer can be very large, output it modulo $10^9 + 7$.

[*] A sequence b is a subsequence of a sequence a if b can be obtained from a by the deletion of several (possibly, zero or all) element from arbitrary positions

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 contains an integer n ($1 \le n \le 300$) — the length of the sequence

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the contents of the sequence.

It is guaranteed that the sum of n over all test cases does not exceed 300.

Output

For each test case, output a single line containing the number of good subsequences modulo $10^9 + 7$.

input

```
4
4
4 2 3 1
7
7 6 1 2 3 3 2
5
1 1 1 1 1
11
7 2 1 9 7 3 4 1 3 5 3
```

output

```
13
73
32
619
```

In the first test case, the subsequences that are **not** good are $[4, 3, 1]$, $[4, 2, 1]$, and $[4, 2, 3, 1]$. Since there are 16 subsequences in total, this means there are $16 - 3 = 13$ good subsequences.

In the third test case, every subsequence is good.

D2. Inversion Graph Coloring (Hard Version)

3 seconds, 512 megabytes

This is the hard version of the problem. The difference between the versions is that in this version, $n \le 2000$. You can hack only if you solved all versions of this problem.

A sequence $b_1, b_2, \ldots, b_k$ is called **good** if there exists a coloring of each index i in red or blue such that for every pair of indices $i < j$ with $b_i > b_j$, the colors assigned to i and j are different.

You are given a sequence $a_1, a_2, \ldots, a_n$. Compute the number of good subsequences of the sequence, including the empty subsequence*. Since the answer can be very large, output it modulo $10^9 + 7$.

* A sequence b is a subsequence of a sequence a if b can be obtained from a by the deletion of several (possibly, zero or all) element from arbitrary positions.

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 contains an integer n ($1 \le n \le 2000$) — the length of the sequence

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the contents of the sequence.

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

Output

For each test case, output a single line containing the number of good subsequences modulo $10^9 + 7$.

input

```
4
4
4 2 3 1
7
7 6 1 2 3 3 2
5
1 1 1 1 1
11
7 2 1 9 7 3 4 1 3 5 3
```

output

```
13
73
```

```
32
619
```

In the first test case, the subsequences that are **not** good are $[4, 3, 1]$, $[4, 2, 1]$, and $[4, 2, 3, 1]$. Since there are 16 subsequences in total, this means there are $16 - 3 = 13$ good subsequences.

In the third test case, every subsequence is good.

E. Make Good

2 seconds, 512 megabytes

You are given a bracket string s of length n. You can apply the following operations arbitrary number of times (possibly, zero) and in any order:

- Choose an index $1 \le i < |s|$ such that $s_i = s_{i+1} = ($, and replace both s_i and s_{i+1} with $)$.
- Choose an index $1 \le i < |s|$ such that $s_i = s_{i+1} =)$, and replace both s_i and s_{i+1} with $($.

Find a regular bracket sequence* t which can be obtained from s by applying the operations described above, or print -1 if there is no such t.

* A regular bracket sequence is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters "`1`" and "`+`" between the original characters of the sequence. For example: the bracket sequences "`()()`" and "`(())`" are regular (the resulting expressions are: "`(1)+(1)`" and "`((1+1)+1)`"); the bracket sequences "`)(`", "`(`" and "`)`" are not.

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 contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the string.

The second line contains a single string s — a sequence of the characters `(` and `)` of length n.

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 regular bracket string t which can be obtained from s or print -1 if there is no such t.

input

```
5
4
()()
6
((())(
10
))(())())(
8
))))))))
1
(
```

output

```
()()
-1
-1
(())(())
-1
```

In the first testcase, the string "`()()`" is already a regular bracket sequence. No changes needed.

In the second and third testcases it's impossible to reach a regular bracket string from s.

In the fourth testcase we can do the following:

"`))))))))`" $\xrightarrow{i=1}$ "`(())))))`"

"`(())))))`" $\xrightarrow{i=5}$ "`(())(())`".

F. Increasing Xor

4 seconds, 512 megabytes

You have been gifted a magic sequence of integers: $a_1, a_2, \ldots, a_n$. However, this is no ordinary sequence — it can modify itself in a specific way!

After observing it carefully, you discovered the rule it follows:

- Repeatedly, two indices $1 \le i \le j \le n$ are chosen.
- Then, the value at index j is updated as: $a_j \leftarrow a_j \oplus a_i$.*

Being afraid of strictly increasing sequences, you start asking yourself questions:

You are given q queries. Each query consists of two integers l and r, and you must determine whether the subarray $a_l, a_{l+1}, \dots, a_r$ can be transformed into a strictly increasing sequence by applying any number of the described operations only within the range l to r, that is, **only using indices** $l \le i \le j \le r$.

* $\oplus$ denotes the bitwise XOR operation.

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

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i < 2^{20}$) — the contents of the sequence.

Each of the next q lines contains two integers l and r ($1 \le l \le r \le n$), representing a query. For each query, you must determine whether the subarray $a_l, a_{l+1}, \dots, a_r$ can be transformed into a strictly increasing sequence using the allowed operations.

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

Output

For each query, output `YES` if the subarray $a_l, a_{l+1}, \dots, a_r$ can be transformed into a strictly increasing sequence using the allowed operations; otherwise, output `NO`.

You may print each letter of `YES` or `NO` in any case. For example, `YES`, `yES`, `YeS` will all be recognized as a positive answer.

input

```
2
4 4
1 2 2 1
1 1
1 2
1 3
1 4
8 6
5 1 1 2 3 2 1 3
1 8
2 2
2 3
2 6
4 8
5 8
```

output

```
YES
YES
YES
YES
NO
YES
YES
NO
NO
YES
```

In the first test case:

For the first query, the sequence is $[1]$, which is increasing, so no changes are needed.

For the second query, the sequence is $[1, 2]$, which is also increasing, no changes required.

For the third query the seqeunce is $[1, 2, 2]$ which is not **strictly** increasing so we have to make some operations. If we apply $i = 1, j = 3$ then $a_3 \leftarrow a_3 \oplus a_1$. The sequence becomes $[1, 2, 3]$ which is now strictly increasing. Thus, the answer is positive.

For the fourth query the sequence is $[1, 2, 2, 1]$. We can do the following:

1. $a_2 \leftarrow a_2 \oplus a_2 (i = 2, j = 2)$. The seqeuence becomes $[1, 0, 2, 1]$.
2. $a_4 \leftarrow a_4 \oplus a_3 (i = 3, j = 4)$. The sequence becomes $[1, 0, 2, 3]$.
3. $a_2 \leftarrow a_2 \oplus a_1 (i = 1, j = 2)$. The sequence becomes $[1, 1, 2, 3]$.
4. $a_1 \leftarrow a_1 \oplus a_1 (i = 1, j = 1)$. The sequence becomes $[0, 1, 2, 3]$ which is strictly increasing.

In the second test case:

For the first query, it is not possible to make the entire sequence strictly increasing.

For the second query, the sequence is $[1]$, which is already strictly increasing, so no changes are needed.

In the third query, the sequence is $[1, 1]$. We can make it strictly increasing by choosing $i = j = 2$ (referring to the indices in the original sequence) and performing the operation $a_2 \leftarrow a_2 \oplus a_2$. This sets $a_2 = 0$, resulting in the subarray from index 2 to 3 becoming $[0, 1]$, which is strictly increasing.

For the final query, we can perform the following operations in this order:

1. $a_6 \leftarrow a_6 \oplus a_5$
2. $a_7 \leftarrow a_7 \oplus a_5$
3. $a_5 \leftarrow a_5 \oplus a_5$

After these operations, the subarray from index 5 to 8 becomes $[0, 1, 2, 3]$, which is strictly increasing.