A. Rectangle Arrangement

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

You are coloring an infinite square grid, in which all cells are initially white. To do this, you are given n stamps. Each stamp is a rectangle of width w_i and height h_i.

You will use **each** stamp exactly **once** to color a rectangle of the same size as the stamp on the grid in black. You cannot rotate the stamp, and for each cell, the stamp must either cover it fully or not cover it at all. You can use the stamp at any position on the grid, even if some or all of the cells covered by the stamping area are already black.

What is the minimum sum of the **perimeters** of the connected regions of black squares you can obtain after all the stamps have been used?

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 ($1 \le n \le 100$).

The i-th of the next n lines contains two integers w_i and h_i ($1 \le w_i, h_i \le 100$).

Output

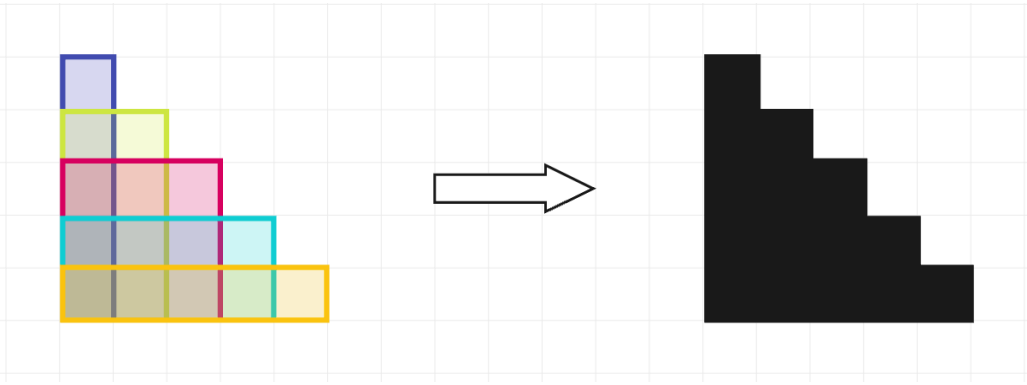
For each test case, output a single integer — the minimum sum of the perimeters of the connected regions of black squares you can obtain after all the stamps have been used.

Standard Input	Standard Output
5 5 1 5 2 4 3 3 4 2 5 1 3 2 2 1 1 1 2 1 3 2 3 100 100 100 100 100 100 4 1 4 2 3 1 5	20 8 10 400 16

3 2	

Note

In the first test case, the stamps can be used as shown on the left. Each stamp is highlighted in its own color for clarity.



After all these stamps are used, there is one black region (as shown on the right), and its perimeter is 20. It can be shown that there is no way of using the stamps that yields a lower total perimeter.

In the second test case, the second and third stamps can be used entirely inside the first one, so the minimum perimeter is equal to 8.

,

B. Stalin Sort

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

Stalin Sort is a humorous sorting algorithm designed to eliminate elements which are out of place instead of bothering to sort them properly, lending itself to an $\mathcal{O}(n)$ time complexity.

It goes as follows: starting from the second element in the array, if it is strictly smaller than the previous element (ignoring those which have already been deleted), then delete it. Continue iterating through the array until it is sorted in non-decreasing order. For example, the array $[1, 4, 2, 3, 6, 5, 5, 7, 7]$ becomes $[1, 4, 6, 7, 7]$ after a Stalin Sort.

We define an array as *vulnerable* if you can sort it in **non-increasing** order by repeatedly applying a Stalin Sort to **any of its subarrays***, as many times as is needed.

Given an array a of n integers, determine the minimum number of integers which must be removed from the array to make it *vulnerable*.

* 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 consists of several test cases. The first line contains a single integer t ($1 \leq t \leq 500$) — the number of test cases. This is followed by descriptions of the test cases.

The first line of each test case contains a single integer n ($1 \leq n \leq 2000$) — the size of the array.

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

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

Output

For each test case, output a single integer — the minimum number of integers which must be removed from the array to make it *vulnerable*.

Standard Input	Standard Output
6 7 3 6 4 9 2 5 2 5 5 4 4 2 2 8 2 2 4 4 6 6 10 10 1 1000 9 6 8 9 10 12 9 7 5 4 7	2 0 6 0 4 2

300000000 600000000 400000000 900000000 200000000 400000000 200000000	

Note

In the first test case, the optimal answer is to remove the numbers 3 and 9. Then we are left with $a = [6, 4, 2, 5, 2]$. To show this array is vulnerable, we can first apply a Stalin Sort on the subarray $[4, 2, 5]$ to get $a = [6, 4, 5, 2]$ and then apply a Stalin Sort on the subarray $[6, 4, 5]$ to get $a = [6, 2]$, which is non-increasing.

In the second test case, the array is already non-increasing, so we don't have to remove any integers.

,

C. Add Zeros

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

You're given an array a initially containing n integers. In one operation, you must do the following:

- Choose a position i such that $1 < i \leq |a|$ and $a_i = |a| + 1 - i$, where $|a|$ is the **current** size of the array.
- Append $i - 1$ zeros onto the end of a.

After performing this operation as many times as you want, what is the maximum possible length of the array a?

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 1000$). The description of the test cases follows.

The first line of each test case contains n ($1 \leq n \leq 3 \cdot 10^5$) — the length of the array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^{12}$).

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output a single integer — the maximum possible length of a after performing some sequence of operations.

Standard Input	Standard Output
4 5 2 4 6 2 5 5 5 4 4 5 1 4 6 8 2 3 1 1	10 11 10 1

Note

In the first test case, we can first choose $i = 4$, since $a_4 = 5 + 1 - 4 = 2$. After this, the array becomes $[2, 4, 6, 2, 5, 0, 0, 0]$. We can then choose $i = 3$ since $a_3 = 8 + 1 - 3 = 6$. After this, the array becomes $[2, 4, 6, 2, 5, 0, 0, 0, 0, 0]$, which has a length of 10. It can be shown that no sequence of operations will make the final array longer.

In the second test case, we can choose $i = 2$, then $i = 3$, then $i = 4$. The final array will be $[5, 4, 4, 5, 1, 0, 0, 0, 0, 0, 0]$, with a length of 11.

,

D1. The Endspeaker (Easy Version)

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

This is the easy version of this problem. The only difference is that you only need to output the minimum total cost of operations in this version. You must solve both versions to be able to hack.

You're given an array a of length n, and an array b of length m ($b_i > b_{i+1}$ for all $1 \leq i < m$). Initially, the value of k is 1. Your aim is to make the array a empty by performing one of these two operations repeatedly:

- Type 1 — If the value of k is less than m and the array a is **not empty**, you can increase the value of k by 1. This does not incur any cost.
- Type 2 — You remove a non-empty prefix of array a, such that its sum does not exceed b_k. This incurs a cost of $m - k$.

You need to minimize the total cost of the operations to make array a empty. If it's impossible to do this through any sequence of operations, output -1. Otherwise, output the minimum total cost of the operations.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 1000$). The description of the test cases follows.

The first line of each test case contains two integers n and m ($1 \leq n, m \leq 3 \cdot 10^5$, $\boldsymbol{1 \leq n \cdot m \leq 3 \cdot 10^5}$).

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

The third line of each test case contains m integers $b_1, b_2, \ldots, b_m$ ($1 \leq b_i \leq 10^9$).

It is also guaranteed that $b_i > b_{i+1}$ for all $1 \leq i < m$.

It is guaranteed that the sum of $\boldsymbol{n \cdot m}$ over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, if it's possible to make a empty, then output the minimum total cost of the operations.

If there is no possible sequence of operations which makes a empty, then output a single integer -1.

Standard Input	Standard Output
5 4 2 9 3 4 3 11 7 1 2 20 19 18 10 2 2 5 2 1 10 3 2 9 9 6 17 9	1 -1 2 10 4

10 11 2 2 2 2 2 2 2 2 2 2 20 18 16 14 12 10 8 6 4 2 1 1 6 10 32 16 8 4 2 1	

Note

In the first test case, one optimal sequence of operations which yields a total cost of 1 is as follows:

- Perform an operation of type 2. Choose the prefix to be $[9]$. This incurs a cost of 1.
- Perform an operation of type 1. The value of k is now 2. This incurs no cost.
- Perform an operation of type 2. Choose the prefix to be $[3, 4]$. This incurs a cost of 0.
- Perform an operation of type 2. Choose the prefix to be $[3]$. This incurs a cost of 0.
- The array a is now empty, and the total cost of all operations is 1.

In the second test case, it's impossible to remove any prefix of the array since $a_1 > b_1$, so array a cannot be made empty by any sequence of operations.

,

D2. The Endspeaker (Hard Version)

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

This is the hard version of this problem. The only difference is that you need to also output the number of optimal sequences in this version. You must solve both versions to be able to hack.

You're given an array a of length n, and an array b of length m ($b_i > b_{i+1}$ for all $1 \leq i < m$). Initially, the value of k is 1. Your aim is to make the array a empty by performing one of these two operations repeatedly:

- Type 1 — If the value of k is less than m and the array a is **not empty**, you can increase the value of k by 1. This does not incur any cost.
- Type 2 — You remove a non-empty prefix of array a, such that its sum does not exceed b_k. This incurs a cost of $m - k$.

You need to minimize the total cost of the operations to make array a empty. If it's impossible to do this through any sequence of operations, output -1. Otherwise, output the minimum total cost of the operations, and the number of sequences of operations which yield this minimum cost modulo $10^9 + 7$.

Two sequences of operations are considered different if you choose a different type of operation at any step, or the size of the removed prefix is different at any step.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 1000$). The description of the test cases follows.

The first line of each test case contains two integers n and m ($1 \leq n, m \leq 3 \cdot 10^5$, $\boldsymbol{1 \leq n \cdot m \leq 3 \cdot 10^5}$).

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

The third line of each test case contains m integers $b_1, b_2, \ldots, b_m$ ($1 \leq b_i \leq 10^9$).

It is also guaranteed that $b_i > b_{i+1}$ for all $1 \leq i < m$.

It is guaranteed that the sum of $\boldsymbol{n \cdot m}$ over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, if it's possible to make a empty, then output two integers. The first should be the minimum total cost of the operations, and the second should be the number of sequences of operations which achieve this minimum cost, modulo $10^9 + 7$.

If there is no possible sequence of operations which makes a empty, then output a single integer -1.

Standard Input	Standard Output
5 4 2 9 3 4 3 11 7 1 2 20	1 3 -1 2 11 10 42 4 1

19 18 10 2 2 5 2 1 10 3 2 9 9 6 17 9 10 11 2 2 2 2 2 2 2 2 2 2 20 18 16 14 12 10 8 6 4 2 1 1 6 10 32 16 8 4 2 1	

Note

In the first test case, there are 3 optimal sequences of operations which yield a total cost of 1:

- All 3 sequences begin with a type 2 operation, removing the prefix $[9]$ to make $a = [3, 4, 3]$, incurring a cost of 1. Then, we perform a type 1 operation to increase the value of k by 1. All subsequent operations now incur a cost of 0.
- One sequence continues by removing the prefixes $[3, 4]$ then $[3]$.
- Another sequence continues by removing the prefixes $[3]$ then $[4, 3]$.
- Another sequence continues by removing the prefixes $[3]$ then $[4]$ then $[3]$.

In the second test case, it's impossible to remove any prefix of the array since $a_1 > b_1$, so array a cannot be made empty by any sequence of operations.

,

E1. Bit Game (Easy Version)

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

This is the easy version of this problem. The only difference is that you need to output the winner of the game in this version, and the number of stones in each pile are fixed. You must solve both versions to be able to hack.

Alice and Bob are playing a familiar game where they take turns removing stones from n piles. Initially, there are x_i stones in the i-th pile, and it has an associated value a_i. A player can take d stones away from the i-th pile if and only if both of the following conditions are met:

- $1 \le d \le a_i$, and
- $x \,\&\, d = d$, where x is the current number of stones in the i-th pile and $\&$ denotes the bitwise AND operation.

The player who cannot make a move loses, and Alice goes first.

You're given the a_i and x_i values for each pile, please determine who will win the game if both players play optimally.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 1000$). The description of the test cases follows.

The first line of each test case contains n ($1 \le n \le 10^4$) — the number of piles.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i < 2^{30}$).

The third line of each test case contains n integers $x_1, x_2, \ldots, x_n$ ($1 \le x_i < 2^{30}$).

It is guaranteed that the sum of n over all test cases does not exceed 10^4.

Output

Print a single line with the winner's name. If Alice wins, print "`Alice`", otherwise print "`Bob`" (without quotes).

Standard Input	Standard Output
7 2 1 6 10 7 3 10 8 15 25 4 14 4 8 32 65 64 7 45 126 94 3 20 40 1	Bob Bob Bob Bob Bob Alice Alice

23 55 1 5 12345 9876 86419 8641 1 6789 54321 7532 97532 1 2 20 64 44 61 3 57 109 55 69 90 85	

Note

In the first test case, neither player can take any stones from the first pile since there is no value of d satisfying the conditions. For the second pile, to begin with, Alice can remove between 1 and 6 stones. No matter which move Alice performs, Bob can remove the rest of the stones on his turn. After Bob's move, there are no more moves that Alice can perform, so Bob wins.

In the second test case, here is one example of how the game might go. Alice moves first, and she decides to remove from the first pile. She cannot take 17 stones, because $17 > 10$, which fails the first condition. She cannot take 10 stones, because $25 \,\&\, 10 = 8$ which fails the second condition. One option is to take 9 stones; now the pile has 16 stones left. On Bob's turn he decides to take stones from the second pile; the only option here is to take all 4. Now, no more stones can be taken from either of the first two piles, so Alice must take some stones from the last pile. She decides to take 12 stones, and Bob then follows by taking the last 2 stones on that pile. Since Alice now has no legal moves left, Bob wins. It can be shown that no matter which strategy Alice follows, Bob will always be able to win if he plays optimally.

,

E2. Bit Game (Hard Version)

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

This is the hard version of this problem. The only difference is that you need to output the number of choices of games where Bob wins in this version, where the number of stones in each pile are not fixed. You must solve both versions to be able to hack.

Alice and Bob are playing a familiar game where they take turns removing stones from n piles. Initially, there are x_i stones in the i-th pile, and it has an associated value a_i. A player can take d stones away from the i-th pile if and only if both of the following conditions are met:

- $1 \le d \le a_i$, and
- $x \,\&\, d = d$, where x is the current number of stones in the i-th pile and $\&$ denotes the bitwise AND operation.

The player who cannot make a move loses, and Alice goes first.

You're given the a_i values of each pile, but the number of stones in the i-th pile has not been determined yet. For the i-th pile, x_i can be any integer between 1 and b_i, inclusive. That is, you can choose an array $x_1, x_2, \ldots, x_n$ such that the condition $1 \le x_i \le b_i$ is satisfied for all piles.

Your task is to count the number of games where Bob wins if both players play optimally. Two games are considered different if the number of stones in any pile is different, i.e., the arrays of x differ in at least one position.

Since the answer can be very large, please output the result modulo $10^9 + 7$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 1000$). The description of the test cases follows.

The first line of each test case contains n ($1 \le n \le 10^4$) — the number of piles.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i < 2^{30}$).

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i < 2^{30}$).

It is guaranteed that the sum of n over all test cases does not exceed 10^4.

Output

Output a single integer, the number of games where Bob wins, modulo $10^9 + 7$.

Standard Input	Standard Output
7 3 1 2 3 3 2 2 1 13	4 4 0 6552 722019507

45 5 5 4 7 8 6 4 4 5 5 5 4 6 4 8 8 12 13 14 12 3 92856133 46637598 12345678 29384774 73775896 87654321 2 65 12 110 31 4 677810235 275091182 428565855 720629731 74522416 889934149 3394714 230851724	541 665443265

Note

In the first test case, no matter which values of x_2 and x_3 we choose, the second and third piles will always be chosen exactly once before no more stones can be taken from them. If $x_1 = 2$, then no stones can be taken from it, so Bob will make the last move. If $x_1 = 1$ or $x_1 = 3$, then exactly one move can be made on that pile, so Alice will make the last move. So Bob wins when $x = [2, 1, 1]$ or $x = [2, 1, 2]$ or $x = [2, 2, 1]$ or $x = [2, 2, 2]$.

In the second test case, Bob wins when $x_1 = 14$ or $x_1 = 30$ by removing $14 - k$ stones, where k is the number of stones Alice removes on her turn. Bob also wins when $x_1 = 16$ or $x_1 = 32$ since Alice does not have any moves to begin with.