A. Two Subsequences

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

You are given a string s. You need to find two non-empty strings a and b such that the following conditions are satisfied:

1. Strings a and b are both **subsequences** of s.
2. For each index i, character s_i of string s must belong to **exactly one** of strings a or b.
3. String a is *lexicographically* minimum possible; string b may be any possible string.

Given string s, print any valid a and b.

Reminder:

A string a (b) is a *subsequence* of a string s if a (b) can be obtained from s by deletion of several (possibly, zero) elements. For example, "`dores`", "`cf`", and "`for`" are subsequences of "`codeforces`", while "`decor`" and "`fork`" are not.

A string x is *lexicographically smaller* than a string y if and only if one of the following holds:

- x is a prefix of y, but $x \neq y$;
- in the first position where x and y differ, the string x has a letter that appears earlier in the alphabet than the corresponding letter in y.

Input

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

The first and only line of each test case contains one string s ($2 \le |s| \le 100$ where $|s|$ means the length of s). String s consists of lowercase Latin letters.

Output

For each test case, print the strings a and b that satisfy the given conditions. If there are multiple answers, print any.

Example

input

```
3
fc
aaaa
thebrightboiler
```

output

```
c f
a aaa
b therightboiler
```

Note

In the first test case, there are only two choices: either $a =$ `f` and $b =$ `c` or $a =$ `c` and $b =$ `f`. And $a =$`c` is lexicographically smaller than $a =$ `f`.

In the second test case, a is the only character in the string.

In the third test case, it can be proven that b is the lexicographically smallest subsequence of s. The second string can be of two variants; one of them is given here.

B. Divine Array

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

Black is gifted with a Divine array a consisting of n ($1 \le n \le 2000$) integers. Each position in a has an initial value. After shouting a curse over the array, it becomes angry and starts an unstoppable transformation.

The transformation consists of infinite steps. Array a changes at the i-th step in the following way: for every position j, a_j becomes equal to the number of occurrences of a_j in a before starting this step.

Here is an example to help you understand the process better:

Initial array:	$2\ 1\ 1\ 4\ 3\ 1\ 2$
After the 1-st step:	$2\ 3\ 3\ 1\ 1\ 3\ 2$
After the 2-nd step:	$2\ 3\ 3\ 2\ 2\ 3\ 2$
After the 3-rd step:	$4\ 3\ 3\ 4\ 4\ 3\ 4$
...	...

In the initial array, we had two 2-s, three 1-s, only one 4 and only one 3, so after the first step, each element became equal to the number of its occurrences in the initial array: all twos changed to 2, all ones changed to 3, four changed to 1 and three changed to 1.

The transformation steps continue **forever**.

You have to process q queries: in each query, Black is curious to know the value of a_x after the k-th step of transformation.

Input

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

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

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the initial values of array a.

The third line of each test case contains a single integer q ($1 \le q \le 100\,000$) — the number of queries.

Next q lines contain the information about queries — one query per line. The i-th line contains two integers x_i and k_i ($1 \le x_i \le n$; $0 \le k_i \le 10^9$), meaning that Black is asking for the value of a_{x_i} after the k_i-th step of transformation. $k_i = 0$ means that Black is interested in values of the initial array.

It is guaranteed that the sum of n over all test cases doesn't exceed 2000 and the sum of q over all test cases doesn't exceed $100\,000$.

Output

For each test case, print q answers. The i-th of them should be the value of a_{x_i} after the k_i-th step of transformation. It can be shown that the answer to each query is unique.

Example

input

```
2
7
2 1 1 4 3 1 2
4
3 0
1 1
2 2
6 1
2
1 1
2
1 0
2 1000000000
```

output

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

Note

The first test case was described ih the statement. It can be seen that:

1. $k_1 = 0$ (initial array): $a_3 = 1$;
2. $k_2 = 1$ (after the 1-st step): $a_1 = 2$;
3. $k_3 = 2$ (after the 2-nd step): $a_2 = 3$;
4. $k_4 = 1$ (after the 1-st step): $a_6 = 3$.

For the second test case,

Initial array:	$1\ 1$
After the 1-st step:	$2\ 2$

After the 2-nd step:	$2\ 2$
...	...

It can be seen that:

1. $k_1 = 0$ (initial array): $a_1 = 1$;
2. $k_2 = 1000000000$: $a_2 = 2$;

C. Array Elimination

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

You are given array $a_1, a_2, \ldots, a_n$, consisting of non-negative integers.

Let's define operation of "elimination" with integer parameter k ($1 \le k \le n$) as follows:

- Choose k distinct array indices $1 \le i_1 < i_2 < \ldots < i_k \le n$.
- Calculate $x = a_{i_1} ~\&~ a_{i_2} ~\&~ \ldots ~\&~ a_{i_k}$, where $\&$ denotes the bitwise AND operation (notes section contains formal definition).
- Subtract x from each of $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$; all other elements remain untouched.

Find all possible values of k, such that it's possible to make all elements of array a equal to 0 using a finite number of elimination operations with parameter k. It can be proven that exists at least one possible k for any array a.

Note that you **firstly choose k and only after that perform elimination operations with value k you've chosen initially**.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). Description of the test cases follows.

The first line of each test case contains one integer n ($1 \le n \le 200\,000$) — the length of array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < 2^{30}$) — array a itself.

It's guaranteed that the sum of n over all test cases doesn't exceed $200\,000$.

Output

For each test case, print all values k, such that it's possible to make all elements of a equal to 0 in a finite number of elimination operations with the given parameter k.

Print them in increasing order.

Example

input

```
5
4
4 4 4 4
4
13 7 25 19
6
3 5 3 1 7 1
1
1
5
0 0 0 0 0
```

output

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

Note

In the first test case:

- If $k = 1$, we can make four elimination operations with sets of indices $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$. Since $\&$ of one element is equal to the element itself, then for each operation $x = a_i$, so $a_i - x = a_i - a_i = 0$.
- If $k = 2$, we can make two elimination operations with, for example, sets of indices $\{1, 3\}$ and $\{2, 4\}$: $x = a_1 ~\&~ a_3 = a_2 ~\&~ a_4 = 4 ~\&~ 4 = 4$. For both operations $x = 4$, so after the first operation $a_1 - x = 0$ and $a_3 - x = 0$, and after the second operation — $a_2 - x = 0$ and $a_4 - x = 0$.
- If $k = 3$, it's impossible to make all a_i equal to 0. After performing the first operation, we'll get three elements equal to 0 and one equal to 4. After that, all elimination operations won't change anything, since at least one chosen element will always be equal to 0.

- If $k = 4$, we can make one operation with set $\{1, 2, 3, 4\}$, because $x = a_1 \& a_2 \& a_3 \& a_4 = 4$.

In the second test case, if $k = 2$ then we can make the following elimination operations:

- Operation with indices $\{1, 3\}$: $x = a_1 \& a_3 = 13 \& 25 = 9$. $a_1 - x = 13 - 9 = 4$ and $a_3 - x = 25 - 9 = 16$. Array a will become equal to $[4, 7, 16, 19]$.
- Operation with indices $\{3, 4\}$: $x = a_3 \& a_4 = 16 \& 19 = 16$. $a_3 - x = 16 - 16 = 0$ and $a_4 - x = 19 - 16 = 3$. Array a will become equal to $[4, 7, 0, 3]$.
- Operation with indices $\{2, 4\}$: $x = a_2 \& a_4 = 7 \& 3 = 3$. $a_2 - x = 7 - 3 = 4$ and $a_4 - x = 3 - 3 = 0$. Array a will become equal to $[4, 4, 0, 0]$.
- Operation with indices $\{1, 2\}$: $x = a_1 \& a_2 = 4 \& 4 = 4$. $a_1 - x = 4 - 4 = 0$ and $a_2 - x = 4 - 4 = 0$. Array a will become equal to $[0, 0, 0, 0]$.

Formal definition of bitwise AND:

Let's define bitwise AND ($\&$) as follows. Suppose we have two non-negative integers x and y, let's look at their binary representations (possibly, with leading zeroes): $x_k \dots x_2 x_1 x_0$ and $y_k \dots y_2 y_1 y_0$. Here, x_i is the i-th bit of number x, and y_i is the i-th bit of number y. Let $r = x \& y$ is a result of operation $\&$ on number x and y. Then binary representation of r will be $r_k \dots r_2 r_1 r_0$, where:

$$r_i = \begin{cases} 1, \text{ if } x_i = 1 \text{ and } y_i = 1 \\ 0, \text{ if } x_i = 0 \text{ or } y_i = 0 \end{cases}$$

D. Frog Traveler

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

Frog Gorf is traveling through Swamp kingdom. Unfortunately, after a poor jump, he fell into a well of n meters depth. Now Gorf is on the bottom of the well and has a long way up.

The surface of the well's walls vary in quality: somewhere they are slippery, but somewhere have convenient ledges. In other words, if Gorf is on x meters below ground level, then in one jump he can go up on any integer distance from 0 to a_x meters inclusive. (Note that Gorf can't jump down, only up).

Unfortunately, Gorf has to take a break after each jump (including jump on 0 meters). And after jumping up to position x meters below ground level, he'll slip exactly b_x meters down while resting.

Calculate the minimum number of jumps Gorf needs to reach ground level.

Input

The first line contains a single integer n ($1 \le n \le 300\,000$) — the depth of the well.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($0 \le a_i \le i$), where a_i is the maximum height Gorf can jump from i meters below ground level.

The third line contains n integers $b_1, b_2, \dots, b_n$ ($0 \le b_i \le n - i$), where b_i is the distance Gorf will slip down if he takes a break on i meters below ground level.

Output

If Gorf can't reach ground level, print -1. Otherwise, firstly print integer k — the minimum possible number of jumps.

Then print the sequence $d_1,\ d_2,\ \dots,\ d_k$ where d_j is the depth Gorf'll reach after the j-th jump, but before he'll slip down during the break. Ground level is equal to 0.

If there are multiple answers, print any of them.

Examples

input

```
3
0 2 2
1 1 0
```

output

```
2
1 0
```

input

```
2
1 1
1 0
```

output

```
-1
```

input

```
10
0 1 2 3 5 5 6 7 8 5
9 8 7 1 5 4 3 2 0 0
```

output

```
3
9 4 0
```

Note

In the first example, Gorf is on the bottom of the well and jump to the height 1 meter below ground level. After that he slip down by meter and stays on height 2 meters below ground level. Now, from here, he can reach ground level in one jump.

In the second example, Gorf can jump to one meter below ground level, but will slip down back to the bottom of the well. That's why he can't reach ground level.

In the third example, Gorf can reach ground level only from the height 5 meters below the ground level. And Gorf can reach this height using a series of jumps $10 \Rightarrow 9 \dashrightarrow 9 \Rightarrow 4 \dashrightarrow 5$ where $\Rightarrow$ is the jump and $\dashrightarrow$ is slipping during breaks.

E. Optimal Insertion

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given two arrays of integers $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$.

You need to insert all elements of b into a in an arbitrary way. As a result you will get an array $c_1, c_2, \ldots, c_{n+m}$ of size $n + m$.

Note that you are not allowed to change the order of elements in a, while you can insert elements of b at arbitrary positions. They can be inserted at the beginning, between any elements of a, or at the end. Moreover, elements of b can appear in the resulting array in any order.

What is the minimum possible number of inversions in the resulting array c? Recall that an inversion is a pair of indices (i, j) such that $i < j$ and $c_i > c_j$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). Description of the test cases follows.

The first line of each test case contains two integers n and m ($1 \le n, m \le 10^6$).

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

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

It is guaranteed that the sum of n for all tests cases in one input doesn't exceed 10^6. The sum of m for all tests cases doesn't exceed 10^6 as well.

Output

For each test case, print one integer — the minimum possible number of inversions in the resulting array c.

Example

input

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

output

```
0
4
6
```

Note

Below is given the solution to get the optimal answer for each of the example test cases (elements of a are underscored).

- In the first test case, $c = [\underline{1}, 1, \underline{2}, 2, \underline{3}, 3, 4]$.
- In the second test case, $c = [1, 2, \underline{3}, \underline{2}, \underline{1}, 3]$.
- In the third test case, $c = [\underline{1}, 1, 3, \underline{3}, \underline{5}, \underline{3}, \underline{1}, 4, 6]$.

F. Difficult Mountain

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

A group of n alpinists has just reached the foot of the mountain. The initial difficulty of climbing this mountain can be described as an integer d.

Each alpinist can be described by two integers s and a, where s is his skill of climbing mountains and a is his neatness.

An alpinist of skill level s is able to climb a mountain of difficulty p only if $p \leq s$. As an alpinist climbs a mountain, they affect the path and thus may change mountain difficulty. Specifically, if an alpinist of neatness a climbs a mountain of difficulty p the difficulty of this mountain becomes $\max(p, a)$.

Alpinists will climb the mountain one by one. And before the start, they wonder, what is the maximum number of alpinists who will be able to climb the mountain if they choose the right order. As you are the only person in the group who does programming, you are to answer the question.

Note that after the order is chosen, each alpinist who can climb the mountain, must climb the mountain at that time.

Input

The first line contains two integers n and d ($1 \leq n \leq 500\,000$; $0 \leq d \leq 10^9$) — the number of alpinists and the initial difficulty of the mountain.

Each of the next n lines contains two integers s_i and a_i ($0 \leq s_i, a_i \leq 10^9$) that define the skill of climbing and the neatness of the i-th alpinist.

Output

Print one integer equal to the maximum number of alpinists who can climb the mountain if they choose the right order to do so.

Examples

input

```
3 2
2 6
3 5
5 7
```

output

```
2
```

input

```
3 3
2 4
6 4
4 6
```

output

```
2
```

input

```
5 0
1 5
4 8
2 7
7 6
3 2
```

output

```
3
```

Note

In the first example, alpinists 2 and 3 can climb the mountain if they go in this order. There is no other way to achieve the answer of 2.

In the second example, alpinist 1 is not able to climb because of the initial difficulty of the mountain, while alpinists 2 and 3 can go up in any order.

In the third example, the mountain can be climbed by alpinists 5, 3 and 4 in this particular order. There is no other way to achieve optimal answer.