A. Min Max Swap

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given two arrays a and b of n positive integers each. You can apply the following operation to them any number of times:

- Select an index i ($1 \le i \le n$) and swap a_i with b_i (i. e. a_i becomes b_i and vice versa).

Find the **minimum** possible value of $\max(a_1, a_2, \ldots, a_n) \cdot \max(b_1, b_2, \ldots, b_n)$ you can get after applying such operation any number of times (possibly zero).

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 100$) — the number of test cases. Description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 100$) — the length of the arrays.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10\,000$) where a_i is the i-th element of the array a.

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le 10\,000$) where b_i is the i-th element of the array b.

Output

For each test case, print a single integer, the **minimum** possible value of $\max(a_1, a_2, \ldots, a_n) \cdot \max(b_1, b_2, \ldots, b_n)$ you can get after applying such operation any number of times.

Example

input

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

output

```
18
9
2
```

Note

In the first test, you can apply the operations at indices 2 and 6, then $a = [1, 4, 6, 5, 1, 5]$ and $b = [3, 2, 3, 2, 2, 2]$, $\max(1, 4, 6, 5, 1, 5) \cdot \max(3, 2, 3, 2, 2, 2) = 6 \cdot 3 = 18$.

In the second test, no matter how you apply the operations, $a = [3, 3, 3]$ and $b = [3, 3, 3]$ will always hold, so the answer is $\max(3, 3, 3) \cdot \max(3, 3, 3) = 3 \cdot 3 = 9$.

In the third test, you can apply the operation at index 1, then $a = [2, 2]$, $b = [1, 1]$, so the answer is $\max(2, 2) \cdot \max(1, 1) = 2 \cdot 1 = 2$.

B. Fun with Even Subarrays

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array a of n elements. You can apply the following operation to it any number of times:

- Select some subarray from a of even size $2k$ that begins at position l ($1 \le l \le l + 2 \cdot k - 1 \le n$, $k \ge 1$) and for each i between 0 and $k - 1$ (inclusive), assign the value a_{l+k+i} to a_{l+i}.

For example, if $a = [2, 1, 3, 4, 5, 3]$, then choose $l = 1$ and $k = 2$, applying this operation the array will become $a = [3, 4, 3, 4, 5, 3]$.

Find the minimum number of operations (possibly zero) needed to make all the elements of the array equal.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Description of the test cases follows.

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

The second line of each test case consists of n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) — the elements of the array a.

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

Output

Print t lines, each line containing the answer to the corresponding test case — the minimum number of operations needed to make equal all the elements of the array with the given operation.

Example

input

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

output

```
0
1
1
2
0
```

Note

In the first test, all elements are equal, therefore no operations are needed.

In the second test, you can apply one operation with $k = 1$ and $l = 1$, set $a_1 := a_2$, and the array becomes $[1, 1]$ with 1 operation.

In the third test, you can apply one operation with $k = 1$ and $l = 4$, set $a_4 := a_5$, and the array becomes $[4, 4, 4, 4, 4]$.

In the fourth test, you can apply one operation with $k = 1$ and $l = 3$, set $a_3 := a_4$, and the array becomes $[4, 2, 3, 3]$, then you can apply another operation with $k = 2$ and $l = 1$, set $a_1 := a_3$, $a_2 := a_4$, and the array becomes $[3, 3, 3, 3]$.

In the fifth test, there is only one element, therefore no operations are needed.

C. And Matching

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a set of n (n is always a power of 2) elements containing all integers $0, 1, 2, \dots, n-1$ exactly once.

Find $\frac{n}{2}$ pairs of elements such that:

- Each element in the set is in exactly one pair.
- The sum over all pairs of the bitwise AND of its elements must be exactly equal to k. Formally, if a_i and b_i are the elements of the i-th pair, then the following must hold:

$$\sum_{i=1}^{n/2} a_i \& b_i = k,$$

where $\&$ denotes the bitwise AND operation.

If there are many solutions, print any of them, if there is no solution, print -1 instead.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 400$) — the number of test cases. Description of the test cases follows.

Each test case consists of a single line with two integers n and k ($4 \le n \le 2^{16}$, n is a power of 2, $0 \le k \le n-1$).

The sum of n over all test cases does not exceed 2^{16}. All test cases in each individual input will be pairwise **different**.

Output

For each test case, if there is no solution, print a single line with the integer -1.

Otherwise, print $\frac{n}{2}$ lines, the i-th of them must contain a_i and b_i, the elements in the i-th pair.

If there are many solutions, print any of them. Print the pairs and the elements in the pairs in any order.

Example

input

```
4
4 0
4 1
4 2
4 3
```

output

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

Note

In the first test, $(0\&3) + (1\&2) = 0$.

In the second test, $(0\&2) + (1\&3) = 1$.

In the third test, $(0\&1) + (2\&3) = 2$.

In the fourth test, there is no solution.

D. Range and Partition

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

Given an array a of n integers, find a range of values $[x, y]$ ($x \le y$), and split a into **exactly** k ($1 \le k \le n$) subarrays in such a way that:

- Each subarray is formed by several continuous elements of a, that is, it is equal to $a_l, a_{l+1}, \ldots, a_r$ for some l and r ($1 \le l \le r \le n$).
- Each element from a belongs to exactly one subarray.
- In each subarray the number of elements inside the range $[x, y]$ (inclusive) is **strictly greater** than the number of elements outside the range. An element with index i is inside the range $[x, y]$ if and only if $x \le a_i \le y$.

Print any solution that minimizes $y - x$.

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 3 \cdot 10^4$) — the number of test cases. Description of the test cases follows.

The first line of each test case contains two integers n and k ($1 \le k \le n \le 2 \cdot 10^5$) — the length of the array a and the number of subarrays required in the partition.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) where a_i is the i-th element of the array.

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

Output

For each test case, print $k + 1$ lines.

In the first line, print x and y — the limits of the found range.

Then print k lines, the i-th should contain l_i and r_i ($1 \le l_i \le r_i \le n$) — the limits of the i-th subarray.

You can print the subarrays in any order.

Example

input

```
3
2 1
1 2
4 2
1 2 2 2
11 3
```

```
5 5 5 1 5 5 1 5 5 5 1
```

output

```
1 2
1 2
2 2
1 3
4 4
5 5
1 1
2 2
3 11
```

Note

In the first test, there should be only one subarray, which must be equal to the whole array. There are 2 elements inside the range $[1, 2]$ and 0 elements outside, if the chosen range is $[1, 1]$, there will be 1 element inside (a_1) and 1 element outside (a_2), and the answer will be invalid.

In the second test, it is possible to choose the range $[2, 2]$, and split the array in subarrays $(1, 3)$ and $(4, 4)$, in subarray $(1, 3)$ there are 2 elements inside the range (a_2 and a_3) and 1 element outside (a_1), in subarray $(4, 4)$ there is only 1 element (a_4), and it is inside the range.

In the third test, it is possible to choose the range $[5, 5]$, and split the array in subarrays $(1, 4)$, $(5, 7)$ and $(8, 11)$, in the subarray $(1, 4)$ there are 3 elements inside the range and 1 element outside, in the subarray $(5, 7)$ there are 2 elements inside and 1 element outside and in the subarray $(8, 11)$ there are 3 elements inside and 1 element outside.

E. Paint the Middle

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

You are given n elements numbered from 1 to n, the element i has value a_i and color c_i, initially, $c_i = 0$ for all i.

The following operation can be applied:

- Select three elements i, j and k ($1 \le i < j < k \le n$), such that c_i, c_j and c_k are all equal to 0 and $a_i = a_k$, then set $c_j = 1$.

Find the maximum value of $\sum\limits_{i=1}^{n} c_i$ that can be obtained after applying the given operation any number of times.

Input

The first line contains an integer n ($3 \le n \le 2 \cdot 10^5$) — the number of elements.

The second line consists of n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$), where a_i is the value of the i-th element.

Output

Print a single integer in a line — the maximum value of $\sum\limits_{i=1}^{n} c_i$ that can be obtained after applying the given operation any number of times.

Examples

input

```
7
1 2 1 2 7 4 7
```

output

```
2
```

input

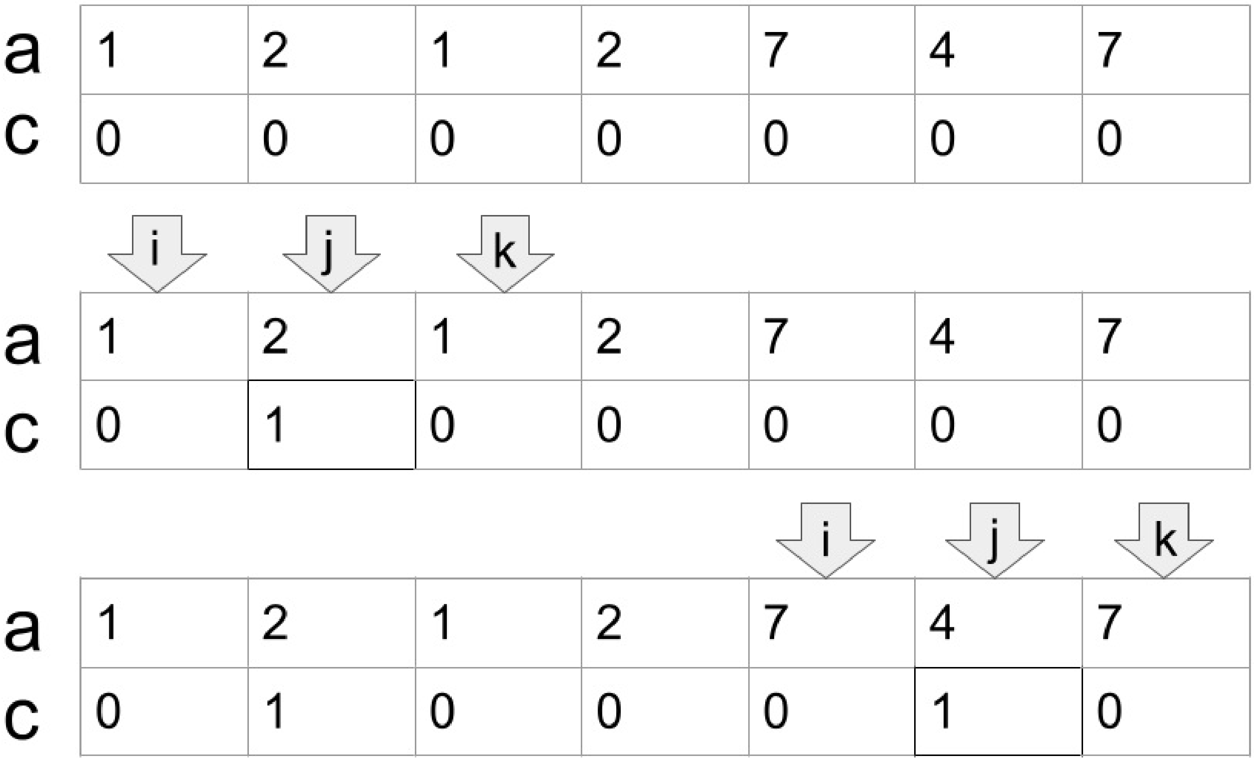
```
13
1 2 3 2 1 3 3 4 5 5 5 4 7
```

output

```
7
```

Note

In the first test, it is possible to apply the following operations in order:



F. Flipping Range

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

You are given an array a of n integers and a set B of m positive integers such that $1 \leq b_i \leq \lfloor \frac{n}{2} \rfloor$ for $1 \leq i \leq m$, where b_i is the i-th element of B.

You can make the following operation on a:

1. Select some x such that x appears in B.
2. Select an interval from array a of size x and multiply by -1 every element in the interval. Formally, select l and r such that $1 \leq l \leq r \leq n$ and $r - l + 1 = x$, then assign $a_i := -a_i$ for every i such that $l \leq i \leq r$.

Consider the following example, let $a = [0, 6, -2, 1, -4, 5]$ and $B = \{1, 2\}$:

1. $[0, 6, -2, -1, 4, 5]$ is obtained after choosing size 2 and $l = 4$, $r = 5$.
2. $[0, 6, 2, -1, 4, 5]$ is obtained after choosing size 1 and $l = 3$, $r = 3$.

Find the maximum $\sum\limits_{i=1}^{n} a_i$ you can get after applying such operation any number of times (possibly zero).

Input

The input consists of multiple test cases. The first line contains a single integer t ($1 \leq t \leq 10^5$) — the number of test cases. Description of the test cases follows.

The first line of each test case contains two integers n and m ($2 \leq n \leq 10^6$, $1 \leq m \leq \lfloor \frac{n}{2} \rfloor$) — the number of elements of a and B respectively.

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

The third line of each test case contains m **distinct** positive integers $b_1, b_2, \ldots, b_m$ ($1 \leq b_i \leq \lfloor \frac{n}{2} \rfloor$) — the elements in the set B.

It's guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

For each test case print a single integer — the maximum possible sum of all a_i after applying such operation any number of times.

Example

input

```
3
6 2
0 6 -2 1 -4 5
1 2
```

```
7 1
1 -1 1 -1 1 -1 1
2
5 1
-1000000000 -1000000000 -1000000000 -1000000000 -1000000000
1
```

output

```
18
5
5000000000
```

Note

In the first test, you can apply the operation $x = 1$, $l = 3$, $r = 3$, and the operation $x = 1$, $l = 5$, $r = 5$, then the array becomes $[0, 6, 2, 1, 4, 5]$.

In the second test, you can apply the operation $x = 2$, $l = 2$, $r = 3$, and the array becomes $[1, 1, -1, -1, 1, -1, 1]$, then apply the operation $x = 2$, $l = 3$, $r = 4$, and the array becomes $[1, 1, 1, 1, 1, -1, 1]$. There is no way to achieve a sum bigger than 5.