Codeforces Round #683 (Div. 2, by Meet IT)

A. Add Candies

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

There are n bags with candies, initially the i-th bag contains i candies. You want all the bags to contain an equal amount of candies in the end.

To achieve this, you will:

- Choose m such that $1 \leq m \leq 1000$
- Perform m operations. In the j-th operation, you will pick one bag and add j candies to all bags apart from the chosen one.

Your goal is to find a valid sequence of operations after which all the bags will contain an equal amount of candies.

- It can be proved that for the given constraints such a sequence always exists.
- You **don't** have to minimize m.
- If there are several valid sequences, you can output **any**.

Input
Each test contains multiple test cases.

The first line contains the number of test cases t ($1 \leq t \leq 100$). Description of the test cases follows.

The first and only line of each test case contains one integer n ($2 \leq n \leq 100$).

Output
For each testcase, print two lines with your answer.

In the first line print m ($1 \leq m \leq 1000$) — the number of operations you want to take.

In the second line print m positive integers $a_1, a_2, \ldots, a_m$ ($1 \leq a_i \leq n$), where a_j is the number of bag you chose on the j-th operation.

Example

input
2 2 3
output
1 2 5 3 3 3 1 2

Note
In the first case, adding 1 candy to all bags except of the second one leads to the arrangement with $[2, 2]$ candies.

In the second case, firstly you use first three operations to add $1 + 2 + 3 = 6$ candies in total to each bag except of the third one, which gives you $[7, 8, 3]$. Later, you add 4 candies to second and third bag, so you have $[7, 12, 7]$, and 5 candies to first and third bag — and the result is $[12, 12, 12]$.

B. Numbers Box

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

You are given a rectangular grid with n rows and m columns. The cell located on the i-th row from the top and the j-th column from the left has a value a_{ij} written in it.

You can perform the following operation any number of times (possibly zero):

- Choose any two adjacent cells and multiply the values in them by -1. Two cells are called adjacent if they share a side.

Note that you can use a cell more than once in different operations.

You are interested in X, the **sum** of all the numbers in the grid.

What is the maximum X you can achieve with these operations?

Input
Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 100$). Description of the test cases follows.

The first line of each test case contains two integers n,m ($2 \le n$, $m \le 10$).

The following n lines contain m integers each, the j-th element in the i-th line is a_{ij} ($-100 \le a_{ij} \le 100$).

Output
For each testcase, print one integer X, the maximum possible sum of all the values in the grid after applying the operation as many times as you want.

Example

input
2 2 2 -1 1 1 1 3 4 0 -1 -2 -3 -1 -2 -3 -4 -2 -3 -4 -5
output
2 30

Note
In the first test case, there will always be at least one -1, so the answer is 2.

In the second test case, we can use the operation six times to elements adjacent horizontally and get all numbers to be non-negative. So the answer is: $2 \times 1 + 3 \times 2 + 3 \times 3 + 2 \times 4 + 1 \times 5 = 30$.

C. Knapsack

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

You have a knapsack with the capacity of W. There are also n items, the i-th one has weight w_i.

You want to put some of these items into the knapsack in such a way that their total weight C is at least half of its size, but (obviously) does not exceed it. Formally, C should satisfy: $\lceil \frac{W}{2} \rceil \le C \le W$.

Output the list of items you will put into the knapsack or determine that fulfilling the conditions is impossible.

If there are several possible lists of items satisfying the conditions, you can output any. Note that you **don't** have to maximize the sum of weights of items in the knapsack.

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 integers n and W ($1 \le n \le 200\,000$, $1 \le W \le 10^{18}$).

The second line of each test case contains n integers $w_1, w_2, \ldots, w_n$ ($1 \le w_i \le 10^9$) — weights of the items.

The sum of n over all test cases does not exceed $200\,000$.

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

If there exists a solution consisting of m items, print m in the first line of the output and m integers j_1, j_2, ..., j_m ($1 \le j_i \le n$, **all** j_i **are distinct**) in the second line of the output — indices of the items you would like to pack into the knapsack.

If there are several possible lists of items satisfying the conditions, you can output any. Note that you **don't** have to maximize the sum of weights items in the knapsack.

Example

input

```
3
1 3
3
6 2
19 8 19 69 9 4
7 12
1 1 1 17 1 1 1
```

output

```
1
1
-1
6
1 2 3 5 6 7
```

Note

In the first test case, you can take the item of weight 3 and fill the knapsack just right.

In the second test case, all the items are larger than the knapsack's capacity. Therefore, the answer is -1.

In the third test case, you fill the knapsack exactly in half.

D. Catching Cheaters

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

You are given two strings A and B representing essays of two students who are suspected cheaters. For any two strings C, D we define their similarity score $S(C, D)$ as $4 \cdot LCS(C, D) - |C| - |D|$, where $LCS(C, D)$ denotes the length of the Longest Common **Subsequence** of strings C and D.

You believe that only some part of the essays could have been copied, therefore you're interested in their **substrings**.

Calculate the maximal similarity score over all pairs of substrings. More formally, output maximal $S(C, D)$ over all pairs (C, D), where C is some substring of A, and D is some substring of B.

If X is a string, $|X|$ denotes its length.

A string a is a **substring** of a string b if a can be obtained from b by deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

A string a is a **subsequence** of a string b if a can be obtained from b by deletion of several (possibly, zero or all) characters.

Pay attention to the difference between the **substring** and **subsequence**, as they both appear in the problem statement.

You may wish to read the Wikipedia page about the Longest Common Subsequence problem.

Input

The first line contains two positive integers n and m ($1 \leq n, m \leq 5000$) — lengths of the two strings A and B.

The second line contains a string consisting of n lowercase Latin letters — string A.

The third line contains a string consisting of m lowercase Latin letters — string B.

Output

Output maximal $S(C, D)$ over all pairs (C, D), where C is some substring of A, and D is some substring of B.

Examples

input

```
4 5
abba
babab
```

output

```
5
```

input

```
8 10
bbbbabab
bbbabaaaaa
```

output

```
12
```

input

```
7 7
uiibwws
qhtkxcn
```

output
0

Note
For the first case:

abb from the first string and abab from the second string have LCS equal to abb.

The result is $S(abb, abab) = (4 \cdot |abb|) - |abb| - |abab| = 4 \cdot 3 - 3 - 4 = 5$.

E. Xor Tree

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

For a given sequence of **distinct** non-negative integers $(b_1, b_2, \ldots, b_k)$ we determine if it is **good** in the following way:

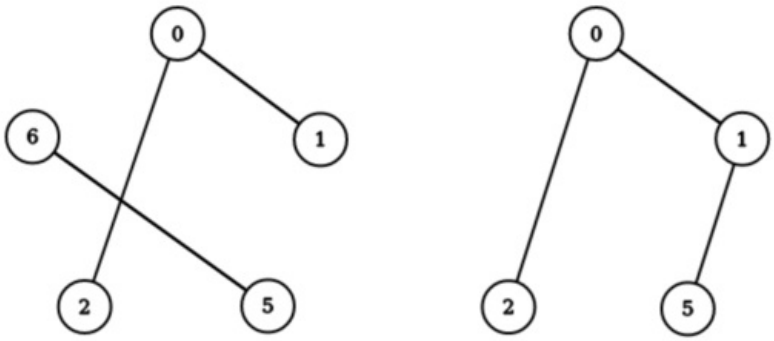
- Consider a graph on k nodes, with numbers from b_1 to b_k written on them.
- For every i from 1 to k: find such j ($1 \le j \le k$, $j \neq i$), for which $(b_i \oplus b_j)$ **is the smallest** among all such j, where $\oplus$ denotes the operation of bitwise XOR (https://en.wikipedia.org/wiki/Bitwise_operation#XOR). Next, draw an **undirected** edge between vertices with numbers b_i and b_j in this graph.
- We say that the sequence is **good** if and only if the resulting graph forms a **tree** (is connected and doesn't have any simple cycles).

It is possible that for some numbers b_i and b_j, you will try to add the edge between them twice. Nevertheless, you will add this edge only once.

You can find an example below (the picture corresponding to the first test case).

Sequence $(0, 1, 5, 2, 6)$ **is not** good as we **cannot** reach 1 from 5.

However, sequence $(0, 1, 5, 2)$ **is** good.



You are given a sequence $(a_1, a_2, \ldots, a_n)$ of **distinct** non-negative integers. You would like to remove some of the elements (possibly none) to make the **remaining** sequence good. What is the minimum possible number of removals required to achieve this goal?

It can be shown that for any sequence, we can remove some number of elements, leaving at least 2, so that the remaining sequence is good.

Input
The first line contains a single integer n ($2 \le n \le 200,000$) — length of the sequence.

The second line contains n **distinct** non-negative integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — the elements of the sequence.

Output
You should output exactly one integer — the minimum possible number of elements to remove in order to make the remaining sequence good.

Examples

input
5 0 1 5 2 6
output
1

input

7 6 9 8 7 3 5 2
output
2

Note
Note that numbers which you remove **don't** impact the procedure of telling whether the resulting sequence is good.

It is possible that for some numbers b_i and b_j, you will try to add the edge between them twice. Nevertheless, you will add this edge only once.

F1. Frequency Problem (Easy Version)

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

This is the easy version of the problem. The difference between the versions is in the constraints on the array elements. You can make hacks only if all versions of the problem are solved.

You are given an array $[a_1, a_2, \ldots, a_n]$.

Your goal is to find the length of the longest subarray of this array such that the most frequent value in it is **not** unique. In other words, you are looking for a subarray such that if the most frequent value occurs f times in this subarray, then at least 2 different values should occur exactly f times.

An array c is a subarray of an array d if c can be obtained from d by 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 a single integer n ($1 \le n \le 200\,000$) — the length of the array.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le min(n, 100)$) — elements of the array.

Output
You should output exactly one integer — the length of the longest subarray of the array whose most frequent value is not unique. If there is no such subarray, output 0.

Examples

input
7 1 1 2 2 3 3 3
output
6

input
10 1 1 1 5 4 1 3 1 2 2
output
7

input
1 1
output
0

Note
In the first sample, the subarray $[1, 1, 2, 2, 3, 3]$ is good, but $[1, 1, 2, 2, 3, 3, 3]$ isn't: in the latter there are 3 occurrences of number 3, and no other element appears 3 times.

F2. Frequency Problem (Hard Version)

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

This is the hard version of the problem. The difference between the versions is in the constraints on the array elements. You can make hacks only if all versions of the problem are solved.

You are given an array $[a_1, a_2, \ldots, a_n]$.

Your goal is to find the length of the longest subarray of this array such that the most frequent value in it is **not** unique. In other words, you are looking for a subarray such that if the most frequent value occurs f times in this subarray, then at least 2 different values should occur exactly f times.

An array c is a subarray of an array d if c can be obtained from d by 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 a single integer n ($1 \le n \le 200\,000$) — the length of the array.

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

Output

You should output exactly one integer — the length of the longest subarray of the array whose most frequent value is not unique. If there is no such subarray, output 0.

Examples

input
7 1 1 2 2 3 3 3
output
6

input
10 1 1 1 5 4 1 3 1 2 2
output
7

input
1 1
output
0

Note

In the first sample, the subarray $[1, 1, 2, 2, 3, 3]$ is good, but $[1, 1, 2, 2, 3, 3, 3]$ isn't: in the latter there are 3 occurrences of number 3, and no other element appears 3 times.