Codeforces Round #683 (Div. 1, by Meet IT)

A. 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.

B. 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 ($1 \le n, m \le 5000$) — 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 $S(abb, abab) = (4 \cdot |abb|)$ - $|abb|$ - $|abab| = 4 \cdot 3 - 3 - 4 = 5$.

C. 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:

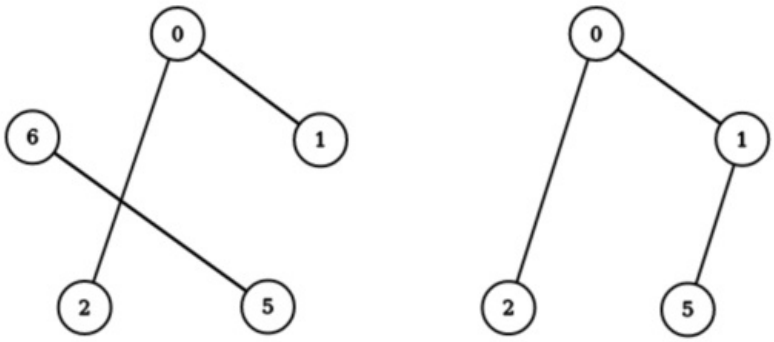
- Consider a graph on k nodes, with numbers from b_1 to b_k written on them.
- For every i from 1 to k: find such j ($1 \le j \le k$, $j \neq i$), for which $(b_i \oplus b_j)$ **is the smallest** among all such j, where $\oplus$ denotes the operation of bitwise XOR (https://en.wikipedia.org/wiki/Bitwise_operation#XOR). 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 ($2 \leq n \leq 200,000$) — length of the sequence.

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

D1. 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.

D2. 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

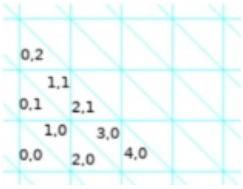
input
1 1
output
0

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.

E. Long Recovery

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A patient has been infected with an unknown disease. His body can be seen as an infinite grid of triangular cells which looks as follows:



Two cells are neighboring if they share a side. Therefore, each cell (x, y) has exactly three neighbors:

- ($x + 1$, y)
- ($x - 1$, y)
- ($x + 1$, $y - 1$) if x is even and ($x - 1$, $y + 1$) otherwise.

Initially some cells are infected, all the others are healthy. The process of recovery begins. Each second, for **exactly one** cell (even though there might be multiple cells that could change its state) one of the following happens:

- A healthy cell with at least 2 infected neighbors also becomes infected.
- An infected cell with at least 2 healthy neighbors also becomes healthy.

If no such cell exists, the process of recovery stops. Patient is considered recovered if the process of recovery has stopped and all the cells are healthy.

We're interested in a **worst-case** scenario: is it possible that the patient never recovers, or if it's not possible, what is the maximum possible duration of the recovery process?

Input
The first line contains one integer n $(1 \le n \le 250000)$ — the number of infected cells.

The i-th of the next n lines contains two space-separated integers x_i and y_i $(0 \le x_i, y_i < 500)$, meaning that cell (x_i, y_i) is infected. All cells (x_i, y_i) are distinct, and all other cells are considered healthy.

Output
If it is possible that the organism never fully recovers from the disease, print SICK. Otherwise, you should print RECOVERED and in the next line an integer k — the longest possible recovery period, modulo 998244353.

Examples

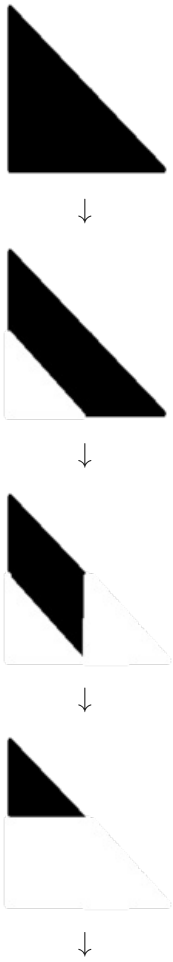
input
4 0 0 1 0 2 0 0 1
output
RECOVERED 4

input
3 1 0 3 0 1 1

output
SICK

Note

For the first testcase, the following drawings describe the longest possible recovery process. It can be proven that there are no recovery periods of length 5 or longer, and the organism always recovers in this testcase.



RECOVERED

For the second testcase, it is possible for the cells $(2, 0)$, $(2, 1)$, $(0, 1)$ to become infected. After that, no cell can change its state, so the answer is SICK, as not all of the cells are healthy.

F. Line Distance

time limit per test: 7.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an integer k and n distinct points with integer coordinates on the Euclidean plane, the i-th point has coordinates (x_i, y_i).

Consider a list of all the $\frac{n(n-1)}{2}$ pairs of points $((x_i, y_i), (x_j, y_j))$ $(1 \le i < j \le n)$. For every such pair, write out the distance from the line through these two points to the origin $(0, 0)$.

Your goal is to calculate the k-th smallest number among these distances.

Input

The first line contains two integers n, k $(2 \le n \le 10^5, 1 \le k \le \frac{n(n-1)}{2})$.

The i-th of the next n lines contains two integers x_i and y_i $(-10^4 \le x_i, y_i \le 10^4)$ — the coordinates of the i-th point. It is guaranteed that all given points are pairwise distinct.

Output

You should output one number — the k-th smallest distance from the origin. Your answer is considered correct if its absolute or relative error does not exceed 10^{-6}.

Formally, let your answer be a, and the jury's answer be b. Your answer is accepted if and only if $\frac{|a-b|}{\max(1, |b|)} \le 10^{-6}$.

Example

input
4 3 2 1 -2 -1 0 -1 -2 4
output
0.707106780737

Note

There are 6 pairs of points:

- Line $1 - 2$: distance 0 from the origin
- Line $1 - 3$: distance $\frac{\sqrt{2}}{2} \approx 0.707106781$ from the origin
- Line $1 - 4$: distance 2 from the origin
- Line $2 - 3$: distance 1 from the origin
- Line $2 - 4$: distance 2 from the origin
- Line $3 - 4$: distance $\frac{2}{\sqrt{29}} \approx 0.371390676$ from the origin

The third smallest distance among those is approximately 0.707106781.