Tsinghua Bootcamp 2024. Day 1

A. Palindromization

4 seconds, 256 megabytes

Palindromes are sequences that read the same both left-to-right and right-to-left. Such sequences play a significant role in string algorithms. A certain programmer believes that working with palindromes is easier than with any other strings or arrays, so whenever he sees an array, he strives to turn it into a palindrome.

Today this programmer received an array a consisting of n integers. In one operation, he can choose any segment of this array and add the same number x from 1 to k inclusive to all elements on this segment.

Despite being accustomed to working with palindromes, this time he encountered problems with turning a into a palindrome with the minimum number of actions. Help him with solving this problem.

Input

The first line of input contains two integers n and k — the size of the original array and the limit on the added values ($1 \le n \le 10^6$; $1 \le k \le 3$; if $k = 3$ then $n \le 1000$).

The second line of input contains n space-separated integers a_i — the elements of the array ($1 \le a_i \le 10^9$).

It's very important to notice that in case of $k = 3$, an additional constraint of $n \le 1000$ is introduced. Meaning that in every test where $k = 3$, also $n \le 1000$ holds true.

Output

Output a single number — the minimum number of operations required to turn the array a into a palindrome.

input
5 1 1 2 3 4 5
output
4

input
6 2 3 6 4 1 2 5
output
4

input
8 3 1 4 3 1 2 1 1 2
output
3

In the first example, you need to add 1 to the segment $[1, 3]$, add 1 to the segment $[1, 2]$, and then add 1 twice to the first element. Thus, after four operations, the array equals $[5, 4, 4, 4, 5]$.

In the second example, you need to add 2 to the segment $[1, 1]$, add 2 to the segment $[4, 5]$, and add 1 to the segment $[4, 5]$, after which add 1 to the last element. Thus, after four operations, the array equals $[5, 6, 4, 4, 6, 5]$. Notice that because $k = 2$, we can't add 3 to the segment $[4, 5]$ in one action.

In the third example, you need to add 1 to the segment $[1, 4]$, add 3 to the segment $[6, 7]$, and add 1 to the segment $[7, 7]$. Thus, after three operations, the array equals $[2, 5, 4, 2, 2, 4, 5, 2]$.

B. Deck for Magic Tricks

2 seconds, 256 megabytes

Recently, decks of playing cards of a new format have been released. The value of each card in the deck is represented by a `base64` string of length k. In the context of this problem, a `base64` string is a string where each character can take one of the following 64 values: '`#`', '`$`', '`0`'–'`9`', '`A`'–'`Z`', and '`a`'–'`z`'.

Each deck consists of $n = 64^k$ cards and contains exactly one card with each possible value. Moreover, in the sealed new deck, the cards are arranged in the order of lexicographic increasing of their values. Recall that a string s is *lexicographically greater* than a string t if and only if there exists an index i such that $s_1 = t_1$, ..., $s_{i-1} = t_{i-1}$, and $s_i > t_i$. In other words, when after their common (possibly empty) prefix, s has a greater character than t.

Famous magician shuffles the deck, repeating the following action an infinite number of times:

1. First, he divides the deck into two halves — the first half contains the cards from position 1 to $\frac{n}{2}$, and the second half contains the cards from $\frac{n}{2} + 1$ to n (the order of the cards does not change).
2. Then, he reassembles the deck, strictly alternating the cards from the two halves, starting with the first half: if we number the cards from 1 to n before shuffling, then after shuffling, the deck will look as follows:

$$\left\langle 1, \frac{n}{2} + 1, 2, \frac{n}{2} + 2, \ldots, \frac{n}{2}, n \right\rangle.$$

He plans to perform m magic tricks. For the i-th trick to succeed, he needs the card with value x_i to be in the position where the card with value y_i was in the original (sealed) deck. Assuming that the magician takes a new sealed deck for each trick, help him determine the minimum number of shuffles needed for each trick.

Input

The first line of input contains two integers k and m — the length of each card value and the number of tricks ($1 \le k \le 2500$; $1 \le m \le 1000$).

The next m lines list the descriptions of the tricks. The i-th line contains two strings x_i and y_i — the values of the cards involved in the i-th trick ($|x_i| = |y_i| = k$). It is guaranteed that x_i and y_i consist only of the characters specified in the problem statement.

Output

For each trick, output a separate line containing the shuffle number after which the card with value x_i will first be in the position y_i.

If the card is initially in the correct position, output 0. If such an event will never occur, output -1.

input
1 7 # # U $ 4 3 t D D t 2 $ $ 2
output
0 1 -1 3 3 4 2

input
3 8 Hd7 CYZ mZ$ 1Z8 iYq poa JlP edh SyR uxw aCp n5O I#g Oq8 wrP tir

output

```
2
13
15
-1
13
17
-1
1
```

The ASCII codes of the characters used in the card values take the values 35 ('#'), 36 ('$'), from 48 to 57 (digits), from 65 to 90 (uppercase Latin letters), and from 97 to 122 (lowercase Latin letters).

C. Unfair Game

2 seconds, 256 megabytes

On a grid of size $n \times m$, consisting of n rows and m columns, there is a token located in cell (r, c), which is at the intersection of the r-th row from the top and the c-th column from the left. You will play a game against the computer on this board, where you can move the token and remove cells from the board.

Each turn is structured as follows:

1. The computer announces an integer $k > 0$.
2. You must do the following exactly k times: choose one of the still unremoved cells that is adjacent (sharing a side) to the cell where the token is currently located, and move the token to that cell. You can move the token to a cell it has already occupied before. If there are no unremoved cells adjacent to the current cell, no movement occurs.
3. The computer announces the coordinates (i, j) of any still unremoved cell on the board, after which it is immediately removed.

If the computer removes the cell where the token is located, the game ends with your victory. Your goal is to win as soon as possible. Since you do not inform the computer of your movements, you can play unfairly: instead of actually moving the token on the board, you can keep track of all possible positions it could be in. In other words, if at some point, when the cell (i, j) is removed, there exists a sequence of moves that would place the token exactly in cell (i, j) at that moment, you can inform the computer that the game is over and that you have won.

Of course, the computer monitors compliance with the rules, so if you inform it of your victory when the token could not have been in that cell at that moment, you automatically lose.

Determine the smallest turn number after which you can inform the computer of your victory, that is, after which the token could have been in the removed cell.

Input

The input consists of a single line containing four integers n, m, r, and c — the dimensions of the board and the coordinates of the initial position of the token ($1 \le r \le n \le 1000$; $1 \le c \le m \le 1000$).

In the following $n \cdot m$ lines, the moves that the computer will make sequentially are provided. The description of the t-th move consists of three integers k_t, i_t, and j_t — the number of moves you need to make with the token, and the coordinates of the cell on the board that needs to be removed afterward ($1 \le k_t \le 10^9$; $1 \le i_t \le n$; $1 \le j_t \le m$).

It is guaranteed that all removed cells are distinct, meaning no cell is removed twice.

Output

Output a single integer from 1 to $n \cdot m$ — the turn number after which you will inform the computer of your victory.

input

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

output

```
2
```

input

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

output

```
9
```

In the first example, for instance, you can move the token from $(1, 1)$ to $(1, 2)$ on the first turn, and then from $(1, 2)$ to $(2, 2)$ and then to $(2, 1)$ on the second turn, thus placing it on the removed cell.

D. Hackathon

2 seconds, 512 megabytes

Imagine yourself organizing a hackathon for a large number of participants. Good hackathons often include food and drinks, which can be enjoyed over the several days that participants spend at the venue.

You have to arrange food for m participants. And for this reason, you have ordered n pizzas which are arranged in a row and numbered from 1 to n. It is known that:

- each pizza consists of an infinite number of slices;
- the first slice of the i-th pizza has a dough thickness of a_i, and each subsequent one is 1 thicker than the previous (i.e., $a_i + 1$, $a_i + 2$, and so on);
- the slices with the least dough thickness seem most attractive to participants;
- any participant can approach the row of pizzas at any point, but afterwards can only move to the left (towards the pizzas with smaller numbers) to avoid collisions with other participants.

At some moments in time, participants get up from their places and approach the row of pizzas to take a slice. Participant number i approaches the pizza numbered s_i at time t_i, after which they immediately find the slice of pizza with the least dough thickness among pizzas from 1 to s_i inclusive and start moving towards it. It takes each participant the same amount of time to move between two adjacent pizzas, which is exactly v seconds. Several participants can be near the same pizza at the same time.

As soon as a participant reaches the pizza they have chosen, they take the slice and return to their seat. If several participants have chosen the same slice and reach it simultaneously, the participant with the lower number takes the slice. If the slice a participant was counting on is taken by someone else, that participant immediately chooses a new target using the same algorithm: they look at all the pizzas from the first to their current position, select the minimum among them, and start moving towards it.

Formally, this process can be described as follows. At every whole number of seconds t:

1. all participants who were already near the row of pizzas at time $t - 1$ move by $\frac{1}{v}$ to the left;
2. all participants with $t_i = t$ approach the pizza number s_i;
3. time "stops";
4. all participants standing at **whole** positions p such that $a_p = \min(a_1, a_2, \ldots, a_p)$ are identified;
5. for each such p, the participant with the minimum number standing at position p takes the slice of pizza with thickness a_p and leaves; a_p is increased by 1;
6. after some slices of pizza are taken, it may happen that the minimum thickness of the slice that some participants can still get has changed, in which case the process repeats from step 4;
7. otherwise, time "starts again" and the process moves to the next second.

Based on the information about participants' approaches to the pizzas, determine for each participant the thickness of the pizza slice they received and how much time they spent on the selection.

Input

The first line of input contains three integers n, m, and v — the number of pizzas, the number of participants, and the time it takes each participant to move between adjacent pizzas ($1 \le n, m \le 10^5$; $1 \le v \le 10^9$).

The second line lists n space-separated integers a_i — the thickness of the first slice of each pizza ($0 \le a_i \le 10^9$).

In the i-th of the following m lines, two integers s_i and t_i are given — the starting position and the time of the participant's approach to the row of pizzas ($1 \le s_i \le n$; $0 \le t_i \le 10^9$).

Output

Output m lines, in the i-th of which print a pair of space-separated integers: the dough thickness of the pizza slice received by the i-th participant and the time it took the participant to reach it.

input

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

output

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

input

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

output

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

In the first example, events will occur in the following order:

1. The first participant appears at position 5 at time 1; their target is pizza number 1.
2. By the time 2, the first participant is at position 4, and the second participant appears at position 3. Both will move towards pizza number 1.
3. In the third second, no changes occur, and the participants continue to move left.
4. At time 4: the second participant is near the first pizza, the first and third participants are together near the second. The second participant takes a slice of pizza with a thickness of 1, a_1 becomes equal to 2.
5. The first and third participants immediately see that they will no longer get a slice of pizza with a thickness of 1, but they can take one with a thickness of 2 at their current position. The first, as the participant with the lower number, takes $a_2 = 2$, after which a_2 becomes equal to 3.
6. The remaining third participant can only move one more to the left and take $a_1 = 2$.

E. Restore Permutation

2 seconds, 256 megabytes

This is a run-twice problem. Your solution will be executed twice.

During the first run, you are given p — a permutation of integers from 1 to n. Your program should output a binary string (a string of zeros and ones) of length not exceeding $m \ge 200$. The exact value of m may differ from test to test. If your program outputs a string longer than m, it will receive a verdict of "`Wrong Answer`".

Between the runs, the jury program will swap two different elements of the original permutation, obtaining permutation q.

During the second run, your program will be given permutation q and the binary string output by you in the first run as input. You are required to restore the original permutation.

Input

During the first run, the first input line contains the number 1 and an integer n — the length of the permutation ($2 \le n \le 10^6$). The second line contains n distinct integers p_i separated by a space — the elements of the permutation ($1 \le p_i \le n$).

During the second run, the first and second lines in the same format contain the number 2, the length of the permutation n, and the permutation q, and the third line contains a binary string of zeros (symbol '`0`') and ones (symbol '`1`') of length not exceeding m.

It is recommended to use $m = 200$ as a reference even if some tests can be passed with longer bit strings of information.

Output

During the first run, output a binary string of length not exceeding m, which will then be used to restore the permutation during the second run.

During the second run, output n distinct integers from 1 to n separated by a space — the elements of the original permutation p.

Interaction

To avoid receiving incorrect verdicts such as "`Idleness Limit Exceeded`" or "`Security Violation`", terminate the output of each line with a newline character ('`\n`').

Input	Output	Note
1 4 4 1 2 3	100001010011	First run
2 4 4 3 2 1 100001010011	4 1 2 3	Second run

Input	Output	Note
1 5 2 3 4 5 1	11110010011101010001111110011000100010011011110001111111101011100100	First run
2 5 2 3 5 4 1 11110010011101010001111110011000100010011011110001111111101011100100	2 3 4 5 1	Second run

In the examples, long binary strings are displayed with line breaks for correct display in the statement. In reality, there are no such line breaks. Also note that the input and output during the second run can depend on the output during the first run and may not match the examples shown in the statement.

F. Self-Descriptive

2 seconds, 512 megabytes

Consider an infinite sequence of integers $a_1, a_2, a_3, \ldots$. The sequence is called a *self-descriptive* if a_i is equal to the number of occurrences of i in this sequence.

If we restrict this sequence to satisfy $a_1 = 1$ and the numbers are non-decreasing (that is, $a_i \le a_{i+1}$ for all i), then the self-descriptive sequence is unique. The first thirty elements are as follows:

$1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9, 10, 1($

Your task is to calculate the range sum queries for this sequence efficiently.

Input
The first line contains an integer t — the number of range sum queries ($1 \le t \le 10^4$).

The following t lines contain two integers l_i and r_i each ($1 \le l_i \le r_i \le 10^{15}$). The i-th query requires you to calculate $\sum_{k=l_i}^{r_i} a_k$.

Output
Print t lines, each one with a single integer — the answer to the range sum query. Because the sum can be large, print it modulo $1\,000\,000\,007$.

input
5 1 1 2 2 3 4 56 56 5 13
output
1 2 5 14 42

G. Space Battleship

2.5 seconds, 256 megabytes

The game board for "Space Battleship" game consists of cells and has a width of w and a height of h. Ships can consist of one, two, or three consecutive cells either vertically or horizontally.

In total, there should be s_1 one-cell ships, s_2 two-cell ships, and s_3 three-cell ships on the board. Ships must not touch each other side by side or overlap, but they **can** touch at the corners.

You have to write a program that helps with anti-cheating, i.e., against players who tactically place ships on the board in real time during the game instead of placing them on the board beforehand.

Given the results of the first shots (several cells with the information either for each of them: there is definitely a ship in that cell, or there is definitely no ship), determine how many possible arrangements of ships there are for the opponent. Since this number can be very large, find it modulo $10^9 + 7$.

Input
The first line contains two integers w and h — the width and height of the game board, respectively ($w \le 100$; $h \le 8$).

The next h lines contain descriptions of the rows of the board. The symbol '`.`' denotes a cell that you have no information about, '`o`' denotes a cell without a ship (a miss), and '`x`' denotes a cell where a ship has been hit.

The last line contains the numbers s_1, s_2, and s_3 — the number of ships of each size that need to be placed on the board ($s_1 \le 5$; $s_2 \le 4$; $s_3 \le 3$).

Output
Output a single number — the number of distinct arrangements of the opponent's ships that satisfy the given data, taken modulo $10^9 + 7$.

Two arrangements are considered different if there is a cell occupied by a ship in one arrangement and free in another.

input
4 2 .ox. x.o. 2 1 0
output
2

input
3 3 .oo x.. .xx 0 2 0
output
1

H. Ancestral Problem

6 seconds, 256 megabytes

Let's take a look into a universe where each inhabitant is a tree described by the number of vertices n and $n - 1$ edges between them.

It is unknown why, but a certain scientist has been sitting and analyzing the *family ties* between the inhabitants of this dimension for three days now.

He believes that tree T_1 can be a descendant of tree T_2 if it is possible to add some (possibly zero) vertices and edges to T_2 and renumber its vertices so that it matches T_1.

You are tasked with helping him in analyzing t pairs of inhabitants from this dimension. For each given pair of trees, check if the first tree can be a descendant of the second tree.

Input
The first line contains an integer t — the number of pairs of trees that are interesting to the scientist ($1 \le t \le 10^4$). Following this, there are t descriptions of pairs of trees.

The first line of each description contains an integer n — the size of the first tree ($2 \le n \le 10^5$). Each of the next $n - 1$ lines contains two integers u_i and v_i — the ends of the i-th edge of the first tree ($1 \le u_i, v_i \le n$). The next two lines describe the second tree with m vertices in the same format.

It is guaranteed that the sum of n across all pairs of trees does not exceed $5 \cdot 10^5$, and the sum of $n \cdot m$ does not exceed 10^7.

Output
For each of the t pairs of trees, print "`YES`" if the first tree can be a descendant of the second tree, and "`NO`" otherwise.

input
2 5 1 2 1 5 2 3 2 4 4 1 2 1 3 1 4 6 1 2 1 3 1 4 5 1 6 1 4 1 2 2 3 3 4
output
YES NO

I. Secret Folder

1.5 seconds, 256 megabytes

A super secret folder is protected by all sorts of locks and passwords.

But you might be surprised that the last layer of protection consists of n ordinary master passwords in the form of strings. The owner must have thought that no one could break all the previous layers of protection.

However, today, out of boredom, the owner wondered what is the shortest string that could serve as a master password instead of all n passwords currently implemented. For a string to qualify as a single master password, it must contain all n master passwords as continuous substrings in any order.

Find the shortest master password that could replace n existing ones.

Input

The first line contains an integer t — the number of test cases ($1 \le t \le 30$). Following this are t sets of input data, each set described by $n + 1$ lines.

The first line of the test case contains an integer n — the number of master passwords ($1 \le n \le 17$).

In the next n lines, the i-th line contains the string s_i — the i-th master password ($1 \le |s_i| \le 5 \cdot 10^4$). The passwords consist of lowercase letters of the Latin alphabet (from 'a' to 'z').

It is guaranteed that the sum of n across all test cases does not exceed 30.

Output

For each test case output the string of minimal length that contains all passwords as continuous substrings in any order in a separate line. If there are multiple possible answers, output any of them.

input
3 3 abacaba baba saba 4 xzy yyxx yyy xx 5 c abcde cde bcde cd

output
sababacaba yyyxxzy abcde

J. Tree Trisection

4 seconds, 256 megabytes

On the territory of a certain popular university, there are many trees that require maintenance. In particular, full binary trees sometimes need to be pruned to prevent them from growing too quickly.

Consider a full binary tree with $n = 2^k$ leaves. Let's assign number 1 to the root, numbers 2 and 3 to its children (from left to right), and so on, numbering each layer of vertices with consecutive integers. The leaves of the tree will have numbers from n to $2n - 1$. The weight of a vertex number i is equal to w_i. The weight of a set of vertices is defined as the sum of the weights of the vertices in that set.

Your task is to determine how to prune the trees efficiently. A *pruning* request is specified by the leaf numbers l and r, and is carried out as follows:

1. First, a set of vertices V is selected — this is the smallest connected set of vertices that contains **all** the leaves from the l-th to the r-th. Let's denote the vertex of the set V that is closest to the root of the tree as $\texttt{root}(V)$.
2. Then, it is required to select in the set V two different vertices x and y such that neither of them is a descendant of the other.
3. Once the vertices x and y are chosen, the edges leading from $\texttt{root}(V)$, x, and y "upwards" (towards the root) are cut. As a result, V is separated from the entire tree and breaks down into three subsets, denoted as $V_{\texttt{root}}$, V_x, and V_y, respectively.
4. After that, all three of these subsets are planted as separate sub-trees to enhance the landscaping of the territory.

The management has several options for how to prune the given tree. Your goal is to determine for each of their requests (l, r) which vertices x and y should be chosen to minimize the maximum weight among $V_{\texttt{root}}$, V_x, and V_y.

You don't have to find the best possible answer. Any approximate answer not worse than jury's obtained via some assumptions (for example, by selecting one of the sub-trees to be close to $\frac{1}{3}$ of the weight of V) will be accepted.

Input

The first line contains two integers n and m — the number of leaves in the tree and the number of management's requests ($2 \le n \le 2^{16}$; $1 \le m \le 10^5$; $n = 2^k$ for some integer k).

The next line lists $2n - 1$ integers w_i — the weights of the tree vertices ($1 \le w_i \le 10^9$).

In the i-th of the following m lines, the i-th management's request is given as two integers l_i and r_i — the number of the first and last leaves involved in the pruning of the tree ($n \le l_i, r_i \le 2n - 1$; $r_i \ge l_i + 1$).

Output

For each request, output in a separate line two space-separated integers x and y — the numbers of additional vertices where the tree should be cut to minimize the maximum weight of the three cut parts.

If there are several options for answers that minimize the maximum weight of the cut part, output any of them.

input
4 4 1 1 1 1 1 1 1 4 5 5 6 6 7 4 7

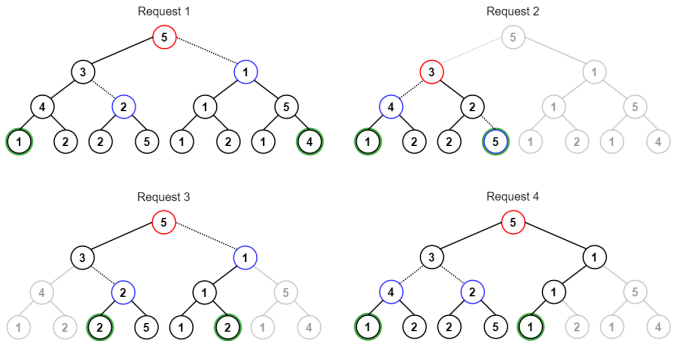
output
4 5 5 3 6 7 5 3

input
8 7 5 3 1 4 2 1 5 1 2 2 5 1 2 1 4 8 15 8 11 10 13 8 12 11 12 9 14 13 15

output
5 3 4 11 5 3 4 5 5 3 5 3 6 15

Below is an illustration for the first four requests in the second example from the problem statement.

The weight of each vertex is marked. Vertices that are not included in the set V are highlighted in gray. The selected leaves l and r are circled in green. The vertex $\texttt{root}(V)$ is circled in red, and the vertices x and y, which need to be chosen as the answer, are circled in blue. The edges that are cut during the corresponding *pruning* are highlighted with a dotted line.



K. Public Transportation

2 seconds, 256 megabytes

The project of a new city involves the construction of a grid of size $n \times m$ cells. A house with $t_{i,j}$ residents will be located at the intersection of the i-th row and the j-th column.

Three houses in cells (i, j), $(i + a, j)$ and $(i, j + b)$ form a *good triangle* suitable for a public transportation line if:

- $a > 0$ and $b > 0$;
- in the house (i, j), exactly $a + b$ residents live;
- in the house $(i + a, j)$, exactly b residents live;
- and in the house $(i, j + b)$, exactly a residents live.

You can change the construction plan by increasing or decreasing all $t_{i,j}$ by the same integer d. If $t_{i,j}$ should become less than zero, consider it to be equal to zero. Let $T + d$ denote the matrix of values $t_{i,j} + d$, and let $\triangle(T + d)$ be the number of good triangles when the city is constructed accordingly.

Find

$$\sum_{d=-\infty}^{+\infty} \triangle(T + d),$$

or in other words, the total number of good triangles for all possible construction plans.

Input

The first line of input contains two integers n and m separated by a space, representing the number of rows and columns of the grid, respectively ($1 \le n, m \le 1000$).

In the i-th of the following n lines, there are m integers $t_{i,j}$ listed, representing the number of residents in each house of the i-th row of the grid ($1 \le t_{i,j} \le 10^9$).

Output

Output a single integer, the number of sets of three cells that satisfy the given conditions.

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

input
2 4 5 4 3 2 4 3 2 1
output
6

L. Crossbreeding

3 seconds, 1024 megabytes

You are the owner of a zoo that currently houses n rare exotic species of snakes, the i-th of which has a *danger* level of a_i.

You are not very keen on keeping animals in captivity, but of course, you cannot release dangerous snakes into the general environment either. To solve this problem, you decided to crossbreed some species, which will not reduce their exoticism but may decrease their danger level. For one crossbreeding, you can choose two species of snakes with indices i and j, and crossbreed them into a new species with a danger level of $a^*_{i,j} = a_i \oplus a_j$, where $\oplus$ denotes the bitwise "exclusive OR" operation (also referred to as `xor` or `^`).

To avoid creating overly similar species, each of the n existing species can participate in only one crossbreeding. It is also impossible to crossbreed two species if at least one of them is already a result of crossbreeding some of the original n species. In other words, only the original species can be crossbred, and only in pairs.

In the end, you will obtain a new set of species, which will include the original species that did not participate in any crossbreeding, as well as all the results of the crossbreedings. That is, the original species that were crossbred with others will not be included in this set (again, because there is no point in allocating a separate space in the zoo for several closely related species — visitors will not see the difference anyway).

Determine which pairs of snake species should be crossbred so that the maximum danger level of the resulting set of species is minimized.

Input

The first line contains an integer n — the number of exotic snake species initially available ($1 \le n \le 10^6$).

The second line lists n integers a_i — the danger levels of these species ($0 \le a_i \le 10^9$).

Output

Output a single integer — the minimum possible value of the maximum danger level that can be achieved through such crossbreeding.

input
3 1 2 3
output
1

input
7 1 4 9 16 25 36 49
output
21

input
6 0 1 2 5 6 7
output
4

In the first example, it is beneficial to crossbreed species 2 and 3, resulting in the original species with $a_1 = 1$ and a new species with $a_{2,3} = 1$.

In the second example, species 16 is crossbred with 25 (resulting in $a_{4,5} = 9$) and species 36 with 49 (resulting in $a_{6,7} = 21$).

In the third example, species 2 is crossbred with 6 and species 5 with 7.