A. Tourist Problem

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

Iahub is a big fan of tourists. He wants to become a tourist himself, so he planned a trip. There are n destinations on a straight road that Iahub wants to visit. Iahub starts the excursion from kilometer 0. The n destinations are described by a non-negative integers sequence $a_1, a_2, ..., a_n$. The number a_k represents that the kth destination is at distance a_k kilometers from the starting point. No two destinations are located in the same place.

Iahub wants to visit each destination only once. Note that, crossing through a destination is not considered visiting, unless Iahub explicitly wants to visit it at that point. Also, after Iahub visits his last destination, he doesn't come back to kilometer 0, as he stops his trip at the last destination.

The distance between destination located at kilometer x and next destination, located at kilometer y, is $|x - y|$ kilometers. We call a "route" an order of visiting the destinations. Iahub can visit destinations in any order he wants, as long as he visits all n destinations and he doesn't visit a destination more than once.

Iahub starts writing out on a paper all possible routes and for each of them, he notes the total distance he would walk. He's interested in the average number of kilometers he would walk by choosing a route. As he got bored of writing out all the routes, he asks you to help him.

Input

The first line contains integer n ($2 \le n \le 10^5$). Next line contains n distinct integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^7$).

Output

Output two integers — the numerator and denominator of a fraction which is equal to the wanted average number. The fraction must be irreducible.

Sample test(s)

input

```
3
2 3 5
```

output

```
22 3
```

Note

Consider 6 possible routes:

- [2, 3, 5]: total distance traveled: |2 – 0| + |3 – 2| + |5 – 3| = 5;
- [2, 5, 3]: |2 – 0| + |5 – 2| + |3 – 5| = 7;
- [3, 2, 5]: |3 – 0| + |2 – 3| + |5 – 2| = 7;
- [3, 5, 2]: |3 – 0| + |5 – 3| + |2 – 5| = 8;
- [5, 2, 3]: |5 – 0| + |2 – 5| + |3 – 2| = 9;
- [5, 3, 2]: |5 – 0| + |3 – 5| + |2 – 3| = 8.

The average travel distance is $\frac{1}{6} \cdot (5 + 7 + 7 + 8 + 9 + 8) = \frac{44}{6} = \frac{22}{3}$.

B. Bubble Sort Graph

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

Iahub recently has learned Bubble Sort, an algorithm that is used to sort a permutation with n elements a_1, a_2, ..., a_n in ascending order. He is bored of this so simple algorithm, so he invents his own graph. The graph (let's call it G) initially has n vertices and 0 edges. During Bubble Sort execution, edges appear as described in the following algorithm (pseudocode).

```
procedure bubbleSortGraph()
    build a graph G with n vertices and 0 edges
    repeat
        swapped = false
        for i = 1 to n - 1 inclusive do:
            if a[i] > a[i + 1] then
                add an undirected edge in G between a[i] and a[i + 1]
                swap( a[i], a[i + 1] )
                swapped = true
            end if
        end for
    until not swapped
    /* repeat the algorithm as long as swapped value is true. */
end procedure
```

For a graph, an independent set is a set of vertices in a graph, no two of which are adjacent (so there are no edges between vertices of an independent set). A maximum independent set is an independent set which has maximum cardinality. Given the permutation, find the size of the maximum independent set of graph G, if we use such permutation as the premutation a in procedure bubbleSortGraph.

Input

The first line of the input contains an integer n $(2 \le n \le 10^5)$. The next line contains n distinct integers a_1, a_2, ..., a_n $(1 \le a_i \le n)$.

Output

Output a single integer — the answer to the problem.

Sample test(s)

input

```
3
3 1 2
```

output

```
2
```

Note

Consider the first example. Bubble sort swaps elements 3 and 1. We add edge (1, 3). Permutation is now [1, 3, 2]. Then bubble sort swaps elements 3 and 2. We add edge (2, 3). Permutation is now sorted. We have a graph with 3 vertices and 2 edges (1, 3) and (2, 3). Its maximal independent set is [1, 2].

C. Iahub and Permutations

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

Iahub is so happy about inventing bubble sort graphs that he's staying all day long at the office and writing permutations. Iahubina is angry that she is no more important for Iahub. When Iahub goes away, Iahubina comes to his office and sabotage his research work.

The girl finds an important permutation for the research. The permutation contains n distinct integers a_1, a_2, ..., a_n $(1 \le a_i \le n)$. She replaces some of permutation elements with -1 value as a revenge.

When Iahub finds out his important permutation is broken, he tries to recover it. The only thing he remembers about the permutation is it didn't have any fixed point. A fixed point for a permutation is an element a_k which has value equal to k $(a_k = k)$. Your job is to proof to Iahub that trying to recover it is not a good idea. Output the number of permutations which could be originally Iahub's important permutation, modulo 1000000007 $(10^9 + 7)$.

Input

The first line contains integer n $(2 \le n \le 2000)$. On the second line, there are n integers, representing Iahub's important permutation after Iahubina replaces some values with -1.

It's guaranteed that there are no fixed points in the given permutation. Also, the given sequence contains at least two numbers -1 and each positive number occurs in the sequence at most once. It's guaranteed that there is at least one suitable permutation.

Output

Output a single integer, the number of ways Iahub could recover his permutation, modulo 1000000007 $(10^9 + 7)$.

Sample test(s)

input

```
5
-1 -1 4 3 -1
```

output

```
2
```

Note

For the first test example there are two permutations with no fixed points are [2, 5, 4, 3, 1] and [5, 1, 4, 3, 2]. Any other permutation would have at least one fixed point.

D. Iahub and Xors

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

Iahub does not like background stories, so he'll tell you exactly what this problem asks you for.

You are given a matrix a with n rows and n columns. Initially, all values of the matrix are zeros. Both rows and columns are 1-based, that is rows are numbered 1, 2, ..., n and columns are numbered 1, 2, ..., n. Let's denote an element on the i-th row and j-th column as $a_{i,j}$.

We will call a submatrix (x_0, y_0, x_1, y_1) such elements $a_{i,j}$ for which two inequalities hold: $x_0 \le i \le x_1$, $y_0 \le j \le y_1$.

Write a program to perform two following operations:

1. Query(x_0, y_0, x_1, y_1): print the xor sum of the elements of the submatrix (x_0, y_0, x_1, y_1).
2. Update(x_0, y_0, x_1, y_1, v): each element from submatrix (x_0, y_0, x_1, y_1) gets xor-ed by value v.

Input

The first line contains two integers: n $(1 \le n \le 1000)$ and m $(1 \le m \le 10^5)$. The number m represents the number of operations you need to perform. Each of the next m lines contains five or six integers, depending on operation type.

If the i-th operation from the input is a query, the first number from i-th line will be 1. It will be followed by four integers x_0, y_0, x_1, y_1. If the i-th operation is an update, the first number from the i-th line will be 2. It will be followed by five integers x_0, y_0, x_1, y_1, v.

It is guaranteed that for each update operation, the following inequality holds: $0 \le v < 2^{62}$. It is guaranteed that for each operation, the following inequalities hold: $1 \le x_0 \le x_1 \le n$, $1 \le y_0 \le y_1 \le n$.

Output

For each query operation, output on a new line the result.

Sample test(s)

input

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

output

```
3
2
```

Note

After the first 3 operations, the matrix will look like this:

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

The fourth operation asks us to compute 1 xor 2 xor 3 xor 3 = 3.

The fifth operation asks us to compute 1 xor 3 = 2.

E. Candies Game

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

Iahub is playing an uncommon game. Initially, he has n boxes, numbered 1, 2, 3, $\dots$, n. Each box has some number of candies in it, described by a sequence a_1, a_2, ..., a_n. The number a_k represents the number of candies in box k.

The goal of the game is to move all candies into **exactly** two boxes. The rest of $n - 2$ boxes must contain **zero** candies. Iahub is allowed to do several (possible zero) moves. At each move he chooses two different boxes i and j, such that $a_i \le a_j$. Then, Iahub moves from box j to box i exactly a_i candies. Obviously, when two boxes have equal number of candies, box number j becomes empty.

Your task is to give him a set of moves such as Iahub to archive the goal of the game. If Iahub can't win the game for the given configuration of boxes, output -1. Please note that in case there exist a solution, you don't need to print the solution using minimal number of moves.

Input

The first line of the input contains integer n ($3 \le n \le 1000$). The next line contains n non-negative integers: $a_1, a_2, \dots, a_n$ — sequence elements. It is guaranteed that sum of all numbers in sequence a is up to 10^6.

Output

In case there exists no solution, output -1. Otherwise, in the first line output integer c ($0 \le c \le 10^6$), representing number of moves in your solution. Each of the next c lines should contain two integers i and j ($1 \le i, j \le n$, $i \ne j$): integers i, j in the kth line mean that at the k-th move you will move candies from the j-th box to the i-th one.

Sample test(s)

input

```
3
3 6 9
```

output

```
2
2 3
1 3
```

input

```
3
0 1 0
```

output

```
-1
```

input

```
4
0 1 1 0
```

output

```
0
```

Note

For the first sample, after the first move the boxes will contain 3, 12 and 3 candies. After the second move, the boxes will contain 6, 12 and 0 candies. Now all candies are in exactly 2 boxes.

For the second sample, you can observe that the given configuration is not valid, as all candies are in a single box and they should be in two boxes. Also, any move won't change the configuration, so there exists no solution.

For the third sample, all candies are already in 2 boxes. Hence, no move is needed.