Codeforces Round #493 (Div. 2)

A. Balloons

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

There are quite a lot of ways to have fun with inflatable balloons. For example, you can fill them with water and see what happens.

Grigory and Andrew have the same opinion. So, once upon a time, they went to the shop and bought $$$n$$$ packets with inflatable balloons, where $$$i$$$-th of them has exactly $$$a_i$$$ balloons inside.

They want to divide the balloons among themselves. In addition, there are several conditions to hold:

- Do not rip the packets (both Grigory and Andrew should get unbroken packets);
- Distribute all packets (every packet should be given to someone);
- Give both Grigory and Andrew at least one packet;
- To provide more fun, the total number of balloons in Grigory's packets should not be equal to the total number of balloons in Andrew's packets.

Help them to divide the balloons or determine that it's impossible under these conditions.

Input
The first line of input contains a single integer $$$n$$$ ($$$1 \le n \le 10$$$) — the number of packets with balloons.

The second line contains $$$n$$$ integers: $$$a_1$$$, $$$a_2$$$, $$$\ldots$$$, $$$a_n$$$ ($$$1 \le a_i \le 1000$$$) — the number of balloons inside the corresponding packet.

Output
If it's impossible to divide the balloons satisfying the conditions above, print $$$-1$$$.

Otherwise, print an integer $$$k$$$ — the number of packets to give to Grigory followed by $$$k$$$ distinct integers from $$$1$$$ to $$$n$$$ — the indices of those. The order of packets doesn't matter.

If there are multiple ways to divide balloons, output any of them.

Examples

input
3 1 2 1
output
2 1 2

input
2 5 5
output
-1

input
1 10
output
-1

Note
In the first test Grigory gets $$$3$$$ balloons in total while Andrey gets $$$1$$$.

In the second test there's only one way to divide the packets which leads to equal numbers of balloons.

In the third test one of the boys won't get a packet at all.

B. Cutting

time limit per test: 2 seconds

memory limit per test: 256 megabytes
input: standard input
output: standard output

There are a lot of things which could be cut — trees, paper, "the rope". In this problem you are going to cut a sequence of integers.

There is a sequence of integers, which contains the equal number of even and odd numbers. Given a limited budget, you need to make maximum possible number of cuts such that each resulting segment will have the same number of odd and even integers.

Cuts separate a sequence to continuous (contiguous) segments. You may think about each cut as a break between two adjacent elements in a sequence. So after cutting each element belongs to exactly one segment. Say, $$$[4, 1, 2, 3, 4, 5, 4, 4, 5, 5]$$$ $$$\to$$$ two cuts $$$\to$$$ $$$[4, 1 | 2, 3, 4, 5 | 4, 4, 5, 5]$$$. On each segment the number of even elements should be equal to the number of odd elements.

The cost of the cut between $$$x$$$ and $$$y$$$ numbers is $$$|x - y|$$$ bitcoins. Find the maximum possible number of cuts that can be made while spending no more than $$$B$$$ bitcoins.

Input
First line of the input contains an integer $$$n$$$ ($$$2 \le n \le 100$$$) and an integer $$$B$$$ ($$$1 \le B \le 100$$$) — the number of elements in the sequence and the number of bitcoins you have.

Second line contains $$$n$$$ integers: $$$a_1$$$, $$$a_2$$$, ..., $$$a_n$$$ ($$$1 \le a_i \le 100$$$) — elements of the sequence, which contains the equal number of even and odd numbers

Output
Print the maximum possible number of cuts which can be made while spending no more than $$$B$$$ bitcoins.

Examples

input
6 4 1 2 5 10 15 20
output
1

input
4 10 1 3 2 4
output
0

input
6 100 1 2 3 4 5 6
output
2

Note
In the first sample the optimal answer is to split sequence between $$$2$$$ and $$$5$$$. Price of this cut is equal to $$$3$$$ bitcoins.

In the second sample it is not possible to make even one cut even with unlimited number of bitcoins.

In the third sample the sequence should be cut between $$$2$$$ and $$$3$$$, and between $$$4$$$ and $$$5$$$. The total price of the cuts is $$$1 + 1 = 2$$$ bitcoins.

C. Convert to Ones

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

You've got a string $$$a_1, a_2, \dots, a_n$$$, consisting of zeros and ones.

Let's call a sequence of consecutive elements $$$a_i, a_{i + 1}, \ldots, a_j$$$ ($$$1\leq i\leq j\leq n$$$) a *substring* of string $$$a$$$.

You can apply the following operations any number of times:

- Choose some substring of string $$$a$$$ (for example, you can choose entire string) and reverse it, paying $$$x$$$ coins for it (for example, «01011101» $$$\to$$$ «01110101»);
- Choose some substring of string $$$a$$$ (for example, you can choose entire string or just one symbol) and replace each symbol to the opposite one (zeros are replaced by ones, and ones — by zeros), paying $$$y$$$ coins for it (for example, «01011101» $$$\to$$$ «01100001»).

You can apply these operations in any order. It is allowed to apply the operations multiple times to the same substring.

What is the minimum number of coins you need to spend to get a string consisting only of ones?

Input
The first line of input contains integers $$$n$$$, $$$x$$$ and $$$y$$$ ($$$1 \leq n \leq 300\,000, 0 \leq x, y \leq 10^9$$$) — length of the string, cost of the first operation (substring reverse) and cost of the second operation (inverting all elements of substring).

The second line contains the string $$$a$$$ of length $$$n$$$, consisting of zeros and ones.

Output
Print a single integer — the minimum total cost of operations you need to spend to get a string consisting only of ones. Print $$$0$$$, if you do not need to perform any operations.

Examples

input
5 1 10 01000
output
11

input
5 10 1 01000
output
2

input
7 2 3 1111111
output
0

Note
In the first sample, at first you need to reverse substring $$$[1 \dots 2]$$$, and then you need to invert substring $$$[2 \dots 5]$$$.

Then the string was changed as follows:

«01000» $$$\to$$$ «10000» $$$\to$$$ «11111».

The total cost of operations is $$$1 + 10 = 11$$$.

In the second sample, at first you need to invert substring $$$[1 \dots 1]$$$, and then you need to invert substring $$$[3 \dots 5]$$$.

Then the string was changed as follows:

«01000» $$$\to$$$ «11000» $$$\to$$$ «11111».

The overall cost is $$$1 + 1 = 2$$$.

In the third example, string already consists only of ones, so the answer is $$$0$$$.

D. Roman Digits

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

Let's introduce a number system which is based on a roman digits. There are digits I, V, X, L which correspond to the numbers $$$1$$$, $$$5$$$, $$$10$$$ and $$$50$$$ respectively. The use of other roman digits is not allowed.

Numbers in this system are written as a sequence of one or more digits. We define the value of the sequence simply as the sum of digits in it.

For example, the number XXXV evaluates to $$$35$$$ and the number IXI — to $$$12$$$.

Pay attention to the difference to the traditional roman system — in our system any sequence of digits is valid, moreover the order of digits doesn't matter, for example IX means $$$11$$$, not $$$9$$$.

One can notice that this system is ambiguous, and some numbers can be written in many different ways. Your goal is to determine how many distinct integers can be represented by **exactly** $$$n$$$ roman digits I, V, X, L.

Input
The only line of the input file contains a single integer $$$n$$$ ($$$1 \le n \le 10^9$$$) — the number of roman digits to use.

Output
Output a single integer — the number of distinct integers which can be represented using $$$n$$$ roman digits `exactly`.

Examples

input
1
output
4

input
2
output
10

input
10
output
244

Note
In the first sample there are exactly $$$4$$$ integers which can be represented — `I`, `V`, `X` and `L`.

In the second sample it is possible to represent integers $$$2$$$ (`II`), $$$6$$$ (`VI`), $$$10$$$ (`VV`), $$$11$$$ (`XI`), $$$15$$$ (`XV`), $$$20$$$ (`XX`), $$$51$$$ (`IL`), $$$55$$$ (`VL`), $$$60$$$ (`XL`) and $$$100$$$ (`LL`).

E. Sky Full of Stars

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

On one of the planets of Solar system, in Atmosphere University, many students are fans of bingo game.

It is well known that one month on this planet consists of $$$n^2$$$ days, so calendars, represented as square matrix $$$n$$$ by $$$n$$$ are extremely popular.

Weather conditions are even more unusual. Due to the unique composition of the atmosphere, when interacting with sunlight, every day sky takes one of three colors: blue, green or red.

To play the bingo, you need to observe the sky for one month — after each day, its cell is painted with the color of the sky in that day, that is, blue, green or red.

At the end of the month, students examine the calendar. If at least one row or column contains only cells of one color, that month is called lucky.

Let's call two colorings of calendar different, if at least one cell has different colors in them. It is easy to see that there are $$$3^{n \cdot n}$$$ different colorings. How much of them are lucky? Since this number can be quite large, print it modulo $$$998244353$$$.

Input
The first and only line of input contains a single integer $$$n$$$ ($$$1 \le n \le 1000\,000$$$) — the number of rows and columns in the calendar.

Output
Print one number — number of lucky colorings of the calendar modulo $$$998244353$$$

Examples

input
1
output
3

input
2
output
63

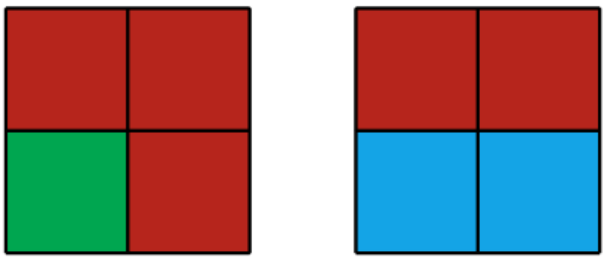
input
3
output

```
9933
```

Note

In the first sample any coloring is lucky, since the only column contains cells of only one color.

In the second sample, there are a lot of lucky colorings, in particular, the following colorings are lucky:



While these colorings are not lucky: