Codeforces Round #525 (Div. 2)

A. Ehab and another construction problem

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

Given an integer x, find 2 integers a and b such that:

- $1 \leq a, b \leq x$
- b divides a (a is divisible by b).
- $a \cdot b > x$.
- $\frac{a}{b} < x$.

Input
The only line contains the integer x $(1 \leq x \leq 100)$.

Output
You should output two integers a and b, satisfying the given conditions, separated by a space. If no pair of integers satisfy the conditions above, print "-1" (without quotes).

Examples

input
10
output
6 3

input
1
output
-1

B. Ehab and subtraction

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

You're given an array a. You should repeat the following operation k times: find the minimum non-zero element in the array, print it, and then subtract it from all the non-zero elements of the array. If all the elements are 0s, just print 0.

Input
The first line contains integers n and k $(1 \leq n, k \leq 10^5)$, the length of the array and the number of operations you should perform.

The second line contains n space-separated integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 10^9)$, the elements of the array.

Output
Print the minimum non-zero element before each operation in a new line.

Examples

input
3 5 1 2 3
output
1 1 1 0 0

input
4 2

10 3 5 3

output
3 2

Note
In the first sample:

In the first step: the array is $[1, 2, 3]$, so the minimum non-zero element is 1.

In the second step: the array is $[0, 1, 2]$, so the minimum non-zero element is 1.

In the third step: the array is $[0, 0, 1]$, so the minimum non-zero element is 1.

In the fourth and fifth step: the array is $[0, 0, 0]$, so we printed 0.

In the second sample:

In the first step: the array is $[10, 3, 5, 3]$, so the minimum non-zero element is 3.

In the second step: the array is $[7, 0, 2, 0]$, so the minimum non-zero element is 2.

C. Ehab and a 2-operation task

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

You're given an array a of length n. You can perform the following operations on it:

- choose an index i $(1 \leq i \leq n)$, an integer x $(0 \leq x \leq 10^6)$, and replace a_j with $a_j + x$ for all $(1 \leq j \leq i)$, which means add x to all the elements in the prefix ending at i.
- choose an index i $(1 \leq i \leq n)$, an integer x $(1 \leq x \leq 10^6)$, and replace a_j with $a_j \% x$ for all $(1 \leq j \leq i)$, which means replace every element in the prefix ending at i with the remainder after dividing it by x.

Can you make the array **strictly increasing** in no more than $n + 1$ operations?

Input
The first line contains an integer n $(1 \leq n \leq 2000)$, the number of elements in the array a.

The second line contains n space-separated integers a_1, a_2, ..., a_n $(0 \leq a_i \leq 10^5)$, the elements of the array a.

Output
On the first line, print the number of operations you wish to perform. On the next lines, you should print the operations.

To print an adding operation, use the format "$1\ i\ x$"; to print a modding operation, use the format "$2\ i\ x$". If i or x don't satisfy the limitations above, or you use more than $n + 1$ operations, you'll get *wrong answer* verdict.

Examples

input
3 1 2 3

output
0

input
3 7 6 3

output
2 1 1 1 2 2 4

Note
In the first sample, the array is already increasing so we don't need any operations.

In the second sample:

In the first step: the array becomes $[8, 6, 3]$.

In the second step: the array becomes $[0, 2, 3]$.

D. Ehab and another another xor problem

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

This is an interactive problem!

Ehab plays a game with Laggy. Ehab has 2 hidden integers (a, b). Laggy can ask a pair of integers (c, d) and Ehab will reply with:

- 1 if $a \oplus c > b \oplus d$.
- 0 if $a \oplus c = b \oplus d$.
- -1 if $a \oplus c < b \oplus d$.

Operation $a \oplus b$ is the bitwise-xor operation of two numbers a and b.

Laggy should guess (a, b) with **at most 62 questions**. You'll play this game. You're Laggy and the interactor is Ehab.

It's guaranteed that $0 \leq a, b < 2^{30}$.

Input
See the interaction section.

Output
To print the answer, print "! a b" (without quotes). **Don't forget to flush the output after printing the answer**.

Interaction
To ask a question, print "? c d" (without quotes). Both c and d must be non-negative integers less than 2^{30}. **Don't forget to flush the output after printing any question**.

After each question, you should read the answer as mentioned in the legend. If the interactor replies with -2, that means you asked more than 62 queries and your program should terminate.

To flush the output, you can use:-

- fflush(stdout) in C++.
- System.out.flush() in Java.
- stdout.flush() in Python.
- flush(output) in Pascal.
- See the documentation for other languages.

Hacking:

To hack someone, print the 2 space-separated integers a and b $(0 \leq a, b < 2^{30})$.

Example

input
1 -1 0
output
? 2 1 ? 1 2 ? 2 0 ! 3 1

Note
In the sample:

The hidden numbers are $a = 3$ and $b = 1$.

In the first query: $3 \oplus 2 = 1$ and $1 \oplus 1 = 0$, so the answer is 1.

In the second query: $3 \oplus 1 = 2$ and $1 \oplus 2 = 3$, so the answer is -1.

In the third query: $3 \oplus 2 = 1$ and $1 \oplus 0 = 1$, so the answer is 0.

Then, we printed the answer.

E. Ehab and a component choosing problem

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

You're given a tree consisting of n nodes. Every node u has a weight a_u. You want to choose an integer k $(1 \leq k \leq n)$ and then choose k connected components of nodes that don't overlap (i.e every node is in at most 1 component). Let the set of nodes you

chose be s. You want to maximize:

$$\frac{\sum_{u \in s} a_u}{k}$$

In other words, you want to maximize the sum of weights of nodes in s divided by the number of connected components you chose. Also, if there are several solutions, you want to **maximize** k.

Note that adjacent nodes **can** belong to different components. Refer to the third sample.

Input
The first line contains the integer n $(1 \le n \le 3 \cdot 10^5)$, the number of nodes in the tree.

The second line contains n space-separated integers a_1, a_2, ..., a_n $(|a_i| \le 10^9)$, the weights of the nodes.

The next $n - 1$ lines, each contains 2 space-separated integers u and v $(1 \le u, v \le n)$ which means there's an edge between u and v.

Output
Print the answer as a **non-reduced** fraction represented by 2 space-separated integers. The fraction itself should be maximized and if there are several possible ways, you should maximize the denominator. See the samples for a better understanding.

Examples

input
3 1 2 3 1 2 1 3
output
6 1

input
1 -2
output
-2 1

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

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

Note
A connected component is a set of nodes such that for any node in the set, you can reach all other nodes in the set passing only nodes in the set.

In the first sample, it's optimal to choose the whole tree.

In the second sample, you only have one choice (to choose node 1) because you can't choose 0 components.

In the third sample, notice that we could've, for example, chosen only node 1, or node 1 and node 3, or even the whole tree, and the fraction would've had the same value (-1), but we want to maximize k.

In the fourth sample, we'll just choose nodes 1 and 3.

F. Ehab and a weird weight formula

time limit per test: 2.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You're given a tree consisting of n nodes. Every node u has a weight a_u. It is guaranteed that there is only one node with minimum weight in the tree. For every node u (except for the node with the minimum weight), it must have a neighbor v such that $a_v < a_u$. You should construct a tree to minimize the weight w calculated as follows:

- For every node u, $deg_u \cdot a_u$ is added to w (deg_u is the number of edges containing node u).
- For every edge $\{u, v\}$, $\lceil log_2(dist(u, v)) \rceil \cdot min(a_u, a_v)$ is added to w, where $dist(u, v)$ is the number of edges in the path from u to v in the given tree.

Input

The first line contains the integer n $(2 \le n \le 5 \cdot 10^5)$, the number of nodes in the tree.

The second line contains n space-separated integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le 10^9)$, the weights of the nodes.

The next $n - 1$ lines, each contains 2 space-separated integers u and v $(1 \le u, v \le n)$ which means there's an edge between u and v.

Output

Output one integer, the minimum possible value for w.

Examples

input
3 1 2 3 1 2 1 3
output
7

input
5 4 5 3 7 8 1 2 1 3 3 4 4 5
output
40

Note

In the first sample, the tree itself minimizes the value of w.

In the second sample, the optimal tree is: