GPC-UPC Official First day contest (Mathforces)

A. Lesha and array splitting

2 seconds, 256 megabytes

One spring day on his way to university Lesha found an array A. Lesha likes to split arrays into several parts. This time Lesha decided to split the array A into several, possibly one, new arrays so that the sum of elements in each of the new arrays is not zero. One more condition is that if we place the new arrays one after another they will form the old array A.

Lesha is tired now so he asked you to split the array. Help Lesha!

Input

The first line contains single integer n ($1 \le n \le 100$) — the number of elements in the array A.

The next line contains n integers $a_1, a_2, ..., a_n$ ($-10^3 \le a_i \le 10^3$) — the elements of the array A.

Output

If it is not possible to split the array A and satisfy all the constraints, print single line containing "NO" (without quotes).

Otherwise in the first line print "YES" (without quotes). In the next line print single integer k — the number of new arrays. In each of the next k lines print two integers l_i and r_i which denote the subarray $A[l_i ... r_i]$ of the initial array A being the i-th new array. Integers l_i, r_i should satisfy the following conditions:

- $l_1 = 1$
- $r_k = n$
- $r_i + 1 = l_{i+1}$ for each $1 \le i < k$.

If there are multiple answers, print any of them.

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

input
8 9 -12 3 4 -4 -10 7 3
output
YES 2 1 2 3 8

input
1 0
output
NO

input
4 1 2 3 -5
output
YES 4 1 1 2 2 3 3 4 4

B. PolandBall and Hypothesis

2 seconds, 256 megabytes

PolandBall is a young, clever Ball. He is interested in prime numbers. He has stated a following hypothesis: "`There exists such a positive integer` n `that for each positive integer` m `number` $n \cdot m + 1$ `is a prime number`".

Unfortunately, PolandBall is not experienced yet and doesn't know that his hypothesis is incorrect. Could you prove it wrong? Write a program that finds a counterexample for any n.

Input

The only number in the input is n ($1 \le n \le 1000$) — number from the PolandBall's hypothesis.

Output

Output such m that $n \cdot m + 1$ is not a prime number. Your answer will be considered correct if you output any suitable m such that $1 \le m \le 10^3$. It is guaranteed the the answer exists.

input
3
output
1

input
4
output
2

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself.

For the first sample testcase, $3 \cdot 1 + 1 = 4$. We can output 1.

In the second sample testcase, $4 \cdot 1 + 1 = 5$. We cannot output 1 because 5 is prime. However, $m = 2$ is okay since $4 \cdot 2 + 1 = 9$, which is not a prime number.

C. Soldier and Number Game

3 seconds, 256 megabytes

Two soldiers are playing a game. At the beginning first of them chooses a positive integer n and gives it to the second soldier. Then the second one tries to make maximum possible number of rounds. Each round consists of choosing a positive integer $x > 1$, such that n is divisible by x and replacing n with n / x. When n becomes equal to 1 and there is no more possible valid moves the game is over and the score of the second soldier is equal to the number of rounds he performed.

To make the game more interesting, first soldier chooses n of form $a! / b!$ for some positive integer a and b ($a \ge b$). Here by $k!$ we denote the *factorial* of k that is defined as a product of all positive integers not large than k.

What is the maximum possible score of the second soldier?

Input

First line of input consists of single integer t ($1 \le t \le 1\,000\,000$) denoting number of games soldiers play.

Then follow t lines, each contains pair of integers a and b ($1 \le b \le a \le 5\,000\,000$) defining the value of n for a game.

Output

For each game output a maximum score that the second soldier can get.

input
2 3 1 6 3
output
2 5

D. Maximum of Maximums of Minimums

1 second, 256 megabytes

You are given an array $a_1, a_2, ..., a_n$ consisting of n integers, and an integer k. You have to split the array into exactly k non-empty subsegments. You'll then compute the minimum integer on each subsegment, and take the maximum integer over the k obtained minimums. What is the maximum possible integer you can get?

Definitions of subsegment and array splitting are given in notes.

Input
The first line contains two integers n and k ($1 \le k \le n \le 10^5$) — the size of the array a and the number of subsegments you have to split the array to.

The second line contains n integers $a_1, a_2, ..., a_n$ ($-10^9 \le a_i \le 10^9$).

Output
Print single integer — the maximum possible integer you can get if you split the array into k non-empty subsegments and take maximum of minimums on the subsegments.

input
5 2 1 2 3 4 5
output
5

input
5 1 -4 -5 -3 -2 -1
output
-5

A subsegment $[l, r]$ ($l \le r$) of array a is the sequence $a_l, a_{l+1}, ..., a_r$.

Splitting of array a of n elements into k subsegments $[l_1, r_1], [l_2, r_2], ..., [l_k, r_k]$ ($l_1 = 1, r_k = n, l_i = r_{i-1} + 1$ for all $i > 1$) is k sequences $(a_{l_1}, ..., a_{r_1}), ..., (a_{l_k}, ..., a_{r_k})$.

In the first example you should split the array into subsegments $[1, 4]$ and $[5, 5]$ that results in sequences $(1, 2, 3, 4)$ and (5). The minimums are $min(1, 2, 3, 4) = 1$ and $min(5) = 5$. The resulting maximum is $max(1, 5) = 5$. It is obvious that you can't reach greater result.

In the second example the only option you have is to split the array into one subsegment $[1, 5]$, that results in one sequence $(-4, -5, -3, -2, -1)$. The only minimum is $min(-4, -5, -3, -2, -1) = -5$. The resulting maximum is -5.

E. List Of Integers

5 seconds, 256 megabytes

Let's denote as $L(x, p)$ an infinite sequence of integers y such that $gcd(p, y) = 1$ and $y > x$ (where gcd is the greatest common divisor of two integer numbers), sorted in ascending order. The elements of $L(x, p)$ are 1-indexed; for example, 9, 13 and 15 are the first, the second and the third elements of $L(7, 22)$, respectively.

You have to process t queries. Each query is denoted by three integers x, p and k, and the answer to this query is k-th element of $L(x, p)$.

Input
The first line contains one integer t ($1 \le t \le 30000$) — the number of queries to process.

Then t lines follow. i-th line contains three integers x, p and k for i-th query ($1 \le x, p, k \le 10^6$).

Output
Print t integers, where i-th integer is the answer to i-th query.

input
3 7 22 1 7 22 2 7 22 3
output
9 13 15

input
5 42 42 42 43 43 43 44 44 44 45 45 45 46 46 46
output
187 87 139 128 141

F. Travelling Salesman and Special Numbers

1 second, 256 megabytes

The Travelling Salesman spends a lot of time travelling so he tends to get bored. To pass time, he likes to perform operations on numbers. One such operation is to take a positive integer x and reduce it to the number of bits set to 1 in the binary representation of x. For example for number 13 it's true that $13_{10} = 1101_2$, so it has 3 bits set and 13 will be reduced to 3 in one operation.

He calls a number *special* if the minimum number of operations to reduce it to 1 is k.

He wants to find out how many special numbers exist which are not greater than n. Please help the Travelling Salesman, as he is about to reach his destination!

Since the answer can be large, output it modulo $10^9 + 7$.

Input
The first line contains integer n ($1 \le n < 2^{1000}$).

The second line contains integer k ($0 \le k \le 1000$).

Note that n is given in its binary representation without any leading zeros.

Output
Output a single integer — the number of special numbers not greater than n, modulo $10^9 + 7$.

input
110 2
output
3

input
111111011 2
output
169

In the first sample, the three special numbers are 3, 5 and 6. They get reduced to 2 in one operation (since there are two set bits in each of 3, 5 and 6) and then to 1 in one more operation (since there is only one set bit in 2).