Problem A. Station Module Hierarchy

Input file:	*standard input*
Output file:	*standard output*
Time limit:	7 seconds
Memory limit:	256 mebibytes

The engineers of an orbital station are designing a hierarchy of modules of size n. The main command module is numbered 1. To keep the structure stable, they decided to calculate the number of *k-module calibration groups* in the hierarchy.

A *station module hierarchy* of size n is defined as a rooted tree with n vertices, rooted at vertex 1, where each vertex has **strictly fewer** direct children than its parent (except the root that doesn't have any parent).

A *k-module calibration group* is a pair (v, S) where:

- $v \in V$ is a vertex of the hierarchy (the *center*),
- $S \subset V \setminus \{v\}$ is a set of vertices of size $k-1$,
- there exists an integer d such that for every $u \in S$, $\text{dist}(v, u) = d$.

The distance $\text{dist}(u, v)$ is defined as the number of edges on the simple path between vertices u and v.

In other words, a k-module calibration group consists of one vertex v (the center) and $k-1$ other vertices that are all at the same distance from v.

Your task is to compute the number of k-module calibration groups in the given hierarchy.

Since the answer may be large, output it modulo 998 244 353.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

Each test case begins with a line containing two integers n and k ($2 \le n \le 5 \cdot 10^5$, $1 \le k \le n$): the number of vertices in the station module hierarchy and the size of the calibration group.

The next line contains $n-1$ integers $p_2, p_3, \ldots, p_n$ ($1 \le p_i < i$), where p_i denotes the parent of vertex i in the station module hierarchy.

The sum of n over all test cases does not exceed $5 \cdot 10^5$.

Output

For each test case, output a single integer: the number of k-module calibration groups modulo 998 244 353.

Example

standard input	*standard output*
3 3 3 1 1 5 2 1 1 2 3 6 3 1 1 1 2 2	1 20 16

Note

In the first test case, the only existing calibration group is centered at vertex 1, and the corresponding set is $S = \{2, 3\}$.

Problem B. Energy Cell Selection

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	256 mebibytes

The orbital station is preparing a new power module for calibration. The engineers have n energy cells in storage. Each cell has two associated values: a_i and b_i. You can select any subset S of exactly k cells and install them together to obtain a calibration score calculated as:

$$\left(\sum_{i \in S} a_i\right) \cdot \left(\sum_{i \in S} b_i\right)$$

Your task is to determine the maximum possible calibration score that can be achieved by installing exactly k energy cells.

It is additionally known that the cell storage is *safety-certified*, meaning no energy cell has both a_i and b_i simultaneously too high. Specifically, for every cell, $\min(a_i, b_i) \leq 100$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). The description of the test cases follows.

Each test case begins with a line containing two integers n and k ($1 \leq n \leq 10^5$, $1 \leq k \leq \min(n, 5)$): the total number of energy cells and the number of cells to install for a single calibration.

Each of the next n lines contains two integers a_i and b_i ($1 \leq a_i, b_i \leq 10^9$), representing the values of an energy cell. For every cell, $\min(a_i, b_i) \leq 100$.

The total number of energy cells across all test cases does not exceed 10^5.

Output

For each test case, print a single integer: the maximum possible calibration score that can be obtained by selecting exactly k energy cells.

Examples

standard input	*standard output*
1 5 5 1 1 2 2 3 3 4 4 5 5	225
1 6 5 1 1 2 6 3 5 4 4 5 3 6 2	400

Note

In the first test case, we use all the energy cells. The score is $(1+2+3+4+5)\cdot(1+2+3+4+5) = 15^2 = 225$.

In the second test case, we have to leave out one energy cell. The set with cells $[2, 3, 4, 5, 6]$ has the value 400.

Station note: the two values of a cell may represent any pair of independent engineering measurements. Only the formula above and the safety-certification condition matter for choosing the cells.

Problem C. Championship And Buses

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

Kostya is going to the French Orienteering Championship. He needs to travel to the championship by train. There are buses of various routes running from Kostya's home to the railway station.

Kostya wants to determine the latest moment when he can board a bus that arrives at the station before boarding for the train ends. Kostya knows, for each bus, in how many minutes from the current moment it departs from his home, and how many minutes the trip on that bus to the station takes.

Determine the maximum number of minutes from the current moment after which Kostya can board a bus and still arrive before boarding for the train ends, or determine that he is guaranteed to be late regardless of which bus he takes.

Input

The first line of the input contains two integers n and x — the number of bus routes and the time until boarding for the train ends, respectively ($1 \le n \le 100$, $2 \le x \le 200$). Each of the next n lines contains two integers s and t — the time until the bus departs, s, and the travel time to the station, t ($1 \le s, t \le 100$).

Output

If it is impossible to arrive before boarding for the train ends, output -1. Otherwise, output the time until departure of the latest suitable bus.

Example

standard input	*standard output*
3 25 18 2 9 9 24 1	24

Problem D. Gateway Code

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

The orbital station has a large system of service gateways, docking corridors, and maintenance airlocks. To make the access system easy to audit, every gateway is assigned a numeric code.

A new gateway is being connected to the station network, but all simple codes have already been used. The chief engineer decides to generate its code from today's access key k and wants to choose some number n as the new gateway code.

The gateway controller will accept a code n if it has the following properties:

- the number n is divisible by k,
- the sum of all digits in n is k.

You already found valid gateway codes for small values of k, so now you need a program to work with larger keys.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 1000$). The description of the test cases follows.

Each test case is given on a single line containing one integer k ($1 \le k \le 10^5$).

The sum of k over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output n ($1 \le n \le 10^{20k}$). In case there are multiple valid gateway codes, output any one of them.

Example

standard input	*standard output*
4 2 8 18 84	2 8 198 84848484848484

Problem E. Euclid Wants to Eat

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 seconds
Memory limit:	1024 mebibytes

Euclid is testing an algorithm he has just invented.

He wrote down q numbers. For each such number n, he wants to find two numbers a, b such that $a+b=n$, $\gcd(a,b)=1$, and $max(a,b)$ is as small as possible. Since there are too many numbers, Euclid wants to finish the work as quickly as possible and start his lunch. Help him and write a program that, for each n, outputs the minimum possible value of the maximum of the numbers a, b satisfying the conditions above.

Input

The first line contains the number of test cases $1 \le q \le 100$. Each of the next q lines contains one test case: a single integer $2 \le n \le 10^9$.

Output

For each test case, output the answer.

Example

standard input	*standard output*
1 3	2

Problem F. Framboise Game

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 seconds
Memory limit:	1024 mebibytes

Alice and Bob are playing the following game. There are n non-empty piles of berries in front of them. They take turns, with Alice moving first. In one move, a player must take some odd number of berries from each non-empty pile and discard them; discarded berries are no longer considered. For example, if $a = [3, 5, 2, 1, 4, 6]$, then in one move a player may take $[3, 1, 1, 1, 3, 5]$ berries from the piles, after which the array a becomes $a = [0, 4, 1, 0, 1, 1]$. A player loses when they cannot make a move, that is, when all piles are empty. Given the array a, determine who wins if both players play optimally.

Input

The first line contains an integer $1 \le n \le 10^5$ — the number of piles. The second line contains n positive integers — the pile sizes, $1 \le a_i \le 10^9$.

Output

Print 1 if Alice wins, and 0 if Bob wins.

Examples

standard input	*standard output*
3 1 2 3	1
2 2 2	0

Problem G. Signal Messages

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	256 mebibytes

The orbital station stores old communication logs from its service modules. Most messages are ordinary maintenance notes, but some of them are generated by an ancient beacon protocol used for repeated calibration signals.

The protocol chooses a non-empty lowercase word $\mathbb{M}$ as a signal key. One valid signal block consists of two consecutive lines. The first line repeats the key in the old compact form, and the second line confirms the same key by adding the fixed prefix `i said`. Formally, given a non-empty word $\mathbb{M}$ consisting of lowercase English letters, the $\mathbb{M}$-signal message is composed of a **block of two lines** that can be repeated multiple times using the same word $\mathbb{M}$. Each block looks like:

$\mathbb{M}$`, `$\mathbb{M}$` `$\mathbb{M}$`ity `$\mathbb{M}$
`i said `$\mathbb{M}$`, `$\mathbb{M}$` `$\mathbb{M}$`ity `$\mathbb{M}$

Given a large multiline station log, determine the maximum possible amount of characters in some sequence of consecutive blocks that forms a valid signal message.

Input

The input consists of multiple lines, consisting of lowercase English letters, commas, and spaces. Each line ends with a newline character. The total number of characters including endlines does not exceed 10^5. There are no empty lines, and lines do not start or end with a space.

Output

Print a single integer: the maximum possible amount of characters in some consecutive sequence of blocks that matches the signal-message pattern. This includes newline characters at the end of each line of the sequence, including the last line of the signal message. Each newline is considered to be 1 character: pay attention to this if you have a Windows setup.

If there are no blocks matching the signal-message pattern, output -1.

Examples

standard input	*standard output*
pulse, pulse pulseity pulse i said pulse, pulse pulseity pulse	63
orbit, orbit orbitity orbit i said orbit, orbit orbitity orbit module, module moduleity module i said module, module moduleity module module, module moduleity module i said module, module moduleity module	142
control room is quiet, signal deck is blue, checksum reports are calm, but no beacon is true	-1

Note

In the first example, the whole log is one valid signal block with key `pulse`, so the answer is 63.

In the second example, the last four lines form two consecutive signal blocks with the same key `module`. This sequence is longer than the single `orbit` block, so the answer is 142.

The third example contains no valid signal blocks.

Problem H. Capsule Passenger Distribution

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

You are managing a row of n transport capsules on an orbital station. There are $n \cdot k$ passengers unevenly distributed over the capsules; more precisely, there are h_i passengers in capsule i. You want to balance the load and have exactly k passengers in every capsule.

At every transfer cycle, some passengers (probably none) can leave their capsule and move to another one, but they cannot move more than d capsules in one transfer.

It's up to you to decide who moves where and when. How many transfer cycles do you need to achieve your goal?

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains two integers n and d ($1 \le d \le n \le 10^6$): the number of capsules and the distance one passenger can move at once.

The second line of each test case contains n integers $h_1, h_2, \ldots, h_n$ ($0 \le h_i \le 10^9$): the distribution of passengers over the capsules. The sum of h_i is divisible by n.

The sum of n over all test cases does not exceed 10^6.

Output

For each test case, output a line with a single integer: how many transfer cycles you need.

Example

standard input	*standard output*
3 1 1 5 3 2 3 0 0 10 1 0 0 10 0 0 0 0 10 0 0	0 1 2

Note

In the first test case, there is a single capsule, so passengers are evenly distributed from the start.

In the second test case, before the first transfer cycle, there are 3 passengers in the first capsule: Aleksei, Dima, and Kostya. During the first cycle, Dima moves to the second capsule, and Kostya moves to the third one, so one cycle is enough to make every capsule have one passenger.

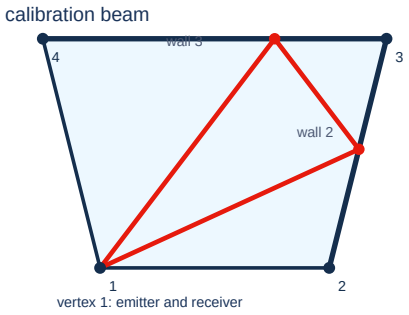
Problem I. Optical Calibrator

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

The orbital station's engineers are installing a new optical calibrator in one of its shielded service modules. The module's cross-section is shaped like a convex polygon with n vertices, and thus has n reflective walls. There is a service hatch located at vertex 1. This hatch already contains a fixed calibration unit that protects walls 1 (between vertices 1 and 2) and n (between vertices n and 1).

However, the remaining reflective walls 2 through $n-1$ still need to be calibrated. Fortunately, the hatch also has a steerable optical emitter and receiver installed at vertex 1. You want to aim the calibration beam in such a way that:

- it starts at vertex 1,
- reflects in this exact order off wall 2, then wall 3, and so on until it reflects off wall $n-1$,
- and finally returns precisely to vertex 1.



All reflections follow the laws of ideal optics (angle of incidence equals angle of reflection), and the polygon is convex. You need to determine if it is possible to align the calibrator in the described way. If so, compute the required firing angle.

The base direction (angle 0°) is the positive x-axis, and angles are measured in degrees counter-clockwise.

Important: To keep the calibration stable, you decided to have all reflection points strictly inside their respective walls: more precisely, to be at least 10^{-5} **relative** units away from both endpoints. Formally, the valid points of reflection p for the wall $[a, b]$ are of the form $p = \alpha a + (1-\alpha)b$, where $\min(\alpha, 1-\alpha) \geq 10^{-5}$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 1000$). The description of the test cases follows.

For each test case, the first line contains a single integer n ($3 \leq n \leq 20$): the number of vertices of the convex polygon.

Each of the next n lines contains two integers x_i and y_i ($-10^9 \leq x_i, y_i \leq 10^9$): the coordinates of the vertices. The vertices are given in counter-clockwise order and form a strictly convex polygon.

Output

For each test case, output a single line "`NO`" if there is no answer.

Otherwise output two lines. The first line must contain "`YES`". On the second line, print a real number d ($-180 \leq d \leq 180$): the angle (in degrees) relative to the positive x-axis (counter-clockwise), at which the beam should be fired towards wall 2.

The trajectory of the ray should hit all the walls in required order at points inside the walls, and end in the 10^{-7}-neighborhood of vertex 1. The 10^{-7}-neighborhood of vertex 1 is defined as any point (x, y) on the walls 1 or n that follows $(x_1 - x)^2 + (y_1 - y)^2 < (10^{-7} \cdot L)^2$, where L is the total length of polygon sides.

If multiple angles satisfy the condition, output any one suitable angle.

Examples

standard input	*standard output*
2 3 2 3 0 0 6 0 4 0 0 2 0 2 3 0 3	YES -90 NO
3 4 0 0 6 0 8 5 -2 5 5 1 0 7 0 10 4 5 9 -1 5 6 -6 0 -3 -5 3 -5 6 0 3 5 -3 5	YES 10.20397372173168 YES 8.28061631487776 NO

Note

Illustration for the first test case of the second example is provided in the statement.

It is guaranteed that, for each test with answer "YES", there exists a shooting angle that has all its reflections at points $p = \alpha a + (1 - \alpha)b$, where $\min(\alpha, 1 - \alpha) \geq 10^{-5} + 10^{-9}$). It is also guaranteed that, for each test with answer "NO", there are no shooting angles that are *almost* valid: this means that at some point there is either no intersection, or $\min(\alpha, 1 - \alpha) < 10^{-5} - 10^{-9}$.

Problem J. Broken Navigation Console

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

It is not enough to plot the route, you still have to make the console obey it.

Put the cursor at the end of a row on the station console. Press Up and Right again and again. You learned a convenient emergency navigation trick.

The orbital station has a route table on its navigation console. During a calibration drill, several control keys stopped responding. The station technician still needs to move the cursor through the table and bring it to the top command row; fortunately, the Up and Right arrows still work.

The route table consists of n rows numbered from 1 to n, where row i contains s_i possible cursor positions from $(i, 1)$ to (i, s_i), denoting the j-th position on the i-th row as (i, j).

Navigation on a damaged console is a bit tricky. Fortunately, the technician still has Up and Right arrows working:

- Up ($\uparrow$): if the cursor is at position (i, j), then if $i = 1$, the cursor stays in place; otherwise, it moves to position $(i - 1, \min(j, s_{i-1}))$, where s_{i-1} is the length of row $i - 1$.
- Right ($\rightarrow$): if the cursor is at position (i, j), then if $j < s_i$, the cursor moves to $(i, j + 1)$; otherwise, it moves to $(i + 1, 1)$; except when at (n, s_n), then it stays in place.

The technician has a fallback routine. They pick an initial cursor position and start to perform an infinite sequence of key presses: u times Up, followed by r times Right, then again u times Up, followed by r times Right, and so on.

For example, let $n = 5$, $u = 2$, $r = 5$, and $s = \{5, 2, 4, 3, 1\}$. Starting from position $(4, 3)$, the first few key presses will move the cursor as follows:

$$(4, 3) \uparrow (3, 3) \uparrow (2, 2) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4) \rightarrow (4, 1) \ldots$$

A position is called *successful* if, starting from this position and performing the infinite routine, the technician will sooner or later have the cursor on the first row.

Your task is to help the station crew count how many successful starting positions exist in the route table.

The table is still being recalibrated, so you also need to handle q updates. In each update, you get two values, *pos* and x, and you have to assign s_{pos} a new value x. You need to give the answer to the problem before any updates and after processing each update.

Input

The first line contains three integers: n, u, and r ($1 \le n \le 10^5$, $1 \le u, r \le 10^{18}$).

The second line contains n integers $s_1, s_2, \ldots, s_n$ ($1 \le s_i \le 10^{12}$): the number of positions in each row of the route table.

The third line contains a single integer q ($1 \le q \le 10^5$), the number of updates.

Each of the next q lines contains two integers, *pos* and x ($1 \le pos \le n$, $1 \le x \le 10^{12}$): the updates.

The updates are consecutive, so each update is applied to the current state of the route table, not the initial one.

Output

Output $q + 1$ integers, each on a separate line: the number of successful starting positions before all the updates and after each one of them.

Examples

standard input	*standard output*
5 2 5 5 2 4 3 1 1 1 1	15 11
5 2 10 5 2 4 3 1 1 2 3	11 12
10 1 42 169 42 42 42 42 42 42 42 42 42 1 2 43	211 254

Note

In the first example, before any updates, starting from position $(5, 1)$ and pressing "2 times Up followed by 5 times Right" in a loop:

Round 1: $(5, 1) \uparrow (4, 1) \uparrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4) \rightarrow (4, 1) \rightarrow (4, 2)$

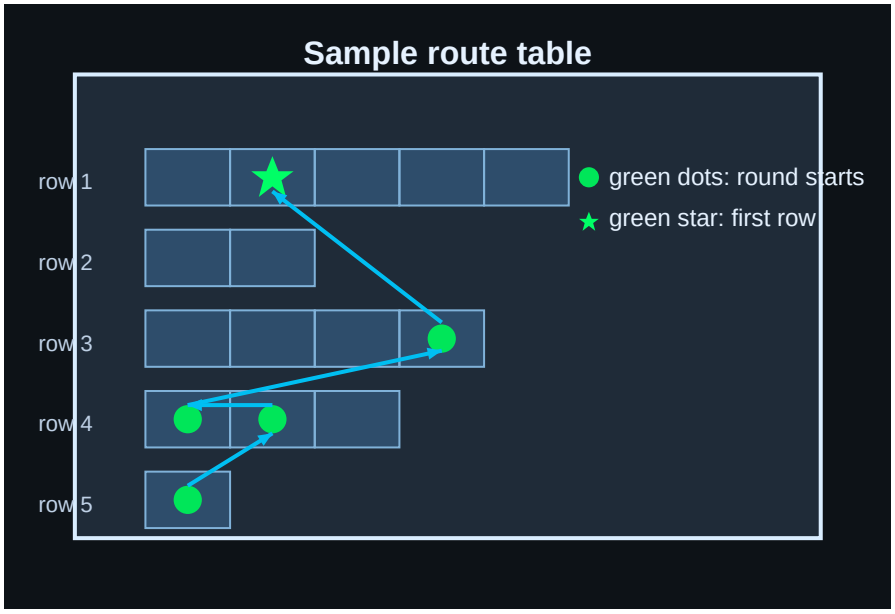
Round 2: $(4, 2) \uparrow (3, 2) \uparrow (2, 2) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4) \rightarrow (4, 1)$

Round 3: $(4, 1) \uparrow (3, 1) \uparrow (2, 1) \rightarrow (2, 2) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4)$

Round 4: $(3, 4) \uparrow (2, 2) \uparrow (\mathbf{1}, \mathbf{2}) \rightarrow \ldots$

Thus, $(5, 1)$ is a successful position.

On the image below you can see the starting positions of each round marked with the green dots and the final position marked with the green star.



Problem K. Power Consumption Optimization

Input file:	*standard input*
Output file:	*standard output*
Time limit:	6 seconds
Memory limit:	1024 mebibytes

The orbital station has a row of n power modules. Module i currently consumes a_i units of power. During a power-saving calibration, the station controller may throttle selected modules one unit at a time.

You are given an array $a = [a_1, a_2, \ldots, a_n]$ of n positive integers.

We define $f_k(b_1, b_2, \ldots, b_m)$ to be the minimum possible product of the array $[b_1, \ldots, b_m]$ that can be obtained by applying the following operation at most k times:

- Select any index i such that $b_i > 1$ and assign $b_i := b_i - 1$.

Your task is to process q maintenance queries. Each query is given as three integers ℓ, r, and k, and asks you to compute the value of $f_k(a_\ell, a_{\ell+1}, \ldots, a_r)$: the minimum possible product of the consumption levels of modules ℓ through r after applying at most k throttling operations, as described above.

Since the answer may be large, output it modulo 998 244 353.

Input

The first line contains two integers n and q ($1 \le n \le 2 \cdot 10^5$, $1 \le q \le 5 \cdot 10^5$): the number of power modules and the number of queries.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 998\,244\,352$): the current power consumption levels of the modules.

Each of the next q lines contains three integers ℓ, r, and k ($1 \le \ell \le r \le n$, $0 \le k \le 10^{18}$) describing a query on the station sector from module ℓ to module r, with at most k allowed throttling operations.

Output

For each query, print one integer: the value of $f_k(a_\ell, \ldots, a_r)$ modulo 998 244 353.

Example

standard input	*standard output*
5 2 1 2 3 4 5 1 2 3 4 5 1	1 15

Note

The product is computed after all throttling operations are applied, and only the final result should be taken modulo 998 244 353.

Problem L. Letter Game and Wise Flea

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

...After the Bytelandian king Baitazar the Cunning once again lost to queen Baitika the Wise in the game of «cities», being unable to name a city starting with the letter «A», he asked for help from a wise flea, who had gathered many clever thoughts while traveling through people's heads and served as his advisor. The flea said that the problem should be solved constructively.

In the Bytelandian language there are v vowels and c consonants. One of the vowels is the letter «A». A word is considered euphonious if no consonant is adjacent to another consonant and no vowel is adjacent to another vowel. Following the flea's advice, the king ordered cities to be founded with all possible euphonious names of lengths from l to r, starting and ending with the letter «A», hoping that this would help him win.

The game of cities is played by the standard rules: on each turn, a player must name a city whose name starts with the letter with which the city named by the other player ends; if a player cannot make a move, they lose.

Assume that all previously known cities have been exhausted (that is, all unnamed cities are exactly those whose names satisfy the property described above, and only those), and the king has to name the next city starting with the letter «A». Given v, c, l, and r, determine who wins with optimal play.

Input

The first line of the input contains four integers c, v, l and r ($1 \le c \le 10^9$, $1 \le v \le 10^9$, $1 \le l \le r \le 10^9$) — the number of consonants and vowels in the Bytelandian language, and the range of name lengths, respectively.

Output

Print `King` if the king wins, and `Queen` if the queen wins.

Examples

standard input	*standard output*
2 3 5 8	Queen
3 5 8 13	King

Problem M. Magic of Art

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 seconds
Memory limit:	1024 mebibytes

Alice enjoys drawing city landscapes. The top points of the buildings on the horizon can be represented as an array a of n positive integers. Alice believes that the *magicness* of a landscape is defined as

$$(a_2 - a_1) \cdot 1 + (a_3 - a_2) \cdot 2 + (a_4 - a_3) \cdot 3 + \ldots + (a_{i+1} - a_i) \cdot i + \ldots + (a_n - a_{n-1}) \cdot (n-1)$$

Alice wants to draw a landscape "inspired by" the city where she is now (with a given array a), but she wants to rearrange the buildings in the picture (and thus permute the elements of array a) so that the magicness of the landscape is equal to a given number x.

Determine whether this is possible, and if it is, find at least one such permuted array.

Input

The first line of the input contains two integers $1 \le n \le 10^5$ and $-10^{15} \le x \le 10^{15}$. The next line contains the elements of the array, $1 \le a_i \le 10^9$, listed in increasing order of indices.

Output

Print -1 if there is no permutation of the given array with magicness equal to x. Otherwise, print n numbers — the permuted array with magicness equal to x. Print the values themselves, not the indices.

Example

standard input	*standard output*
8 16 4 1 8 9 3 7 6 2	4 1 8 9 3 6 2 7