Problem A. Adjusted Average

Source file name:	Adjusted.c, Adjusted.cpp, Adjusted.java, Adjusted.py
Input:	`Standard`
Output:	`Standard`

As a student of the Biology And Probability Course at your university, you have just performed an experiment as part of the practical assignments. However, your results do not look very nice: you had hoped that the average of your samples would be different from what it is now.

To improve your results, you decide to let some of your samples "magically disappear" (i.e., dump them in the waste bin). In order to not raise suspicion with your teacher, you can remove only a few of your samples. How close can you possibly get to your desired average?

"If you don't reveal some insights soon, I'm going to be forced to slice, dice, and drill!"

Torturing data. CC BY
by Timo Elliott on timoelliott.com

Input

The input consists of:

- One line with three integers n, k, and $\overline{x}$ ($2 \leq n \leq 1500$, $1 \leq k \leq 4$, $k < n$, $|\overline{x}| \leq 10^9$), the number of samples, the number of samples that may be removed, and the average you think looks the nicest.
- One line with n integers x ($|x| \leq 10^9$), representing the samples.

Output

Output the minimal absolute difference between $\overline{x}$ and the average you can obtain by removing at most k samples from the dataset.

Your answer should have an *absolute* error of at most 10^{-4}.

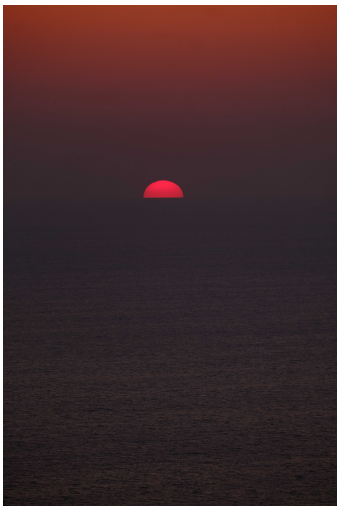
Example

Input	Output
5 2 2 1 2 3 100 200	0
5 4 -5 -6 -3 0 6 3	0.5
4 1 4 1 3 3 7	0.333333333333333

Problem B. Bellevue

Source file name:	Bellevue.c, Bellevue.cpp, Bellevue.java, Bellevue.py
Input:	`Standard`
Output:	`Standard`

As any photographer knows, any good sunset photo has the sun setting over the sea. In fact, the more sea that is visible in the photo, the prettier it is!



Sunset over the Mediterranean sea.
CC0 by Ragnar Groot Koerkamp

You are currently visiting the island Bellevue, and you would like to take a photo of either the sunrise to the east or the sunset to the west to submit it to Bellevue's Astonishing Photography Competition. By carefully studying the topographic maps, you managed to find the east-west profile of the island. Now you would like to know the maximal amount of sea that you could capture in a photo, measured as the viewing angle covered by water.

The profile of the island is given as a piecewise linear function consisting of $n-1$ segments between n points. The island starts and ends at sea level. As an example, Figure 1 shows the profile of the first sample case.

Note that the viewing angle of your lens is not large enough to capture the ocean to the east and west of the island in one shot. Also, the viewing angle of sea at sea level is 0 degrees.

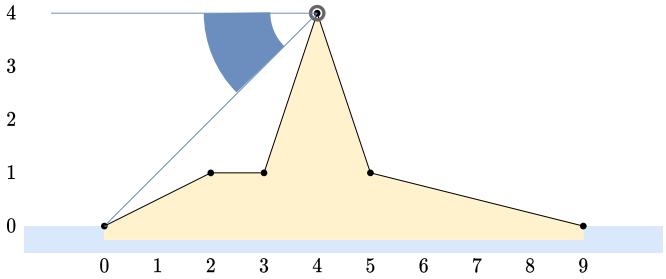


Figure 1: The east-west profile of the island in the first sample case.

Input

The input consists of:

- One line with an integer n ($3 \leq n \leq 50\,000$), the number of points.
- n lines with two integers x_i and y_i ($0 \leq x_i, y_i \leq 50\,000$), a point in the east-west profile of the island.

It is guaranteed that the points are given from left to right ($x_1 < x_2 < \cdots < x_n$) and that the island starts and ends at sea level ($y_1 = y_n = 0$). The interior of the island is all above sea level ($y_i > 0$ for $1 < i < n$).

Output

Output the maximal viewing angle of sea you can see, in degrees.

Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
6 0 0 2 1 3 1 4 4 5 1 9 0	45
5 1 0 5 4 6 1 8 2 9 0	63.4349488

Problem C. Cleaning Robot

Source file name:	Cleaning.c, Cleaning.cpp, Cleaning.java, Cleaning.py
Input:	`Standard`
Output:	`Standard`

A company wishes to purchase a square-shaped cleaning robot to clean a rectangularly shaped room. Some parts of the room are obstructed.

There are different robots of different sizes. Each robot can move horizontally and vertically in the room if no part of the robot intersects an obstruction. They are incapable of changing orientation, so movements are always axis-aligned. Larger robots will get the job done faster, but are more likely to be hindered by obstructions. The robot must always remain fully in the room with no portion extending past the edges of the rectangle.

What is the largest robot the company can buy that will be able to clean all the squares of the room not occupied by obstructions?

Input

The first line of input contains three integers n, m ($3 \leq n, m$ and $n \cdot m \leq 5 \cdot 10^6$) and k ($0 \leq k < n \cdot m$, $k < 10^6$), where n and m are the dimensions of the room in inches, and k is the number of obstructions.

Each of the next k lines contains two integers i and j ($1 \leq i \leq n$, $1 \leq j \leq m$). This specifies that the one-inch square at $(i,\ j)$ is obstructed. All obstructed squares are distinct.

Output

Output a single integer, which is the maximum length of one side of the largest square-shaped robot that could clean the entire room, or -1 if no such robot could clean the entire room.

Example

Input	Output
10 7 1 8 3	2

Problem D. Dimensional Debugging

Source file name:	Debugging.c, Debugging.cpp, Debugging.java, Debugging.py
Input:	`Standard`
Output:	`Standard`

After struggling with this *one* problem for *days*, you have had enough! You are determined to find the bug in your algorithm once and for all! To do so, you will start all over. From scratch. At least you are sure you know the correct answer in the most trivial case: the answer in $(0, 0, \ldots, 0)$ is 0.

You will re-solve the problem, which takes k parameters, using n simpler but slower algorithms. Each algorithm has two bounds for every parameter i (L_i and H_i). An algorithm is only fast enough to run on inputs $(x_1, \ldots, x_k)$ where $x_i \leq H_i$ for all parameters i. You are confident the implementation of an algorithm is correct if you can verify its correctness at least once on an input $(x_1, \ldots, x_k)$ where $x_i \geq L_i$ for all parameters i. To do so, you will need another algorithm that you already proved to be correct and can handle such large inputs, or your knowledge of the answer for $(0, 0, \ldots, 0)$.

Given a list of algorithms and their bounds, find the number of algorithms you are sure are correctly implemented.

As an example, consider the first sample case shown in Figure 2 on the left. The first algorithm (red, bottom left) can be used to verify the correctness of the second (yellow, top left) and third (blue, bottom right) algorithms. No algorithm can be used to verify the correctness of the fourth algorithm (grey, top right).

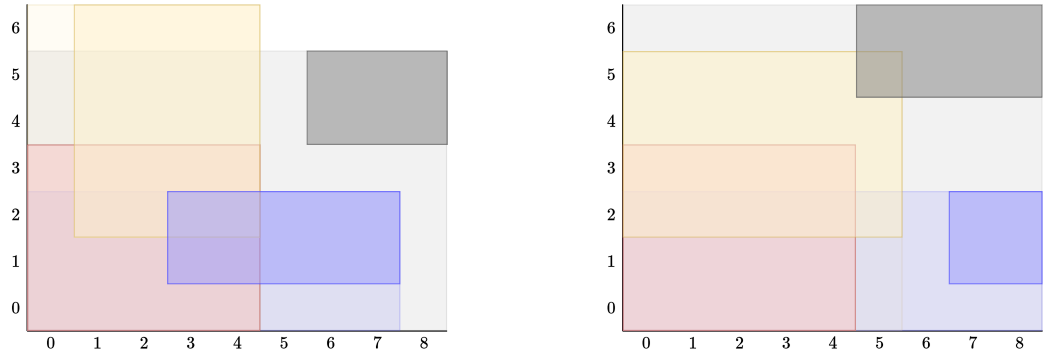


Figure 2: The algorithms to be tested in examples 1 and 2, respectively. The boxes indicate the parameters where an algorithm must be tested, while the lighter background indicates the region where an algorithm can be used to verify other algorithms.

Input

The input consists of:

- One line with two integers n and k ($1 \leq n \leq 1000$, $1 \leq k \leq 10$), the number of algorithms to test and the number of parameters.
- Then follow n pairs of lines:
  - One line with k integers $L_i, \ldots, L_k$ ($0 \leq L_i \leq 10^9$ for all i).
  - One line with k integers $H_1, \ldots, H_k$ ($0 \leq H_i \leq 10^9$ for all i).

  It is guaranteed that $L_i \leq H_i$ for all $1 \leq i \leq k$.

Output

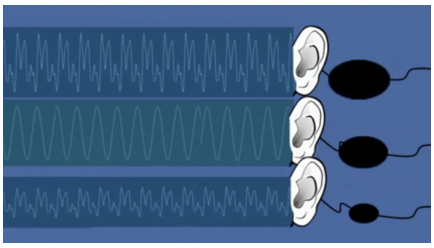
Output the number of algorithms of which you can verify the correctness.

Example

Input	Output
4 2 0 0 4 3 1 2 4 6 3 1 7 2 6 4 8 5	3
4 2 0 0 4 3 0 2 5 5 7 1 8 2 5 5 8 6	4
3 1 1 10 10 100 0 1	3
3 3 0 0 1 2 2 1 1 0 0 2 3 4 0 1 0 3 4 5	0

Problem E. Equalising Audio

Source file name:	Equalising.c, Equalising.cpp, Equalising.java, Equalising.py
Input:	`Standard`
Output:	`Standard`



CC BY-SA 4.0 by Rburtonresearch on Wikipedia

As a radio engineer at the Balanced Audio Podcast © your job is to deliver an equal listening experience at all times. You did a poll among the listeners and they are especially concerned about fluctuations in loudness. To resolve this you bought a transformer to equalise the audio, but alas, its software got corrupted during transport.

Your job is to rewrite the equalising software. As input the transformer gets n amplitudes $a_1, \ldots, a_n$, with an average perceived loudness of $\frac{1}{n}\sum_{i=1}^{n} a_i^2$. The output should contain the same amplitudes, but renormalised by some constant positive factor, such that the average perceived loudness is x. There is one exception: total silence should always be preserved.

Input

The input consists of:

- One line with a two integers n and x ($1 \leq n \leq 10^5$, $0 \leq x \leq 10^6$), the number of amplitudes and the average perceived loudness to achieve.
- One line with n integers $a_1, \ldots, a_n$ ($|a_i| \leq 10^6$), the amplitudes.

Output

Output one line containing n numbers, the renormalised amplitudes with an average perceived loudness of x.

Your answers should have an absolute error of at most 10^{-6}.

Example

Input
5 6 0 1 -2 3 -4
Output
0 1 -2 3 -4
Input
4 1 1 3 3 7
Output
0.242535625 0.7276068751 0.7276068751 1.697749375

Problem F. Failing Flagship

Source file name:	Failing.c, Failing.cpp, Failing.java, Failing.py
Input:	`Standard`
Output:	`Standard`

Ahoy! You are sailing towards the next "Boats Are Pretty Cool" convention to sell your latest gadget: a new type of compass.

On a normal compass, it is difficult to read off the precise wind direction. However, your new type of compass lets you read off wind directions to a much higher precision! The display can display strings of at most 1000 characters.

Unfortunately, you have encountered some bad weather. After a few hours of heavy winds and big waves, you can finally look at your compass again. You read off the wind direction X you are going and know in which wind direction Y you need to go. However, to make the ship turn you have to enter the degrees of the angle the ship has to make in the control system. What is the smallest turn, in degrees, you have to make to get back on the right course?

The conversion of a wind direction to degrees goes as follows. The four basic wind directions are N, E, S, and W pointing at 0, 90, 180, and 270 degrees, respectively. There are also four wind directions consisting of two letters: NE, SE, SW, and NW, pointing at 45, 135, 225, and 315 degrees, respectively.

A wind direction can also consist of $k \geq 3$ letters $l_1 l_2 \dots l_k$. In that case, the last two letters indicate one of the four two-letter wind directions, i.e., $l_{k-1} l_k \in \{\text{NE}, \text{SE}, \text{SW}, \text{NW}\}$ and the other letters are equal to one of these, i.e., $l_i \in \{l_{k-1}, l_k\}$ for all $i \leq k-2$. This wind direction points precisely in the middle of the following two wind directions:

- wind direction $l_2 \dots l_k$,
- the first wind direction of at most $k-1$ letters you encounter when starting in $l_2 \dots l_k$ and move along the circle towards l_1.

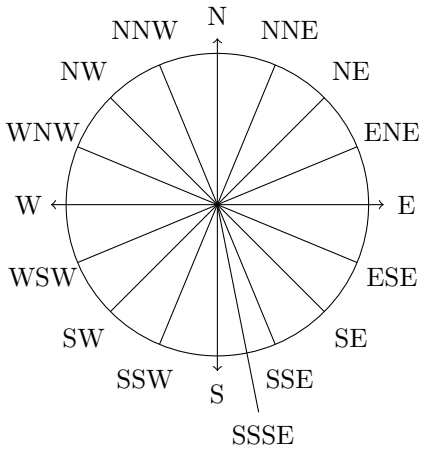


Figure 3: Wind directions

For example, the wind direction SSSE points in the middle of SSE and S, because S is the first wind direction with at most 3 letters when moving from SSE towards S, as can also been seen in Figure 3.

Input

The input consists of:

- One line with two strings X and Y ($1 \leq |X|, |Y| \leq 1000$), indicating the wind directions as described above.

Output

Output the smallest turn you have to make to go from direction X to Y.

Your answer should have an *absolute* error of at most 10^{-6}.

Example

Input	Output
N S	180
NNE SSSE	146.25
ENE NW	112.5

Problem G. Grinding Gravel

Source file name:	Grinding.c, Grinding.cpp, Grinding.java, Grinding.py
Input:	`Standard`
Output:	`Standard`

During the renovation of your garden, you decide that you want a gravel path running from the street to your front door. Being a member of the Boulders And Pebbles Community, you want this path to look perfect. You already have a regular grid to put the gravel in, as well as a large container of gravel containing exactly as much as the total capacity of the grid.

Perfectly ground gravel in a perfect grid.
CC BY-NC 2.0 by markjowen66 on Flickr

There is one problem: the gravel does not yet fit perfectly into the grid. Each grid cell has the same (fixed) capacity and every piece of gravel has a certain weight. You have a grindstone that can be used to split the stones into multiple pieces, but doing so takes time, so you want to do a minimal number of splits such that the gravel can be exactly distributed over the grid.

As an example, consider the first sample case. There are three grid cells of size 8, which can be filled as follows. Put the stones of weight 2 and 6 in the first cell. Now grind the stone of weight 7 into two pieces of weight 3 and 4. Then the other two grid cells get filled by weights $3, 5$ and $4, 4$ respectively.

Input

The input consists of:

- One line with two integers n and k ($1 \leq n \leq 100$, $1 \leq k \leq 8$), the number of pieces of gravel and the capacity per grid cell.
- One line with n integers $w_1, \ldots, w_n$ ($1 \leq w_i \leq 10^6$ for all i), the weight of each piece of gravel.

It is guaranteed that $w_1 + w_2 + \cdots + w_n$ is a multiple of k.

Output

Output the minimal number of times a stone needs to be split into two, such that all the pieces of gravel can be used to fill all the grid cells perfectly.

Example

Input	Output
5 8 2 4 5 6 7	1
2 5 12 13	4

Problem H. Heavy Hauling

Source file name:	Hauling.c, Hauling.cpp, Hauling.java, Hauling.py
Input:	`Standard`
Output:	`Standard`

The Parcel Retriever Robot.
CC BY 2.0 by Enrique Dans on Flickr

The warehouse of the Boxes And Parcels Center (BAPC) just received an official warning from the inspector: apparently, it does not conform to the latest safety requirements. In the past, it was allowed to stack multiple boxes at the same shelf location, but due to the potential fire hazard, this is no longer allowed. In a hurry, all employees of the BAPC are roused to move the boxes to distinct positions.

After moving the boxes, the automated parcel retriever robot needs to be reprogrammed such that it knows the correct location of the boxes. Per box that is moved d positions, it takes d^2 time to do this reprogramming. Of course, the BAPC should be up and running as soon as possible after moving the boxes, so the boxes should be moved in such a way that this total reprogramming time is as small as possible. Calculate the minimal time for the reprogramming for an optimal moving of boxes.

The warehouse of the BAPC is unbounded in both directions.

As an example, consider Figure 4, corresponding to the first example case. One box at position -1 is moved to the left, which costs 1 time for the reprogramming. The box at position 4 is moved one position to the right, to make place for one of the boxes at position 3, costing 1 time as well. Two boxes at position 3 are moved to the left (costing 1 and 4), and one box at position 3 is moved to the right (costing 1), making the total cost $1 + 1 + 1 + 4 + 1 = 8$.

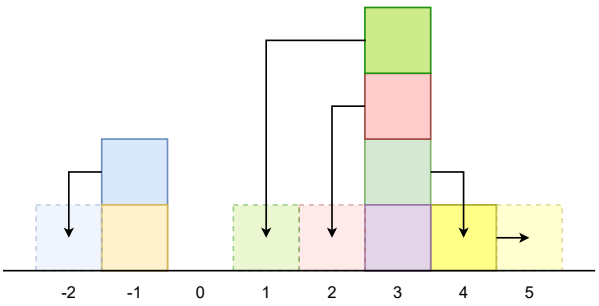


Figure 4: Visualisation of the first example case.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 10^6$), the number of boxes.
- One line with n integers x ($|x| \leq 10^9$), the position of each box. The box positions are ordered non-decreasingly.

Output

Output the minimal time to reprogram the parcel retriever robot for an optimal moving of boxes.

Example

Input	Output
7 -1 -1 3 3 3 3 4	8
8 2 2 2 2 2 2 4 4	24

Problem I. Ice Growth

Source file name:	Icegrowth.c, Icegrowth.cpp, Icegrowth.java, Icegrowth.py
Input:	`Standard`
Output:	`Standard`

Ice growth is dependent on temperature. A rule of thumb is that every 5 degrees of frost (on average in a 24-hour period) contributes to 1 cm of ice growth, whilst every 5 degrees above zero removes 1 cm of ice. For example, if there are three days with average temperatures of -3, 1 and -7 degrees Celsius there will be a total of $3 - 1 + 7 = 9$ degrees of frost and thus 1.8 cm of ice growth at the end of day 3. Of course, the ice thickness cannot be negative. If there is enough ice, people can skate on it. The required ice thickness depends on the person, as different persons have different perceptions of safety.

CC0, source: freesvg.org/girl-skating

There is currently no ice, but the weather report for the next n days has just come in, and a group of k people wants you to figure out how many of these days they can skate on the ice at the end of the day.

Input

The input consists of:

- One line containing two integers, n ($1 \leq n \leq 10^5$) the number of days and k ($1 \leq k \leq 10^5$) the number of people.
- One line with n integers $a_1, \ldots, a_n$ ($-10^6 \leq a_i \leq 10^6$ for all i), the average temperature on day i.
- One line with k integers $b_1, \ldots, b_k$ ($1 \leq b_j \leq 10^6$ for all j), the required minimal ice thickness in cm before person j can skate on the ice.

Output

Output a line with k integers $c_1, \ldots, c_k$, where c_j is the number of days that person j can skate on the ice at the end of the day.

Example

Input	Output
3 2 -3 1 -7 1 2	1 0
5 3 -5 -5 15 -5 -5 1 2 3	4 2 0

Problem J. Jabbing Jets

Source file name:	Jabbing.c, Jabbing.cpp, Jabbing.java, Jabbing.py
Input:	`Standard`
Output:	`Standard`

You have just gotten a new job at the Bathroom Accessories Production Company. The first task you are given is to jab holes into showerheads. To prove yourself, you have decided you want to create as many holes as possible.

However, you cannot just randomly drill holes everywhere in the showerhead.[1] In order to ensure that the showerheads look aesthetically pleasing, the company has composed some guidelines which you will have to follow. See Figure 5 for some examples of aesthetically pleasing showerheads.

- The holes should be arranged in concentric circles of radii $r_1, r_2, \ldots, r_n$: the center of every hole should be on one of these circles.
- The distance between the centers of any two holes should be at least e.

How many holes can you make at most?

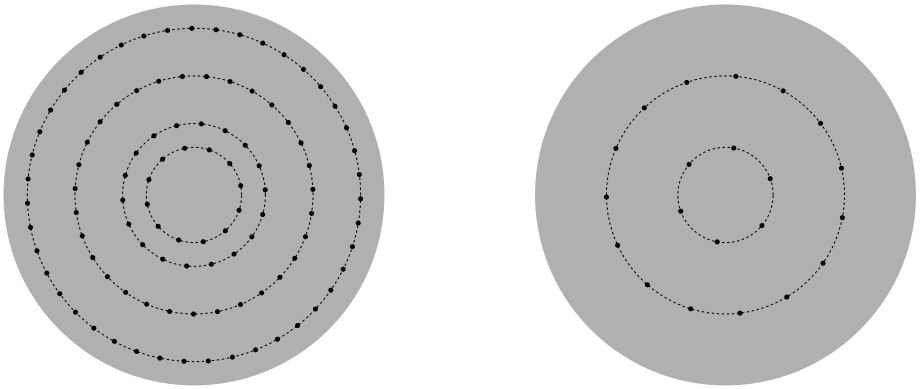


Figure 5: Possible aesthetically pleasing showerheads for the first two examples.

Input

The input consists of:

- One line with two integers n and e ($1 \leq n, e \leq 10^4$), the number of circles and the minimal distance between holes.
- One line with n integers $r_1, \ldots, r_n$ ($1 \leq r_i \leq 10^4$), the radii of the circles.

It is guaranteed that the numbers r_i are given in increasing order, and that $r_{i+1} - r_i \geq e$. Furthermore, it is guaranteed that increasing any radius r_i by at most 10^{-6} will not change the final answer.

Output

Output the maximal number of holes that you can make in the showerhead.

[1] At least, not without getting fired.

Example

Input	Output
4 1 2 3 5 7	104
2 2 2 5	21
3 20 14 53 80	44

Problem K. Knitting Patterns

Source file name:	Knitting.c, Knitting.cpp, Knitting.java, Knitting.py
Input:	`Standard`
Output:	`Standard`

It is the most relaxing hobby of anyone's grandma: knitting! Your grandma uses a lot of wool, even for the most simple patterns that she is knitting. You are sure that she can be more efficient with her wool, so you decide to write a program that calculates the most efficient use of wool.

Knitting Grandma.
Pixabay License

A knitting pattern is a long list of stitches, where each stitch can use a different colour.[2] There is a cost for starting or ending the use of a certain colour, for using the wool in a stitch, and for letting it strand through the back unused. For a given knitting pattern, calculate the least possible amount of wool required for every colour of wool.

As an example, consider the first example case. There are three colours of wool (red, green, and blue). The red wool is used at the start and at the end of the pattern: it is most efficient to start and stop using the red wool for both parts separately ($4 \cdot 4 = 16$ cost), and it is used in ten stitches ($10 \cdot 2 = 20$ cost), giving a total cost of 36. The green and blue wool have the same cost: start once (4 cost), use for five stitches ($5 \cdot 2 = 10$ cost), strand along the back for four stitches ($4 \cdot 1 = 4$ cost), and stop using the colour (4 cost), giving a total cost of 22.

Input

The input consists of:

- One line with three integers a, b, and c ($1 \leq a < b < c \leq 1000$), the cost of letting the wool strand through the back unused, the cost of using the wool in a stitch, and the cost of starting or ending the use of a colour of wool.
- One line with a string w ($1 \leq |w| \leq 26$), representing the unique letters used for denoting stitch colours.
- One line with a string p ($1 \leq |p| \leq 10^6$), representing the stitch colours of the knitting pattern.

All stitch colours use English lowercase letters (`a-z`).

Output

Output, for every colour of wool, in the order as they are in w, the amount of wool used in this pattern.

[2] Most real-life knitting patterns are two-dimensional, but since you zig-zag through such a pattern while knitting, the input is one-dimensional for simplicity.

Example

Input	Output
1 2 4 rgb rrrrrgbgbgbgbgbrrrrr	36 22 22
2 4 1000 ab abbbbbbbba	2024 2032

Problem L. Lots of Liquid

Source file name:	Liquid.c, Liquid.cpp, Liquid.java, Liquid.py
Input:	`Standard`
Output:	`Standard`

You work at a warehouse that sells chemical products, where somebody just placed an order for all the Boron Acetate Phosphoric Carbonate (BAPC) that you have in store. This liquid is stored in many separate lots, in cube-shaped containers, but your client requires the order to be delivered in a single cube-shaped container that fits all the BAPC liquid perfectly. What should be the size of this container?

Some of the cube-shaped containers. Used with permission from BeautifulChemistry.net

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 10^5$), the number of cube-shaped containers that you have in store.
- One line with n floating-point numbers c ($1 \leq c \leq 10^9$), the length of one of the sides for each of these containers.

Output

Output the length of one of the sides of the cube-shaped container that will contain all the BAPC liquid.

Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
3 21 28 35	42
3 22.10 2022 1337	2200.6131345362505
3 1.41421356 2.718281828 3.1415926535	3.777901284526486