Problem A. Atomic Energy

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

The *Next Wave Energy Research Club* is looking at several atoms as potential energy sources, and has asked you to do some computations to see which are the most promising.

Plasma ball by Halacious, Unsplash
https://unsplash.com/photos/0gvqXGL7XO4

Although an atom is composed of various parts, for the purposes of this method only the number of neutrons in the atom is relevant[1]. In the method, a laser charge is fired at the atom, which then releases energy in a process formally called *explodification*. Exactly how this process proceeds depends on the number of neutrons k:

- If the atom contains $k \leq n$ neutrons, it will be converted into a_k joules of energy.
- If the atom contains $k > n$ neutrons, it will decompose into two atoms with i and j neutrons respectively, satisfying $i, j \geq 1$ and $i + j = k$. These two atoms will then themselves explodificate.

When an atom with k neutrons is explodificated, the total energy that is released depends on the exact sequence of decompositions that occurs in the explodification process. Modern physics is not powerful enough to predict exactly how an atom will decompose—however, for explodification to be a reliable energy source, we need to know the minimum amount of energy that it can release upon explodification. You have been tasked with computing this quantity.

Input

The input consists of:

- One line with two integers n and q ($1 \leq n \leq 100$, $1 \leq q \leq 10^5$), the neutron threshold and the number of experiments.
- One line with n integers $a_1, \ldots, a_n$ ($1 \leq a_i \leq 10^9$ for each i), where a_i is the amount of energy released when an atom with i neutrons is explodificated.
- Then q lines follow, each with an integer k ($1 \leq k \leq 10^9$), asking for the minimum energy released when an atom with k neutrons is explodificated.

Output

For each query k, output the minimum energy released when an atom with k neutrons is explodificated.

[1] In fact, for this problem you might want to forget everything you thought you knew about chemistry.

Example

Input	Output
4 5 2 3 5 7 2 3 5 6 8	3 5 8 10 13
1 3 10 1 2 100	10 20 1000

Problem B. Bulldozer

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

You are tasked with bulldozing some buildings that stand along a long, straight road. The buildings are modelled as evenly spaced stacks of identical square blocks along an infinite line. Your powerful bulldozer is capable of moving any one of these blocks one unit of distance to the left or to the right. This may push other blocks out of the way, and blocks which sit atop moving blocks will move along. Blocks which are pushed over a gap fall down until they reach either the ground or another block.

For instance, consider the stacks of blocks shown on the left in Figure 1 below. If you push the block labelled C to the right, the blocks D and E would be pushed along to the right, since they are in the way. Blocks A, B and F would also move along because they are sitting on top of moving blocks. After pushing C to the right, E would be sitting over a gap, so E and F drop down to fill that gap. The resulting stacks are shown in the middle of Figure 1. Pushing block C one further step to the right would result in the configuration shown on the right.

Your goal is to *level* all the buildings: bulldoze until all stacks are of height at most 1, i.e., all blocks are on the ground. Note that the road stretches out infinitely far on either side, so this is always possible.

Given the initial heights of the stacks, determine the smallest number of moves you need to make to level all the buildings, where a move consists of using the bulldozer to push one block one step to the left or right.

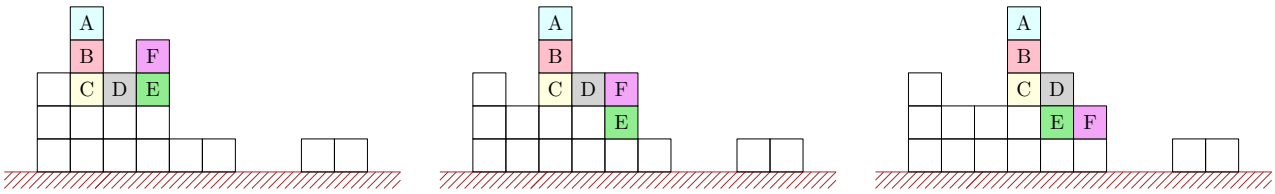


Figure 1: Illustration of a configuration of stacks of blocks, and the results of pushing the block labelled C towards the right twice (the blocks labelled A–F are coloured and labelled only for illustrative purposes).

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 2 \cdot 10^5$), the number of stacks of blocks.
- One line with n integers $a_1, \ldots, a_n$ ($0 \leq a_i \leq 10^9$ for each i), the initial heights of the stacks from left to right.

The example shown on the left in Figure 1 could be given by $3, 5, 3, 4, 1, 1, 0, 0, 1, 1$, but it could also be left- or right-padded with additional zeros.

Output

Output the minimum number of moves required to level every building.

Example

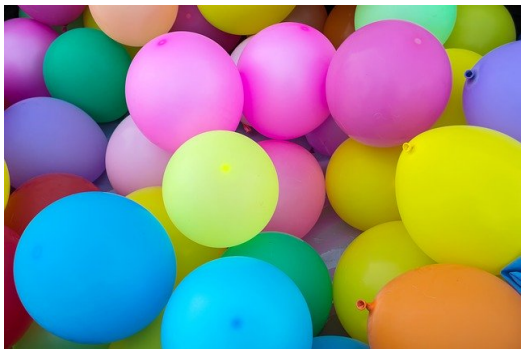
Input	Output
5 1 1 2 1 1	2
5 1 4 3 1 1	7
9 1 0 0 0 6 0 0 0 1	5
10 1 3 0 0 1 9 1 1 1 1	13

Problem C. Contest Struggles

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

Lotte is competing in a programming contest. Her team has already solved k out of the n problems in the problem set, but as the problems become harder, she begins to lose focus and her mind starts to wander.

She recalls hearing the judges talk about the difficulty of the problems, which they rate on an integer scale from 0 to 100, inclusive. In fact, one of the judges said that "*the problem set has never been so tough, the average difficulty of the problems in the problem set is d!*"



Balloons by Pexels, Pixabay
https://pixabay.com/images/id-1869790/

She starts thinking about the problems her team has solved so far, and comes up with an estimate s for their average difficulty. In hope of gaining some motivation, Lotte wonders if she can use this information to determine the average difficulty of the remaining problems.

Input

The input consists of:

- One line with two integers n and k ($2 \leq n \leq 10^6$, $0 < k < n$), the total number of problems and the number of problems Lotte's team has solved so far.
- One line with two integers d and s ($0 \leq d, s \leq 100$), the average difficulty of all the problems and Lotte's estimate of the average difficulty of the problems her team has solved.

Output

Assuming Lotte's estimate is correct, output the average difficulty of the unsolved problems, or "`impossible`" if the average difficulty does not exist. Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
2 1 70 50	90.00
10 3 80 90	75.7142857
2 1 100 10	impossible

Problem D. Dyson Circle

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

A Dyson Sphere is a theoretical construction around the sun or another star, that captures the entire energy output of the star. Science fiction writers have speculated that advanced civilizations will eventually build such a sphere, as the energy demands of such a society continue to grow without bounds. In our three-dimensional space, Dyson spheres are still in the realm of fiction. *Dy & Son*, the main energy company of your dimensional neighbours, has tasked you with carrying out a feasibility study in the two-dimensional world of Flatland.

Dy & Son has developed a modular Dyson Circle. It consists of independent square Dyson Units that can chain together to form a closed loop that gathers energy. Your task is to figure out how many of these Dyson units they actually need to enclose the star or stars they are interested in. Note that they want a single Dyson Circle, not a separate one for each star.

For convenience, both the stars Dy & Son wants to enclose and the Dyson Units used for this are modeled as squares of exactly 1 by 1 *Intergalactic Unit*, aligned to the *Intergalactic Coordinate System*. Your Dyson units connect if they have at least a corner in common. See Figure 2 for an example.

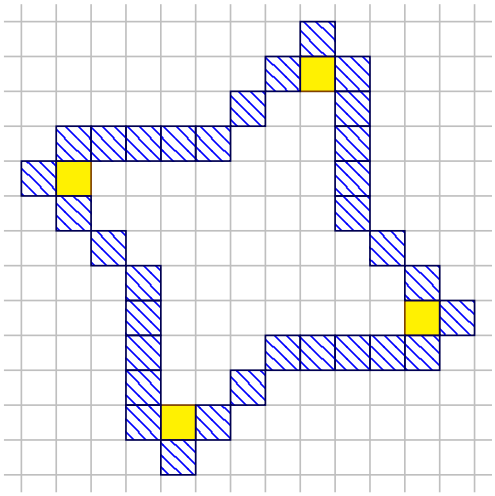


Figure 2: Illustration of Example Input 1: four stars (yellow squares) and an optimal Dyson Circle (dashed blue squares) surrounding them, and the remaining blackness of space shown in white.

Formally, select some squares in the plane to turn into Dyson Units, such that the remaining squares can be split into *inside* and *outside* squares. All the stars must be inside squares. The inside squares must form a contiguous region (connected via edges) and not connect to the outside squares (via edges). The outside squares form a contiguous region stretching off to infinity.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 2 \cdot 10^5$), the number of stars.
- n lines, each containing two integers x and y ($-10^6 \leq x, y \leq 10^6$), the location of the center of a star.

No two stars are at the same location.

Output

Output the least number of Dyson units required to capture the energy of all stars in the input.

Example

Input	Output
4 2 5 -5 2 -2 -5 5 -2	32
2 1 1 3 2	8
2 2 3 4 5	9

Problem E. Eruption

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

A volcano has recently erupted in Geldingadalur, Iceland. Fortunately this eruption is relatively small, and—unlike the infamous Eyjafjallajökull eruption—is not expected to cause delayed international flights or global outrage.

Geldingadalur eruption by Jon Schow, CC BY-SA

There is some concern about the magmatic gas that has been released as part of the eruption, as it could pose danger to human populations in the surrounding area. Scientists have estimated the total amount of gas emitted, which, due to lack of wind, has spread out uniformly across a circular area around the centre of the volcano. The authorities have evacuated the area, and would now like to close it off by surrounding the perimeter with barrier tape.

Input

The input consists of:

- One line with an integer a ($1 \leq a \leq 10^{18}$), the total area covered by gas in square metres.

Output

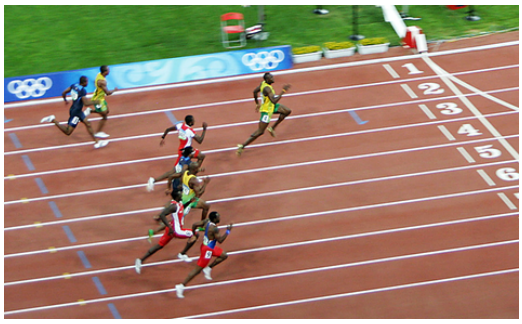
Output the total length of barrier tape needed to surround the area covered by gas, in metres. Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
50	25.066282746
1234	124.526709336

Problem F. Flatland Olympics

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



2008 Beijing Olympics by PhotoBobil
https://en.wikipedia.org/wiki/File:Usain_Bolt_winning.jpg

It is the day after Olympia, and you—as the organizer—are happy that *everything* worked well in these troublesome times. Well, not everything....

Since this morning e-mails have been filling up your inbox, containing complaints about obscured views during the most important race: the 100-meter dash. They demand their money back, or threaten exposing you on social media. To make things worse, spectators have not just complained once, but they have sent you a separate e-mail for every person that blocked their view at some point during the race! They even wrote multiple e-mails when two or more people blocked their view at the same time. And not only that, some visitors complained to the main sponsor *Dy & Son* who in turn has urged you to improve the situation.

Since you expect that a greater number of visitors will be allowed to spectate at the next Olympic games, you assume that there will be even more complaints if you do not address this issue. If the situation will be too bad, you may even lose your sponsor Dy & Son. Therefore, you decide to count the number of complaints beforehand. To do this, you model the running track as a straight line segment, and count the maximal number of complaints you could get based on the seating of the visitors. Depending on the number of complaints you expect, you will determine if you need to rework the seating or just reconfigure your spam blocker and try to find a new sponsor.

Input

The input consists of:

- One line containing four integers x_s, y_s, x_e and y_e ($|x_s|, |y_s|, |x_e|, |y_e| \leq 10^9$), where $s = (x_s, y_s)$ is the starting point of the running track and $e = (x_e, y_e)$ is the end point of the running track. Both s and e belong to the running track.
- One line containing an integer n ($1 \leq n \leq 10^5$), the number of visitors.
- n lines, each containing two integers x and y ($|x|, |y| \leq 10^9$), where (x, y) is the location of the seat of a visitor.

It is guaranteed that the track has a positive length, i.e. $s \neq e$. Further, you can assume that all visitors are seated at distinct locations and that no visitor is seated on the track.

Output

Output the total number of complaints that you would receive for the given seating.

Example

Input	Output
0 0 100 0 4 50 20 50 30 50 50 120 0	3
0 0 100 0 5 50 20 50 30 50 -20 50 -30 100 30	2

Explanation

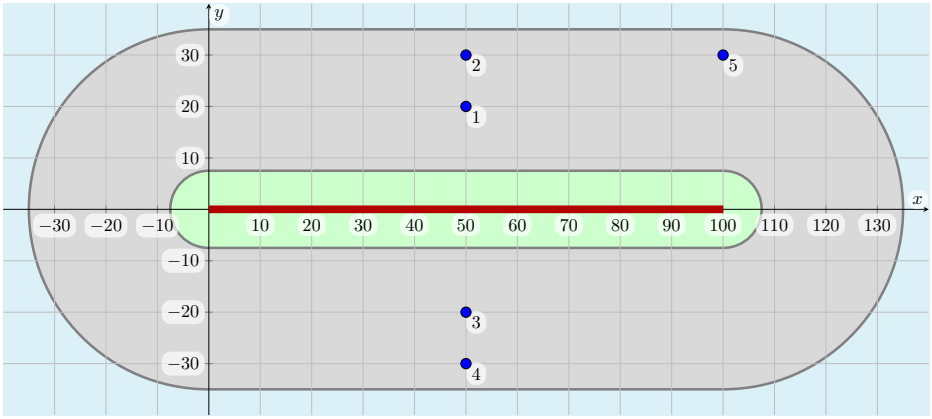


Figure 3: Illustration of Example Input 2. The running track is drawn as a red line and the seats of the visitors are highlighted in blue. The second visitor will complain about the first visitor and the fourth visitor will complain about the third visitor.

Problem G. Great Expectations

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

A *speedrun* is a playthrough of a game with the intention to complete it as quickly as possible. When speedrunning, you usually follow a pre-planned path through the game. Along this path, there may be some places where you have to pull off a difficult technique, or *trick*, which may cause a delay if you fail to pull it off successfully. Luckily you can *reset* the game at any time: if you have made a few mistakes, you can start a new run, losing your progress but instantaneously starting over with a clean slate. You can do this as often as you like.

The game you are currently speedrunning has a record of r seconds, which you intend to beat. You have discovered a path through the game that, in the best case, takes $n < r$ seconds. There are some tricks along the way, though: you know exactly where along the run they occur, what the probability is that you will pull them off successfully, and how many seconds you have to spend to recover if they fail.

Given this data, you want to find the optimal strategy for when to reset the game to minimise the expected time to set a new record. Write a program to determine what this smallest possible expected time is.

Input

The input consists of:

- One line with three integers n, r and m ($2 \leq n < r \leq 5\,000$, $1 \leq m \leq 50$), where n and r are as described above and m is the number of tricks.
- m lines, each containing three numbers describing a trick:
  - An integer t ($1 \leq t < n$), the time in the route (assuming no failed tricks before) at which the trick occurs,
  - a real number p ($0 < p < 1$ and p has at most 6 digits after the decimal point), the probability that the trick succeeds, and
  - an integer d ($1 \leq d \leq 1\,000$), the number of seconds required to recover in case the trick fails.

The tricks are given in sorted order by t, and no two tricks occur at the same time t in the route.

You may assume that, without resetting, a single playthrough has a probability of at least 1 in 50 000 to succeed at improving the record.

Output

Output the expected time you will have to play the game to set a new record, assuming an optimal strategy is used. Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
100 111 5 20 0.5 10 80 0.5 2 85 0.5 2 90 0.5 2 95 0.5 2	124
2 4 1 1 0.5 5	3
10 20 3 5 0.3 8 6 0.8 3 8 0.9 3	18.9029850746
10 50 1 5 0.5 30	15

Explanation

Example Input 1:

The record for this game is 111 seconds, and your route takes 100 seconds if everything goes right.

After playing for 20 seconds, there is a trick with a 50% success rate. If it succeeds, you keep playing. If it fails, you incur a 10 second time loss: now the run will take at least 110 seconds. It is still possible to set a record, but every other trick in the run has to be successful. It turns out to be faster on average to reset after failing the first trick.

Thus you repeat the first 20 seconds of the game until the trick is successful: with probability 1/2, it takes 1 attempt; with probability 1/4, it takes 2 attempts; and so on. On average, you spend 40 seconds on the first 20 seconds of the route.

Once you have successfully performed the first trick, you want to finish the run no matter the result of the other tricks: it takes 80 seconds, plus on average 1 second loss from each of the remaining 4 tricks. So the expected time until you set a record is 124 seconds.

Problem H. Heating Up

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

Jonas just entered his first chilli-eating contest. He is presented with a pizza consisting of n slices, numbered from 1 to n, each containing a selection of chilli peppers. Initially slices i and $i+1$ are adjacent on the plate (where $1 \leq i < n$), and so are slices 1 and n. According to the contest rules only one slice can be consumed at a time, and the slice must be finished in its entirety before a new slice is started. Jonas is allowed to pick any slice to eat first, but after that he is only allowed to eat slices that have at most one remaining adjacent slice.

The spiciness of each slice is measured in Scoville Heat Units (SHU). Jonas has a certain spiciness tolerance, also measured in SHU, which corresponds to the spiciness of the spiciest slice that Jonas can tolerate eating. He has also noticed that, after eating a slice of k SHU, his tolerance immediately increases by k.

Chilli pizza by Rahul Upadhyay, Unsplash
https://unsplash.com/photos/yDKHJxfiWDk

In order to win the contest, Jonas would like to finish all the slices of his pizza. Help him determine the minimum initial spiciness tolerance necessary to do so while abiding by the contest rules.

Input

The input consists of:

- One line with an integer n ($3 \leq n \leq 5 \cdot 10^5$), the number of pizza slices.
- One line with n integers $s_1, s_2, \ldots, s_n$ ($0 \leq s_i \leq 10^{13}$), where s_i is the spiciness of the ith slice in SHU.

Output

Output the minimum initial spiciness tolerance in SHU that Jonas needs in order to be able to eat all slices of the pizza.

Example

Input	Output
5 5 0 10 6 1	4
7 20 23 7 2 3 7 1	2

Problem I. Icelandic Corporation for Parcel Circulation

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

The *Icelandic Corporation for Parcel Circulation* is the leading carrier for transporting goods between Iceland and the rest of the world. Their newest innovation is a drone link connecting to mainland Europe that has a number of drones travelling back and forth along a single route.

Drone by Hyeri Kim, Pixabay
https://pixabay.com/images/id-1134764/

The drones are equipped with a sophisticated system that allows them to fly evasive manoeuvres whenever two drones come close to each other. Unfortunately, a software glitch has caused this system to break down and now all drones are flying along the route with no way of avoiding collisions between them.

For the purposes of this problem, the drones are considered as points moving along an infinite straight line with constant velocity. Whenever two drones are at the same location, they will collide, causing them to fall off their flight path and plummet into the Atlantic Ocean. The flight schedule of the drones is guaranteed to be such that at no point will there be three or more drones colliding at the same location.

You know the current position of each drone as well as their velocities. Your task is to assess the damage caused by the system failure by finding out which drones will continue flying indefinitely without crashing.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 10^5$), the number of drones. The drones are numbered from 1 to n.
- n lines, the ith of which contains two integers x_i and v_i ($-10^9 \leq x_i, v_i \leq 10^9$), the current location and the velocity of the ith drone along the infinite straight line.

The drones are given by increasing x coordinate and no two drones are currently in the same position, i.e. $x_i < x_{i+1}$ for each i. You may assume that there will never be a collision involving three or more drones.

Output

The output consists of:

- One line with an integer m ($0 \leq m \leq n$), the number of drones that never crash.
- One line with m space-separated integers, the indices of the drones that never crash in numerically increasing order.

Example

Input	Output
3 10 15 30 5 50 -1	1 3
6 0 3 2 2 3 1 4 3 5 2 6 3	2 1 6
10 -8 1 -4 1 -3 3 -2 -9 2 -3 4 1 5 1 6 4 8 4 10 3	4 1 6 7 8

Problem J. Jet Set

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

Your wealthy friends love to brag about all their travelling. Every time you see them, they have visited some new exotic place you have never heard of. All of them are all too happy to tell you they have been *all* around the world—but you're not so sure about that. Have these jetsetters made a real *circumnavigation*?

There exist many different definitions of what exactly constitutes a circumnavigation, but for the purposes of this problem we consider a circumnavigation a journey starting and ending at the same point and visiting all meridians (lines of longitude) along the way. Note that the North and South Pole are part of every meridian.

Illustration of Example Input 1, giving a circumnavigation starting and ending in Reykjavík, with additional waypoints in Athens, Jakarta, Honolulu and Chicago.

Amelia, one of your rich friends, gave you a log of her flights in the form of a list of waypoints. Her trip started at the first waypoint, visited the remaining waypoints in order, and finally went back from the last waypoint to the first. Between consecutive waypoints, Amelia always travelled along the shortest circular arc connecting the two points. Find out whether Amelia's trip can be considered a circumnavigation in the above sense, and if not find a meridian that Amelia never visited.

Input

The input consists of:

- One line with an integer n ($2 \leq n \leq 1\,000$), the number of waypoints.
- n lines, each with two integers ϕ and λ ($-90 < \phi < 90, -180 \leq \lambda < 180$), the latitude and longitude of one of the waypoints.

No two consecutive waypoints along the route are equal or antipodes (opposite points on the sphere) of each other.

Output

If the route is a valid circumnavigation, output `yes`. Otherwise, output `no`.

Example

Input	Output
5 64 -22 38 24 -6 107 21 -158 42 -88	yes
2 80 30 75 -150	yes
4 45 0 0 -170 -45 0 0 170	no

Problem K. Knitpicking

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

Kattis has many pairs of nice, warm, knit socks in her sock drawer that are perfect for the winter. These socks come in a wide range of colours and types, and have all been mixed together. Each morning Kattis needs to pick two matching socks.

To find matching socks, she simply randomly takes single socks out of the drawer until she has a matching pair. It may take a long time, for example when she keeps drawing right socks without a matching left one. How long does she need to keep drawing socks until she is guaranteed to have a pair to wear?

Gillie in one of his resting places By Dwight Sipler (cc by-sa)

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 1\,000$), the number of groups of identical socks.
- n lines, each describing a group of identical socks with the following:
  - A string i, the type of the socks in the group. The type i consists of between 1 and 20 lowercase English letters. Socks with the same type are considered compatible for fashion purposes.
  - A string j, the fit of the socks in the group, which is either `left`, `right` or `any`, indicating whether the socks fit on the left foot, the right foot or any foot.
  - An integer k ($1 \leq k \leq 1\,000$), the number of socks in the drawer that are of this type and fit.

A given fit of a given type of sock appears at most once in the input.

Output

Output the minimum number of socks Kattis needs to draw to be guaranteed to get a matching pair. If it is not possible to get a matching pair at all, output `impossible`.

Example

Input	Output
3 fuzzy any 10 wool left 6 wool right 4	8
3 sports any 1 black left 6 white right 6	impossible
2 warm any 5 warm left 3	4

Problem L. Lucky Shirt

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

Stack of clothes via pxfuel.com

You, a frantic competitive programmer, have collected a large number of T-shirts by competing in various programming contests. In fact, you have so many, that these are the only T-shirts you wear anymore. You keep them neatly folded in a large stack in your oversized closet. Each morning, you pick the top shirt from your stack of shirts to wear that day. At the end of the day, you throw the shirt in the laundry basket.

In order to keep a fresh supply of clean shirts, you sometimes also do laundry at night, washing all shirts in the laundry basket (including the one you wore that day). This is not according to some neat schedule however; the number of days between your wash cycles is a uniformly random integer between 1 (in which case you would wash only a single shirt) and the number of shirts you have. After washing your clothes, you put them back on top of the stack in a uniformly random order.

It is now the night after a successful programming contest, and you decide that the T-shirt you got there is your lucky shirt from now on. You wonder when you will be able to wear it again, and entertain yourself with thoughts about the good fortune you will receive when you do. You just completed your laundry and put all your shirts on the stack. Knowing the current position of your lucky shirt in the stack, what is the expected position of your lucky shirt after k more washing cycles?

Input

The input consists of:

- One line containing three integers n ($1 \leq n \leq 10^6$), the number of shirts you have, i ($1 \leq i \leq n$), the position of your lucky shirt counted from the top, and k ($1 \leq k \leq 10^6$), the number of washing cycles.

Output

Output the expected position of your lucky shirt after k washing cycles. Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
3 2 2	1.833333333
5 1 1	2.0
10 7 100	5.499986719