Thirty-seventh Annual University of Central Florida

High School Programming Tournament

Problems

Problem Name	**Filename**
Princess Plum Chiffon	princess
Stronghold Triangulation	stronghold
Jailhouse Rock	jail
Dencker's Dice	dice
I-4 Eyesore	eyesore
Hexes and Spells	hex
A Spy on the Inside	spy
You Come and Go	chameleon
Coin Drop	coin
Cut the Cake	cake
Ghastly Tenant	horror
Ultimate Hardcore Mindset	uhc

Call your program file: *filename*.c, *filename*.cpp, *filename*.java, or *filename*.py
Call your Java class: *filename*

For example, if you are solving Ghastly Tenant:
Call your program file: horror.c, horror.cpp, horror.java, or horror.py
Call your Java class: horror

Notes

1. All solutions must read from standard input and write to standard output. The judges will ignore all output sent to standard error.

2. All input sets used by the judges will follow the input specification in each problem. You do not need to test for input that violates the input format specification.

3. Submitted programs will be tested on additional judge cases and not just the sample cases given in the problem.

4. If you need use the value for π, use the value 3.141592653589793.

5. If you want to know how to compute the absolute error, given the actual value of a quantity, x, and the measured value of a quantity, x_0, then the absolute error value, Δx, is calculated as:

$$\Delta x = |x_0 - x|$$

6. The relative error is defined as the ratio of the absolute error of the measurement to the actual measurement. If you want to know how to compute the relative error, given the actual value of a quantity, x, and the measured value of a quantity, x_0, then the relative error value, r, is calculated as:

$$r = \frac{|x_0 - x|}{x} = \frac{\Delta x}{x}$$

7. Images used are owned by either the original author, the University of Central Florida, or used under appropriate licenses (such as various Creative Commons licenses).

Princess Plum Chiffon

Filename: `princess`

Princess Plum Chiffon has been kidnapped by the fearsome Biscuit! Before she was hidden away completely, the princess cast her last lucky star into the air. The star coursed through the skies before landing at the feet of the princess's love, Earl Gray. Earl tucked the star into his pocket solemnly. He told himself that he would find the princess, even if it meant searching every kingdom in the land.

There are n kingdoms scattered throughout the land, numbered 1 to n. There also exists a series of m bidirectional roads: each road connects a pair of kingdoms and takes a certain amount of time to travel. The lucky star Earl received from the princess is an artifact that he can activate at most once, at any point along his journey. If Earl uses the lucky star while he is at kingdom i, then he will be instantly transported to kingdom $n - i + 1$.

Earl is currently at kingdom 1, but he does not know where the princess has been taken. For each index i from 2 to n, Earl wishes to know the shortest amount of time it will take him to travel from kingdom 1 to kingdom i.

The Problem:

Determine how long it will take Earl to get from kingdom 1 to every other kingdom in the land. Note that each of these journeys are to be evaluated independently, and Earl has a lucky star that can be used at most once per journey.

The Input:

The first line of input contains two integers, n ($2 \leq n \leq 10^5$) and m ($1 \leq m \leq 10^5$), representing the number of kingdoms and the number of roads, respectively. The following m lines each contain three integers, u ($1 \leq u \leq n$), v ($1 \leq v \leq n$) and w ($1 \leq w \leq 10^9$), indicating that there exists a road that connects kingdoms u and v that takes w minutes to travel.

The input guarantees that there will never exist a road that connects a kingdom to itself, nor will there exist more than one road connecting the same pair of kingdoms.

The Output:

Output a line of n - 1 integers: for each index i from 2 to n, the shortest amount of time in minutes it will take Earl to travel from kingdom 1 to kingdom i. If Earl is unable to reach a kingdom, print -1.

Sample Input 1:

```
6 4
1 2 3
2 4 100
1 3 2
3 6 10
```

Sample Output 1:

```
3 2 2 3 0
```

Sample Input 2:

```
5 4
1 5 1
2 4 1
2 3 1
3 4 1
```

Sample Output 2:

```
-1 -1 -1 0
```

Stronghold Triangulation

Filename: `stronghold`

Recently, Steve and Alex have been playing a lot of Minecraft. They've built up their base and already have full diamond armor, so now all that's left for them to do is to defeat the final boss: the Ender Dragon. The only issue is they cannot find a stronghold! Every time they throw an Eye of Ender, it breaks, completely crushing their dreams. However, after much heartbreak, Steve came up with a brilliant idea: triangulating the location of the stronghold using only two Eyes of Ender.

Once thrown, Eyes of Ender begin moving in the direction of the nearest stronghold for a short distance. After throwing an Eye of Ender from position A and recording the angle it travels in, Steve can move to a different position B and throw another Eye. Using the angle of both Eyes, he can find where these vectors eventually intersect, leading them straight to the stronghold! In the case that the vectors never intersect, the Eyes must be moving towards two different strongholds, making triangulation impossible. Unfortunately, geometry is neither Steve's nor Alex's strong suit, so they need your help!

The Problem:

Given two distinct pairs of (x, y) coordinates and the angle (direction) that the Eye traveled at each coordinate pair, determine the (x, y) coordinates of the stronghold.

The Input:

The input will contain two lines, one for each Eye of Ender. Each line consists of an ordered pair of integers, (x, y) ($-10,000 \leq x \leq 10,000$; $-10,000 \leq y \leq 10,000$), representing the position of the throw, and an integer, a ($0 \leq a \leq 359$), representing the angle that the Eye of Ender traveled. Every angle is recorded with the player facing directly North and follows standard compass headings: 0° is North, 90° is East, 180° is South, and 270° is West.

For each attempt, it is guaranteed that the vectors formed by the two angles will not be parallel, and the two positions (A and B) will be different. It is also guaranteed that neither of the two given positions will be the final location of the stronghold.

The Output:

If triangulation is possible, output two space-separated real numbers, representing the x and y coordinate of the stronghold, respectively. If triangulation is impossible, output -1 instead. Each coordinate must be accurate to an absolute error of 10^{-3}.

Sample Input 1:

```
-300 42 45
42 -300 0
```

Sample Output 1:

```
42.00 384.00
```

Sample Input 2:

```
-35 50 280
35 50 90
```

Sample Output 2:

```
-1
```

Jailhouse Rock

Filename: `jail`

Peridot has hatched a daring plan to break out of jail. He has obtained a spare prison guard uniform, and all he needs now is a fake ID badge. Peridot has observed that each guard has a string of lowercase English letters on their badge.

Thanks to inside help, Peridot knows that the string on each guard's badge is generated according to a rule enforced by the prison warden. The warden broadcasts a string, *s*, and a positive integer, *x*, for all of the guards to see. Each guard then writes a string on their badge that contains string s as a substring exactly *x* times (no more and no less).

A substring is a contiguous sequence of characters within a string. For example, the strings "spaced" and "ace" are substrings of "spacedust" but "sus" and "sad" are not.

If Peridot generates his own string that follows the warden's rule, then he can pretend to be a prison guard without arousing suspicion.

The Problem:

Given the string *s* and the integer *x* given to the prison guards to generate their badges, generate a string that contains a string that Peridot can use as a fake ID badge.

The Input:

The input is a single line consisting of a string, *s* (whose length is between 1 and 20, inclusive), and an integer, *x* ($1 \le x \le 100$), describing how the guards' badges are generated. The string *s* will consist only of lowercase English letters.

The Output:

Output a string, *t*, representing a valid guard badge that Peridot may use. The string must be between 1 and 10^4 in length (inclusive), consist of only lowercase English letters, and contain the string *s* as a substring exactly *x* times.

Sample Input 1:	**Sample Output 1:**
`aaa 2`	`aaaa`

Sample Input 2:	**Sample Output 2:**
`abc 5`	`abcabcabcabcabc`

Sample Input 3:	**Sample Output 3:**
`letsrock 1`	`everybodyletsrock`

Dencker's Dice

Filename: `dice`

When he can't play the board game "Ticket to Ride," Professor Dencker needs a way to occupy himself, and he's come up with just the game to do so! At the start of his game, Professor Dencker will pick an integer, *t*, called the target value. He will then start rolling a fair die (meaning it can roll any value on it with equal probability) with sides labeled 1 through *s* (inclusive). The number it lands on will be added to his total (which, of course, initially starts at zero). If his total is ever equal to the target value, he wins the game! If the total is ever greater than the target value, he will lose the game. Professor Dencker will roll the die until he either wins or loses the game. Professor Dencker would like to know: given a target value and the number of sides on the die, what is the probability that he wins the game?

The Problem:

Given an integer target value and the number of sides of the die, determine the probability that Professor Dencker will win the game by reaching a total exactly equal to his target value.

The Input:

A scenario will start with a line containing two integers, s ($1 \le s \le 10^6$), representing the number of sides the die will have, and, v ($1 \le v \le 10^4$), representing the number of games (target values) Professor Dencker would like to know the probability of him winning with the given die. The following will be v lines each containing an integer, t ($1 \le t \le 10^6$), representing the target value for that game.

The Output:

For each target value, output a single line containing the percent chance that Professor Dencker will reach that target value. Output this as a single real number from 0 to 100. Answers within 10^{-6} relative or absolute error of the answer will be considered correct.

Sample Input 1:

```
6 2
1
2
```

Sample Output 1:

```
16.66666667
19.44444444
```

Sample Input 2:

```
3 1
3
```

Sample Output 2:

```
59.25925925
```

I-4 Eyesore

Filename: `eyesore`

In metropolitan Orlando, along the infamous Interstate 4 (I-4) highway, there lies a building constantly ridiculed for how long it has been left incomplete. This building has jokingly been deemed the "I-4 Eyesore."

Dr. Hunts Mittleshmirtz, after getting kicked out of the Quad-State Area due to the Alphabet-Rearrange-Inator shenanigans, has set eyes on his new home: the I-4 Eyesore. However, he is getting quite fed up with everyone calling it an "eyesore." To get his revenge, he created a brand new device: The Eyesore-Inator! With this, he now has the power to turn all of Orlando into an eyesore!

Luckily for us Orlandoans, Agent B is on the case! However, Dr. Hunts Mittleshmirtz is back with his crafty traps. As soon as Agent B broke through the door, he was dropped into a room surrounded by mirrors with only one way to escape: speaking the passcode. Foolish as always, Dr. Hunts Mittleshmirtz left the code written at a certain point in the room. Agent B needs your help to determine the minimum angle he has to turn to see the code.

The mirror room can be modeled as an axis-aligned square where the bottom left corner is at the origin, (0, 0). Each wall acts as a perfect mirror where the angle of incidence (θ_i) is equal to the angle of reflection (θ_r). This is modeled in Figure 1. However, whenever text is reflected, it becomes harder to read, so Agent B can only read the code within a certain number of reflections. No matter the orientation of the text after reflection, Agent B will be able to read it thanks to his training. Agent B can rotate either clockwise or counterclockwise to read the code, and also can see through himself (luckily!). Due to the nature of the trap, Agent B will always be facing directly towards the left wall and positioned in the center of the room. Some of Agent B's options for the first sample case are modeled in Figure 2.

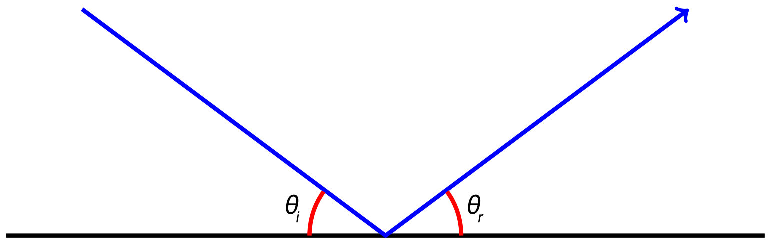

Figure 1: Perfect mirror reflection

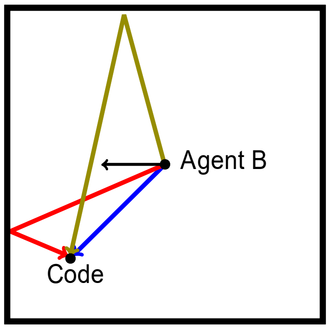

Figure 2: First sample

The Problem:

Given the room dimensions, the coordinates of the code, and the number of reflections allowed, determine the minimum angle that Agent B has to turn to see the code.

The Input:

A room will have a single line containing four integers, s, k, x and y ($0 \le s \le 1{,}000$; $0 \le k \le 100$; $0 \le x \le s$; $0 \le y \le s$), where s is the length of the square room, k is the maximum number of reflections, and x and y are the coordinates of the passcode, respectively. It is guaranteed that the code will not be located at the center of the room (not even Dr. Hunts Mittleshmirtz would be that foolish!).

The Output:

Output a single line containing a single floating point number: the minimum angle (in degrees) that Agent B must rotate to see the code. Each angle must be accurate to an absolute or relative error of 10^{-3}.

Sample Input 1:	**Sample Output 1:**
`5 1 1 1`	`23.19859`

Sample Input 2:	**Sample Output 2:**
`4 0 2 3`	`90.00000`

Sample Input 3:	**Sample Output 3:**
`6 2 4 2`	`5.19442`

Hexes and Spells

Filename: `hex`

Double, double toil and trouble;

Fire burn and cauldron bubble.

They may say witches cause rubble,

But some want to be loveable.

Helga the Witch is quite fed up with being viewed as a disgusting and evil old hag. All the other witches want to get revenge on a foe, or curse those in power, or just cause absolute mayhem. But Helga wants to make the world a better place. If she uses her talents for good, though, the other witches will banish her, so she has to sneak this benevolent spell into their slew of malevolent hexes. Luckily, it is her turn to arrange the hexes! Hexes and spells require a certain number of instructions to be carried out. If she can arrange the hexes in some manner such that the instructions of her spell appear consecutively, the spell will be cast. Can you help her determine if her magnanimous spell can be casted?

The instructions can be represented as a string of characters. So if one of the witches wants to cast a hex where first you add the scale of a dragon, then add the eye of newt, followed by howlet's wing, this hex can be represented by "BAC." Luckily for Helga, all spells and hexes are guaranteed to have the same number of instructions. For example, if she has to arrange the hexes "ABA" and "BAC" and wants to make the spell "ACA," she can arrange the hexes as "BAC" then "ABA" to get an overall arrangement of "BACABA," which contains the string "ACA," therefore successfully casting Helga's spell. Hexes can only be used once. However, multiple hexes and spells may have the same instructions.

The Problem:

Given a list of hexes the witches will cast, and the spell that Helga wants to cast, determine if there is some arrangement of the hexes such that Helga can sneak her spell in.

The Input:

A casting will start with a line containing two integers, n ($1 \leq n \leq 100$), representing the number of hexes Helga must consider and, s ($1 \leq s \leq 50$), representing the length of both the hexes and spells, respectively. The following n lines will consist of a string of s characters, representing the instructions for that hex. The last line of the casting will contain another string of length s representing the spell Helga wants to cast. Hexes and spells will be composed of only uppercase alphabetic characters.

The Output:

For the casting, output "`Helga is loveable`" if Helga can rearrange the hexes to make her spell or "`Toil and Trouble`" if she cannot.

Sample Input 1:

```
2 3
ABA
BAC
ACA
```

Sample Output 1:

```
Helga is loveable
```

Sample Input 2:

```
3 3
ABB
DAC
CAB
DDB
```

Sample Output 2:

```
Toil and Trouble
```

Sample Input 3:

```
4 4
ABBA
DABA
CAFA
CCDD
DDDA
```

Sample Output 3:

```
Helga is loveable
```

A Spy on the Inside

Filename: `spy`

HQ has dispatched a team of highly trained spies to infiltrate the Candy Cane Factory and discover the secret to delicious, colorful stripes. Although morale is high, victory is not assured: there is an elite squadron of guards that has established a close watch over the entire compound.

The floor plan of the compound may be modeled as a rectangular grid of characters. Each character is equal to one of the following:

- `#` that indicates a wall cell
- `.` that indicates a floor cell that is not occupied by a person
- `P` that indicates a floor cell that is occupied either by a spy or a guard (note that the spies have all donned stolen uniforms, and are now indistinguishable from the guards)

Astonishingly, HQ has misplaced their mission log, and no one can recall the exact number of spies that were sent into the compound. The only information HQ has is the following:

- A guard will never be directly adjacent to (i.e., share a border with) more than one wall cell, because, otherwise, their view of the compound would be too obstructed
- A spy will always be directly adjacent to at least one wall cell, because, otherwise, they would be too exposed

HQ has broken into the command center and now has access to the video feed of the entire compound. To avoid jeopardizing the mission any further, they must determine the number of spies they sent into the Candy Cane Factory as accurately as possible.

The Problem:

Given the floor map of the compound, determine the minimum and maximum possible number of spies that were on the team dispatched by HQ.

The Input:

The first line of input contains two integers, n and m ($1 \leq n \leq 100$; $1 \leq m \leq 100$), representing the height and width of the floor plan of the compound, respectively. A description of the floor plan will be on the following n lines. Each of the n lines will contain m characters. It is guaranteed that each character in the floor plan is either a `#`, a `.`, or a `P`.

The Output:

Output two space-separated integers: the minimum and maximum possible number of spies that were on the team dispatched by HQ.

Sample Input 1:

```
5 5
#####
#P..#
#.PP#
#...#
#####
```

Sample Output 1:

```
1 2
```

Sample Input 2:

```
4 4
###P
P#PP
P#PP
####
```

Sample Output 2:

```
4 6
```

Sample Input 3:

```
7 7
#######
#..P..#
#.....#
#P...P#
#.....#
#..P..#
#######
```

Sample Output 3:

```
0 4
```

Sample Input 4:

```
5 7
#######
#.....#
#..P..#
#.....#
#######
```

Sample Output 4:

```
0 0
```

You Come and Go

Filename: `chameleon`

Kiwi the Chameleon is about to embark on her world tour as a legendary rock opera artist. On her tour, Kiwi will visit a number of different locations in a predetermined order. Each location has a predominant color palette, represented by an integer value.

As everyone knows, chameleons enjoy changing colors to match their surroundings. Whenever Kiwi arrives in a place that is a different color from the place she came from, she will change her color to match the new place. In addition, Kiwi will always change her color upon arriving at the first location in her tour.

To ensure that her costume changes are as seamless as possible, Kiwi would like to know how many times she will change color on her tour.

The Problem:

Given the color of each location Kiwi will visit in order, determine the number of times she will change color on her tour.

The Input:

The first line of input contains an integer, n ($1 \le n \le 100$), representing the number of locations Kiwi will visit on her tour. The second line of input contains n integers, a_i ($1 \le a_i \le 100$), representing the color of each location in the order she visits them.

The Output:

Output a single integer: the number of times Kiwi will change color.

Sample Input 1:

```
8
1 2 3 4 5 6 7 8
```

Sample Output 1:

```
8
```

Sample Input 2:

```
12
1 1 1 2 2 2 3 3 3 1 1 1
```

Sample Output 2:

```
4
```

Sample Input 3:

```
1
100
```

Sample Output 3:

```
1
```

Coin Drop

Filename: `coin`

Brian has dropped a coin represented as a circle onto a square table. He hasn't looked to see where on the table it's landed yet. He knows that it has landed flat on the table since he did not hear the distinctive ring of the metal piece hitting the floor. Brian also knows that the center of the coin must lie on the table because if it didn't, the coin would have toppled and proceeded to fall on the floor.

Brian prefers that a portion of the coin hangs off the table as it will be easier to pick up (he'll be able to partly lift it from underneath). As Brian hates mild inconveniences, he's much too terrified to check how the coin has landed and therefore would like you to write a program determining the probability that some portion of the coin is hanging off the table. Every location of the coin in which the coin's center remains on the table is equally likely to occur. All locations that do not meet that requirement cannot occur.

The Problem:

Given a coin and a table, determine the probability that when the coin is dropped on the table in a uniformly random location such that its center lies on the table, some portion of the coin will be hanging off the table.

The Input:

The input will consist of a single line with two integers, r ($1 \le r \le 10{,}000$) and s ($1 \le s \le 1{,}000$), representing the radius of the coin dropped (in inches), and the side lengths (also in inches) of the square table it has been dropped on, respectively.

The Output:

Output the probability that Brian will be able to pick up the coin in a convenient manner after it is dropped. Output this as a real number from 0 to 1 (inclusive). Answers that are within 10^{-3} absolute error of the answer will be accepted.

Sample Input 1:	Sample Output 1:
`1 5`	`0.64000000000`

Sample Input 2:	Sample Output 2:
`8 18`	`0.98765432098`

Cut the Cake

Filename: `cake`

Tyler just baked the most delicious, scrumptious-looking, incredible-smelling Tres Leches cake. Being a generous man, he wants to share this delicacy with his roommates. To do so, he and his roommates will cut the cake into many pieces of size 1.

All of Tyler's roommates are programmers. Programmers hate floating point numbers because they lead to all sorts of precision issues (yuck!). They also love dividing things by two, and can't stand to make imperfect divisions (that is, divisions with non-integer results). Thus, whenever they make a cut, they divide the piece into equal halves. If the size of the piece is odd, Tyler will first bite off a sliver of size 1 and then they perform the division. Tyler loves Tres Leches cake!

Tyler is wondering how many times they will divide cake pieces in half until every piece is of size 1. He does not count the times he bites off a sliver to make an even-sized piece, as this results in him getting to eat Tres Leches cake, so he doesn't mind.

The Problem:

Given the initial size of the cake, determine the number of cuts that will be made until every cake piece is of size 1.

The Input:

The input will be a single line containing a single integer, s ($0 < s < 10^9$), representing the size of the cake.

The Output:

Output the number of cuts that will be made until every cake piece is of size 1.

Sample Input 1:	**Sample Output 1:**
`10`	`7`

Sample Input 2:	**Sample Output 2:**
`1`	`0`

Sample Input 3:	**Sample Output 3:**
`4`	`3`

Ghastly Tenant

Filename: `horror`

There's a ghost in your house! At first you were afraid. You were petrified! You kept thinking you could never live with him by your side.

But then, you realized that the ghost was essentially harmless! Well, except when he keeps you up all night with his incessant wailing. You were searching for a new place to live when the ghost considerately informed you that he will only be haunting you for a finite number of nights. You decide to wait and, in the meantime, find a place to stay on his frightful nights.

Every evening, the ghost leaves you a message on your bathroom mirror via conveniently washable red paint. After recording all the messages thus far, you have noticed a pattern. On nights when the ghost keeps you up all night, the ghost's message contains at least one occurrence of two consecutive identical characters. On the days where this does not occur, the ghost instead opts for silently sliding all the furniture in your residence exactly three inches to the left, thereby allowing you to get a good night's sleep.

The Problem:

Given the ghost's daily message, determine if there is at least one occurrence of two consecutive identical characters.

The Input:

The first and only line of input will contain the ghost's message as a string of length between 1 and 100 (inclusive). The strings will only include lowercase English letters (a-z) and spaces. There will be no consecutive spaces and no leading or trailing spaces

The Output:

For the message, determine if the string has at least one occurrence of two consecutive identical characters. If there is a consecutive valid pair, output "`No sleep here.`". Otherwise, instead output "`Goodnight!`"

Sample Input 1:	**Sample Output 1:**
`rest the whole night`	`Goodnight!`

Sample Input 2:	**Sample Output 2:**
`beware the horrors`	`No sleep here.`

Sample Input 3:	**Sample Output 3:**
`no sleeeeeeeep tonight`	`No sleep here.`

Ultimate Hardcore Mindset

Filename: `uhc`

Steve was feeling ambitious, and wanted to conquer a task that's been on his mind for ages: beating Minecraft by defeating the Ender Dragon in Ultra Hardcore mode. That is, defeating the Ender Dragon while locked in hard difficulty, with no natural health regeneration, and losing all progress upon death. This task may be daunting for some, but not for Steve; Steve simply never got around to it. With this newfound motivation, he wanted to just get it over with so he could brag to his friends and move on to other activities. As such, he plays with a greedy mindset. If there is any shortcut he can take, no matter the risk, he will take it. Cave mining with no armor? Taking on 3 blazes at a time? MLG Water Bucket Trick? No problem, or so he thought.

To defeat the Ender Dragon, Steve has chosen a set of tasks he must complete in order. Each task takes a certain amount of time to complete, and has a certain probability of success. Upon failing a task, Steve must restart from the beginning. Note that if Steve fails a task, he still uses up the amount of time required for that task.

After numerous failed attempts, and with his goal still hopelessly out of reach, he realized: maybe slow and steady does win the race.

The Problem:

Help Steve determine the expected amount of time it will take him to beat Ultra Hardcore Minecraft if he keeps up with this greedy mindset!

The Input:

The first line of input contains one integer, n ($1 \le n \le 10$), representing the number of tasks Steve must face. The following line contains n real numbers, p_i ($0.2 \le p_i \le 1$), representing the probability of passing each task in the order they must be completed. The following line contains n integers, a_i ($1 \le a_i \le 100$), representing the time it takes to complete each task.

The Output:

Output a single real number that represents the expected amount of time Steve will take to defeat the Ender Dragon. Answers within 10^{-3} absolute or relative error of the answer will be accepted.

Sample Input 1:	Sample Output 1:

`3`
`0.5 0.5 0.8`
`1 10 100`
	`155`

Sample Input 2:	Sample Output 2:

`4`
`0.3 0.9 0.2023 0.99`
`5 3 5 10`
	`144.17457746946394`