A. Avoiding Zero

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array of n integers $a_1, a_2, \ldots, a_n$.

You have to create an array of n integers $b_1, b_2, \ldots, b_n$ such that:

- The array b is a rearrangement of the array a, that is, it contains the same values and each value appears the same number of times in the two arrays. In other words, the multisets $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_n\}$ are equal.
  For example, if $a = [1, -1, 0, 1]$, then $b = [-1, 1, 1, 0]$ and $b = [0, 1, -1, 1]$ are rearrangements of a, but $b = [1, -1, -1, 0]$ and $b = [1, 0, 2, -3]$ are not rearrangements of a.
- For all $k = 1, 2, \ldots, n$ the sum of the first k elements of b is nonzero. Formally, for all $k = 1, 2, \ldots, n$, it must hold

$$b_1 + b_2 + \cdots + b_k \neq 0\,.$$

If an array $b_1, b_2, \ldots, b_n$ with the required properties does not exist, you have to print NO.

Input

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

The first line of each testcase contains one integer n ($1 \leq n \leq 50$) — the length of the array a.

The second line of each testcase contains n integers $a_1, a_2, \ldots, a_n$ ($-50 \leq a_i \leq 50$) — the elements of a.

Output

For each testcase, if there is not an array $b_1, b_2, \ldots, b_n$ with the required properties, print a single line with the word NO.

Otherwise print a line with the word YES, followed by a line with the n integers $b_1, b_2, \ldots, b_n$.

If there is more than one array $b_1, b_2, \ldots, b_n$ satisfying the required properties, you can print any of them.

Example

input
4 4 1 -2 3 -4 3 0 0 0 5 1 -1 1 -1 1 6 40 -31 -9 0 13 -40

output
YES 1 -2 3 -4 NO YES 1 1 -1 1 -1 YES -40 13 40 0 -9 -31

Note

Explanation of the first testcase: An array with the desired properties is $b = [1, -2, 3, -4]$. For this array, it holds:

- The first element of b is 1.
- The sum of the first two elements of b is -1.
- The sum of the first three elements of b is 2.
- The sum of the first four elements of b is -2.

Explanation of the second testcase: Since all values in a are 0, any rearrangement b of a will have all elements equal to 0 and therefore it clearly cannot satisfy the second property described in the statement (for example because $b_1 = 0$). Hence in this case the answer is NO.

Explanation of the third testcase: An array with the desired properties is $b = [1, 1, -1, 1, -1]$. For this array, it holds:

- The first element of b is 1.
- The sum of the first two elements of b is 2.
- The sum of the first three elements of b is 1.
- The sum of the first four elements of b is 2.
- The sum of the first five elements of b is 1.

Explanation of the fourth testcase: An array with the desired properties is $b = [-40, 13, 40, 0, -9, -31]$. For this array, it holds:

- The first element of b is -40.
- The sum of the first two elements of b is -27.
- The sum of the first three elements of b is 13.
- The sum of the first four elements of b is 13.
- The sum of the first five elements of b is 4.
- The sum of the first six elements of b is -27.

B. Chess Cheater

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You like playing chess tournaments online.

In your last tournament you played n games. For the sake of this problem, each chess game is either won or lost (no draws). When you lose a game you get 0 points. When you win you get 1 or 2 points: if you have won also the previous game you get 2 points, otherwise you get 1 point. If you win the very first game of the tournament you get 1 point (since there is not a "previous game").

The outcomes of the n games are represented by a string s of length n: the i-th character of s is W if you have won the i-th game, while it is L if you have lost the i-th game.

After the tournament, you notice a bug on the website that allows you to change the outcome of **at most** k of your games (meaning that at most k times you can change some symbol L to W, or W to L). Since your only goal is to improve your chess rating, you decide to cheat and use the bug.

Compute the maximum score you can get by cheating in the optimal way.

Input

Each test contains multiple test cases. The first line contains an integer t ($1 \leq t \leq 20,000$) — the number of test cases. The description of the test cases follows.

The first line of each testcase contains two integers n, k ($1 \leq n \leq 100,000$, $0 \leq k \leq n$) – the number of games played and the number of outcomes that you can change.

The second line contains a string s of length n containing only the characters W and L. If you have won the i-th game then $s_i =$ W, if you have lost the i-th game then $s_i =$ L.

It is guaranteed that the sum of n over all testcases does not exceed $200,000$.

Output
For each testcase, print a single integer – the maximum score you can get by cheating in the optimal way.

Example

input
8 5 2 WLWLL 6 5 LLLWWL 7 1 LWLWLWL 15 5 WWWLLLWWWLLLWWW 40 7 LLWLWLWWWLWLLWLWWWLWLLWLLWLLLLWLLWWWLWWL 1 0 L 1 1 L 6 1 WLLWLW

output
7 11 6 26 46 0 1 6

Note
Explanation of the first testcase. Before changing any outcome, the score is 2. Indeed, you won the first game, so you got 1 point, and you won also the third, so you got another 1 point (and not 2 because you lost the second game).

An optimal way to cheat is to change the outcomes of the second and fourth game. Doing so, you end up winning the first four games (the string of the outcomes becomes `WWWWL`). Hence, the new score is $7 = 1 + 2 + 2 + 2$: 1 point for the first game and 2 points for the second, third and fourth game.

Explanation of the second testcase. Before changing any outcome, the score is 3. Indeed, you won the fourth game, so you got 1 point, and you won also the fifth game, so you got 2 more points (since you won also the previous game).

An optimal way to cheat is to change the outcomes of the first, second, third and sixth game. Doing so, you end up winning all games (the string of the outcomes becomes `WWWWWW`). Hence, the new score is $11 = 1 + 2 + 2 + 2 + 2 + 2$: 1 point for the first game and 2 points for all the other games.

C. The Hard Work of Paparazzi

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are a paparazzi working in Manhattan.

Manhattan has r south-to-north streets, denoted by numbers $1, 2, \ldots, r$ in order from west to east, and r west-to-east streets, denoted by numbers $1, 2, \ldots, r$ in order from south to north. Each of the r south-to-north streets intersects each of the r west-to-east streets; the intersection between the x-th south-to-north street and the y-th west-to-east street is denoted by (x, y). In order to move from the intersection (x, y) to the intersection (x', y') you need $|x - x'| + |y - y'|$ minutes.

You know about the presence of n celebrities in the city and you want to take photos of as many of them as possible. More precisely, for each $i = 1, \ldots, n$, you know that the i-th celebrity will be at the intersection (x_i, y_i) in exactly t_i minutes from now (and he will stay there for a very short time, so you may take a photo of him only if at the t_i-th minute from now you are at the intersection (x_i, y_i)). You are very good at your job, so you are able to take photos instantaneously. You know that $t_i < t_{i+1}$ for any $i = 1, 2, \ldots, n - 1$.

Currently you are at your office, which is located at the intersection $(1, 1)$. If you plan your working day optimally, what is the maximum number of celebrities you can take a photo of?

Input
The first line of the input contains two positive integers r, n ($1 \le r \le 500$, $1 \le n \le 100,000$) – the number of south-to-north/west-to-east streets and the number of celebrities.

Then n lines follow, each describing the appearance of a celebrity. The i-th of these lines contains 3 positive integers t_i, x_i, y_i ($1 \le t_i \le 1,000,000$, $1 \le x_i, y_i \le r$) — denoting that the i-th celebrity will appear at the intersection (x_i, y_i) in t_i minutes from now.

It is guaranteed that $t_i < t_{i+1}$ for any $i = 1, 2, \ldots, n - 1$.

Output
Print a single integer, the maximum number of celebrities you can take a photo of.

Examples

input
10 1 11 6 8

output
0

input
6 9 1 2 6 7 5 1 8 5 5 10 3 1 12 4 4 13 6 2 17 6 6 20 1 4 21 5 4

output
4

input
10 4 1 2 1 5 10 9 13 8 8 15 9 9

output
1

input
500 10 69 477 122 73 186 235 341 101 145 372 77 497 390 117 440 494 471 37 522 300 498 682 149 379

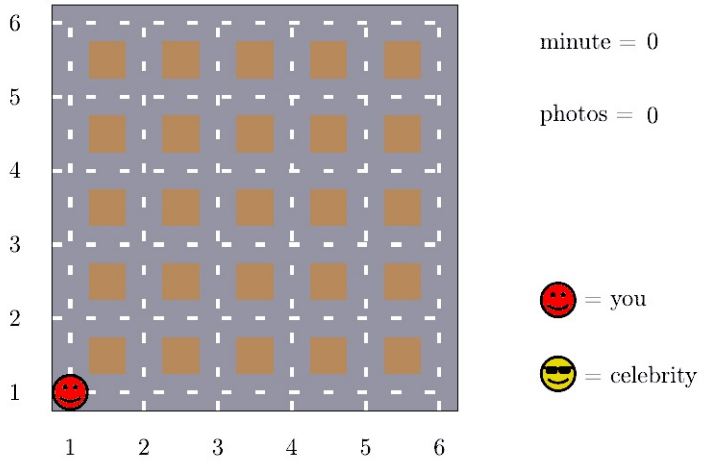
821 486 359 855 157 386
output
3

Note

Explanation of the first testcase: There is only one celebrity in the city, and he will be at intersection $(6, 8)$ exactly 11 minutes after the beginning of the working day. Since you are initially at $(1, 1)$ and you need $|1 - 6| + |1 - 8| = 5 + 7 = 12$ minutes to reach $(6, 8)$ you cannot take a photo of the celebrity. Thus you cannot get any photo and the answer is 0.

Explanation of the second testcase: One way to take 4 photos (which is the maximum possible) is to take photos of celebrities with indexes $3, 5, 7, 9$ (see the image for a visualization of the strategy):

- To move from the office at $(1, 1)$ to the intersection $(5, 5)$ you need $|1 - 5| + |1 - 5| = 4 + 4 = 8$ minutes, so you arrive at minute 8 and you are just in time to take a photo of celebrity 3.
- Then, just after you have taken a photo of celebrity 3, you move toward the intersection $(4, 4)$. You need $|5 - 4| + |5 - 4| = 1 + 1 = 2$ minutes to go there, so you arrive at minute $8 + 2 = 10$ and you wait until minute 12, when celebrity 5 appears.
- Then, just after you have taken a photo of celebrity 5, you go to the intersection $(6, 6)$. You need $|4 - 6| + |4 - 6| = 2 + 2 = 4$ minutes to go there, so you arrive at minute $12 + 4 = 16$ and you wait until minute 17, when celebrity 7 appears.
- Then, just after you have taken a photo of celebrity 7, you go to the intersection $(5, 4)$. You need $|6 - 5| + |6 - 4| = 1 + 2 = 3$ minutes to go there, so you arrive at minute $17 + 3 = 20$ and you wait until minute 21 to take a photo of celebrity 9.



Explanation of the third testcase: The only way to take 1 photo (which is the maximum possible) is to take a photo of the celebrity with index 1 (since $|2 - 1| + |1 - 1| = 1$, you can be at intersection $(2, 1)$ after exactly one minute, hence you are just in time to take a photo of celebrity 1).

Explanation of the fourth testcase: One way to take 3 photos (which is the maximum possible) is to take photos of celebrities with indexes $3, 8, 10$:

- To move from the office at $(1, 1)$ to the intersection $(101, 145)$ you need $|1 - 101| + |1 - 145| = 100 + 144 = 244$ minutes, so you can manage to be there when the celebrity 3 appears (at minute 341).
- Then, just after you have taken a photo of celebrity 3, you move toward the intersection $(149, 379)$. You need $|101 - 149| + |145 - 379| = 282$ minutes to go there, so you arrive at minute $341 + 282 = 623$ and you wait until minute 682, when celebrity 8 appears.
- Then, just after you have taken a photo of celebrity 8, you go to the intersection $(157, 386)$. You need $|149 - 157| + |379 - 386| = 8 + 7 = 15$ minutes to go there, so you arrive at minute $682 + 15 = 697$ and you wait until minute 855 to take a photo of celebrity 10.

D. Unshuffling a Deck

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a deck of n cards numbered from 1 to n (not necessarily in this order in the deck). You have to sort the deck by repeating the following operation.

- Choose $2 \le k \le n$ and split the deck in k nonempty contiguous parts $D_1, D_2, \ldots, D_k$ (D_1 contains the first $|D_1|$ cards of the deck, D_2 contains the following $|D_2|$ cards and so on). Then reverse the order of the parts, transforming the deck into $D_k, D_{k-1}, \ldots, D_2, D_1$ (so, the first $|D_k|$ cards of the new deck are D_k, the following $|D_{k-1}|$ cards are D_{k-1} and so on). The internal order of each packet of cards D_i is unchanged by the operation.

You have to obtain a sorted deck (i.e., a deck where the first card is 1, the second is 2 and so on) performing at most n operations. It can be proven that it is always possible to sort the deck performing at most n operations.

Examples of operation: The following are three examples of valid operations (on three decks with different sizes).

- If the deck is `[3 6 2 1 4 5 7]` (so 3 is the first card and 7 is the last card), we may apply the operation with $k = 4$ and $D_1 =$ `[3 6]`, $D_2 =$`[2 1 4]`, $D_3 =$`[5]`, $D_4 =$`[7]`. Doing so, the deck becomes `[7 5 2 1 4 3 6]`.
- If the deck is `[3 1 2]`, we may apply the operation with $k = 3$ and $D_1 =$`[3]`, $D_2 =$`[1]`, $D_3 =$`[2]`. Doing so, the deck becomes `[2 1 3]`.
- If the deck is `[5 1 2 4 3 6]`, we may apply the operation with $k = 2$ and $D_1 =$`[5 1]`, $D_2 =$`[2 4 3 6]`. Doing so, the deck becomes `[2 4 3 6 5 1]`.

Input

The first line of the input contains one integer n ($1 \le n \le 52$) — the number of cards in the deck.

The second line contains n integers $c_1, c_2, \ldots, c_n$ — the cards in the deck. The first card is c_1, the second is c_2 and so on.

It is guaranteed that for all $i = 1, \ldots, n$ there is exactly one $j \in \{1, \ldots, n\}$ such that $c_j = i$.

Output

On the first line, print the number q of operations you perform (it must hold $0 \le q \le n$).

Then, print q lines, each describing one operation.

To describe an operation, print on a single line the number k of parts you are going to split the deck in, followed by the size of the k parts: $|D_1|, |D_2|, \ldots, |D_k|$.

It must hold $2 \le k \le n$, and $|D_i| \ge 1$ for all $i = 1, \ldots, k$, and $|D_1| + |D_2| + \cdots + |D_k| = n$.

It can be proven that it is always possible to sort the deck performing at most n operations. If there are several ways to sort the deck you can output any of them.

Examples

input
4 3 1 2 4
output

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

input
6 6 5 4 3 2 1

output
1 6 1 1 1 1 1 1

input
1 1

output
0

Note

Explanation of the first testcase: Initially the deck is `[3 1 2 4]`.

- The first operation splits the deck as `[(3) (1 2) (4)]` and then transforms it into `[4 1 2 3]`.
- The second operation splits the deck as `[(4) (1 2 3)]` and then transforms it into `[1 2 3 4]`.

Explanation of the second testcase: Initially the deck is `[6 5 4 3 2 1]`.

- The first (and only) operation splits the deck as `[(6) (5) (4) (3) (2) (1)]` and then transforms it into `[1 2 3 4 5 6]`.

E. Xum

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have a blackboard and initially only an **odd** number x is written on it. Your goal is to write the number 1 on the blackboard.

You may write new numbers on the blackboard with the following two operations.

- You may take two numbers (not necessarily distinct) already on the blackboard and write their sum on the blackboard. The two numbers you have chosen remain on the blackboard.
- You may take two numbers (not necessarily distinct) already on the blackboard and write their bitwise XOR on the blackboard. The two numbers you have chosen remain on the blackboard.

Perform a sequence of operations such that at the end the number 1 is on the blackboard.

Input

The single line of the input contains the odd integer x ($3 \le x \le 999,999$).

Output

Print on the first line the number q of operations you perform. Then q lines should follow, each describing one operation.

- The "sum" operation is described by the line "a + b", where a, b must be integers already present on the blackboard.
- The "xor" operation is described by the line "a ^ b", where a, b must be integers already present on the blackboard.

The operation symbol (+ or ^) must be separated from a, b by a whitespace.

You can perform at most $100,000$ operations (that is, $q \le 100,000$) and all numbers written on the blackboard must be in the range $[0, 5 \cdot 10^{18}]$. It can be proven that under such restrictions the required sequence of operations exists. You can output any suitable sequence of operations.

Examples

input
3

output
5 3 + 3 3 ^ 6 3 + 5 3 + 6 8 ^ 9

input
123

output
10 123 + 123 123 ^ 246 141 + 123 246 + 123 264 ^ 369 121 + 246 367 ^ 369 30 + 30 60 + 60 120 ^ 121

F. Boring Card Game

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

When they are bored, Federico and Giada often play the following card game with a deck containing $6n$ cards.

Each card contains one number between 1 and $6n$ and each number appears on exactly one card. Initially the deck is sorted, so the first card contains the number 1, the second card contains the number 2, ..., and the last one contains the number $6n$.

Federico and Giada take turns, alternating; Federico starts.

In his turn, the player takes 3 contiguous cards from the deck and puts them in his pocket. The order of the cards remaining in the deck is not changed. They play until the deck is empty (after exactly $2n$ turns). At the end of the game both Federico and Giada have $3n$ cards in their pockets.

You are given the cards in Federico's pocket at the end of the game. Describe a sequence of moves that produces that set of cards in Federico's pocket.

Input

The first line of the input contains one integer n ($1 \le n \le 200$).

The second line of the input contains $3n$ numbers $x_1, x_2, \ldots, x_{3n}$ ($1 \le x_1 < x_2 < \ldots < x_{3n} \le 6n$) – the cards in Federico's pocket at the end of the game.

It is guaranteed that for each test there is at least one sequence of moves that produces such set of cards in Federico's pocket.

Output

Print $2n$ lines, each containing 3 integers.

The i-th line should contain, in increasing order, the integers $a_i < b_i < c_i$ written on the three cards taken by the player during the i-th turn (so taken by Federico if i is odd and by Giada if i is even).

If there is more than one possible sequence of moves, you can print any.

Examples

input
2 2 3 4 9 10 11

output
9 10 11 6 7 8 2 3 4 1 5 12

input
5 1 2 3 4 5 9 11 12 13 18 19 20 21 22 23

output
19 20 21 24 25 26 11 12 13 27 28 29 1 2 3 14 15 16 18 22 23 6 7 8 4 5 9 10 17 30

Note

Explanation of the first testcase: Initially the deck has $12 = 2 \cdot 6$ sorted cards, so the deck is $[1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12]$.

- During turn 1, Federico takes the three cards $[9\ 10\ 11]$. After his move, the deck is $[1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 12]$.
- During turn 2, Giada takes the three cards $[6\ 7\ 8]$. After her move, the deck is $[1\ 2\ 3\ 4\ 5\ 12]$.
- During turn 3, Federico takes the three cards $[2\ 3\ 4]$. After his move, the deck is $[1\ 5\ 12]$.
- During turn 4, Giada takes the three cards $[1\ 5\ 12]$. After her move, the deck is empty.

At the end of the game, the cards in Federico's pocket are $[2\ 3\ 4\ 9\ 10\ 11]$ and the cards in Giada's pocket are $[1\ 5\ 6\ 7\ 8\ 12]$.

G. One Billion Shades of Grey

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have to paint with shades of grey the tiles of an $n \times n$ wall. The wall has n rows of tiles, each with n tiles.

The tiles on the boundary of the wall (i.e., on the first row, last row, first column and last column) are already painted and you shall not change their color. All the other tiles are not painted. Some of the tiles are broken, you shall not paint those tiles. It is guaranteed that the tiles on the boundary are not broken.

You shall paint all the non-broken tiles that are not already painted. When you paint a tile you can choose from 10^9 shades of grey, indexed from 1 to 10^9. You can paint multiple tiles with the same shade. Formally, painting the wall is equivalent to assigning a shade (an integer between 1 and 10^9) to each non-broken tile that is not already painted.

The contrast between two tiles is the absolute value of the difference between the shades of the two tiles. The total contrast of the wall is the sum of the contrast of all the pairs of adjacent non-broken tiles (two tiles are adjacent if they share a side).

Compute the minimum possible total contrast of the wall.

Input

The first line contains n ($3 \le n \le 200$) – the number of rows and columns.

Then n lines, each containing n integers, follow. The i-th of these lines describe the i-th row of tiles. It contains the n integers a_{ij} ($-1 \le a_{ij} \le 10^9$). The value of a_{ij} described the tile on the i-th row and j-th column:

- If $a_{ij} = 0$, then the tile is not painted and shall be painted.
- If $a_{ij} = -1$, then the tile is broken and shall not be painted.
- If $1 \le a_{ij} \le 10^9$, then the tile is already painted with the shade a_{ij}.

It is guaranteed that the tiles on the boundary are already painted, the tiles not on the boundary are not already painted, and the tiles on the boundary are not broken.

Output

Print a single integer – the minimum possible total contrast of the wall.

Examples

input
3 1 7 6 4 0 6 1 1 1

output
26

input
3 10 100 1 1 -1 100 10 10 10

output
396

input
5 6 6 5 4 4 6 0 0 0 4 7 0 0 0 3 8 0 0 0 2 8 8 1 2 2

output
34

input
7 315055237 841510063 581663979 148389224 405375301 243686840 882512379 683199716 -1 -1 0 0 0 346177625 496442279 0 0 0 0 0 815993623 223938231 0 0 -1 0 0 16170511 44132173 0 -1 0 0 0 130735659 212201259 0 0 -1 0 0 166102576 123213235 506794677 467013743 410119347 791447348 80193382 142887538

output
10129482893

Note

Explanation of the first testcase: The initial configuration of the tiles is (tiles to paint are denoted by ?):

1 7 6
4 ? 6
1 1 1

A possible way to paint the tile achieving the minimum possible contrast of 26 is:

1 7 6
4 5 6
1 1 1

Explanation of the second testcase: Since all tiles are either painted or broken, there is nothing to do. The total contrast is 396.

Explanation of the third testcase: The initial configuration of the tiles is (tiles to paint are denoted by ?):

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

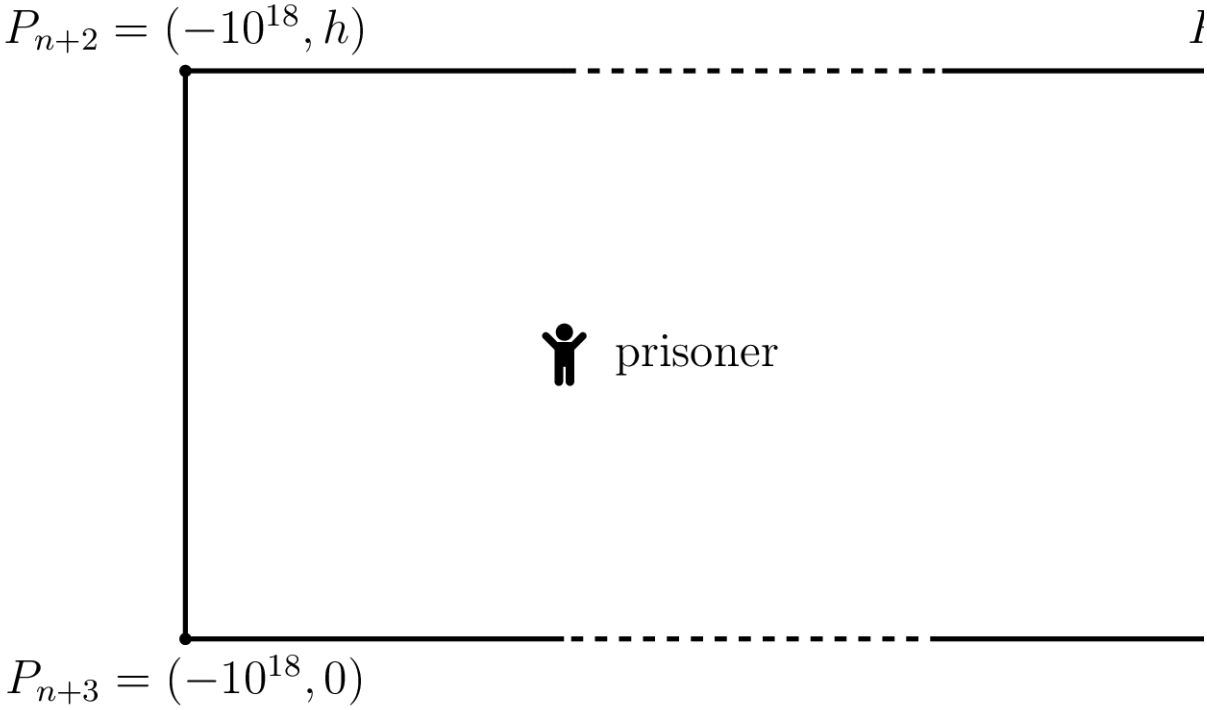
A possible way to paint the tiles achieving the minimum possible contrast of 34 is:

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

H. Prison Break

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A prisoner wants to escape from a prison. The prison is represented by the interior of the convex polygon with vertices $P_1, P_2, P_3, \ldots, P_{n+1}, P_{n+2}, P_{n+3}$. It holds $P_1 = (0, 0)$, $P_{n+1} = (0, h)$, $P_{n+2} = (-10^{18}, h)$ and $P_{n+3} = (-10^{18}, 0)$.



The prison walls $P_{n+1}P_{n+2}$, $P_{n+2}P_{n+3}$ and $P_{n+3}P_1$ are very high and the prisoner is not able to climb them. Hence his only chance is to reach a point on one of the walls $P_1P_2, P_2P_3, \ldots, P_nP_{n+1}$ and escape from there. On the perimeter of the prison, there are two guards. The prisoner moves at speed 1 while the guards move, **remaining always on the perimeter of the prison**, with speed v.

If the prisoner reaches a point of the perimeter where there is a guard, the guard kills the prisoner. If the prisoner reaches a point of the part of the perimeter he is able to climb and there is no guard there, he escapes immediately. Initially the prisoner is at the point $(-10^{17}, h/2)$ and the guards are at P_1.

Find the minimum speed v such that the guards can guarantee that the prisoner will not escape (assuming that both the prisoner and the guards move optimally).

Notes:

- At any moment, the guards and the prisoner can see each other.
- The "climbing part" of the escape takes no time.
- You may assume that both the prisoner and the guards can change direction and velocity instantly and that they both have perfect reflexes (so they can react instantly to whatever the other one is doing).
- The two guards can plan ahead how to react to the prisoner movements.

Input

The first line of the input contains n ($1 \le n \le 50$).

The following $n + 1$ lines describe $P_1, P_2, \ldots, P_{n+1}$. The i-th of such lines contain two integers x_i, y_i ($0 \le x_i, y_i \le 1,000$) — the coordinates of $P_i = (x_i, y_i)$.

It is guaranteed that $P_1 = (0, 0)$ and $x_{n+1} = 0$. The polygon with vertices $P_1, P_2, \ldots, P_{n+1}, P_{n+2}, P_{n+3}$ (where P_{n+2}, P_{n+3} shall be constructed as described in the statement) is guaranteed to be convex and such that there is no line containing three of its vertices.

Output

Print a single real number, the minimum speed v that allows the guards to guarantee that the prisoner will not escape. Your answer will be considered correct if its relative or absolute error does not exceed 10^{-6}.

Examples

input
2 0 0 223 464 0 749
output
1

input

```
3
0 0
2 2
2 4
0 6
```

output

```
1.0823922
```

input

```
4
0 0
7 3
7 4
5 7
0 8
```

output

```
1.130309669
```

input

```
5
0 0
562 248
460 610
281 702
206 723
0 746
```

output

```
1.148649561
```

input

```
7
0 0
412 36
745 180
747 184
746 268
611 359
213 441
0 450
```

output

```
1.134745994
```