Plan for the Worst! (airstrike)

The latest *Trample*'s air raid against the *United Nations of Antarctica* (UNA) has destroyed everything! Now it's time for Edoardo, the emperor of the UNA, to plan the reconstruction of all the new N buildings of the continent. The i-th building will be constructed in a strategic position at coordinates (X_i, Y_i).

Figure 1: One of Trample's formidable fighter jets.

Everyone knows that Trample's fighter jets are very powerful. In particular, a single airstrike is able to destroy all the buildings that have a given X or Y coordinate in a matter of seconds. Before giving his approval, Edoardo would like to know if the plan is resistant to air attacks. Can you help him find out the minimum number of airstrikes needed to destroy all the new buildings?

☞ Among the attachments of this task you may find a template file `airstrike.*` with a sample incomplete implementation.

Input

The first line contains the only integer N, the number of buildings. Each of the next N lines contains two integers X_i and Y_i, the coordinates of the i-th building.

Output

You need to write a single line with an integer: the minimum number of airstrikes needed to destroy all the buildings.

Constraints

- $1 \le N \le 10\,000$.
- $1 \le X_i, Y_i \le N$ for each $i = 0 \ldots N-1$.
- No two buildings have the same coordinates.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.
- **Subtask 2** (10 points) $N \leq 4$.
- **Subtask 3** (10 points) $N \leq 9$.
- **Subtask 4** (10 points) $N \leq 17$.
- **Subtask 5** (10 points) $X_i, Y_i \leq 10$ for each $i = 0 \ldots N-1$.
- **Subtask 6** (15 points) $X_i, Y_i \leq 17$ for each $i = 0 \ldots N-1$.
- **Subtask 7** (15 points) $Y_i \neq Y_j$ for each i, j such that $i \neq j$.
- **Subtask 8** (30 points) No additional limitations.

Examples

input	output

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

4
1 2
2 1
3 3
4 4
	4

Explanation

In the **first sample case**, Trample is able to destroy all the buildings using only two airstrikes. In particular, he can destroy the first three buildings with a single airstrike, by targeting all the buildings that have $X_i = 1$. Then, he can destroy the last two buildings with another airstrike that targets all the buildings that have $Y_i = 3$.

In the **second sample case**, no two buildings have the same X or Y coordinate. This means that Trample needs to use at least four airstrikes, one for each building.

Pay That Box! Home Edition (`boardgame`)

Giorgio is organizing an amazing board game night. He has invited N friends, and they are going to play *Pay That Box! Home Edition*, the board game version of the homonymous game show. Giorgio really loves it: he owns several copies of it, so that there can be a lot of matches going on at the same time.

Figure 1: Some friends have already arrived and are eager to play!

The rules of the game state that it must be played by at least L and at most U players. Of course, Giorgio needs to make sure that he can split his friends into many groups such that each group has the correct number of players. He also needs to check that he has enough copies of the game. It would be a shame if someone would not be able to play!

Given the number of friends, can you help Giorgio find out whether everybody will be able to play? If so, can you tell him the minimum number of copies of the game he needs?

☞ Among the attachments of this task you may find a template file `boardgame.*` with a sample incomplete implementation.

Input

The first line contains three integers N, L and U: the number of friends and the minimum and maximum number of players of the game.

Output

You need to write a single line with an integer. If Giorgio cannot split his friends into many groups such that everybody is able to play, write `-1`. Otherwise, write the minimum number of copies of the game he must have in order to make everyone play.

Constraints

- $1 \leq N, L, U \leq 10^9$.
- $L \leq U$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (15 points) $N, L, U \leq 5$.

– **Subtask 3** (15 points) $L = 1$.

– **Subtask 4** (15 points) $L = U$.

– **Subtask 5** (15 points) $N, L, U \leq 10^5$.

– **Subtask 6** (40 points) No additional limitations.

Examples

input	output
`6 2 5`	`2`
`6 4 5`	`-1`

Explanation

In the **first sample case** Giorgio's friends can be divided into two groups, the first one with four people and the second one with two.

In the **second sample case** there is no way to split Giorgio's friends into groups such that each of them has between 4 and 5 people.

Coin Change (`coinchange`)

When in-presence university was still a thing, Marco loved going to the canteen for lunch. Since the price of the meal was constantly changing, he ended up with tons of coins in his wallet! Thus, he decided to go to a *coin changer*, a professional that can optimize the space used by your money.

Figure 1: Marco trusting the coin changer.

When you visit a coin changer, he first takes all of your money, and then he gives you back the same amount of money, but composing it with the least total amount of pieces. Coin changers handle all existing sizes of euros: 1, 2, 5, 10, 20 and 50 *cents* (hundreds of an euro) and 1, 2, 5, 10, 20, 50, 100, 200 and 500 euros. Marco is skeptical about the honesty of those exchanges, hence he wants to be prepared before visiting. Help him compute the number and type of pieces that an honest coin changer should give him back!

☞ Among the attachments of this task you may find a template file `coinchange.*` with a sample incomplete implementation.

Input

The first line contains 15 integers V_i, the number of pieces from 1 cent to 500 euros.

Output

You need to write a single line with 15 integers: the number of pieces S_i returned by the coin changer, by type from 1 cent to 500 euro. The output should be such that $\sum_{i=0}^{14} S_i$ is minimized.

Constraints

- $0 \leq V_i \leq 10^9$ for each $i = 0 \ldots 14$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (18 points)	The coin changer will give you only 500 euro pieces.
– **Subtask 3** (31 points)	$V_i \leq 10^3$.
– **Subtask 4** (22 points)	$V_i = 0$ for $i = 0 \ldots 5$ (no cents, only whole euros).
– **Subtask 5** (29 points)	No additional limitations.

Examples

input	output
`100 10 50 10 0 13 11 10 1 10 100 1000 200 1 1`	`0 0 0 0 1 0 0 1 1 0 2 0 1 1 145`
`10 20 0 0 0 1 199 10 16 0 15 0 0 2 1`	`0 0 0 0 0 0 0 0 0 0 0 0 0 0 3`

Explanation

In the **first sample case** Marco gives to the coin changer:

$$\begin{aligned}
100 \times 0.01€ &= 1.00€ \\
10 \times 0.02€ &= 0.20€ \\
50 \times 0.05€ &= 2.50€ \\
10 \times 0.10€ &= 1.00€ \\
0 \times 0.20€ &= 0.00€ \\
13 \times 0.50€ &= 6.50€
\end{aligned}$$

$$\begin{aligned}
11 \times 1.00€ &= 11.00€ \\
10 \times 2.00€ &= 20.00€ \\
1 \times 5.00€ &= 5.00€ \\
10 \times 10.00€ &= 100.00€ \\
100 \times 20.00€ &= 2000.00€ \\
100 \times 20.00€ &= 2\,000.00€ \\
1000 \times 50.00€ &= 50\,000.00€ \\
200 \times 100.00€ &= 20\,000.00€ \\
1 \times 200.00€ &= 200.00€ \\
1 \times 500.00€ &= 500.00€
\end{aligned}$$

The total of 74 847.20€ can then be optimally distributed in 145 pieces of 500€, 2 of 20€ and one of 200€, 100€, 5€, 2€ and 0.20€.

In the **second sample case** the total amount of money is 1500.00€, which is equivalent to 3 pieces of 500€.

Delayed Signals (`delay`)

Giorgio loves playing with *delays* to compose electronic songs! A delay is an electrical component that takes two inputs: a musical signal, which is a sequence S_t for $t = 0 \dots N-1$ of integer values, and the delay amount, which is also a sequence D_t for $t = 0 \dots N-1$ of integers (it can change over time!).

As output, the delay component produces the exact same values appearing in the input musical signal, in the correct order, possibly skipping some of them and repeating others, in order to match as much as possible the requested delay. We say that the integer O_t in output at time t has a delay of x if it is equal to the input musical signal S_{t-x}. At every time step t, the delay component chooses the value of x closer to D_t which produces a value coming no earlier than O_{t-1} in the original sequence.

This delayed version of the input musical signal is very useful to create psychedelic echo effects.

Giorgio had a pedal applying delays to his musical tracks, but it just broke and now he wants to replace it with a software version. Help Giorgio compute the output of a delay component!

Figure 1: Giorgio's broken delay pedal.

For example, suppose that the delay amount is fixed to 2 and the input sequence is:

`0 5 8 9 7 10 4 1.`

Then, the delayed output is:

`0 0 0 5 8 9 7 10,`

since it reports the same values as the original sequence in order, repeating the first three times and skipping the last two, with every integer in output being produced with the requested delay of two (except for the first with a delay of zero and the second with a delay of one, which are the maximum delays possible).

Suppose now that the delay amount is varying over time: `5 4 3 2 1 0 2 1`, with the same input sequence. Then, the most accurate output is:

`0 0 0 5 9 10 10 4.`

The first three values are 0, with delays 0, 1, and 2 respectively, which is as close as you can get to the requested 5, 4, 3. The following values 5, 9, 10 match exactly the requested delays of 2, 1 and 0 respectively. The following value, 10, has a delay of 1 (instead of the requested 2), which is as much as possible given that you have to respect the order of the original sequence. Finally, 4 conclude the sequence with the requested delay of 1.

☞ Among the attachments of this task you may find a template file `delay.*` with a sample incomplete implementation.

Input

The first line contains the only integer N. The second line contains N integers S_t. The third line contains N integers D_t.

Output

You need to write a single line with N integers O_t: the delayed signal.

Constraints

- $1 \le N \le 100\,000$.
- $-10^9 \le S_t \le 10^9$ for each $t = 0 \dots N-1$.
- $0 \le D_t < N$ for each $t = 0 \dots N-1$.
- The integers in the input sequence S_t are all distinct.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (30 points)	$N \le 100$ and $D_t = D_{t'}$ for every t, t'.
– **Subtask 3** (20 points)	$N \le 1000$.
– **Subtask 4** (15 points)	$D_t = D_{t'}$ for every t, t'.
– **Subtask 5** (35 points)	No additional limitations.

Examples

input	output

`8`
`0 5 8 9 7 10 4 1`
`2 2 2 2 2 2 2 2`
	`0 0 0 5 8 9 7 10`

`8`
`0 5 8 9 7 10 4 1`
`5 4 3 2 1 0 2 1`
	`0 0 0 5 9 10 10 4`

Quantum Magic (`magic`)

Luca has recently discovered an ancient book[1] revealing a new candidate source of unlimited energy: *quantum magic*. Quantum magic is *quantized*, as it can only come in nine discrete levels of energy, corresponding to the integers from 1 to 9. But most importantly it is also *magic*, so that it can create free energy from nowhere!

Figure 1: Quantum magic flux across nodes and resonators.

Unfortunately, quantum magic occurs naturally only in very few specific locations, all along the most mystical of the *ley lines*. This line is L miles long and contains N quantum magic nodes, each at a specific (integer) number of miles X_i along the line (with the first being at the start $X_0 = 0$), and each with a fixed energy level E_i. The total energy produced by all these nodes is barely enough to power a light bulb, but Edoardo has a trick up his sleeve: quantum magic *resonators*!

As the magic itself, resonators also come in discrete levels from 1 to 9, and are effective only if placed in integer number of miles along the ley line. A resonator of level ℓ in point x can only be activated by a magic source in location $x' = x - \ell$ and energy level ℓ', and only if ℓ' is a divisor or multiple of ℓ. If so happens, the resonator becomes a quantum magic source itself, creating the corresponding energy from nowhere, and possibly activating further resonators in locations $x'' > x$.

Help Marco compute how much total energy could be harnessed at most with a carefully planned array of resonators, recalling that (i) magic is only possible in integer miles $x \in 0 \ldots L - 1$ along the ley line; (ii) at most one magic source (node or resonator) can be present in each integer mile x; (iii) a resonator becomes a source of energy only if activated according to the conditions above.

☞ Among the attachments of this task you may find a template file `magic.*` with a sample incomplete implementation.

[1] H. Tinfoil and T. Roll, *What they don't want you to know.* Conspiracy Editors, 1666. ISBN 667-3-71-698969-0.

Input

The first line contains integers N and L. The second line contains N integers X_i. The third line contains N integers E_i.

Output

You need to write a single line with L integers from 0 to 9: the energies produced in each location by nodes or resonators in your plan.

Constraints

- $1 \leq N \leq L \leq 200$.
- $0 \leq X_i < L$ for each $i = 0 \ldots N-1$.
- $1 \leq E_i \leq 9$ for each $i = 0 \ldots N-1$.
- The values X_i are all distinct, in increasing order, and $X_0 = 0$.
- The optimal plan of resonators does not need to be unique and you can return any one of them (and you can also get a partial score for a sub-optimal plan).

Scoring

Your program will be tested against several test cases grouped in subtasks. Your score on a subtask will be equal to the minimum score for one of its testcases multiplied by the value of the subtask. Your score on a test case will be zero if the output does not represent a valid resonator plan. If the output is valid, let T_{out} be the sum of the energies reported, and T_{cor} be the optimal total energy. Your score on a test case will be 1 if $T_{\text{out}} = T_{\text{cor}}$, or $0.5 \cdot (T_{\text{out}}/T_{\text{cor}})^3$ otherwise.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (10 points) $N = L$.

– **Subtask 3** (40 points) $L \leq 15$.

– **Subtask 4** (30 points) $L \leq 50$.

– **Subtask 5** (20 points) No additional limitations.

Examples

input	output

`1 8`
`0`
`7`
	`7 1 1 2 3 4 5 7`

`3 8`
`0 3 5`
`2 3 1`
	`2 1 2 3 4 1 6 6`

Explanation

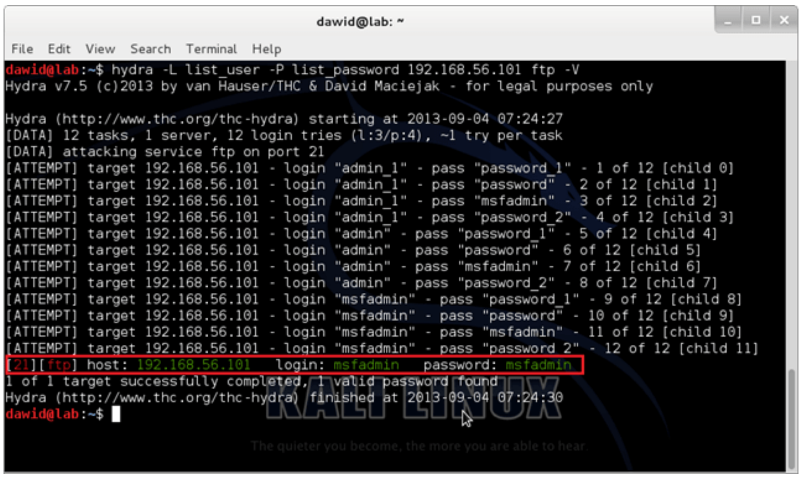
In the **first sample case**, there is a single quantum magic node of level 7, and we have space for 7 resonators after it. At position $x = 1$, a resonator can only get his energy from the node at $x = 0$, and it can do so if (and only if) its level is 1. At positions $x = 2 \dots 6$, no resonator can gather its energy from the node, since the resonator level needed would not be a multiple or divisor of the node level 7. In follows that the optimal choice for the resonators is to gather their energy from the resonator in position $x = 1$, and they can do so with levels $1 \dots 5$ respectively. Finally, at position $x = 7$ it is again possible to gather energy from the node, and that is the optimal choice.

In the **second sample case**, there are three nodes and we have space for other 5 resonators. At positions $x = 1, 2$ resonators can gather their energy from the first node, of level 2. At position $x = 3$ there is the second node, of level 3. At position $x = 4$ a resonator can again get its energy from the first node; while at position $x = 5$ the only option is to get energy from the adjacent resonator. Finally, at positions $x = 6, 7$ resonators of level 6 can get their energy from the node at $x = 0$ and the resonator at $x = 1$.

Passphrase Obfuscation (passphrase)

To better protect the server containing the future tasks of this competition, Luca is choosing a long *passphrase* to encrypt the tasks archive. Due to the current pandemic restrictions, Luca is unable to meet the rest of the team and directly communicate to them a newly chosen passphrase: he is left with no other option than sending it using a potentially insecure channel, and for this reason he must make it unrecognizable to external malicious eyes.

After a careful evaluation of potential attackers, Luca devised an infallible plan to safely communicate the passphrase: he is going to remove exactly K characters from the sequence of passphrases used by the team so far, joined together. These combined passphrases form a string of length N that everybody in the team knows; he can then only communicate the removed characters, making the new passphrase unintelligible to others while still being reconstructible by his colleagues.


dawid@lab: ~
File Edit View Search Terminal Help
KALI LINUX
The quieter you become, the more you are able to hear

```
dawid@lab:~$ hydra -L list_user -P list_password 192.168.56.101 ftp -V
Hydra v7.5 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2013-09-04 07:24:27
[DATA] 12 tasks, 1 server, 12 login tries (l:3/p:4), ~1 try per task
[DATA] attacking service ftp on port 21
[ATTEMPT] target 192.168.56.101 - login "admin_1" - pass "password_1" - 1 of 12 [child 0]
[ATTEMPT] target 192.168.56.101 - login "admin_1" - pass "password" - 2 of 12 [child 1]
[ATTEMPT] target 192.168.56.101 - login "admin_1" - pass "msfadmin" - 3 of 12 [child 2]
[ATTEMPT] target 192.168.56.101 - login "admin_1" - pass "password_2" - 4 of 12 [child 3]
[ATTEMPT] target 192.168.56.101 - login "admin" - pass "password_1" - 5 of 12 [child 4]
[ATTEMPT] target 192.168.56.101 - login "admin" - pass "password" - 6 of 12 [child 5]
[ATTEMPT] target 192.168.56.101 - login "admin" - pass "msfadmin" - 7 of 12 [child 6]
[ATTEMPT] target 192.168.56.101 - login "admin" - pass "password_2" - 8 of 12 [child 7]
[ATTEMPT] target 192.168.56.101 - login "msfadmin" - pass "password_1" - 9 of 12 [child 8]
[ATTEMPT] target 192.168.56.101 - login "msfadmin" - pass "password" - 10 of 12 [child 9]
[ATTEMPT] target 192.168.56.101 - login "msfadmin" - pass "msfadmin" - 11 of 12 [child 10]
[ATTEMPT] target 192.168.56.101 - login "msfadmin" - pass "password_2" - 12 of 12 [child 11]
[21][ftp] host: 192.168.56.101   login: msfadmin   password: msfadmin
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2013-09-04 07:24:30
dawid@lab:~$
```

Figure 1: An example of a dictionary attack that Luca wants to prevent.

Additionally, a security expert consulted by Luca suggested that the new passphrase should be as close as possible to the end of a dictionary to improve its resistance to the so-called *dictionary attacks*. More formally, this means that he should prefer a passphrase p_1 over another passphrase p_2 whenever, in the lexicographic order, p_1 **comes after** p_2. What is the best (i.e., later in the dictionary) passphrase that Luca can produce by removing K characters to the given string of passphrases?

☞ Among the attachments of this task you may find a template file `passphrase.*` with a sample incomplete implementation.

Input

The first line contains two integers N and K. The second line contains the given string of passphrases.

Output

You need to write a single line containing the best passphrase that Luca can choose.

Constraints

- $2 \leq N \leq 20\,000$.
- $1 \leq K < N$.
- The given string consists of N lowercase letters of the English alphabet.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (10 points) $K = 1$.

– **Subtask 3** (30 points) $N \leq 50$.

– **Subtask 4** (35 points) $N \leq 2000$.

– **Subtask 5** (25 points) No additional limitations.

Examples

input	output

`11 4`
`coronavirus`
	`rovirus`

`16 2`
`programmingisfun`
	`rorammingisfun`

Explanation

In the **first sample case** we obtain the most secure passphrase by deleting characters `c`, `o` (at second position), `n`, `a`. No other passphrase, after removing 4 characters, comes lexicographically after this one.

In the **second sample case** we obtain the most secure passphrase by deleting characters `p` and `g`. No other passphrase, after removing 2 characters, comes lexicographically after this one.

Police Investigation 2 (police2)

After managing to escape from the police in Terror Street, William is again hiding from them, this time in Crime Avenue. Crime Avenue is structured similarly to Terror Street, and consists of N houses numbered from 0 to $N-1$. William knows how the police works! They are searching for him going house by house: if William is not found in the i-th house, the people living there will be interrogated until they reveal to the officers that William may be hiding in house V_i. The police will then go there, and repeat the search.

Figure 1: Crime Avenue.

It's easy to prove that, no matter the hints of the inhabitants, the police will eventually reach an already visited house and therefore start looping! When the police notices that they are going in a loop, they stop and restart from a new, unvisited house. Eventually, the police will visit every house in the street.

William wants to know *how long is the longest loop* the police will stumble upon.

☞ Among the attachments of this task you may find a template file `police2.*` with a sample incomplete implementation.

Input

The first line contains the only integer N, the number of houses in the street. The second line contains N integers V_i, the number of the house the police will go to after visiting the house i.

Output

You need to write a single line with an integer: the number of houses in the longest loop.

Constraints

- $1 \leq N \leq 100\,000$.
- $0 \leq V_i < N$ for each $i = 0 \ldots N-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

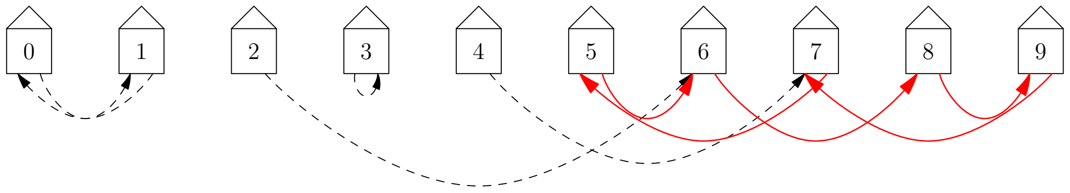
– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (10 points)	$N \leq 10$.
– **Subtask 3** (25 points)	There is only one loop.
– **Subtask 4** (25 points)	Every house is part of a loop.
– **Subtask 5** (20 points)	$N \leq 1000$.
– **Subtask 6** (20 points)	No additional limitations.

Examples

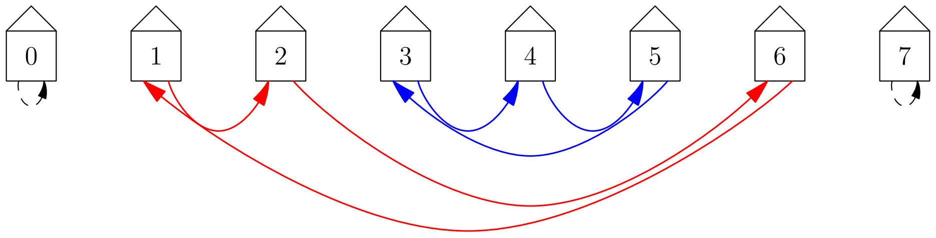
input	output
10 1 0 6 3 7 6 8 5 9 7	5
8 0 2 6 4 5 3 1 7	3

Explanation

In the **first sample case** the longest loop is 5 houses long: 5, 6, 8, 9, 7.



In the **second sample case** there are two loops 3 houses long: 1, 2, 6 and 3, 4, 5.

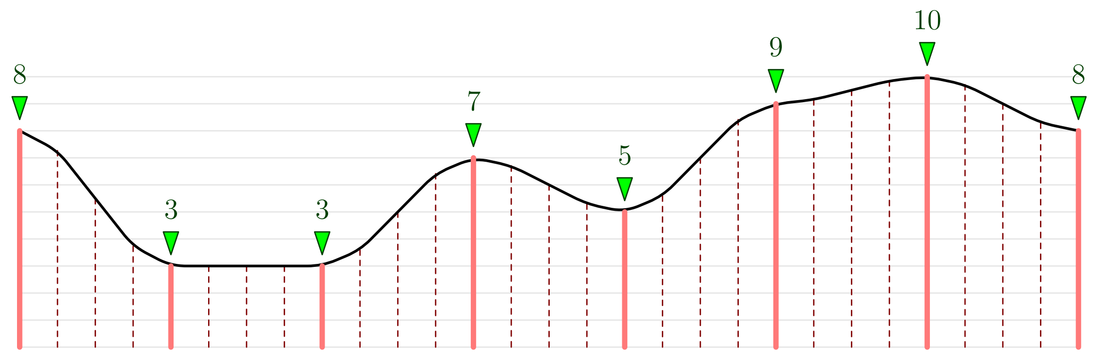


Crazy Roller Coaster (`rollercoaster2`)

William is again playing his favorite game, *RollerCoaster Typhoon*. He just loves it when family-friendly fun and natural catastrophes mix together.

In the game, there are some predefined *tracks* that can be selected by the player. A track is formed by N junctions, numbered from 1 to N, which are then automatically connected by roller coaster sections.

The i-th junction is located at a specific height H_i. For example, suppose that $H = \{8, 3, 3, 7, 5, 9, 10, 8\}$. For such junction height values, the roller coaster would look something like this:



William loves to scare as much as possible the tiny virtual people going on his roller coaster. He calls his favorite tracks *The Crazy Tracks*. We will say that a track is *crazy* if every consecutive triplet of junctions is **alternating**, that is: for every i such that $2 \leq i \leq N-1$, one of the following is true:

- $H_{i-1} < H_i > H_{i+1}$.
- $H_{i-1} > H_i < H_{i+1}$.

In the example above, there are 3 consecutive triplets of junctions that are **not alternating**:

- {$8, 3, 3$, 7, 5, 9, 10, 8}
- {8, $3, 3, 7$, 5, 9, 10, 8}
- {8, 3, 3, 7, $5, 9, 10$, 8}

This means that the track is not *crazy*. Thankfully, the game lets the player change the height of any of the junctions, but this action has a cost: William will have to pay x^2 coins to increase or decrease a junction's height by x units. What is the minimum number of coins that he will have to pay in order to make a given track crazy?

☞ Among the attachments of this task you may find a template file `rollercoaster2.*` with a sample incomplete implementation.

Input

The first line contains the only integer N. The second line contains N integers H_i.

Output

You need to write a single line with an integer: the minimum number of coins to make the track crazy.

Constraints

- $3 \leq N \leq 500$.
- $0 \leq H_i \leq 2000$ for each $i = 1 \ldots N$.
- The minimum number of coins needed is always smaller than 10^9.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (15 points)	$H_i \leq 1$ for all i.
– **Subtask 3** (35 points)	$N \leq 10$ and $H_i \leq 5$ for all i.
– **Subtask 4** (20 points)	$H_i \leq 100$.
– **Subtask 5** (30 points)	No additional limitations.

Examples

input	output

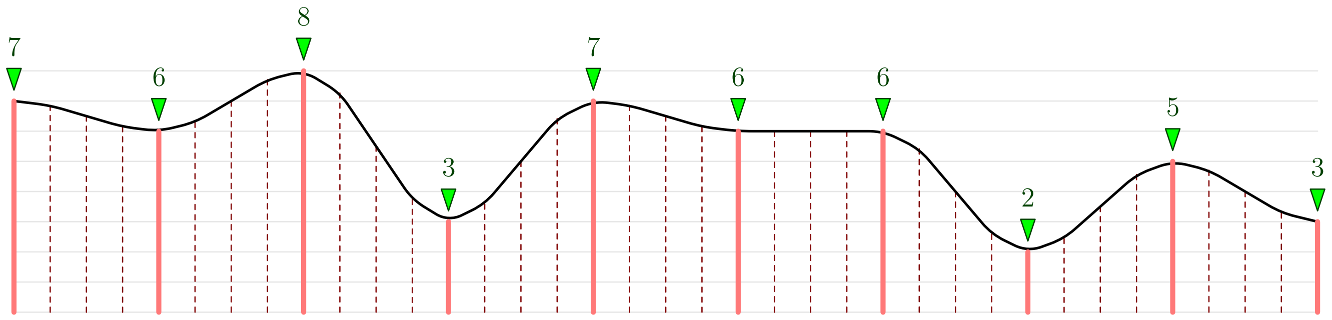
`8`
`8 3 3 7 5 9 10 8`
	`23`

`10`
`7 6 8 3 7 6 6 2 5 3`
	`1`

Explanation

The **first sample case** is the one pictured in the problem statement. It's possible to turn this track *crazy* with 23 coins: decrease $8 \longrightarrow 5$ in first position (9 coins), increase $3 \longrightarrow 6$ in second position (9 coins), decrease $10 \longrightarrow 8$ in second last position (4 coins), increase $8 \longrightarrow 9$ in last position (1 coin).

In the **second sample case**, represented as follows, it's enough to increase the rightmost 6 to 7.



7
6
8
3
7
6
6
2
5
3