Vectors, Lists and Iterators

1. Merge Trains

At a certain train station, trains arrive on two tracks – Track A and Track B – and are merged onto a single track. Each railcar has a number, and in each train the railcars are ordered by number – the front railcar has the lowest number, the back railcar has the highest number.

Given the railcar numbers on Track A and Track B, given from the back car to the front car, print the order in which railcars from Track A and Track B should be merged into the single track, then print the new train configuration, starting from the last railcar and finishing at the first railcar. Railcars are moved starting from the front to the back (i.e. you can only move the front car of a train). Each time you move a railcar from the parallel tracks A and B to the merge track, it pushes forward any railcars that are already there.

Input

- The first line of the console will contain the numbers of the railcars on Track A, from the last railcar to the first railcar
- The second line of the console will contain the numbers of the railcars on Track B, from the last railcar to the first railcar

Output

- If a railcar from Track A should be moved to the single track, print A. Otherwise, print B.
- On a separate line print the final configuration of the train (numbers separated by spaces, representing the railcar numbers from the last railcar to the first)

The "input" railcars will always be correctly ordered (i.e. will be a line of descending positive integer numbers). The input will be such that the result will never have any railcars with the same numbers.

Examples

Input	Output	Comments
11 4 2 1 5 3	AABABA 11 5 4 3 2 1	We first move from A, railcar 1, and the result becomes: **1** We again move from A, railcar 2, which pushes railcar 1 more to the right: **2 1** Now we need B (the rightmost of A is 4, the rightmost of B is 3), which again pushes the railcars in the merge: **3 2 1** We now again need A: **4 3 2 1** Back to B, that's the last railcar there: **5 4 3 2 1** Finally we move the last from A and get the result: **11 5 4 3 2 1** The sequence was AABABA

`2`
`1`
	`BA`
	`2 1`
		`Only two railcars, line B has the smaller railcar, so move that first, then move from A`

2. Brackets

Write a program which reads a single line from the console, containing brackets for a mathematical expression (only the brackets will appear in the input), and determines whether the brackets in the expression are correct. There are 3 types of brackets – **{}**, **[]** and **()**. **{}** can contain **{}**, **[]** and **()** brackets. Square brackets **[]** can contain **[]** and **()** brackets. Curved **()** brackets can contain only **()** brackets. Said simply, each type of brackets can contain the same type of brackets inside, or a “lower” type of brackets (**()** is lower than **[]** which is lower than **{}**). If a bracket of one type is opened, it needs to be closed before a bracket of another type is closed.
Print **valid** if the brackets in the expression are valid and **invalid** if they are not.

Examples

Input	Output
`[()]{}{[()()]()}`	`valid`
`[(])`	`invalid`
`([])`	`invalid`
`()[[[()]]]{[()]}`	`valid`

3. Pipes

The company "Water You Waiting For" provides hot water to houses in a village through a series of underground pipes, each house having its own pipe for hot water. But the pipes corrode over periods of years and need to be replaced. Since the company wants to save money, it only does checkups of the pipes once a year, always on the same date (April 1st). Pipes are replaced only during checkups.

But the company wants to save even more money by skipping checkups when they are not necessary. Since the pipes corrode at a constant speed, it is enough to have 2 measurements of a pipe’s strength and calculate how much time remains before it needs to be replaced. This needs to be rounded-down to years, since the repair can only happen during a checkup – so if a pipe is going to break after 2 years and 11 months, the company needs to replace it after 2 years, because otherwise the pipe will be broken and leak water for 1 month until the 3rd year checkup.

Write a program which, given two arrays of consecutive strength measurements (measured when the pipes were installed and the year after that) of all the pipes, calculates the lifetimes the pipes (i.e. how many years after it was installed it needs to be replaced).

Input

The first line of the standard input will contain a single positive integer number **N** – the number of pipes. The second line of the standard input will contain an array of **N** positive integer numbers, separated by single spaces, representing the measurements of each pipe, made during this year’s checkup – let’s call them **`checkup`**.

The second line of the standard input is analogous to the first but contains the measurements from last year – when the pipes were installed – let’s call it **`installation`**.

So, the strength of pipe **`i`** last year was **`installation[i]`** and this year it is **`checkup[i]`**.

Output

A single line, containing integers separated by single spaces, representing the years remaining until the corresponding pipe described in the input arrays must be replaced (counting from **`installation`**). If we call this array **`lifetimes`**, then pipe **`i`** has to be replaced **`lifetimes[i]`** years after **`installation[i]`** was measured.

Restrictions

`0 < installation[i] <= 1000000000`; **`0 <= checkup[i] < installation[i]`**;

N will be a positive number less than or equal to **`500`**.

The total running time of your program should be no more than **`0.1s`**

The total memory allowed for use by your program is **5MB**

Examples

Input	Output	Comments

3
3 2 2
5 4 3
	2 2 3	Pipe 0 has suffered 5 – 3 = 2 damage – next year it will have 3 – 2 = 1 strength remaining and needs to be replaced, otherwise it will fail ~1.5 years from now (so it has a lifetime of 2 years counting from installation). Pipe 1 will break exactly 1 year from now, which means 2 years since it was installed.

5
1 1 1 1 11
2 3 4 5 12
	2 1 1 1 12	Pipe 0 and Pipe 4 get 1 damage per year (i.e. their lifetimes in years are equal to their initial strength), but the others lose more than half their strength per year – they need to be replaced each year

4. Bus

Captain Grant needs your help. He's currently on leave, but needs to get back to his ship soon. To do that, he needs to catch a bus to the train station, and from there take a train to the naval base. But captain Grant hates waiting – he has a certain train he has to catch, but can pick from several busses, and he wants to pick a bus which arrives as close to the train departure as possible.

The transport company, which operates the busses to the station and the trains at the station, has a list of bus arrival times at the station, as well as information on the train departure time. Of course, since the company works with the military, the arrival times and the train departure time are in military time format – **4**-digit numbers, the first two digits represent the hours (**`00`** to **23**), the next two digits represent the minutes (**`00`** to **59**). For example, two o'clock in the morning is **`0200`**, twenty minutes past four in the afternoon is **`1620`**, two minutes to midnight (*the time, not the Iron Maiden song*) is **`2358`**, etc.

Write a program which, given a list of bus arrival times and a train departure time, in military time format, finds the minimum amount of time – in minutes – between a bus arrival and the train departure (i.e. the time Grant would have to wait if he picks the "best" bus) and prints the position of the bus in the bus arrival times list.

Note that **`0`** waits is possible, but negative wait times aren't possible.

Hint: you can convert the military time format numbers into minutes (minutes elapsed since midnight) before calculating the time between an arrival time and the train time

Input

The first line of the standard input will contain the number **N** – the number of bus arrival times.

The first line of the standard input will contain a sequence of bus arrival times, in military time format, separated by single spaces.

The second line of the standard input will contain the train departure time, in military time format.

Output

A single line containing a single non-negative integer – the number/position of the bus in the input sequence of bus arrival times, for which the wait time is minimal.

Restrictions

N will be at least **`1`** and at most **`20`**.

The input data will be such that there will always be a valid (non-negative) minimum wait time. There will always be a bus that arrives before the train leaves.

The total running time of your program should be no more than **`0.1s`**

The total memory allowed for use by your program is **`5MB`**

Example I/O

Input	Output	Comments

`4`
`2013 0130 0004 0012`
`2122`
	`1`	`The best bus is the one arriving at 2013 (20:13) – i.e. the` **`first`** `bus in the list (NOTE: the answer is the position of the bus in the input, not in their chronological order)`

`3`
`1205 1241 1708`
`1241`
	`2`	`The train leaves at 1241 and the` **`second`** `bus arrives then (0 minutes wait, so it is the best option)`