Taxi Express

You have created your own taxi company called "Taxi Express". You want to analyze how well your taxi drivers are doing by calculating how much time they need to tend the customers.

You will receive a **list of the cutomers** (**numbers** seperated by comma and space **", "**) on the first line and **list of your taxis** (**numbers** seperated by comma and space **", "**).

Each number from the **customer** list represents how much **time** it takes to **drive the customer** to his/her **destination.**

Each number from the **taxis** list represents how much **time** they can drive, before they need to refill their tanks.

Keep track of the **total time** passed to drive all the customers to their destinations (values of all customers).

Each time you tend customers you should put the **first customer** in the **last taxi** until there are **no customers left**.

- If the **taxi can drive** the customer to his/her destination, **he does** and you must add the **time** passed to drive the customer to his/her destination (the **value of the current customer**) to the **total time**. **Remove both** the customer and the taxi.
- If the **taxi cannot drive** the customer to his/her destination, **leave** the **customer** at the **beginning** of the queue and **remove the taxi**.

At the end if you have **successfully** driven all the customers to their destinations, print

```
All customers were driven to their destinations
Total time: {total_time} minutes
```

Otherwise, if you **ran out** of **taxis** and there are still some **customers left** print

```
Not all customers were driven to their destinations
Customers left: {left_customers joined by ", "}
```

Input

- On the **first line** you are given the **customers** – **numbers** seperated by comma and space **", "**
- On the **second line** you are given the **taxis** – **numbers** seperated by comma and space **", "**

Output

- Print the output as described above

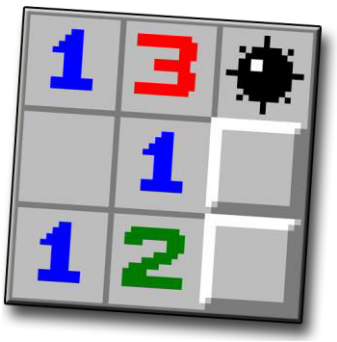
Constraints

- You will always have **at least one customer** and **at least one taxi**

Examples

Input	Output
4, 6, 8, 5, 1 1, 9, 15, 10, 6	All customers were driven to their destinations Total time: 24 minutes
10, 5, 8, 9 2, 4, 5, 8	Not all customers were driven to their destinations Customers left: 10, 5, 8, 9
2, 8, 4, 3, 11, 7 10, 15, 4, 6, 3, 10, 2, 1	All customers were driven to their destinations Total time: 35 minutes

Minesweeper Generator



Everybody remembers the old mines game. Now it is time to create your own.

You will be given an integer **n** for the **size** of the mines field with **square** shape and another one for the number of bombs that you have to place in the field. On the next **n** lines, you will receive the **position for each bomb**. Your task is to **create the game field** placing the **bombs** at the correct positions and mark them with "*", and **calculate the numbers** in each cell of the field. Each **cell** represents a **number** of all **bombs** directly near it (**up, down, left, right and the 4 diagonals**).

*	2	*
2	3	2
1	*	1

*	2	*
2	3	2
1	*	1

*	2	*
2	3	2
1	*	1

Input

- On the first line, you are given the integer **n** – the size of the **square** matrix.
- On the second line – the **number** of the **bombs**.
- The **next n lines** holds the position of each **bomb**.

Output

- Print the matrix you've created.

Constraints

- The size of the **square** matrix will be between **[2…15].**

Examples

Input	Output

4 4 (0, 3) (1, 1) (2, 2) (3, 0)	1 1 2 * 1 * 3 2 2 3 * 1 * 2 1 1
5 3 (1, 1) (2, 4) (4, 1)	1 1 1 0 0 1 * 1 1 1 1 1 1 1 * 1 1 1 1 1 1 * 1 0 0

Numbers search

Write a function called **numbers_searching** which receives a **different amount** of parameters. All parameters will be integer **numbers** forming a sequence of consecutive numbers. Your task is to **find an unknown amount** of **duplicates** from **the given sequence** and **a missing value**, such that **all the duplicate values** and **the missing value** are **between the smallest and the biggest** received number.

The function should **return** a list with the **last missing number** as a first argument and a **sorted list,** containing the duplicates found, in **ascending** order.

For example: if we have the following numbers: 1, 2, 4, 2, 5, 4 will return 3 as missing number and 2, 4 as duplicate numbers in the following format: [3, [2, 4]]

Input

- There will be **no input**
- **Parameters** will be passed to your function

Output

- The function should **return a list** in the following format: **[missing number, [duplicate_numbers separated with comma and space]]**

Constraints

- The missing number will always be between the smallest and the biggest received number

Examples

Input	Output
`print(numbers_searching(1, 2, 4, 2, 5, 4))`	`[3, [2, 4]]`
`print(numbers_searching(5, 5, 9, 10, 7, 8, 7, 9))`	`[6, [5, 7, 9]]`
`print(numbers_searching(50, 50, 47, 47, 48, 45, 49, 44, 47, 45, 44, 44, 48, 44, 48))`	`[46, [44, 45, 47, 48, 50]]`