Question - 1
Find the First Repeated Word in a Sentence

A sentence is minimally defined as a word or group of words. We consider a word to be a sequence of letters (i.e.: *a-zA-Z*) delimited by a space or non-letter character. A *repeated word* is a case-sensitive word that appears more than once in a sentence (e.g.: *'had' ≠ 'Had'*). Because substrings of a word are *not* delimited, they are *not* considered to be words.

Complete the *firstRepeatedWord* function in your editor. It has *1* parameter:
1. A string, *s*, describing a sentence.

It must return a string containing the first *repeated word* in s.

Input Format
The locked stub code in your editor reads a single string, *s*, from stdin and passes it to your function.

Constraints
- $0 < |s| < 1024$
- The following characters are delimiters between words: space, tab, comma (`,`), colon (`:`), semicolon (`;`), dash (`-`), and period (`.`).
- It is guaranteed that each sentence *s* contains one or more repeated words.
- Each word is separated by one or more delimiters.

Output Format
Your function must return the first repeated word in sentence *s*. This will be printed to stdout by the locked stub code in your editor.

Sample Input 1
```
He had had quite enough of this nonsense.
```

Sample Output 1
```
had
```

Explanation
Sample Case 1: 'had' is the first (and only) word to appear twice in the sentence.

Question - 2
Can You Sort?

An array of integers, *arr*, of size *N* is defined as $\{a_0, a_1, ..., a_{N-1}\}$.

Complete the *customSort* function declared in your editor. It must take *arr* as a parameter, sort its elements in order of ascending frequency, and then print each element of the sorted array as a new line of output. If *2* or more elements have the same frequency, this subset of elements should be sorted in non-decreasing order.

The locked stub code in the editor handles reading input from stdin, assembling it into an array of integers (*arr*), and calling the *sort* function. The first line of input contains an integer, N (the number of elements). Each line i of the N subsequent lines describes array element *arr[i]*.

Constraints

- $1 \le N \le 2 \times 10^5$
- $1 \le a_i \le 10^6$

Output Format

Your *customSort* function should print the sorted (in order of *non-decreasing* frequency) elements of array *arr*. If *2* or more elements have the same frequency, this subset of elements should be sorted in non-decreasing order. Each element must be printed on a new line.

Sample Input 1

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

Sample Output 1

```
1
3
4
2
2
```

Sample Input 2

```
10
8
5
5
5
5
1
1
1
4
4
```

Sample Output 2

```
8
4
4
1
1
1
5
5
5
5
```

Explanation

Sample Case 1

$N = 5$, $arr = \{3, 1, 2, 2, 4\}$

First, we separate our numbers by *frequency.*

The subset of numbers having frequency 1 is $\{3, 1, 4\}$.

The subset of numbers having frequency 2 is $\{2, 2\}$.

Our partially sorted data (with respect to and in ascending order of frequency) can be expressed as $\{\{3, 1, 4\}, \{2, 2\}\}$.

Then we sort each subset of elements having the same

frequency in non-decreasing order, resulting in *{{1, 3, 4}, {2, 2}}*.
Sample Case 2
N = 10, arr = {8, 5, 5, 5, 5, 1, 1, 1, 4, 4}
First, we separate our numbers by *frequency.*
The subset of numbers having frequency *1* is *{8}*.
The subset of numbers having frequency *2* is *{4}*.
The subset of numbers having frequency *3* is *{1}*.
The subset of numbers having frequency *4* is *{5}*.
Our partially sorted data (with respect to and in ascending order of frequency) can be expressed as *{{8},{4, 4},{1, 1, 1},{5, 5, 5, 5}}*.

Question - 3
ReplaceWithPlus

Given a string and a non-empty word string, return a version of the original String where all chars have been replaced by pluses ("+"), except for appearances of the word string which are preserved unchanged.

hint:String buffer

Examples:

```
plusOut("12xy34", "xy") → "++xy++"
plusOut("12xy34", "1") → "1+++++"
plusOut("12xy34xyabcxy", "xy") → "++xy++xy+++xy"
```

Question - 4
Modify Prices

Michael is a shop owner who keeps n list, *L*, of the name and sale price for each item in inventory. The store employees record the name and sale price of every item sold. Michael suspects his manager, Alex, of embezzling money and modifying the sale prices of some of the items. Write a program that finds the number of times Alex recorded an incorrect sale price.

Complete the *verifyItems* function provided in your editor so that it returns the number of incorrect sale prices recorded by Alex. It has *4* parameters:

1. *origItems*: An array of strings, where each element is an item name.
2. *origPrices*: An array of floating point numbers, where each element contains the original (correct) price of the item in the corresponding index of *origItems*.
3. *items*: An array of strings containing the name of the items with sales recorded by Alex.
4. *prices*: An array of floating point numbers, where each element contains the sale price recorded by Alex for the item in the corresponding index of *items*.

Note: Where required by the language, there may also be *2* additional integer parameters for passing the array sizes (*N* and *M*).

Input Format
The locked stub code in your editor processes the following inputs and passes the necessary arguments to the *verifyItems* function:
The first line contains an integer, *N*, the size of the *origItems* array. Each line *i* (where *0 ≤ i < N*) of the *N* subsequent lines describes element *i* in *origItems*. The next line contains an integer, *N*, the size of the *origPrices* array. Each line *i* of the *N*

subsequent lines describes element *i* in *origPrices*. The next line contains an integer, *M*, the size of the *items* array. Each line *j* (where $0 \le j < M$) of the *M* subsequent lines describes element *j* in *items*. The next line contains an integer, *M*, the size of the *prices* array. Each line *j* of the *M* subsequent lines contains the price of element *j* in *items*.

Constraints

- $1 \le N \le 10^5$
- $1 \le M \le N$
- $1.00 \le origPrices_i, prices_j \le 100000.00$, where $0 \le i < N$, and $0 \le j < M$

Output Format

Return the number of items whose sale prices were incorrectly recorded by Alex.

Sample input 0

```
4
rice
sugar
wheat
cheese
4
16.89
56.92
20.89
345.99
2
rice
cheese
2
18.99
400.89
```

Sample Output 0

```
2
```

Sample Input 1

```
3
chocolate
cheese
tomato
3
15.00
300.90
23.44
3
cheese
tomato
chocolate
3
300.90
23.44
10.00
```

Sample Output 1

```
1
```

Explanation

Sample Case 0: N = 4 , M = 2
origItems = {"rice", "sugar", "wheat"," cheese"}
origPrices = {16.89, 56.92, 20.89, 345.99}
items = {"rice", "cheese"}
prices = {18.99, 400.89}
The prices for *rice* and *cheese* do not match the original price

list, so we return *2* (the number of incorrectly recorded sale prices).

Sample Case 1: N = 3, M = 3
origItems = {"chocolate", "cheese", "tomato"}
origPrices = {15, 300.90, 23.44}
items = {"chocolate", "cheese", "tomato"}
prices = {15, 300.90, 10}
The price for *tomato* does not match the original price list, so we return *1* (the number of incorrectly recorded sale prices).

Question - 5
lastNChar

Given a string and an int n, return a string made of n repetitions of the last n characters of the string. You may assume that n is between 0 and the length of the string, inclusive.

repeatEnd("Hello", 3) → "llollollo"
repeatEnd("Hello", 2) → "lolo"
repeatEnd("Hello", 1) → "o"

Question - 6
Employees 2

You are given two tables: Employees and Employee_pan. Column '**id**' denotes the ID of the employee in both tables.
Print the UIN and NAME of all employees who are younger than 25.

Table: **EMPLOYEES**

```
+-------------+--------------+
| Field       | Type         |
+-------------+--------------+
| ID (PK)     | int          |
| Name        | char(20)     |
| Users       | int          |
| Age         | int          |
| Address     | char(25)     |
| Salary      | decimal(18,2)|
+-------------+--------------+
```

Table: **EMPLOYEE_PAN**

```
+-------------+--------------+
| Field       | Type         |
+-------------+--------------+
| ID (PK)     | int          |
| UIN         | int          |
+-------------+--------------+
```

Question - 7
Customers 3

You are given a Customers table.

Print the entire row information of those customers whose combinedName length is shorter than 40 characters and also order them by the combinedName length in ascending order. (In case of a tie in combinedName length, use lexicographic ordering by customerName)

Note: combinedName = contactFirstName +customerName + contactLastName

Table: **CUSTOMERS**

```
+---------------------------+--------------+
| Field                     | Type         |
+---------------------------+--------------+
| customerNumber (PK)       | int          |
| customerName              | char(50)     |
| contactLastName           | char(50)     |
| contactFirstName          | char(50)     |
| addressLine1              | char(50)     |
| addressLine2              | char(50)     |
| city                      | char(50)     |
| state                     | char(50)     |
| postalCode                | char(50)     |
| country                   | char(50)     |
| salesRepEmployeeNumber    | int          |
|  creditLimit              | decimal      |
+---------------------------+--------------+
```

Question - 8
Gaining Hackos

You are given a table *Hacker_details*, which has the schema described below.

Hacker_details
H_id is the id of the coder (hacker), *Name* is the name of the hacker, *Time* is the time in months since they started coding and *Hackos* represent the points that the coder gains per month.

Column	*Type*
H_id	Integer
Name	String
Time	Integer
Hackos	Integer

Write a query to print the names of all the hackers who have gained more than 2000 hackos in less than 10 months. Print the output in ascending order of H_id.

Question - 9
Employees 3

You are given two tables: Employees and Employee_pan. Column '**id**' denotes the ID of the employee in both tables.
Print the employee NAME and UIN in each line. If no UIN is present print NULL in its place.

Table: **EMPLOYEES**

```
+-------------+--------------+
| Field       | Type         |
+-------------+--------------+
| ID (PK)     | int          |
| Name        | char(20)     |
| Users       | int          |
| Age         | int          |
| Address     | char(25)     |
| Salary      | decimal(18,2)|
+-------------+--------------+
```

Table: **EMPLOYEE_PAN**

```
+-------------+--------------+
| Field       | Type         |
+-------------+--------------+
| ID (PK)     | int          |
| UIN         | int          |
+-------------+--------------+
```

Question - 10
Orders 3

You are given the table *orders,* which has the schema described below.

Find the 5 oldest (earliest) orders which are not yet shipped.

Table: **ORDERS**

```
+-----------------+--------------+
| Field           | Type         |
+-----------------+--------------+
| orderNumber (PK)| int          |
| orderDate       | date         |
| requiredDate    | date         |
| shippedDate     | date         |
| status          | char(15)     |
| comments        | char(200)    |
| customerNumber  | int          |
+-----------------+--------------+
```

Note: If shipped status = 'Shipped'

Question - 11
Students and Departments

A university uses *2* data tables, *Students* and *Departments*, to store data about its students and the departments associated with each major. Write a query to print the respective *department name* and *number of students* majoring in each department for *all* departments in the *Departments* table (even ones with no current students). Sort your results by descending *number of students*; if two or more departments have same number of students, then sort those departments alphabetically by *department name*.

Input Format

The *Students* and *Departments* tables are described as follows:

Students

Column Name	*Type*
STUDENT_ID	Integer
STUDENT_NAME	String
GENDER	Character
DEPT_ID	Integer

where *STUDENT_ID* is the student's ID number, *STUDENT_NAME* is the student's name, *GENDER* is their gender, and *DEPT_ID* is the department ID associated with their declared major.

Departments

Column Name	*Type*
DEPT_ID	Integer
DEPT_NAME	String

where *DEPT_ID* is the department's ID number and *DEPT_NAME* is the department name.