# Exercises: Array and List Algorithms

Problems for exercises and homework for the [“Programming Fundamentals Extended” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

You can check your solutions here: <https://judge.softuni.bg/Contests/427/>

## Shoot List Elements

Until you receive the command "stop", you will either receive an **integer**, or the command bang on every line.

If you receive an integer, place it at the **front** of the list. If, however, you receive the command bang, **remove** the first element whose value is **smaller** than the **average** of the elements of the list, **print** "shot {element}" and then **decrement** every element in the list by **1**.

If you receive "bang" and there are no elements left in the list, print "nobody left to shoot! last one was {lastRemovedInt}" and end the program. When you finally do receive the command "end", either **print** "survivors: {elements, separated by space}" if there are any, or "you shot them all. last one was {lastRemovedInt}".

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** |  |
| 3  4  5  6  2  bang  bang  bang  bang  bang  bang | shot 2  shot 3  shot 1  shot 2  shot 2  nobody left to shoot! last one was 2 | Insert 3 at front 🡺 3  Insert 4 at front 🡺 4 3  Insert 5 at front 🡺 5 4 3  Insert 6 at front 🡺 6 5 4 3  Insert 2 at front 🡺 2 6 5 4 3  bang 🡺 average: 4 🡺 remove 2  after removal 🡺 6 5 4 3  decrement everything 🡺 5 4 3 2  bang 🡺 average: 3.5 🡺 remove 3  after removal 🡺 5 4 2  decrement everything 🡺 4 3 1  bang 🡺 average: 2.66 🡺 remove 1  after removal 🡺 4 3  decrement everything 🡺 3 2  bang 🡺 average: 2.5 🡺 remove 2  after removal 🡺 3  decrement everything 🡺 2  bang 🡺 last element 🡺 remove 2 |
| 56  12  33  915  bang  bang  bang  bang  stop | shot 33  shot 11  shot 54  shot 912  you shot them all. last one was 912 | Insert 56 at front 🡺 56  Insert 12 at front 🡺 12 56  Insert 33 at front 🡺 33 12 56  Insert 915 at front 🡺 915 33 12 56  bang 🡺 average: 254 🡺 remove 33  after removal 🡺 915 12 56  decrement everything 🡺 914 11 55  bang 🡺 average: 326.66 🡺 remove 11  after removal 🡺 914 55  decrement everything 🡺 913 54  bang 🡺 average: 483.5 🡺 remove 54  after removal 🡺 913  decrement everything 🡺 912  bang 🡺 last element 🡺 remove 912  stop 🡺 nothing left in list |
| 99  150  3450  819  bang  bang  stop | shot 819  shot 149  survivors: 3448 97 | Insert 99 at front 🡺 99  Insert 150 at front 🡺 150 99  Insert 3450 at front 🡺 3450 150 99  Insert 819 at front 🡺 819 3450 150 99  bang 🡺 average: 1129.5 🡺 remove 819  after removal 🡺 3450 150 99  decrement everything 🡺 3449 149 98  bang 🡺 average: 1232 🡺 remove 149  after removal 🡺 3449 98  decrement everything 🡺 3448 97  stop 🡺 survivors: 3448 97 |

## Average Character Delimiter

You will receive an array of **strings** as input. Your task is to find the **average character** within every string in the array. As in, take every character’s **ASCII** code, **sum** it and **divide** the result by the **sum** of the **count** of all the letters in the array.

**Example**:

a b ab abc abc 🡺 a, b, a, b a, b, c, a, b, c 🡺 sum: 978 🡺 Divide it by 10: 97.8 🡺 **97**

After you find the average sum, convert it to its **ASCII** character representation, **convert** it to **uppercase** and **print** the **original array**, **delimited** by that character.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| a b ab abc abc | aAbAabAabcAabc | Sum of a+b+a+b+a+b+c+a+b+c: 978  Divided by 10: 97.8 🡺97 🡺a  To uppercase: A  original array, delimited by a:  aAbAabAabcAabc |
| gosho pesho 1 2 3 | gosho\_pesho\_1\_2\_3 | Sum of letters: 1237  Divided by 13: 95 🡺 \_  Original array, delimited by \_:  gosho\_pesho\_1\_2\_3 |
| SOFTWARE University | SOFTWARE^University | Sum of letters: 1709  Divided by 18: 94 🡺 ^  Original array, delimited by ^:  SOFTWARE^University |

## Sort Array of Strings

You are given an array of strings (one line, space-delimited). Sort the array using the **Bubble sort** or **Insertion sort** algorithms. Instead of comparing if one string is smaller than the other (which you can’t do), you can use the string.CompareTo(str) method. Once the array is sorted, **print** it on the console, separating the elements by spaces.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| abc abc test 123 123 test | 123 123 abc abc test test |
| 8 6 7 2 pesho 9 | 2 6 7 8 9 pesho |
| softuni SOFTUNI software university | SOFTUNI softuni software university |

## Array Histogram

You will be given a **string array** on the console (single line, **space-separated**). Your task is to make statistics about the elements of the array. Find out **how many times** each word **occurs** in the array. After which, **sort** the result by the **count** of occurrences in **descending** order and print statistics about every word in the following format:

{word} -> {count} times ({percentage:F2}%)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| hello hello pesho hello four two gosho five pesho six gosho | hello -> 3 times (27.27%)  pesho -> 2 times (18.18%)  gosho -> 2 times (18.18%)  four -> 1 times (9.09%)  two -> 1 times (9.09%)  five -> 1 times (9.09%)  six -> 1 times (9.09%) |
| get the get request from the get url of get dot com | get -> 4 times (33.33%)  the -> 2 times (16.67%)  request -> 1 times (8.33%)  from -> 1 times (8.33%)  url -> 1 times (8.33%)  of -> 1 times (8.33%)  dot -> 1 times (8.33%)  com -> 1 times (8.33%) |
| trep trep trep trep trepni trepni trepni | trep -> 4 times (57.14%)  trepni -> 3 times (42.86%) |

### Hints

You can solve the problem in the following way:

* Make two lists: one for the **words** and one for their **occurrences count**.
* Iterate over every element in the initial array:
  + If it’s **not** contained in the **words** list, **add it**, and **add 1** to the **occurrences count** list.
  + If it **is** contained in the **words** list, increment the number at the **occurrences count** list (which is placed on the same index as the word itself in the **words** list)
* After you iterate over the entire initial array, you should have the **unique words** in the **words list** and their **occurrence count** in the **occurrences count** list.
* **Sort** the **words list** and **occurrence count list** at the **same time** by **modifying** a standard **sorting algorithm**:
  + First – reverse the swapping logic in the algorithm, so that the elements are sorted in **descending order**.
  + Second – instead of swapping just the elements in the **occurrences count list**, swap the elements in the **words list** at the **same time**, ensuring that when you move one element at one index in one list, the other element in the other list will have the **same index after it’s swapped**.
* After both lists are sorted at the same time, **iterate** over **either** of them and print the elements at each index in the aforementioned format. Since you’re sure a word in one list will **always** have its occurrences at the **same index** of the other list, you can iterate over either list.

## Decode Radio Frequencies

You are given an array of **doubles** (one line, space-separated). Your task is to **convert** the double values to their **character representations** and insert them into a list like so:

* Example: 83.105
* Extract the left part of the number and convert it to its ASCII code representation: 83 🡺 S
* Extract the right part of the number and convert I to its ASCII code representation: 105 🡺 i
* Insert the left character at the position equal to the index of the double number in the original array. 🡺 Index of double: 0 🡺 insert S at index 0
* Insert the right character at the position equal to the index of the double number in the original array, **counted in reverse**: index of the double: 0 🡺 insert “i” at index 0, counted **in reverse** (so, the end of the list)

Repeat the aforementioned algorithm for each element of the double array, until you run out of elements. When you do, print the list with **no delimiter**.

If any of the parts of the double number are 0 (such as 86.0 or 0.97), **ignore** them and do not insert them anywhere.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 83.105 111.110 102.85 116.0 | SoftUni | 83.105:  Left part 🡺 83 🡺 S, right part 🡺105 🡺i  Index of 83.105 🡺 0  Insert S at index 0 🡺S  Insert I at index 0, backwards 🡺S i  111.110:  Left part 🡺 111 🡺 o, right part 🡺110 🡺n  Index of 111.110 🡺 1  Insert o at index 1 🡺 S o i  Insert I at index 1, backwards 🡺S o n i  102.85:  Left part 🡺 102 🡺 f, right part 🡺85 🡺U  Index of 102.85 🡺 2  Insert o at index 2 🡺 S o f n i  Insert I at index 2, backwards 🡺S o f t n i  116.0:  Left part 🡺 116 🡺 U, right part 🡺0 🡺ignore  Index of 116.0 🡺 3  Insert U at index 3 🡺 S o f t U n i |
| 66.33 105.97 114.0 0.0 | Bira! | 66.33:  Left part 🡺 66 🡺 B, right part 🡺 33 🡺!  Index of 66.33 🡺 0  Insert B at index 0 🡺B  Insert ! at index 0, backwards 🡺B !  105.97:  Left part 🡺 105🡺 i, right part 🡺 97 🡺a  Index of 105.97 🡺 1  Insert i at index 1 🡺B i !  Insert a at index 1, backwards 🡺B i a !  114.0:  Left part 🡺 114 🡺 r, right part 🡺0 🡺ignore  Index of 114.0 🡺 2  Insert r at index 2 🡺B i r a !  0.0:  Left part 🡺 0 🡺 ignore, right part 🡺 0 🡺ignore |
| 110.115 111.113 116.101 32.114 114.102 97.32 100.111 105.0 | not radio freqs | The space character has an ascii code of 32 |
| 72.46 101.100 114.105 101.107 39.32 115.44 32.117 108.111 111.121 111.32 107.116 105.97 110.32 103.0 | Here's looking at you, kid. | ' has an ascii character code of 39 and , has a character code of 44. The period has an ascii code of 46. |

## Batteries

You are in a battery manufacturing plant. Your task is to stress test the latest batch of batteries for longevity.

You will receive an array of **doubles** on the console (first line, space-separated), indicating the **capacities** of the different batteries in the batch (in mAh). On the next line, you will receive the **usage per hour** for each battery as an array of **doubles** (second line, space-separated).

Next, you will receive the amount of hours you have to stress test each battery for (as an **integer**). Each of the batteries drains by its capacity until either the testing hours are over, or the battery dies (reaches 0 capacity).

Print the status of all the batteries at the end of the testing session (in percentage). If any batteries die, along with the percentage, print **how many hours** it took. The **capacity** and **percentage** are **rounded** to the 2nd decimal point.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3000 3600 800.56 6000  350 400 200 600  8 | Battery 1: 200.00 mAh (6.67)%  Battery 2: 400.00 mAh (11.11)%  Battery 3: dead (lasted 5 hours)  Battery 4: 1200.00 mAh (20.00)% |
| 1200 9128.22 110 3129.11  120 400 200 600  9 | Battery 1: 120.00 mAh (10.00)%  Battery 2: 5528.22 mAh (60.56)%  Battery 3: dead (lasted 1 hours)  Battery 4: dead (lasted 6 hours) |
| 81279 9128.22 110 99812  1500 400 200 1372.40  72 | Battery 1: dead (lasted 55 hours)  Battery 2: dead (lasted 23 hours)  Battery 3: dead (lasted 1 hours)  Battery 4: 999.20 mAh (1.00)% |