# Exercises: Arrays Advanced

Problems for exercises and homework for the [“Technology Fundamentals” course @ SoftUni](https://softuni.bg/trainings/2056/technology-fundamental-september-2018#lesson-9620).

You can check your solutions here: [Arrays-Advanced-Exercise](https://judge.softuni.bg/Contests/1299/Arrays-Advanced-Exercise)

## Train

You will be given an **array of strings**. The first element will be **list of wagons** (numbers). Each number represents **the number of passengers that are currently in each wagon**. The next element will be **the max capacity of each wagon** (single number). The next elements will be one of the following:

* **Add** {**passengers**} – add a wagon to the end with the given number of passengers.
* {**passengers**} - find an existing wagon to **fit all the passengers** (**starting from the first wagon**)

At the end **print the final state** of the train (all the wagons separated by a space)

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['32 54 21 12 4 0 23',  '75',  'Add 10',  'Add 0',  '30',  '10',  '75'] | 72 54 21 12 4 75 23 10 0 |
| ['0 0 0 10 2 4',  '10',  'Add 10',  '10',  '10',  '10',  '8',  '6'] | 10 10 10 10 10 10 10 |

## Distinct Array

You will be given an **array of integers** on the first line of the input (**space-separated**). Remove all **repeating elements** from the array.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| [1, 2, 3, 4] | 1 2 3 4 | No repeating elements |
| [7, 8, 9, 7, 2, 3, 4, 1, 2] | 7 8 9 2 3 4 1 | 7 and 2 are already present in the array 🡺remove them |
| [20, 8, 12, 13, 4, 4, 8, 5] | 20 8 12 13 4 5 | 4 and 8 are already present in the array 🡺remove them |

## House Party

Write a JS Function that keeps track of guests that are going to a house party. You will be given an **array of strings**. Each string will be one of the following:

* **"{name} is going!"**
* **"{name} is not going!"**

If you receive the **first type of input**, you have to **add** the person if he/she **is not** in the list (If he/she is in the list print on the console: "**{name} is already in the list!**"). If you receive the **second type of input**, you have to **remove** the person if he/she **is** in the list (if not print: "**{name} is not in the list!**"). **At the end print all the guests.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Allie is going!',  'George is going!',  'John is not going!',  'George is not going!'] | John is not in the list!  Allie |
| ['Tom is going!',  'Annie is going!',  'Tom is going!',  'Garry is going!',  'Jerry is going!'] | Tom is already in the list!  Tom  Annie  Garry  Jerry |

## Sorting

Write a function that sorts an **array of numbers** so that the first element is the biggest one, the second is the smallest one, the third is the second biggest one, the fourth is the second smallest one and so on.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [1, 21, 3, 52, 69, 63, 31, 2, 18, 94] | 94 1 69 2 63 3 52 18 31 21 |

## Sort an Array by 2 Criteria

Write a JS function that orders a **given array of strings**, by **length** in **ascending order** as **primary criteria**, and by **alphabetical value** in **ascending order** as **second criteria**. The comparison should be **case-insensitive**.

The **input** comes as array of strings.

The **output** is the ordered array of strings.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| ["alpha", "beta", "gamma"] | beta  alpha  gamma |  | ["Isacc", "Theodor", "Jack", "Harrison", "George"] | Jack  Isacc  George  Theodor  Harrison | ['test', 'Deny', 'omen', 'Default'] | Deny  omen  test  Default |

### Hints

* An array can be sorted by passing a comparing function to the **Array.sort()** function.
* Creating a comparing function by 2 criteria can be achieved by first comparing by the **main criteria**, if the 2 items are different (the result of the compare is not 0) - return the result as the result of the comparing function. If the two items are the same by the **main criteria** (the result of the compare is 0), we need to compare by the **second criteria** and the result of that comparison is the result of the comparing function.

## Bomb Numbers

Write a JS Function that **receives two parameters: sequence of numbers** and **special bomb number** with a certain **power**. Your task is to **detonate every occurrence of the special bomb number** and according to its power **his neighbors from left and right**. Detonations are performed from left to right and all detonated numbers disappear. Finally print the **sum of the remaining elements** in the sequence.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| [1, 2, 2, 4, 2, 2, 2, 9],  [4, 2] | 12 | Special number is **4** with power 2. After detontaion we left with the sequence [1, 2, 9] with sum 12. |
| [1, 4, 4, 2, 8, 9, 1],  [9, 3] | 5 | Special number is **9** with power 3. After detontaion we left with the sequence [1, 4] with sum 5. Since the 9 has only 1 neighbour from the right we remove just it (one number instead of 3). |
| [1, 7, 7, 1, 2, 3],  [7, 1] | 6 | Detonations are performed from left to right. We could not detonate the second occurance of 7 because its already destroyed by the first occurance. The numbers [1, 2, 3] survive. Their sum is 6. |
| [1, 1, 2, 1, 1, 1, 2, 1, 1, 1],  [2, 1] | 4 | The red and yellow numbers disappear in two sequential detonations. The result is the sequence [1, 1, 1, 1]. Sum = 4. |

## Search for a Number

You will receive two **arrays** of **integers**. The second **array is** with exactly **three** **numbers**. **First** number represents the **number** of **elements** you have to **take** from the first **array** (**starting** from the **first** **one**). **Second** number represents the **number** of **elements** you have to **delete** from the numbers you took (**starting** from the **first** **one**). **Last** **number** is the **number** we search in our **collection** after the manipulations.

**Output:** how many times that **number** occurs in our array.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| [5, 2, 3, 4, 1, 6],  [5, 2, 3] | "Number 3 occurs 1 time." | First we take **5** **elements** from the array. Delete the first **2 elements**.  Then we search for the **number** **3**. |

## . \*Array Manipulator

Write a JS Function that **receives an array of integers** and **array of string commands** and **executes them over the array**. The commands are as follows:

* **add <index> <element>** – adds element at the specified index (elements right from this position inclusively are shifted to the right).
* **addMany <index><element 1> <element 2> … <element n>** – adds a set of elements at the specified index.
* **contains <element>** – prints the index of the first occurrence of the specified element (if exists) in the array or **-1** if the element is not found.
* **remove <index>** – removes the element at the specified index.
* **shift <positions>** – **shifts every element** of the array the number of positions **to the** **left** (with rotation).
  + For example, [1, 2, 3, 4, 5] -> shift 2 -> [3, 4, 5, 1, 2]
* **sumPairs** – sums the elements in the array by pairs (first + second, third + fourth, …).
  + For example, [1, 2, 4, 5, 6, 7, 8] -> [3, 9, 13, 8].
* **print** – stop receiving more commands and print the last state of the array.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [1, 2, 4, 5, 6, 7],  ['add 1 8', 'contains 1', 'contains 3', 'print'] | 0  -1  [ 1, 8, 2, 4, 5, 6, 7 ] |
| [1, 2, 3, 4, 5],  ['addMany 5 9 8 7 6 5', 'contains 15', 'remove 3', 'shift 1', 'print'] | -1  [ 2, 3, 5, 9, 8, 7, 6, 5, 1 ] |

## 9. \*Gladiator Inventory

As a gladiator, Pesho has cool Inventory. He loves to buy new equipment. You are given Pesho's inventory with all of his equipment -> strings, separated by whitespace.

You may receive the following commands:

* Buy {equipment}
* Trash {equipment}
* Repair {equipment}
* Upgrade {equipment}-{upgrade}

If you receive **Buy command**, you should **add** the equipment at last position in the inventory, but only if it isn't bought already.

If you receive **Trash command**, **delete** the equipment if it exists.

If you receive **Repair command**, you should **Repair** the equipment if it exists and place it on **last position**.

If you receive **Upgrade command**, you should check if the equipment exists and **insert** after it the upgrade in the following format: "**{equipment}:{upgrade}";**

### Input / Consrtaints

You will receive an **array of strings**. Each element of the array is a command.

* In the **first input element,** you will receive Pesho's **inventory** – sequence of equipment names, separated by space.

### Output

* As output you must print Pesho's **inventory**.

### Constraints

* The **command will always be valid.**
* The **equipment** and **Upgrade** will be strings and will contain any character, except **'-'**.
* Allowed working **time** / **memory**: **100ms** / **16MB**.

***Scroll down to see examples.***

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| ['SWORD Shield Spear',  'Buy Bag',  'Trash Shield',  'Repair Spear',  'Upgrade SWORD-Steel'  ] | SWORD SWORD:Steel Bag Spear | We receive the inventory => SWORD, Shield, Spear  We Buy Bag => SWORD, Shield, Spear, Bag  Trash Shield => SWORD, Spear, Bag  Repair Spear => SWORD, Bag, Spear  We add Upgrade => SWORD, SWORD:Steel, Bag,Spear  We print the inventory. |
| ['SWORD Shield Spear',  'Trash Bow',  'Repair Shield',  'Upgrade Helmet-V'] | SWORD Spear Shield |  |

## \*Build a Wall

Write a JS program that keeps track of the construction of a **30-foot** wall. You will be given an **array of strings** that must be parsed as **numbers**, representing the initial height of mile-long sections of the wall, in feet. Each section has its own construction crew that can **add 1** foot of height per day by using 195 cubic yards of concrete. All crews work simultaneously (see examples), meaning all sections that aren't completed (are less than 30 feet high) grow by 1 foot every day. When a section of the wall is complete, its crew is relieved.

Your program needs to keep track of how much concrete is used **daily** until the completion of the entire wall. At the end, print on a single line, separated by comma and space, the amount of concrete used each day, and on a second line, the **final cost** of the wall. One cubic yard of concrete costs 1900 pesos.

### Input

Your program will receive an **array of strings** representing **numbers as a parameter**.

### Output

Print on the console on **one line** the **amount of concrete used each day separated by comma and space**, and on a **second line**, the **final cost** of the wall.

### Constraints

* The wall may contain up to 2000 sections (2000 elements in the initial array)
* Starting height for each section is within range [0…30]

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| [21, 25, 28] | 585, 585, 390, 390, 390, 195, 195, 195, 195  5928000 pesos |

### Explanation

On the first day, all three crews work, each adding 1 foot to their section, 585 cubic yards total (3 x 195). On the second day, it's the same with the last section reaching 30 feet and its crew being relieved (marked in green while they don't work). On the third day, only two crews work, using up 390 cubic yards total. This continues for 2 more days, with the second section reaching 30 feet. In the remaining 4 days, only 1 crew works, using 195 cubic yards every day. Over the entire period, 3120 cubic yards of concrete were used, costing 5'928'000 pesos. And that was for just 3 miles, imagine 2000!

|  |  |
| --- | --- |
| **Starting** | **[21, 25, 28]** |
| **Day 1** | **[22, 26, 29]** |
| **Day 2** | **[23, 27, 30]** |
| **Day 3** | **[24, 28, 30]** |
| **Day 4** | **[25, 25, 30]** |
| **Day 5** | **[26, 30, 30]** |
| **Day 6** | **[27, 30, 30]** |
| **Day 7** | **[28, 30, 30]** |
| **Day 8** | **[29, 30, 30]** |
| **Day 9** | **[30, 30, 30]** |

***Scroll down for more examples.***

|  |  |
| --- | --- |
| **Input** | **Output** |
| [17] | **195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195, 195**  **4816500 pesos** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| [17, 22, 17, 19, 17] | **975, 975, 975, 975, 975, 975, 975, 975, 780, 780, 780, 585, 585**  **21489000 pesos** |