# Exercise: Arrays, Lists, Array and List Algorithms

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

# Anonymous Threat

The Anonymous have created a cyber hypervirus which steals data from the CIA. You, as the lead security developer in CIA, have been tasked to analyze the software of the virus and observe its actions on the data. The virus is known for his innovative and unbeleivably clever technique of merging and dividing data into partitions.

You will receive a **single input line** containing **STRINGS** separated by **spaces**.   
The strings may contain **any ASCII** character except **whitespace**.

You will then begin receiving commands in one of the following formats:

* merge {startIndex} {endIndex}
* divide {index} {partitions}

Every time you receive the merge command, you must merge all elements from the startIndex, till the endIndex. In other words, you should concatenate them.   
**Example**: {abc, def, ghi} -> merge 0 1 -> {abcdef, ghi}

If **any** of the **given indexes** is **out of the array**, you must take **ONLY** the **range** that is **INSIDE** the **array** and **merge** it.

Every time you receive the divide command, you must **DIVIDE** the **element** at the **given index**, into **several small substrings** with **equal length**. The **count** of the **substrings** should be **equal** to the **given partitions**.

**Example**: {abcdef, ghi, jkl} -> divide 0 3 -> {ab, cd, ef, ghi, jkl}

If the string **CANNOT** be **exactly** **divided** into the **given partitions**, **make all partitions** **except** the **LAST** with **EQUAL LENGTHS**, and make the **LAST one** – **the** **LONGEST**.

**Example**: {abcd, efgh, ijkl} -> divide 0 3 -> {a, b, cd, efgh, ijkl}

The **input ends** when you receive the command “3:1”. At that point you must print the **resulting elements**, **joined** by a **space**.

### Input

* The **first input line** will contain the **array** of **data**.
* On the **next several input** lines you will **receive commands** in the **format specified above**.
* The **input ends** when you receive the command “3:1”.

### Output

* As output you must print a single line containing the elements of the array, **joined** by a **space**.

### Constrains

* The **strings** in the **array** may contain any **ASCII character** except **whitespace**.
* The startIndex and the endIndex will be in **range [-1000, 1000]**.
* The endIndex will **ALWAYS** be **GREATER** than the startIndex.
* The index in the divide command will **ALWAYS** be **INSIDE** the array.
* The partitions will be in **range [0, 100]**.
* Allowed working **time/memory**: **100ms / 16MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Ivo Johny Tony Bony Mony  merge 0 3  merge 3 4  merge 0 3  3:1 | IvoJohnyTonyBonyMony |
| abcd efgh ijkl mnop qrst uvwx yz  merge 4 10  divide 4 5  3:1 | abcd efgh ijkl mnop qr st uv wx yz |

# Icarus

Icarus is the majestic phoenix who has been alive from the beginning of creation. Icarus travels through different planes. When Icarus travels through a plane, he damages Reality itself with his overwhelming, beyond godlike flames.

You will receive a **sequence** of **integers** – the **plane**. After that you will receive **1** **integer** – an **index** in that **sequence**, which is Icarus’s **starting position**. Icarus’s **INITIAL DAMAGE** is **1**.

You will then begin **receiving** **commands** in the following format: “{direction} {steps}”. The direction will be either “left” or “right”, and the **steps** will be an **integer**. Depending on the direction, Icarus must **step** through the sequence of **integers to the left** or **right**. Each time he **steps** on a **NEW** **position**, he **damages** it. In other words, he **SUBTRACTS** his **current damage** **from** the **integer** at **that** **position**. Walking left and right has its conditions though:

* If Icarus **passes beyond** the **start** of the **sequence** (index: -1) while going **left**, he must go at the **end** of the **sequence** (index: length – 1).
* If Icarus **passes beyond** the **end** of the **sequence** (index: length - 1) while going **right**, he must go at the **start** of the **sequence** (index: 0).

If **1** of the **2 cases** **stated above** happens, Icarus **increments** his **damage** by **1**.

The input ends when you receive the command “Supernova”. When that happens you must print what is **left** of the **sequence**.

### Input

* On the **first input line** you will get the **sequence** of **integers**, **separated** by **spaces**.
* On the **second input line** you will get Icarus’s **starting position**.
* On the **next several input lines** you will get the **commands**.

### Output

* As output you must print a **single line** containing the **remaining elements** of the **sequence**, **separated** by **spaces**.

### Constrains

* The **integers** in the **sequence** will be in **range [0, 1000]**.
* The **initial position** of Icarus will **always** be **valid** and **inside** the **sequence’s indexes**.
* The **direction** will always be either “left” or “right”.
* The **steps** will be in **range [0, 1000]**.
* There will be **NO invalid** input lines.
* Allowed working time / memory: **100ms / 16MB**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 50 50 25 50 50  3  left 2  right 2  left 2  right 2  Supernova | 50 48 21 48 50 | Initial index: 3  Initial state:  50 50 25 50 50  Go left 2 steps:  50 50 24 50 50  50 49 24 50 50  Go right 2 steps:  50 49 23 50 50  50 49 23 49 50  Go left 2 steps:  50 49 22 49 50  50 48 22 49 50  Go right 2 steps:  50 48 21 49 50  50 48 21 48 50  Final state:  50 48 21 48 50 |
| 5 3 5 5 5  2  left 5  left 5  Supernova | 2 0 0 0 0 | Initial index: 2  Initial state:  5 3 5 5 5  Go left 5 steps:  5 2 5 5 5  4 2 5 5 5  4 2 5 5 3  4 2 5 3 3  4 2 3 3 3  Go left 5 steps:  4 0 3 3 3  2 0 3 3 3  2 0 3 3 0  2 0 3 0 0  2 0 0 0 0  Final state:  2 0 0 0 0 |

# Pokemon Don’t Go

Ely likes to play Pokemon Go a lot. But Pokemon Go bankrupted … So the developers made Pokemon Don’t Go out of depression. And so Ely now plays Pokemon Don’t Go. In Pokemon Don’t Go, when you walk to a certain pokemon, those closer to you, naturally get further, and those further from you, get closer.

You will receive a **sequence** of **integers**, separated by **spaces** – the distances to the pokemons.  
Then you will begin **receiving integers**, which will **correspond** to **indexes** in **that** **sequence**.

When you **receive** an **index**, you must **remove** the **element** at **that index** from the **sequence** (as if you’ve captured the pokemon).

* You must **INCREASE** the **value** of **all elements** in the sequence which are **LESS** or **EQUAL** to the **removed element**, with the **value** of the **removed element**.
* You must **DECREASE** the **value** of **all elements** in the sequence which are **GREATER** than the **removed element**, with the **value** of the **removed element**.

If the **given index** is **LESS** than **0**, **remove** the **first element** of the **sequence**, and **COPY** the **last element** to its place.

If the **given index** is **GREATER** than the **last index** of the **sequence**, **remove** the **last element** from the sequence, and **COPY** the **first element** to its place.

The **increasing** and **decreasing** of elements should be done in these cases, **also**. The **element**, whose value you should use is the **REMOVED** element.

The program **ends** when the **sequence** has **no elements** (there are no pokemons left for Ely to catch).

### Input

* On the **first line** of input you will receive a **sequence** of **integers**, **separated** by **spaces**.
* On the **next several** lines you will receive **integers** – the **indexes**.

### Output

* When the program ends, you must print on the console, the **summed up** **value** of **all REMOVED elements**.

### Constrains

* The input data will consist **ONLY** of **valid integers** in the **range [-2.147.483.648, 2.147.483.647]**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 4 5 3  1  1  0 | 14 | The **array** is {4, 5, 3}. The index is 1.  We **remove** 5, and we **increase all** **lower** than it and **decrease all higher** than it.  In this case there are **no higher** than 5.  The result is {9, 8}.  The **index** is 1. So we remove 8, and **decrease all higher** than it.  The result is {1}.  The **index** is 0. So we remove 1.  There are **no elements** **left**, so we print the **sum** of **all removed elements**.  5 + 8 + 1 = 14. |
| 5 10 6 3 5  2  4  1  1  3  0  0 | 51 | **Step 1**: {11, 4, 9, 11}  **Step 2**: {22, 15, 20, 22}  **Step 3**: {7, 5, 7}  **Step 4**: {2, 2}  **Step 5**: {4, 4}  **Step 6**: {8}  **Step 7**: {} **(empty).**  **Result** = 6 + 11 + 15 + 5 + 2 + 4 + 8 = 51. |

## C:\Users\inter\AppData\Local\Microsoft\Windows\INetCache\Content.Word\maxresdefault.jpg

# Hornet Assault

The hornets are preparing an assault on beehives. It takes very little amount of hornets to annihilate a beehive, but the bees are far from defenseless. You must calculate how many beehives, can the hornets annihilate, before they die.

You will be given a **sequence of integers**, separated by a **space**. The integers will represent the **beehives** and the **amount of bees** in them. You will then receive **another** sequence of integers, which will represent the **hornets**, and their **power**. The **power** indicates **how many bees** a hornet can **kill**.

The hornets must **start attacking** the beehives **one** by **one**. If the bees are **more** or **equal to** the **summed-up power** of the **hornets**, the **first** (**entered**) hornet, **currently alive**, **dies** in the assault of the **current beehive**.

When the hornets assault a beehive, they **kill** an **amount of bees**, **equal** to their **summed-up power**. This means that you must **decrease** the **integer** of the **beehive**, with the **value** of the summed-up power, of the **live hornets**.

After you’ve successfully assaulted all beehives, you must print the result of the **winning side**. If there are **ANY** **beehives** with **live bees** inside them, you must print them. If there aren’t, you must print the **live hornets**.

### Input

* On the first line of input you will receive a sequence of integers, separated by **spaces** – the **beehives**.
* On the second line of input you will receive a sequence of integers, separated by **spaces** – the **hornets**.

### Output

* Depending on the case of printing you must either print the **live** beehives, or the **live** hornets.
* They are sequences of integers, so in both cases you must print them **separated** by a **space**.

### Constrains

* The input will consist only of integers in **range [0; 1,000,000]**.
* There will be **NO** **STALEMATE** situations.
* The input sequences may consist of up to **3000** elements.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 20 10 20 30  5 10 5 3 | 7 | The **summed power** of the **hornets** is 23. They kill the first **3 beehives**, due to overwhelming power.  The last beehive has **higher value**, and its left with **7 bees** **alive** inside it. The **first hornet** (**5**) **dies**.  All other beehives **are dead**, so we print **only this one**. |
| 10 20 10 15  5 6 7 | 2 2 | The **summed power** is **18**. The first beehive dies. In the second one, 18 bees die, but due to higher power, the **first hornet** (**5**) **dies**.  Now the **summed power** is **13**. The third beehive dies, but the fourth one has **higher** value. **13 bees die** from the **fourth** beehive and the **current first hornet** (**6**) **dies**.  We have the **second** and the **fourth** beehive with **2 bees**, each, so we print them. |
| 20 100 40 45 20 10  40 10 5 40 5 | 10 5 40 5 |