

## Exercises: Lists

Problems for exercises and homework for the [“Programming Fundamentals” course @ SoftUni](#).

You can check your solutions here: <https://judge.softuni.bg/Contests/398/Lists-Exercises>.

### 1. Max Sequence of Equal Elements

Read a **list of integers** and find the **longest sequence of equal elements**. If several exist, print the **leftmost**.

#### Examples

| Input           | Output |
|-----------------|--------|
| 3 4 4 5 5 5 2 2 | 5 5 5  |
| 7 7 4 4 5 5 3 3 | 7 7    |
| 1 2 3 3         | 3 3    |

#### Hints

- Scan positions **p** from left to right and keep the **start** and **length** of the current sequence of equal numbers ending at **p**.
- Keep also the currently best (longest) sequence (**bestStart** + **bestLength**) and update it after each step.

### 2. Change List

Write a program, which reads a **list of integers** from the **console** and receives **commands**, which **manipulate** the list. Your program may receive the following commands:

- Delete {element}** – delete all elements in the array, which are equal to the given element
- Insert {element} {position}** – insert element and the given position

You should stop the program when you receive the command **Odd** or **Even**. If you receive **Odd** → print all **odd** numbers in the array separated with **single** whitespace, otherwise print the **even** numbers.

#### Examples

| Input   | Output |
|---|--------|
| 1 2 3 4 5 5 5 6<br>Delete 5<br>Insert 10 1<br>Delete 5<br>Odd | 1 3    |

| Input  | Output              |
|--|---------------------|
| 20 12 4 319 21 31234 2 41 23 4<br>Insert 50 2<br>Insert 50 5<br>Delete 4<br>Even | 20 12 50 50 31234 2 |

### 3. Search for a Number

On the **first line**, you will receive a **list of integers**. On the **next** line, you will receive an **array** with exactly **three numbers**. **First** number represents the **number of elements** you have to **take** from the **list** (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. If it is present print: **“YES!”**, otherwise print **“NO!”**

## Examples

| Input                | Output |
|----------------------|--------|
| 1 2 3 4 5 6<br>5 2 3 | YES!   |

| Input                                  | Output |
|--|--------|
| 12 412 123 21 654 34 65 3 23<br>7 4 21 | NO!    |

## 4. \*\* Longest Increasing Subsequence (LIS)

Read a **list of integers** and find the **longest increasing subsequence (LIS)**. If several such exist, print the **leftmost**.

### Examples

| Input                               | Output         |
|-------------------------------------|----------------|
| 1                                   | 1              |
| 7 3 5 8 -1 0 6 7                    | 3 5 6 7        |
| 1 2 5 3 5 2 4 1                     | 1 2 3 5        |
| 0 10 20 30 30 40 1 50 2 3 4 5 6     | 0 1 2 3 4 5 6  |
| 11 12 13 3 14 4 15 5 6 7 8 7 16 9 8 | 3 4 5 6 7 8 16 |
| 3 14 5 12 15 7 8 9 11 10 1          | 3 5 7 8 9 11   |

### Hints

- Assume we have **n** numbers in an array **nums[0...n-1]**.
- Let **len[p]** holds the length of the longest increasing subsequence (LIS) ending at position **p**.
- In a for loop, we shall calculate **len[p]** for **p = 0 ... n-1** as follows:
  - Let **left** is the leftmost position on the left of **p** (**left < p**), such that **len[left]** is the largest possible.
  - Then, **len[p] = 1 + len[left]**. If **left** does not exist, **len[p] = 1**.
  - Also, save **prev[p] = left** (we hold if **prev[]** the previous position, used to obtain the best length for position **p**).
- Once the values for **len[0...n-1]** are calculated, restore the LIS starting from position **p** such that **len[p]** is maximal and go back and back through **p = prev[p]**.
- The table below illustrates these computations:

| index  | 0   | 1      | 2     | 3        | 4           | 5       | 6         | 7           | 8              | 9              | 10  |
|--------|-----|--------|-------|----------|-------------|---------|-----------|-------------|----------------|----------------|-----|
| nums[] | 3   | 14     | 5     | 12       | 15          | 7       | 8         | 9           | 11             | 10             | 1   |
| len[]  | 1   | 2      | 2     | 3        | 4           | 3       | 4         | 5           | 6              | 6              | 1   |
| prev[] | -1  | 0      | 0     | 2        | 3           | 2       | 5         | 6           | 7              | 7              | -1  |
| LIS    | {3} | {3,14} | {3,5} | {3,5,12} | {3,5,12,15} | {3,5,7} | {3,5,7,8} | {3,5,7,8,9} | {3,5,7,8,9,11} | {3,5,7,8,9,10} | {1} |

## 5. \* Array Manipulator

Write a program that **reads an array of integers** from the console and **set of 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<br>add 1 8<br>contains 1<br>contains -3<br>print                    | 0<br>-1<br>[1, 8, 2, 4, 5, 6, 7]  |
| 1 2 3 4 5<br>addMany 5 9 8 7 6 5<br>contains 15<br>remove 3<br>shift 1<br>print | -1<br>[2, 3, 5, 9, 8, 7, 6, 5, 1] |
| 2 2 4 2 4<br>add 1 4<br>sumPairs<br>print                                       | [6, 6, 6]                         |
| 1 2 1 2 1 2 1 2 1 2 1 2<br>sumPairs<br>sumPairs<br>addMany 0 -1 -2 -3<br>print  | [-1, -2, -3, 6, 6, 6]             |

## 6. Sum Reversed Numbers

Write a program that reads sequence of numbers, reverses their digits, and prints their sum.

### Examples

| Input             | Output | Comments                          |
|-------------------|--------|-----------------------------------|
| 123 234 12        | 774    | 321 + 432 + 21 = 774              |
| 12 12 34 84 66 12 | 220    | 21 + 21 + 43 + 48 + 66 + 21 = 220 |
| 120 1200 12000    | 63     | 21 + 21 + 21 = 63                 |

## 7. Bomb Numbers

Write a program that **reads 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   |
|---|--------|--|
| <div> <div>1</div> <div>2</div> <div>2</div> <div>4</div> <div>2</div> <div>2</div> <div>2</div> <div>9</div> </div> <div> <div>4</div> <div>2</div> </div>                           | 12     | Special number is <b>4</b> with power 2. After detonation we left with the sequence [1, 2, 9] with sum 12.   |
| <div> <div>1</div> <div>4</div> <div>4</div> <div>2</div> <div>8</div> <div>9</div> <div>1</div> </div> <div> <div>9</div> <div>3</div> </div>  | 5      | Special number is <b>9</b> with power 3. After detonation 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).  |
| <div> <div>1</div> <div>7</div> <div>7</div> <div>1</div> <div>2</div> <div>3</div> </div> <div> <div>7</div> <div>1</div> </div>   | 6      | Detonations are performed from left to right. We could not detonate the second occurrence of 7 because its already destroyed by the first occurrence. The numbers [1, 2, 3] survive. Their sum is 6. |
| <div> <div>1</div> <div>1</div> <div>2</div> <div>1</div> <div>1</div> <div>1</div> <div>2</div> <div>1</div> <div>1</div> <div>1</div> </div> <div> <div>2</div> <div>1</div> </div> | 4      | The red and yellow numbers disappear in two sequential detonations. The result is the sequence [1, 1, 1, 1]. Sum = 4.  |