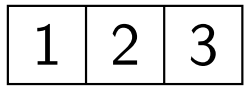
# Array

Arnaud Malapert, Gilles Menez, Marie Pelleau

Master Informatique, Université Côte d'Azur

# Array

- A collection of items stored at contiguous memory locations
- Data structure to store multiple items of the same type

arr : 


## In Python

```
arr = [1, 2, 3]
```

## In C

```
int[] arr = {1, 2, 3};
```

## In R

```
arr <- c(1, 2, 3)
```

## In Java

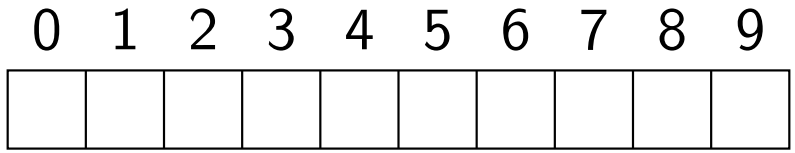
```
int[] arr = {1, 2, 3};
```

# Exercise 1: Making Book

## Statement

Given two numbers $A$ and $B$, count the number of occurrences of each digit in the range between $A$ and $B$ included

## Representation




## Example

Input:

10 15

Output:

Case 1: 0:1 1:7 2:1 3:1 4:1 5:1 6:0 7:0 8:0 9:0

# Exercise 1: Making Book

## Statement

Given two numbers $A$ and $B$, count the number of occurrences of each digit in the range between $A$ and $B$ included

## What problems can arise?

- What do we know of $A$ and $B$?
- Can $A > B$?
- Can $A = B$?
- How great can $A$ and $B$ be?
- How great can the number of occurrences be?
  $\Longrightarrow$ Are integers big enough for the solution?
  $\Longrightarrow$ What type of array for the solution?

# Exercise 1: Making Book

## Solution 1: Brut force

```
read A and B on the standard input
if A > B then
    exchange their value

initialize the solution array with 0

foreach page between A and B
    foreach digit in page
        increment the corresponding cell

print the result
```

# Exercise 1: Making Book

## Solution 2: Arithmetic

```
read A and B on the standard input
if A > B then exchange their value

diff ← B − A + 1
initialize the solution array with ⌊diff/10⌋

if diff mod 10 ≠ 0 then
    deal with the unity

A ← ⌊A/10⌋
B ← ⌊B/10⌋
deal with the ten

print the result
```

# Exercise 1: Making Book

## More test cases

### Input:

```
10 15
15 104
220 202
912 912
900 999
0
```

### Output:

```
Case 1: 0:1 1:7 2:1 3:1 4:1 5:1 6:0 7:0 8:0 9:0
Case 2: 0:14 1:19 2:19 3:19 4:19 5:19 6:19 7:19 8:19 9:19
Case 3: 0:10 1:11 2:22 3:2 4:2 5:2 6:2 7:2 8:2 9:2
Case 4: 0:0 1:1 2:1 3:0 4:0 5:0 6:0 7:0 8:0 9:1
Case 5: 0:20 1:20 2:20 3:20 4:20 5:20 6:20 7:20 8:20 9:120
```

# Exercise 2: It's a Murder

## Statement

Given an array of integers, for each number sum the previous strictly smaller numbers

## Representation

| 1 | 5 | 3 | 6 | 4 |
|---|---|---|---|---|

$$\sum$$

| 0 | 1 | 1 | 9 | 4 |
|---|---|---|---|---|

$$= 15$$

## Example

| Input: | Output: |
|---|---|
| 1<br>5<br>1 5 3 6 4 | 15 |

# Exercise 2: It's a Murder

## Statement

Given an array of integers, for each number sum the previous strictly smaller numbers

## What problems can arise?

- What do we know of the data?
  - $\Longrightarrow$ Are integers big enough for the solution?
  - $\Longrightarrow$ What type for the solution?

# Exercise 2: It's a Murder

## Solution: Brut force

```
read the array tab on standard input

result ← 0

foreach value in tab
    result ← result + sum of previous values strictly smaller
        than value

print result
```

# Exercise 2: It's a Murder

More test cases

Input:

2
5
1 5 3 6 4
7
1 3 5 2 6 7 4

Output:

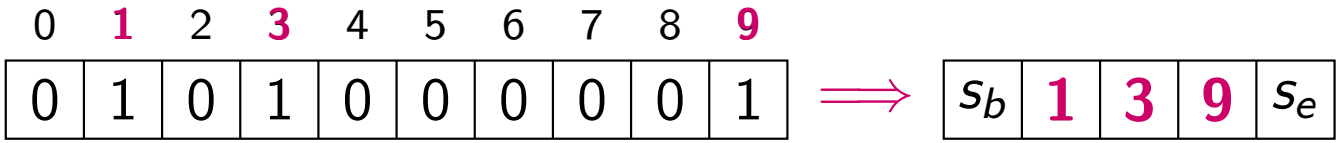
15
40

# Exercise 3: Longest Connected Subsequence

## Statement

Given a binary sequence and a number $k > 0$ find how long is the longest connected subsequence which contains at most $k$ ones

## Representation




## Example

Input:

10 1

0101000001

Output:

7

# Exercise 3: Longest Connected Subsequence

## Statement

Given a binary sequence and a number $k > 0$ find how long is the longest connected subsequence which contains at most $k$ ones

## What problems can arise?

- What do we know of $k$?
- How great can $k$ be?
- How long can the sequence be?
  $\Longrightarrow$ Are integers big enough for the solution?
  $\Longrightarrow$ What type of array for the solution?

# Exercise 3: Longest Connected Subsequence

## Solution

```
read T on the standard input

foreach test case
    read n and k on the standard input
    read the sequence on the standard input
    create array to represent the sequence
    max ← 0
    for i from 0 to n − offset
        if max < array[i + offset] − array[i]
            max ← array[i + offset] − array[i]

print max
```

# Exercise 3: Longest Connected Subsequence

More test cases

Input:

```
3
3 1
000
4 2
1111
5 3
01110
```

Output:

```
3
2
5
```