# Exercise 1

You are given an array of integers that must be sorted in ascending order.

Write **pseudocode** for an algorithm that sorts the array using the **Bubble Sort** technique. Your pseudocode must clearly describe how adjacent elements are compared and swapped, how the algorithm progresses over the array, and the condition under which it terminates.

Bubble Sort is known to be simple but inefficient.
Name **another sorting algorithm** that is asymptotically more efficient than Bubble Sort and **explain why it is better**, referring to the **time complexity** of both algorithms and to the main idea behind the more efficient approach.

## Example

**Input**
```
[4, 3, 2, 1]
```

**Intermediate steps**

```
[3, 4, 2, 1]
[3, 2, 4, 1]
[3, 2, 1, 4]
[2, 3, 1, 4]
[2, 1, 3, 4]
[1, 2, 3, 4]
```

**Output**
```
[1, 2, 3, 4]
```

## What to deliver

- Pseudocode for the Bubble Sort algorithm.
- The name of a more efficient sorting algorithm and a justification of why it is better.
- The time complexity of Bubble Sort and of the algorithm you selected.

## Exercise 2 — Filling a DNA Fragment

You are given an integer `n` representing the length of a DNA fragment.

The fragment must be filled using **building blocks** of fixed lengths:

- a **single nucleotide**, which covers **1 position**;
- a **codon**, which covers **2 consecutive positions**.

The order in which the blocks are placed matters.

Write **pseudocode** for an algorithm that computes the number of distinct **ways** to fill a fragment of length `n` using these building blocks.
Your pseudocode must clearly describe:

- the base cases;
- the recurrence relation used to compute the solution.

### Example

**Input**
```
n = 4
```

**Intermediate values**

```
ways[0] = 1
ways[1] = 1
ways[2] = 2
ways[3] = 3
ways[4] = 5
```

**Output**
```
 5
```

### Hint

Consider how a fragment of length `n` can be obtained by appending:

- a single nucleotide to a valid fragment of length `n - 1`, or
- a codon to a valid fragment of length `n - 2`.

Note that, we are counting *the number of ways to fill a fragment not the number of elements*.

### What to deliver

- Pseudocode for an algorithm that computes the number of possible fillings.
- The time complexity and space complexity of your algorithm.

## Exercise 3

You are given $k$ simple undirected graphs provided in edge-list format.

A **4-cycle** in a graph is a simple cycle that visits **exactly four distinct vertices** $a,\ b,\ c,\ d$ such that all of the edges $(a, b), (b, c), (c, d), (d, a)$ appear in the graph.

For each graph $G_i$, your task is to determine whether it contains **any 4-cycle**.
Return **1** if it does, and **−1** if it does not.

### Input format

You receive as input a file .txt containing:

- in the first line, a positive integer $k \leq 20$,
- from the second line on, $k$ simple undirected graphs (separated by an empty line) with $n$ vertices in the edge list format.

### Example

#### Input

```
2


4 5
3 4
4 2
3 2
3 1
1 2


4 4
1 2
3 4
2 4
4 1
```

#### Sample Output

```
1 -1
```

## Task

Write **pseudocode** for an algorithm that solves this problem for one file with more graph representations.
Your approach should clearly specify:

- **build** the graphs given a string describing a graphs as input,
- the **logic** to detect a 4-cycle

## Hint

You may assume the availability of a library like **NetworkX** to create/edit graphs and of a function that computes **all simple cycles of a graph**. In particular, you can use in your pseudocode a function with the following behavior:

`simple_cycles(G)`

**Input of the function**

- `G`: a graph.

**Output of the function**

- A list of cycles, where each cycle is represented as a list of vertices.
- Each cycle contains **no repeated vertices** (simple cycle).

The time complexity of this routine is $O(n + m)$.

## What to deliver

- A clear description or pseudocode of the algorithm that detects whether an undirected graph contains a 4-cycle.
- A brief explanation of why your algorithm correctly identifies a 4-cycle.
- The time complexity of your method.