Exercise 1

You are given two strings `s` and `t`, representing biological sequences (e.g., DNA or protein strings).

The **edit distance** between `s` and `t` is defined as the minimum number of single-character operations required to transform `s` into `t`.
The allowed operations are:

- insertion of a character,
- deletion of a character,
- substitution of one character with another.

Each operation has a cost equal to 1.

Example

Input

```
s = "PLEASANTLY"
```

```
t = "MEANLY"
```

Expected output

```
5
```

(The minimum number of edit operations needed to transform `s` into `t` is 5.)

What to deliver

1. **Pseudocode** for an algorithm that computes the edit distance between two strings `s` and `t` using **dynamic programming**.
    - Clearly define the meaning of each entry in the DP table.
    - Specify the base cases.
    - Specify the recurrence used to fill the table.
    - Indicate what value is returned as the final result.
2. The **time complexity** and **space complexity** of your algorithm, expressed using Big-O notation.

Exercise 2

You are given a **list of dictionaries**, each representing the result of an experiment conducted in a single laboratory.

Each experiment dictionary contains the following fields:

- `experiment_id`: unique identifier of the experiment
- `duration`: duration of the experiment in minutes
- `status`: either `"success"` or `"failure"`

Your task is to use the **MapReduce paradigm** (i.e., `map`, `filter`, and `reduce`) to:

1. Compute the **average duration of successful experiments**.
2. Return the result as a dictionary **only if** the average duration is **strictly greater than 60 minutes**. Otherwise, return an empty list.

Example input

```
experiments = [

    {"experiment_id": 1, "duration": 80, "status": "success"},

    {"experiment_id": 2, "duration": 45, "status": "failure"},

    {"experiment_id": 3, "duration": 70, "status": "success"}

]
```

Expected output

```
[

    {"average_duration": 75.0}

]
```

Notes

- Use `reduce` to compute **the sum** of durations.
- A solution that does **not** follow the MapReduce paradigm is **not valid**.

Exercise 3 — Mark and Toys

Mark has a budget of `k` units of currency and wants to buy the maximum number of toys. There are `n` toys, each with a given price. Each toy can be purchased **at most once**.

The goal is to determine the **maximum number of toys** Mark can buy without exceeding the budget.

Input format

- An integer `k` representing the available budget.
- A list `prices` of `n` integers, where `prices[i]` is the price of the *i*-th toy.

Example

Input

1. `prices = [1, 12, 5, 111, 200, 1000, 10]`
2. `k = 50`

Expected output

```
4
```

(He can buy 4 toys with prices 1, 5, 10, and 12.)

What to deliver

1. **Write pseudocode** for an algorithm that computes the maximum number of toys Mark can buy.

2. **Explain which type of algorithm** you are using (e.g., greedy, dynamic programming, etc.) and **justify why it is appropriate** for this problem.

3. Give the **time complexity** and **space complexity** of your solution in Big-O notation.