Exercise 1

You are given a **sorted array of integers** `A` and a target value `x`.
Write pseudocode for an algorithm that searches for `x` in `A` using the **binary search technique**.
At each step, the algorithm must:

1. Compute the middle index `mid` of the current search interval.
2. Compare `A[mid]` with `x`.
3. Restrict the search to the left or right half of the array accordingly.

After **each iteration**, print:

- the current values of `low`, `high`, and `mid`,
- the value `A[mid]`.

The algorithm must terminate when:

- `x` is found (return its index), or
- the search interval becomes empty (return `-1`).

Example

Input:

```
A = [1, 3, 5, 7, 9, 11, 13]
x = 9
```

Expected output:

```
low=0 high=6 mid=3 A[mid]=7
low=4 high=6 mid=5 A[mid]=11
low=4 high=4 mid=4 A[mid]=9
→ target found at index 4
```

What to deliver

- The pseudocode of the algorithm.
- A brief explanation of how binary search works.
- The **time complexity** and **space complexity** of the algorithm.
- A short explanation of **why the array must be sorted**.

Exercise 2

You are given a list of dictionaries, each representing an **order** with the following fields:

- `order_id`: unique identifier of the order
- `amount`: total amount of the order
- `customer_type`: either `"regular"` or `"premium"`

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

1. Compute the **average order amount for each customer type**
2. Return a list of dictionaries containing **only the customer types whose average order amount is strictly greater than 100**

Example input

```
orders = [
    {"order_id": 1,  "amount": 120, "customer_type": "regular"},
    {"order_id": 2,  "amount": 80,  "customer_type": "regular"},
    {"order_id": 3,  "amount": 200, "customer_type": "premium"},
    {"order_id": 4,  "amount": 150, "customer_type": "premium"},
    {"order_id": 5,  "amount": 90,  "customer_type": "regular"},
    {"order_id": 6,  "amount": 60,  "customer_type": "regular"},
    {"order_id": 7,  "amount": 180, "customer_type": "premium"},
    {"order_id": 8,  "amount": 170, "customer_type": "premium"},
    {"order_id": 9,  "amount": 70,  "customer_type": "regular"},
    {"order_id": 10, "amount": 50,  "customer_type": "regular"},
    {"order_id": 11, "amount": 140, "customer_type": "vip"},
    {"order_id": 12, "amount": 160, "customer_type": "vip"},
    {"order_id": 13, "amount": 110, "customer_type": "vip"},
    {"order_id": 14, "amount": 95,  "customer_type": "vip"},
    {"order_id": 15, "amount": 145, "customer_type": "vip"}
]
```

Expected output

```
[
    {"customer_type": "premium", "average_amount": 175.0},
    {"customer_type": "vip", "average_amount": 130.0}
]
```

Notes

- A solution that does **not** use `map` and `filter` is **not valid.** `reduce` can be optional.
- Grouping by `customer_type` can be done using standard Python constructs before applying MapReduce

Exercise 3

In a synthetic biology experiment, a micro-organism is transmitting a signal across a chain of molecular checkpoints.

At each checkpoint, the signal propagation is controlled by **two independent subsystems**:

- a **local subsystem**, influenced by what happened at checkpoint `n - 1`;
- a **delayed subsystem**, influenced by what happened at checkpoint `n - 3`.

Because the two subsystems operate independently, the total number of valid signaling cascades at checkpoint `n` is obtained by **multiplying** the number of possibilities of the two subsystems.

Base cases

- `ways(0) = 1`
  (there is exactly one trivial way to transmit no signal)
- `ways(1) = 2`
  (two distinct elementary activation patterns)
- `ways(2) = 3`
  (three valid short signaling cascades)

Recurrence

For `n ≥ 3`:

```
ways(n) = ways(n - 1) * ways(n - 3)
```

Problem Description

Given an integer n, compute the number of distinct signaling cascades ways(n) according to the multiplicative recurrence above.

Function Description

```
def countSignalWaysProduct(n: int) -> int:
        # Write your code here
```

Parameters

- n: target checkpoint index ($n \geq 0$)

Returns

- int: number of distinct signaling cascades at checkpoint n

Example

Input

5

Computation

```
ways(3) = ways(2) * ways(0) = 3 * 1 = 3
ways(4) = ways(3) * ways(1) = 3 * 2 = 6
ways(5) = ways(4) * ways(2) = 6 * 3 = 18
```

Output

18

Hint

This is still a 1-dimensional dynamic programming problem.

Unlike additive recurrences, here each state combines *independent contributions*, so the number of solutions grows **multiplicatively**.

Use an iterative DP array to avoid recomputation.

Test

Input

10

Output

102036672