2. Lab 2: Simple sorting

2.1. Objectives

i. Know how, in reality, three simple sorting methods work.
ii. Know how to use analysis tool to compare performance of sorting algorithms

2.2. Problem 1: BubbleSortApp.java

- Trace the algorithm (display the array inside after inner or outer loop)
- Display the number of swaps after the inner loop
- Display the number of comparisons after the inner loop and the total number of comparisons, and estimate the algorithms' complexity (n*(n-1)/2, O(n^2))

2.3. Problem 2: SelectSortApp.java

- Trace the algorithm (display the array after the inner loop)
- Print the items that are swapped. Are swaps always needed?
- Display the number of comparisons after the inner loop and the total number of comparisons, and estimate the algorithms' complexity (n*(n-1)/2, O(n^2))

2.4. Problem 3: InsertSortApp.java

- Trace the algorithm (display the array after each pass of the outer loop)
- Display the number of passes of the inner loop and total number of passes, and estimate the algorithms' complexity (n*(n-1)/4, O(n^2))

2.5. Problem 4

Create an array of integer numbers, fill the array with random data and print the number of **comparisons, copies, and swaps** made for sorting 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000 and 50000 items and fill in the table below. Analyze the trend for the three different algorithms.

COPIES/ COMPARISONS/ SWAPS			
	Bubble Sort	**Selection Sort**	**Insertion Sort**
10000			
15000			
20000			
25000			
30000			
35000			
40000			
45000			
50000			

2.6. Problem 5: ObjectSortApp.java (sort the array by first name or by age)

(Option 2) Given the class Person.java that has variables of first name, last name, grade

- Add a main() method and add create an array of 10 people

- Add methods to sort the array by first name, last name, and by age.

2.7. Problem 6: E-Commerce Order Fulfillment

In this problem, you are tasked with designing a system that manages order fulfillment for a large e-commerce company. The company has multiple warehouses, and each order can be fulfilled from a specific warehouse depending on stock availability. Your goal is to sort and assign orders to the most suitable warehouse, minimizing the time it takes to fulfill each order.

Problem Description:

Given:

1. A list of orders, where each order has:
    - A unique order ID.
    - A delivery deadline (in days).
    - The number of items requested.
2. A list of warehouses, where each warehouse has:
    - A unique warehouse ID.
    - A processing speed, which represents how many items the warehouse can ship per day.

Constraints:

1. Orders with earlier deadlines should be prioritized.
2. If multiple warehouses can fulfill an order, assign the order to the warehouse that can process it the fastest (i.e., the warehouse with the highest processing speed).
3. If an order cannot be fulfilled by any warehouse before the deadline, it should be marked as "unfulfilled."

Input:

- A list of orders, where each order is a tuple of the form (order_id, deadline, num_items).
- A list of warehouses, where each warehouse is a tuple of the form (warehouse_id, processing_speed).

Output:

- A schedule showing which warehouse is responsible for fulfilling each order.
- A list of unfulfilled orders.

Example:

Input:

Orders:
[("O1", 3, 50), ("O2", 1, 30), ("O3", 2, 40)]

Warehouses:
[("W1", 20), ("W2", 40)]

Output:

Warehouse W2: [("O2", 1, 30), ("O3", 2, 40)]
Warehouse W1: [("O1", 3, 50)]
Unfulfilled: []

Key Challenges:

1. **Sorting by Deadline:** Students need to prioritize orders based on their delivery deadlines to ensure the most urgent orders are fulfilled first.

2. **Matching Orders to Warehouses:** Students must assign orders to the most appropriate warehouse based on processing speed, ensuring that orders are fulfilled in the shortest possible time.
3. **Handling Unfulfilled Orders:** If a warehouse cannot fulfill an order before the deadline, it should be marked as "unfulfilled." This introduces a layer of complexity where students must track whether each order can be completed on time.

Bonus Challenge:

- **Multiple Items Per Order:** Modify the problem so that each order consists of multiple items that may need to be shipped from different warehouses. This will require splitting orders and further optimizing the fulfillment process.