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

An integer array b holds values in the following order:

6, 23, 33, 12, 5, 15, 7, 27, 2

Part a.

Using **Selection Sort**, complete the following table so that every row contains the state of **b** after each iteration of the outer loop

(Initial)	**6**	**9**	**43**	**12**	**5**	**13**	**5**	**27**	**8**
Step 1									
Step 2									
Step 3									
Step 4									
Step 5									
Step 6									
Step 7									
Step 8									

Part b.

Using **Insertion Sort**, create a similar table to part (a), ensuring that every row contains the state of **b** after each iteration of the outer loop