**LAB 01**

**OOP Reviews and Array.**

Nguyễn Minh Đạt – ITDSIU22166

Problem 1: BubbleSortApp.java:

A computer screen shot of a program code

AI-generated content may be incorrect.

Problem 2: SelectSortApp.java

A screen shot of a computer program

AI-generated content may be incorrect.

Problem 3: InsertSortApp.java

A computer screen shot of a program code

AI-generated content may be incorrect.

Problem 4: COPIES/ COMPARISONS/ SWAPS



**Trend Analysis**

**1. Bubble Sort**

* **Comparisons**: Always n(n-1)/2 (quadratic).
  + For n = 10,000, comparisons = 10,000 \* 9,999 / 2 = 49,995,000.
  + For n = 50,000, comparisons = 50,000 \* 49,999 / 2 = 1,249,975,000.
* **Swaps**: Approximately n(n-1)/4 (quadratic).
  + For n = 10,000, swaps = 10,000 \* 9,999 / 4 = 24,997,500.
  + For n = 50,000, swaps = 50,000 \* 49,999 / 4 = 624,987,500.
* **Copies**: None.
* **Trend**:
  + Both comparisons and swaps grow quadratically with input size.
  + Bubble Sort is inefficient for large datasets due to its high number of swaps and comparisons.

**2. Selection Sort**

* **Comparisons**: Always n(n-1)/2 (quadratic).
  + For n = 10,000, comparisons = 10,000 \* 9,999 / 2 = 49,995,000.
  + For n = 50,000, comparisons = 50,000 \* 49,999 / 2 = 1,249,975,000.
* **Swaps**: Exactly n-1 (linear).
  + For n = 10,000, swaps = 9,999.
  + For n = 50,000, swaps = 49,999.
* **Copies**: None.
* **Trend**:
  + Comparisons grow quadratically, but swaps grow linearly.
  + Selection Sort is more efficient than Bubble Sort in terms of swaps but still inefficient for large datasets due to its quadratic comparisons.

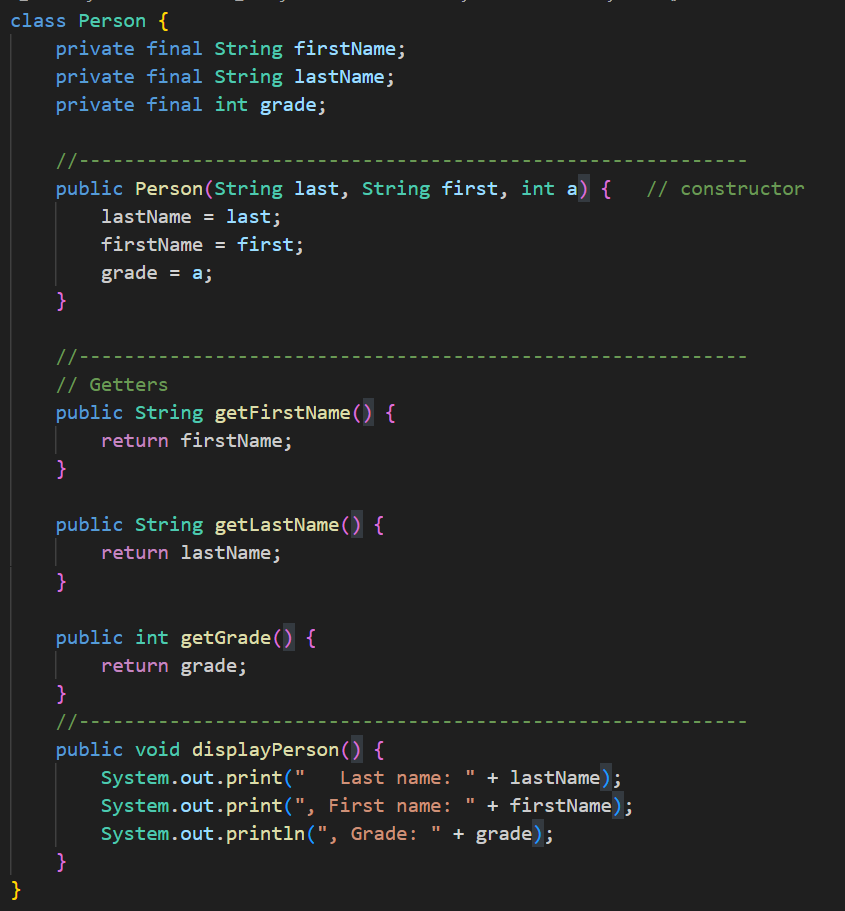
**3. Insertion Sort**

* **Comparisons**: Approximately n(n-1)/4 (quadratic).
  + For n = 10,000, comparisons = 10,000 \* 9,999 / 4 = 24,997,500.
  + For n = 50,000, comparisons = 50,000 \* 49,999 / 4 = 624,987,500.
* **Copies**: Approximately n(n-1)/4 (quadratic).
  + For n = 10,000, copies = 10,000 \* 9,999 / 4 = 24,997,500.
  + For n = 50,000, copies = 50,000 \* 49,999 / 4 = 624,987,500.
* **Swaps**: None.
* **Trend**:
  + Both comparisons and copies grow quadratically with input size.
  + Insertion Sort performs fewer comparisons and copies on average compared to Bubble Sort and Selection Sort but still has quadratic complexity.

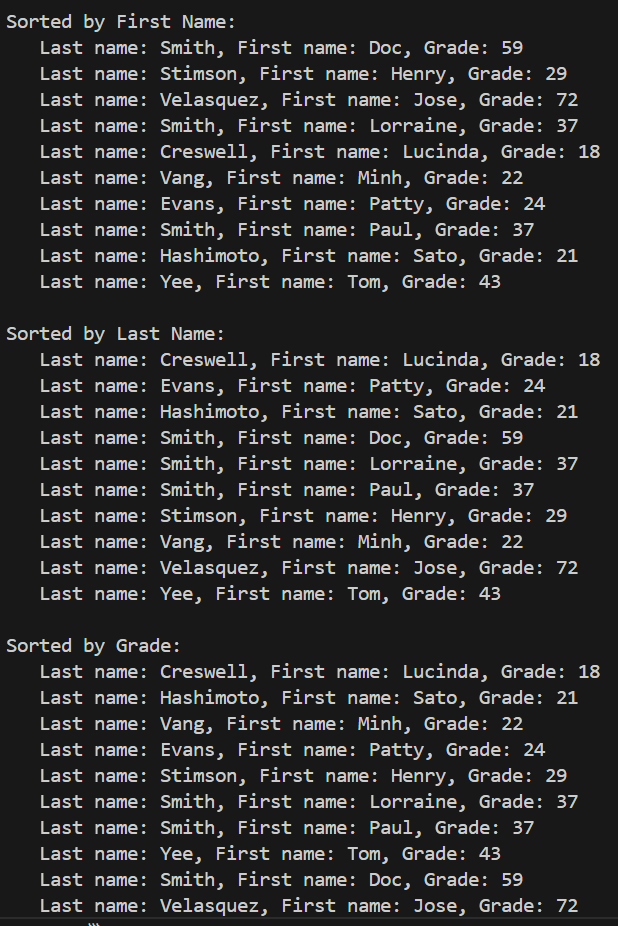
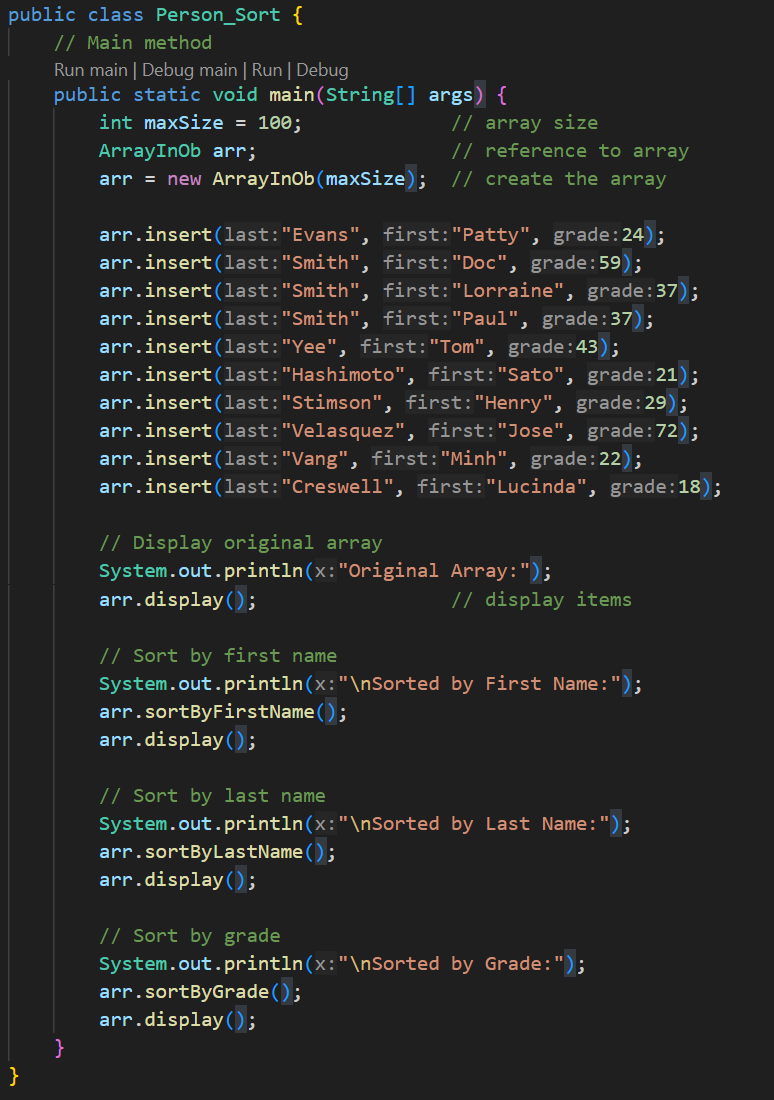
**Performance Comparison**

1. **Bubble Sort**:
   * High number of swaps and comparisons.
   * Inefficient for large datasets.
   * Simple to implement but rarely used in practice.
2. **Selection Sort**:
   * Fewer swaps than Bubble Sort but still high comparisons.
   * More efficient than Bubble Sort for datasets with high swap costs (e.g., large objects).
3. **Insertion Sort**:
   * Fewer comparisons and copies than Bubble Sort and Selection Sort on average.
   * Efficient for small datasets or nearly sorted datasets.
   * Often used as a building block for more advanced algorithms (e.g., Timsort).

Problem 5: Person.java

A screenshot of a computer program

AI-generated content may be incorrect.



Problem 6: Selection Sort vs Insertion Sort

1. Selection Sort:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Selection Sort | | | | | | | | | |
| Initial | 6 | 23 | 33 | 12 | 5 | 15 | 7 | 27 | 2 |
| Step 1 | 9 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 |
| Step 2 | 9 | 5 | 3 | 4 | 2 | 6 | 7 | 8 | 1 |
| Step 3 | 3 | 5 | 9 | 4 | 2 | 6 | 7 | 8 | 1 |
| Step 4 | 3 | 5 | 9 | 7 | 2 | 6 | 4 | 8 | 1 |
| Step 5 | 3 | 7 | 9 | 5 | 2 | 6 | 4 | 8 | 1 |
| Step 6 | 3 | 7 | 9 | 5 | 2 | 6 | 4 | 8 | 1 |
| Step 7 | 3 | 7 | 9 | 5 | 2 | 6 | 4 | 8 | 1 |
| Step 8 | 3 | 7 | 9 | 5 | 2 | 6 | 4 | 8 | 1 |

1. Insertion Sort

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Insertion Sort | | | | | | | | | |
| Initial | 6 | 23 | 33 | 12 | 5 | 15 | 7 | 27 | 2 |
| Step 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Step 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Step 3 | 1 | 3 | 4 | 2 | 5 | 6 | 7 | 8 | 9 |
| Step 4 | 2 | 4 | 5 | 3 | 1 | 6 | 7 | 8 | 9 |
| Step 5 | 2 | 5 | 6 | 3 | 1 | 4 | 7 | 8 | 9 |
| Step 6 | 2 | 6 | 7 | 4 | 1 | 5 | 3 | 8 | 9 |
| Step 7 | 2 | 6 | 8 | 4 | 1 | 5 | 3 | 7 | 9 |
| Step 8 | 3 | 7 | 9 | 5 | 2 | 6 | 4 | 8 | 1 |