Lab 06 Computer Science 1027B Summer 2025

Introduction

The purpose of these labs is to provide practice problems for you to get used to the concepts being covered in class. It is very important that you do these labs on your own without relying on available AI tools. Questions on exams are often very similar to exercises presented in these labs, so understanding them and solving these problems gives you a huge advantage for the exams. Make sure that you complete both the code as well as provide written answers to the blue questions within the boxes.

When you have concluded this lab, you should submit the following files to OWL Brightspace:

- BubbleSortModifications.java

- CustomSort.java

- written_answers.txt (for short answer questions)

- OPTIONAL: images for question 03 (rather than text in written_answers.txt)

Topics Covered

- Sorting algorithms

Exercise 01

In the lecture video for sorting algorithms, I showed some code that implements the bubble sort algorithm. Let's make some modifications to that original algorithm. In a class called **BubbleSortModifications**, let's write a number of static methods.

We will start with the original bubble sort implementation from the lecture, a static method called **bubbleSort** that takes an array of ints and sorts the array using bubble sort (you can copy and paste the code provided in the week 4 content tab).

Exercise 01.1

Sort in descending order:

- In a new static method called '**bubbleSortDescending**', Modify the bubble sort implementation to sort the array in descending order.

Exercise 01.2

Sort Strings:

- In a new static method called '**bubbleSortStrings**', Modify the bubble sort implementation to sort an array of Strings in lexicographical order.

Exercise 01.3

Analyze and compare performance:

- Write a new static method, **bubbleSortCompareArrays**, to generate three large random arrays (1000 or so elements) and compare the performance of bubble sort. You may use System.nanoTime() to measure the execution time. For example:
    - long startTime = System.nanoTime(); // start time
      bubbleSort(arrayToSort); // sort one of the generated arrays
      long endTime = System.nanoTime(); // end time
      System.out.println((endTime – startTime) + "ns"); // total execution time

Q 01.3.1

Why do you think that randomly generated arrays of the same size may have taken different amounts of time? What overarching statements can you make about how the initial state of the array impacts the time it takes to be sorted?

Exercise 02

You have teleported back to the late 1800s, and the world of sorting algorithms has largely been unexplored. This gives you a fantastic opportunity to write a sorting algorithm and have it named after yourself.

Write a method that takes an array of ints and sorts it in increasing order. The emphasis is on creativity and understanding the sorting process, not on efficiency. Thus, it does not need to be an in-place algorithm (or even memory-efficient or time-efficient at all). You could even make it intentionally bad just for fun.

Conceptualize your algorithm: Think about different ways you could sort an array of integers. Consider how you might repeatedly find the smallest or largest elements, how you might swap elements, or any other method you can imagine. Will it be in-place or will you use more than one array while sorting? Will you just keep randomly shuffling the elements until the array is sorted? It's up to you!

Outline your algorithm: Write down the steps your algorithm will take in pseudocode. Make sure to clearly define how the algorithm will determine the order of elements.

Implement your algorithm: Write a static method that implements your sorting algorithm in the class '**CustomSort'**. Use the following method signature:

```
public static void customSort(int[] array)
```

Now, let's compare it to Java's Arrays.sort() method by pasting the following code into a new file, **SortComparison.java**:

```
import java.util.Arrays;

public class SortComparison {
    public static void main(String[] args) {
        int[] array = {64, 25, 12, 22, 11};

        // clone the array to use the same data for both sorts
        int[] arrayForCustomSort = array.clone();
        int[] arrayForJavaSort = array.clone();

        // measure time for custom Sort
        long startTime = System.nanoTime();
        CustomSort.customSort(arrayForCustomSort);
        long endTime = System.nanoTime();
        long customSortTime = endTime - startTime;
        System.out.println("Custom Sort Time: " + customSortTime + " ns");

        // measure time for Java's Arrays.sort()
        startTime = System.nanoTime();
        Arrays.sort(arrayForJavaSort);
        endTime = System.nanoTime();
```

```
        long javaSortTime = endTime - startTime;
        System.out.println("Java's Arrays.sort() Time: " + javaSortTime + " ns");

        // print sorted arrays
        System.out.println("Custom Sorted array: " + Arrays.toString(arrayForCustomSort));
        System.out.println("Java Sorted array: " + Arrays.toString(arrayForJavaSort));
    }
}
```

Q 02.1

What is printed when the above code is run?

Q 02.2

What do the results tell you about your sorting algorithm versus the sort method provided by the Java standard library? Why does memory-efficiency matter? Why does time-efficiency matter?

Exercise 03

Q 03.1

Given the array **{6, 12, 82, 1, -6}**, manually perform bubble sort step-by-step. Write down the array after each pass. **Note that no coding is required here**.

Q 03.2

Given the array **{6, 12, 82, 1, -6}**, manually perform insertion sort step-by-step. Write down the array after each pass and highlight (or bold or underline) the key element being inserted. **Note that no coding is required here**.

Q 03.3

Given the array **{6, 12, 82, 1, -6}**, manually perform selection sort step-by-step. Write down the array after each pass and highlight (or bold or underline) the elements being swapped. You can separate the sorted and unsorted subarrays with a '|'. **Note that no coding is required here**.

**** Note that if you find it easier, you can create a visual in whatever way you'd like. You can even draw it on paper and take a picture and then attach it as 03_1.<some extension>, Q3_2.<some extension>, and Q3_3.<some extension>.**

Exercise 04

Q 04.1

Describe what you think the worst-case scenarios are for quick sort and merge sort in terms of the initial array to be sorted. How could this potentially be mitigated?