Computer Science 385
Design and Analysis of Algorithms
Siena University
Spring 2026

Emprical Study 1

Group Formation: 4:00 PM, Friday, February 27, 2026
Due: 4:00 PM, Wednesday, March 11, 2026 (code)
and 4:00 PM, Monday, March 16, 2026 (writeup)

You may work alone or in a group of up to 4 members on this assignment.

All GitHub repositories must be created with all group members having write access and all group member names specified in the `README.md` file by 4:00 PM, Friday, February 27, 2026. This applies to those who choose to work alone as well!

You should expect to spend substantially more time on the data gathering and writeup than the coding. This assignment is one for which AI assistance can be very helpful, and that is encouraged, but its use must be fully documented and you are responsible for ensuring that the work for which AI assistance was used is correct.

Getting Set Up

In Canvas, you will find a link to follow to set up your GitHub repository, which will be named `sortingstudy-yourgitname`, for this lab. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

Submitting

Your submission requires that all required code deliverables and a PDF file of your writeup are committed and pushed to the main branch for your repository's origin on GitHub. If you see everything you intend to submit when you visit your repository's page on GitHub, you're set.

Empirical Analysis of Sorting Algorithms

Your task, worth a total of 75 points in the problem sets category, is to conduct an empirical study of the three sorting algorithms we have considered so far: bubble sort, selection sort, and insertion sort. We will soon see others.

A sample empirical study, both code and writeup, is available in a GitHub repository (`https://github.com/SienaCSISAlgorithms/SampleEmpiricalStudy`) that you are welcome to clone or fork so you can use it as a model for your own study and writeup. Suggestions for improvements to this sample study are welcome, ideally in the form of Issues and Pull Requests to that repository.

A Sorting Framework for Gathering Timings

Your first task, for 50 points, is to develop a program to perform an empirical study of the efficiency of these sorting algorithms. Your program should operate on arrays of `int` values. It should have command-line options to set the array size, the number of trials (to improve timing accuracy), the ability to generate initial data that is sorted, nearly sorted, completely random, and reverse sorted (implement the algorithms you wrote for a previous assignment for array population). Design your program to make it easy to implement additional sorting algorithms later. You will be adding those later in the semester.

- Use command line parameters rather than prompts, as this makes it much easier when running many (likely hundreds or even thousands) of trials to generate timing results. In Java, `args[]` has what you need! If you don't know or remember how to run with command-line parameters inside your IDE, run your Java program at the command line. That's what you'll want to do when generating timing results anyway.
- You will need one or more methods to implement each sorting algorithm. You should write your own code, but you may base it on any descriptions of these algorithms you wish (pseudocode from our text might be a good choice), and you may use AI assistance with appropriate documentation. For bubble sort, only an iterative implementation is needed, but for selection sort and insertion sort, implement both iterative and recursive options so you can compare their speeds and see how recursive implementations can run out of memory more quickly than their iterative counterparts. Don't overthink these!
- Write one big program rather than lots of little ones. This will help you avoid repeated code as you implement each of the sorting algorithms.
- Even though a single program will be able to run a variety of sorting algorithms on different sizes and initial ordering of input, a single execution of your program should only run one such combination (though possibly many trials of that same run to improve timing accuracy).
- Be careful that you don't reuse an array of values for multiple runs without re-populating it, since all but the first could end up having already-sorted data as input.
- Your program should measure and report both the elapsed time and basic operation counts for each problem instance. The basic operation of interest for these comparison-based sorting algorithms are the number of comparisons of data values **and** modification of array elements.
- A simple tabular format of output will help you manage the creation of tables and/or graphs that you'll need later. Something like

  ```
  10000 bubble random .034693 49995000
  ```

  might indicate for an input size of 10,000, using a bubble sort on random input took .034693 seconds and made 49,995,000 comparisons. Note that your output would also include the number of modifications of array elements.
- Bash or other scripts can be used to run the program multiple times. AI assistance in creation of the scripts is likely to be helpful.

First Empirical Analysis Study

Perform an empirical analysis study for bubble sort (iterative only), selection sort (both iterative and recursive implementations) and insertion sort (both iterative and recursive implementations), on an appropriate range of data sizes and distributions. Use your sorting algorithm program to generate timing data. Compare your timing results with your expectations based on our (theoretical) efficiency analysis of these algorithms. Present your results briefly but formally. (40 points)

- Include as many of the details of your test environment as you can: the type of processor or processors in the computer including clock speed, cache sizes, memory sizes, the operating system and version running on the computer, and the Java version (or whatever other language) you are using.
- Include a brief description of the methodology for the tests. Describe the set of runs you are going to perform and state the expected results based on the theory (*e.g.*, n or n^2 running times).
- If you find discrepencies between your theoretical expectations and the timings and operation counts you gather, explain to the best of your ability what you believe caused it.
- The runs should vary the input array size for each combination of sorting algorithm and input data type. To get meaningful results, you want a pretty wide range of sizes. You might start with an array of size 1000 (or better yet $2^{10} = 1024$) and double the size until you have an input size of 1,000,000 (or better yet $2^{20} = 1,048,576$). Take the average or best times (and justify your choice) for some number of runs, probably a few dozen to a few hundred. Then, plot your results. Make sure your graphs have a meaningful title, legend, and axis labels. See if the numbers fit the expected, *e.g.*, n or n^2, behavior. AI is likely to be helpful as an assistant in creation of your graphs.
- Include a summary of your raw timing results and operation counts (in tabular form), your graphs of those results and your analysis of the results in your writeup. Full sets of raw numbers are useful, but should be submitted separately in a big text file or spreadsheet, rather than as part of the main writeup. There are likely to be too many numbers for anyone to want to look at them all, but it is valuable to have them available.

The formal writeup should follow the format of the sample empirical study. It should have a section that includes a brief introduction to the task, including test environment information. There should then be a section for each algorithm that describes specifics of that algorithm, the theoretical expectations (including best/average/worst cases, where applicable), a summary of your timings and counts in tabular format, graphs illustrating these results, followed by more text that analyzes the results, comparing with the theoretical expectations. A brief conclusion section should summarize the study's results.

Deliverables

A complete submission will include the source code for the program used to generate the results, a PDF document that contains the formal writeup, and any spreadsheets or other files that include the raw data.

Grading

This assignment will be graded out of 90 points.

Feature	Value	Score
Code	50	
Writeup	40	
Total	90	