SIENAcollege
Computer Science

# Computer Science 385
# Design and Analysis of Algorithms
Siena College
Spring 2024

# Problem Set 7
**Due: 4:00 PM, Wednesday, April 24, 2024**

Your task for this problem set is to extend your previous empirical study of sorting algorithms.

---

## Getting Set Up

Please work within your repository from Problem Set 3, since this is an extension of that problem set. If there are any changes to group membership from Problem Set 3, contact me so we can create or modify permissions on repositories as needed.

---

## Extended Empirical Analysis of Sorting Algorithms

- Add some advanced sorting algorithms to your empirical study: mergesort, quicksort, and heapsort. For quicksort, you will need to consider three versions: the standard version that chooses the pivot value as the first element of the subarray being sorted, a version that chooses a pivot randomly from among all of the elements of the subarray being sorted, and the "median of three" pivot choice which chooses the pivot as the middle value of the first, middle, and last elements of the subarray. A good implementation will support all three without replicating large amounts of code by having the latter two move their chosen pivot to the first position then applying the standard quicksort.
- You are permitted to use or base your code on existing algorithm implementations, or get help from generative AI in writing them. For any code that is not entirely your own work, you must be sure you are not violating any copyrights, and you must cite the source(s) both in your code and in your writeup.
- Expectations are the same as they were for the earlier study in terms of coding the algorithms, running tests to gather timings and comparison counts, presenting those timings in tabular and graph formats, and a writeup that states your expectations for the results, what you observed, and how these align. For each algorithm and problem size, you should test with random data, sorted data, reverse sorted data, and nearly sorted data.
- Avoid recursive implementations for efficiency purposes. This is especially true for quicksort when the worst-case behaviors come up. You can probably get away with recursion for mergesort.
- Run each case several times, as you did before. Choose your range of problem sizes so that the smallest cases run for at least a second (but not much more), and the largest cases run for a few minutes at most. Choose a range of several powers of two for problem sizes. You will have a large number of runs to make and it will take a significant amount of time even with this restriction. Plan accordingly.

- Just stating "the graph looks like $O(n \log n)$" isn't good enough to match your timing and comparison count results to the theory. Show how as $n$ changes, the trend grows like $n \log n$ or $n^2$ or whatever is appropriate.
- If comparison counts do not match up precisely with theoretical expectations, briefly explain why that might be the case (e.g., one of the lists in mergesort's merge method will sometimes become empty long before the other)

**Deliverables**

A 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. It would make sense to simply extend your code and writeup documents from the previous problem set.

---

## Grading

This assignment will be graded out of 60 points.

| Feature | Value | Score |
|---|---|---|
| Code | 30 | |
| Writeup | 30 | |
| Total | 60 | |