||

MODULE 3-2 MILESTONE THREE

ENHANCEMENT ONE: ALGORYTHEM AND DATA STRUCTURE

Shekhar Chaudhary

CS-499 Computer Science

September 25, 2025

||

The artifact I selected for Category Two: Algorithms and Data Structures is my sorting and searching program from CS-300. This program was originally created to demonstrate fundamental algorithmic principles, including linear search, binary search, and basic sorting methods. While the original version applied these concepts successfully, it did not fully explore efficiency and scalability when working with larger datasets. I chose this artifact for enhancement because it provides a strong foundation for demonstrating algorithm design, performance trade-offs, and optimization—skills essential for developing scalable systems and preparing for real-world software engineering and AI-related roles.

For enhancement, I focused on expanding the program beyond simple sorting by implementing **hash map indexing** for O (1) lookups, **binary search** over a sorted vector for O (log n) searches, and adding a **MergeSort algorithm** to complement Selection Sort and QuickSort. These additions created opportunities to analyze algorithmic trade-offs in stability, memory usage, and runtime performance. For example, MergeSort is stable but requires additional memory, QuickSort is generally fast but unstable, and Selection Sort serves as a baseline for understanding inefficiency. To provide measurable evidence, I replaced the clock () timing mechanism with std::chrono::steady\_clock and added CSV output so results could be benchmarked consistently. Preliminary tests using the 17,937-row dataset showed QuickSort averaging 42ms, MergeSort 57ms, and Selection Sort more than 12,000ms, directly validating theoretical complexity analysis with practical results.

These enhancements directly support multiple course outcomes. They demonstrate my ability to **design and evaluate computing solutions using algorithmic principles while managing trade-offs** in performance and memory. They also reflect **problem-solving skills** by adapting the program to handle large-scale datasets and **innovative approaches** by integrating benchmarking and visualization to communicate results effectively. In addition, by structuring the program so algorithms can be swapped and compared, I have shown the importance of modular design in applied computing.

Reflecting on the process, I learned that effective enhancement is not about adding complexity for its own sake but about making thoughtful, purposeful design choices. Implementing the hash map required balancing collision handling and memory reallocation strategies, while building the binary search index emphasized the importance of sorted data. MergeSort added an opportunity to discuss algorithm stability, which is often overlooked in entry-level programs. One challenge was ensuring benchmark results remained consistent and interpretable, which I resolved by running multiple trials and exporting results for visualization. Through this work, I deepened my understanding of theoretical algorithm analysis, practical performance testing, and the communication of results in ways meaningful to both technical and professional audiences.

Here is high impact enhancement I will performing to make this benchmark more effectiveness. Specifically, I will me making rigorous enhancement in this file called VectorSorthing.cpp.

A screen shot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

**High-Impact Enhancements Planned**

To maximize the effectiveness of this artifact, I am implementing several high-impact enhancements that transform the program from a basic sorting demonstration into a comprehensive algorithms and data structures showcase.

1. **Introduce advanced searching structures (HashMap and Binary Search):**  
   After loading bids, I will build a hash map index for O(1) average-case lookup by bidId and maintain a sorted vector to enable O(log n) binary search by title. This enhancement allows me to compare constant-time lookups against logarithmic searches, reinforcing the importance of selecting appropriate data structures for specific use cases. It also demonstrates my ability to apply algorithmic efficiency principles to real-world problems.
2. **Add a third sorting algorithm with stability discussion (MergeSort):**  
   In addition to Selection Sort and QuickSort, I will implement MergeSort, which provides stability at the cost of additional memory. By comparing QuickSort’s speed, MergeSort’s stability, and Selection Sort’s inefficiency, I can highlight trade-offs in time complexity, space usage, and output reliability. This discussion showcases my understanding of algorithmic principles and problem-solving techniques.
3. **Implement reproducible benchmarking with modern timing tools:**  
   To provide measurable evidence, I will replace clock() with std::chrono::steady\_clock and export results to a CSV file for analysis. This change ensures accuracy, reproducibility, and professional-level reporting of runtime performance. Preliminary benchmarks already show that QuickSort handles 17,937 bids in ~42ms, while MergeSort requires ~57ms, and Selection Sort takes over 12,000ms. These results directly connect theoretical Big-O analysis to real-world outcomes and strengthen the artifact’s value in my ePortfolio.

Moreover, By making these enhancements, my artifact will evolve from a simple sorting demonstration into a robust showcase of **algorithmic design, efficiency analysis, and data structure trade-offs**. Instead of only running Selection Sort and Quicksort on bids, I will now demonstrate how **multiple sorting algorithms** (Selection, Quick, Merge) compare in terms of stability, memory use, and runtime performance. I will also introduce **advanced searching structures** like hash maps and binary search on sorted vectors, showing how the choice of data structure directly impacts performance when handling large datasets. With the addition of reproducible benchmarks and timing outputs, I will have measurable evidence that connects theory (Big-O complexity) with practice (real-world runtime on thousands of records). The outcome is an artifact that not only meets but exceeds the course outcome of designing and evaluating computing solutions using algorithmic principles, while also giving me a polished, professional piece for your ePortfolio that highlights my ability to manage **trade-offs in speed, space, and stability**—skills directly relevant to software engineering and AI-focused roles.

Reference:

Southern New Hampshire University. Milestone Two Guidelines and Rubric Enhancement One: Software Design/Engineering. Retrieved from Module 3-2 Milestone Two: Algorithm and Data structure: https://learn.snhu.edu/d2l/le/content/2019777/viewContent/43174444/View