CS-499: Computer Science Capstone

4-2 Milestone Three: Enhancement Two: Algorithms & Data Structure

Emily Black

November 18, 2025

ARTIFACT DESCRIPTION & ORIGIN

The artifact I enhanced for Milestone Three originates from my CS-300 Data Structures & Algorithms Project One, completed in October 2024. The original project consisted of pseudocode and analysis comparing the runtime behavior of three data structures: the vector, hash table, and binary search tree (BST). While the original submission demonstrated conceptual understanding of Big-O analysis, it lacked executable code, real performance measurements, and empirical insight. Because the artifact aligned directly with the "Algorithms and Data Structures" outcome, it provided an ideal foundation for enhancement during the capstone experience.

SUMMARY OF PLANNED ENHANCEMENTS & HOW THEY WERE IMPLEMENTED

According to my enhancement plan that I completed in module one, the goals for this artifact were to 1) convert the pseudocode into fully working C++ implementations of each data structure, 2) measure performance by timing the loading, searching, and sorting operations, 3) export the empirical results to a CSV file, 4) visualize performance using Python and matplotlib, and 5) implement logic that recommends the most efficient data structure depending on dataset size and usage patterns; these enhancements were completed for Milestone Three.

First, I implemented each structure in its own modular C++ class: a `VectorDS` class for vector-based storage and sorting, a `HashTableDS` class using `std::unordered_map`, and a `BSTree` class implementing node insertion, recursive search, and in-order traversal. These implementations followed modern C++ standards and included detailed inline documentation. Next, I added

performance measurement functionality using the `chrono::high_resolution_clock` class to measure nanosecond-level operation durations. Each structure's load, search, and, where applicable, sort operations were timed and written to a CSV file for analysis.

To complete the enhancement plan, I wrote a companion Python script that imports the CSV output, constructs runtime comparison charts, and displays the relative efficiency of each data structure. Finally, I added logic to the C++ driver program that generates a "recommended structure" statement, consistent with algorithmic guidance such as the differences between BST and hash table operations described by GeeksforGeeks (2024), which fully satisfies the enhancement steps I originally outlined.

JUSTIFICATION FOR ENHANCEMENT & SKILLS DEMONSTRATED

The enhanced artifact demonstrates algorithmic reasoning, performance evaluation, empirical testing, and professional-level documentation. Converting pseudocode into functional C++ code required the application of data structure design principles, memory organization, recursion, and STL container usage. Adding empirical timing and CSV generation shows the ability to evaluate algorithmic trade-offs with real data, not just theoretical analysis. The final recommendation logic demonstrates the ability to interpret runtime results in a meaningful way, which is similar to how data analysts synthesize quantitative metrics to support decision-making.

This enhancement aligns closely with the outcome: "Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards while managing the trade-offs involved in design choices." It also demonstrates my ability to use well-founded tools to deliver value, another key outcome. As a future data analyst and potential business intelligence analyst, the skills demonstrated, such as structured

experimentation, runtime analysis, and visual communication, mirror the exact competencies required for analyzing datasets, interpreting performance metrics, and making evidence-based recommendations.

REFLECTION ON THE ENHANCEMENT PROCESS

This enhancement challenged me to think far more deeply about algorithmic performance than the original assignment required. Implementing a binary search tree from scratch reinforced my understanding of recursion, tree traversal, and pointer-based memory management. Working with vectors and hash tables in parallel helped highlight the subtle differences in constant factors and access patterns that affect real performance, even when theoretical complexity appears similar. Incorporating Python visualization also demonstrated the importance of communicating technical findings clearly and effectively.

The biggest challenge was managing the benchmarking code in a way that produced consistent, trustworthy measurements. This required isolating timing logic, minimizing overhead, and ensuring that repeated tests were conducted under controlled conditions. Through this process, I gained a greater appreciation for how professional engineers evaluate algorithmic performance using carefully designed experiments. Overall, the enhancement strengthened my algorithmic reasoning, improved my confidence with C++ data structures, and further reinforced the analytical thinking that aligns with my path toward data and BI analytics.

REFERENCES

GeeksforGeeks. (2024). *Data structures comparison: Arrays, linked lists, stacks, queues, trees, and hash tables*. https://www.geeksforgeeks.org

Microsoft. (2024). *chrono::high_resolution_clock class*. https://learn.microsoft.com/cpp