CS-499: Computer Science Capstone

24 November 2024

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

The artifact is a **sorting algorithm** implemented to demonstrate my ability to design and evaluate an efficient sorting solution using algorithmic principles. Specifically, I created an implementation of the **Merge Sort** algorithm, a divide-and-conquer algorithm known for its efficiency and predictable time complexity. This artifact was created in **2024** as part of my studies in algorithms and data structures, where I aimed to showcase how to solve sorting problems using optimized algorithms.

I selected this artifact because it represents a key aspect of my understanding of **algorithms and data structures**—the ability to design and implement efficient sorting algorithms. This artifact demonstrates several critical components of algorithms and data structures:

* **Algorithm Design**: By implementing Merge Sort, I’m showcasing my ability to design efficient algorithms for problem-solving. The algorithm splits the array into smaller subarrays, sorts them recursively, and merges them back together, highlighting my understanding of recursive problem-solving.
* **Time and Space Complexity Analysis**: This artifact emphasizes my understanding of algorithm performance, particularly the **O(n log n)** time complexity of Merge Sort and its efficient handling of large datasets.
* **Data Structure Usage**: The algorithm makes use of arrays and recursive calls, reinforcing my ability to use appropriate data structures for efficient computation.

The artifact was improved by optimizing the recursive function for better space management. Initially, the implementation used extra memory for each recursive call, but I restructured the code to minimize memory usage and improve efficiency, while maintaining the original sorting capabilities.

Yes, the artifact meets the course outcomes I planned to meet with this enhancement, particularly in the following areas:

* **Outcome 3**: *Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices.*

This outcome is addressed by my careful design of the Merge Sort algorithm and analysis of its efficiency compared to other sorting algorithms, such as Quick Sort or Bubble Sort. I evaluated Merge Sort’s trade-offs in terms of time complexity versus space complexity.

Additionally, I’m updating my outcome-coverage plans to also emphasize **Outcome 4**: *Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.*

By refining the sorting algorithm and ensuring its optimal implementation, I am demonstrating my capability to apply well-founded techniques to solve computational problems efficiently, which aligns with the skills needed in industry settings.

Throughout the process of enhancing and modifying the Merge Sort implementation, I gained valuable insights into algorithm optimization and memory management. Key lessons learned include:

* **Optimization**: I learned how to balance algorithm efficiency with memory usage. While Merge Sort is generally efficient in terms of time complexity, its space complexity can be a limiting factor. By modifying the implementation to handle recursive calls more efficiently, I learned how to improve both the speed and memory usage of the algorithm.
* **Debugging and Testing**: The process of debugging and ensuring that the algorithm worked correctly across various input sizes and edge cases taught me the importance of testing algorithms under different conditions. I used test cases with both small and large datasets to validate the accuracy and efficiency of the sorting process.
* **Algorithmic Trade-offs**: I gained a deeper understanding of the trade-offs between time complexity and space complexity. While Merge Sort is efficient in terms of time, it requires additional space, which I had to carefully manage during the optimization process.

One of the challenges I faced was managing the recursive calls efficiently to prevent excessive memory usage, especially when handling large datasets. Initially, the algorithm would consume too much memory during recursion, but I restructured the way the subarrays were merged and utilized memory more effectively.

Overall, the enhancement process helped solidify my understanding of algorithmic principles and reinforced the importance of considering performance and resource utilization when designing algorithms.