**Module 2: Portfolio Milestone**

**Portfolio Project Proposal**

Ryan Thompson

Colorado State University - Global

CSC 506

Dr. Holbert

8 September 2024

**Introduction**

After reviewing the provided portfolio project options, I find myself leaning towards the sorting algorithm comparator. I’ve always wanted to develop an algorithm comparator. Therefore, this seems to be the perfect opportunity to do so. I find the comparison between the algorithms interesting because of how they can be used to implement and optimize programs.

Sorting is a fundamental operation in computer science, essential for organizing data efficiently. As a result, numerous sorting algorithms have been developed, each with unique characteristics and performance implications. To evaluate and compare these algorithms, it is useful to build a sorting algorithm comparator program. This program allows for a systematic comparison of various sorting algorithms, providing insights into their relative efficiencies and practical applicability.

**Concept**

A sorting algorithm comparator program is designed to measure and compare the performance of different sorting algorithms under controlled conditions. The primary algorithms often compared include Bubble Sort, Merge Sort, and Quick Sort, each representing different approaches to sorting data. Bubble Sort is a simple comparison-based algorithm, Merge Sort is a divide-and-conquer algorithm, and Quick Sort is an efficient but complex algorithm that also uses divide-and-conquer principles.

The comparator program generally operates by implementing each sorting algorithm and then measuring their performance using a variety of input sizes and types. This approach enables the comparison of algorithms based on real execution times and resource usage.

**Implementation**

To build a sorting algorithm comparator, one must first implement the sorting algorithms. Bubble Sort repeatedly compares adjacent elements and swaps them if they are out of order, continuing until the list is sorted. This simplicity, however, comes at the cost of efficiency, as Bubble Sort has a time complexity of .

In contrast, Merge Sort is more efficient, operating with a time complexity of . It works by recursively dividing the list into smaller sublists, sorting those sublists, and then merging them back together in a sorted manner. This method ensures that Merge Sort performs well even with large datasets.

Quick Sort, another algorithm, selects a 'pivot' element and partitions the list into elements less than and greater than the pivot. It then recursively sorts the partitions. Quick Sort is often faster in practice compared to Merge Sort due to its in-place sorting and efficient partitioning, although its worst-case time complexity can reach if the pivot selection is poor.

Once the algorithms are implemented, the comparator program must gather performance data. This involves generating lists of varying sizes and types, including random, sorted, and reverse-sorted lists. The sorting algorithms are then executed, and the time taken for each sort operation is measured.

**Analysis**

After collecting execution times for the different sorting algorithms, the data is analyzed to compare their performance. Key factors include execution time as a function of input size and how each algorithm scales with increasing data volume. Bubble Sort's performance is expected to degrade significantly with larger inputs due to its quadratic time complexity, while Merge Sort and Quick Sort should show more favorable scaling due to their complexities.

The analysis also considers the efficiency of each algorithm in terms of memory usage. Merge Sort, while efficient in time, requires additional memory for merging operations. Quick Sort, on the other hand, is typically more memory-efficient as it sorts in place. Bubble Sort's memory usage is minimal but becomes impractical for large datasets due to its poor time complexity.

Additionally, the stability of the algorithms is evaluated. Stability refers to an algorithm's ability to preserve the relative order of equal elements. Merge Sort and Bubble Sort are stable, which can be crucial for certain applications where the order of equal elements matters. Quick Sort, in its basic form, is not stable, though stability can be achieved with modifications.

**Conclusion**

Building a sorting algorithm comparator program provides valuable insights into the performance and practical applicability of different sorting algorithms. Through systematic implementation and performance measurement, one can understand the relative strengths and weaknesses of algorithms like Bubble Sort, Merge Sort, and Quick Sort. This comparison highlights the trade-offs between time complexity, memory usage, and stability, guiding the selection of the most suitable sorting algorithm for a given application. Such a comparator is a powerful tool for both educational purposes and practical software development, enhancing our understanding of algorithmic efficiency and optimization.

**References**

Comparison of sorting algorithms. AfterAcademy. (n.d.). https://afteracademy.com/blog/comparison-of-sorting-algorithms/

GeeksforGeeks. (2024, January 23). Analysis of different sorting techniques. https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/