**Module 4: Portfolio Milestone**

**Sorting Algorithms**

Ryan Thompson

Colorado State University - Global

CSC 506

Dr. Holbert

17 September 2024

**Sorting Algorithms Differentiation**

Sorting algorithms are fundamental to the efficient management and retrieval of data in computer science. In my sorting algorithm comparison program, I implemented three well-known sorting algorithms: Bubble Sort, Merge Sort, and Quick Sort. This paper explores the operational principles of these algorithms, details the input and testing methods used in the comparison, and discusses the performance outcomes observed during the evaluation.

**Sorting Algorithms Overview**

Bubble Sort is one of the most elementary sorting algorithms, characterized by its simplicity and ease of understanding. It operates by repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order. This process is repeated until the list is sorted. Despite its straightforward approach, Bubble Sort has a time complexity of in the worst and average cases, which makes it impractical for large datasets. The algorithm’s main advantage lies in its ability to detect a sorted list early, potentially reducing the number of required operations in best-case scenarios.

Merge Sort, on the other hand, is a more sophisticated sorting algorithm based on the divide-and-conquer paradigm. It works by recursively dividing the input array into two halves, sorting each half, and then merging the sorted halves to produce a fully sorted array. Merge Sort has a time complexity of in both the average and worst cases, which ensures consistent performance even for large datasets. However, it requires additional space proportional to the size of the input array, which can be a drawback in memory-constrained environments.

Quick Sort is another divide-and-conquer algorithm renowned for its average-case efficiency. It sorts an array by selecting a 'pivot' element and partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot. These sub-arrays are then sorted recursively. Quick Sort typically exhibits a time complexity of on average, although it can degrade to in the worst case if the pivot selection is poor. Despite this potential for inefficiency, Quick Sort is often preferred in practice due to its in-place sorting and generally superior performance compared to other algorithms.

**Input and Testing Methods**

To evaluate the performance of Bubble Sort, Merge Sort, and Quick Sort, I designed a comprehensive testing framework that involved various input scenarios. The datasets used for testing included arrays of differing sizes, ranging from small to large, and various types, such as random arrays, nearly sorted arrays, and arrays with many duplicate elements. This diverse set of test cases was aimed at thoroughly assessing each algorithm’s efficiency across different conditions.

The testing methodology involved measuring the time taken by each algorithm to sort arrays of different sizes and types. I used Python’s built in timer technology to record these times accurately. Additionally, memory usage was monitored to evaluate space efficiency. To ensure the reliability of the results, each algorithm was tested multiple times with each dataset, and the results were sometimes averaged to mitigate the impact of any anomalies.

**Results and Analysis**

The results from the comparative study highlighted distinct differences in performance among the three algorithms. Bubble Sort consistently exhibited poor performance, particularly with larger arrays, due to its time complexity. While it performed adequately with small datasets, its inefficiency became apparent as the dataset size increased. Merge Sort demonstrated robust performance across all test cases, maintaining its time complexity and handling large datasets effectively, though at the cost of higher memory usage.

Quick Sort emerged as the most efficient algorithm on average, outperforming both Bubble Sort and Merge Sort in most scenarios. Its in-place sorting and average-case time complexity of contributed to faster sorting times. However, its performance varied depending on the choice of pivot, with some runs showing degraded performance.

**Conclusion**

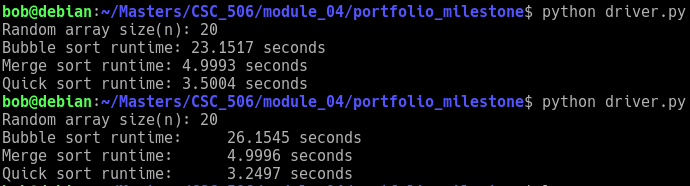
[The comparison of Bubble Sort, Merge Sort, and Quick Sort](https://drive.google.com/file/d/12SFeh1hOOLAOqVvrqA7oYH_kqNemZLIz/view?usp=drive_link) underscores the trade-offs involved in choosing a sorting algorithm. Bubble Sort, while simple, is inefficient for larger datasets. Merge Sort offers consistent performance and is well-suited for large data, despite its higher memory requirements. Quick Sort generally provides the best performance, although its efficiency can be influenced by pivot selection. This comparative study highlights the importance of selecting the appropriate algorithm based on the specific needs of the application and the characteristics of the data.

**Future Expansion**

In the previously linked [demo video](https://drive.google.com/file/d/12SFeh1hOOLAOqVvrqA7oYH_kqNemZLIz/view?usp=drive_link), you can see this study only compares three sorting algorithms, as only three were implemented. Fortunately, three algorithms appear to be an adequate amount in order to illustrate their respective complexities. However, going forward, there will be more sorting algorithms thrown into the comparison. Additionally, there will be multiple forms/types of input rather than a single array of random integers. Introducing different data types as well as more data in general, will allow for a deeper analysis on sorting algorithm comparisons. Furthermore, the program is destined to be expanded into a web based flask application. Which ought to aid in the visual comparison.



*Main Driver Script*



*Program Output*

**References**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching* (2nd ed.). Addison-Wesley.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.