**TOPIC:**

**STUDENT NAME:**

**UNIVERSITY NAME:**

**COURSE NAME:**

**INSTRUCTOR NAME:**

**DATE OF SUBMISSION:**

**Application Context and Selected Data Structures**

This project is a sorting algorithm visualizer designed for educational purposes, helping users explore and understand the behavior of different sorting algorithms interactively. The application uses Streamlit to create a web-based platform where users can select from five sorting methods—Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort—and observe the sorting process in real-time. The visualizations respond instantly to user inputs, making the learning experience intuitive and engaging, suitable even for beginners or younger audiences (Shaffer, 2022).

Streamlit’s user-friendly components, like dropdowns and sliders, make the interface dynamic and interactive. The user can choose an algorithm from a dropdown menu and set the array size using a slider. Based on these selections, the app generates a random array and begins sorting it step-by-step once the user initiates the sorting. This gradual visualization lets the user see each swap, partition, or merge as it happens, providing a hands-on view of how the chosen algorithm works internally.

The primary data structure used here is an array, represented as a Python list. Arrays are ideal for sorting visualizations since they support direct access to elements by index, enabling operations like comparison and swapping, which are fundamental to many sorting algorithms. As the array is sorted, users can witness the ongoing changes, gaining a deeper understanding of each algorithm's flow and mechanics.

**Rationale for Data Structure Choice**

Arrays are the best-suited data structure for this application due to their accessibility and versatility. Python lists (arrays) support direct access by index, enabling efficient element comparisons and swaps, which are core operations in sorting algorithms. For instance, in Bubble Sort, the algorithm continuously compares adjacent elements, swapping them as needed. Arrays allow constant-time access to elements, making this process straightforward and efficient (Kristo et al., 2020).

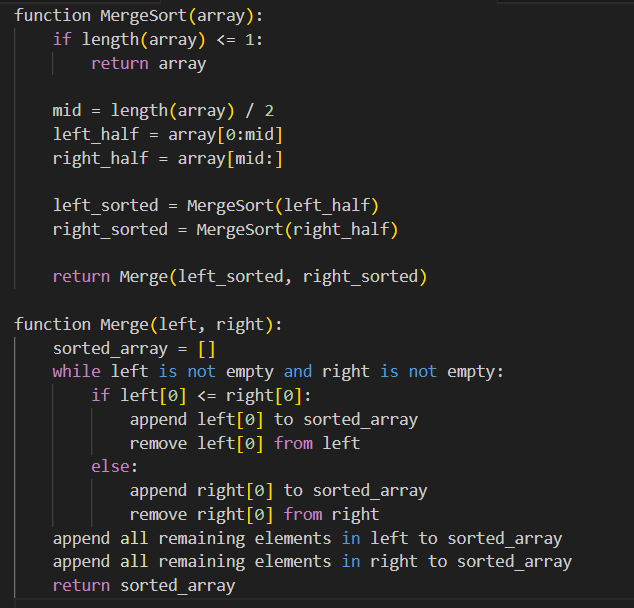
In algorithms like Merge Sort, arrays allow for easy subdivision, a critical step in the divide-and-conquer strategy. Merge Sort works by splitting the array into two halves, sorting each half, and then merging them back together. Python provides efficient methods for slicing arrays, which simplifies the process of creating subarrays and merging them, crucial for Merge Sort’s recursive operations.

Similarly, in Quick Sort, arrays facilitate efficient partitioning. The algorithm selects a pivot element and then organizes the array so that elements smaller than the pivot are on one side, and larger elements are on the other. Arrays allow straightforward swapping and rearrangement based on index positions, making partitioning easy to visualize.

**Overview of the Python Implementation**

The sorting algorithm visualizer uses Python and Streamlit to create an interactive interface. Users can select one of five sorting algorithms and adjust the array size via a slider. The program generates a random array according to the chosen parameters and visualizes the sorting process in real-time.

**Pseudo Code**

****

**Challenges and Limitations**

Performance Bottlenecks: Sorting visualizations for larger arrays may slow down due to frequent screen updates. Sorting algorithms like Bubble Sort, with high time complexities, can be particularly slow for larger data sets, impacting user experience.

Visual Clarity with Larger Arrays: With larger arrays, the display may become cluttered, making it challenging for users to track sorting steps accurately (Khoirom et al., 2020).

Complexity of Advanced Algorithms: Algorithms like Quick Sort and Merge Sort involve complex operations, such as partitioning and recursive merges. Simplified pseudocode may not fully represent the intricacies required in practice.

Scalability Constraints: The app is effective for small to medium array sizes but may require optimizations or different visualization techniques to handle larger data sets effectively.

These challenges highlight the importance of balancing educational clarity with performance and scalability to deliver an engaging and functional user experience.

**References**

Kristo, A., Vaidya, K., Çetintemel, U., Misra, S., & Kraska, T. (2020, June). The case for a learned sorting algorithm. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data* (pp. 1001-1016). <https://dl.acm.org/doi/pdf/10.1145/3318464.3389752>

Shaffer, C. A. (2022). Data structures and algorithm analysis. <http://123.200.31.10:8080/jspui/bitstream/123456789/105/1/Data%20Structure%20%26%20Algorithm%20Analysis.pdf>

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). Comparative analysis of Python and Java for beginners. *Int. Res. J. Eng. Technol*, *7*(8), 4384-4407. <https://www.academia.edu/download/94738677/IRJET-V7I8755.pdf>