Introduction:

In Week 6 of my Data Structures course, the primary focus was on internal sorting techniques. I tested both elementary sorting algorithms – such as Insertion Sort, Bubble Sort and Selection Sort – and advanced ones like Shellsort, Mergesort, Binsort, Heapsort, Quicksort, and Radix Sort. A large portion of this week's study was to know the working of each algorithm which had a focus on time complexities of each type of sorting algorithms used and how they are becoming efficient in sorting the given data. I developed a stronger grasp of how asymptotic analysis is applied to compare these algorithms based on best, average, and worst-case scenarios. The most in-depth concept I engaged with was Quicksort, given its divide-and-conquer nature and strong practical efficiency.

Difficulties Faced:

One of the challenges I encountered was differentiating between the multiple sorting algorithms, especially remembering their unique characteristics and performance metrics. Understanding why some algorithms perform better under certain conditions than others require deeper engagement with the theoretical underpinnings. Additionally, implementing Quicksort in Java and integrating the exchange count logic added a layer of complexity, particularly when debugging recursion and ensuring the pivot partitioning logic worked as expected. I also had to overcome confusion surrounding the worst-case scenarios for algorithms like Quicksort and how pivot selection can drastically affect performance.

Activities Performed:

During this week, I participated in a discussion post where I analyzed Quicksort and elaborated on the importance of pivot selection. I explored techniques such as the median-of-three and randomized pivot selection to avoid worst-case $O(n^2)$ scenarios (Sedgewick & Wayne, 2011). I also submitted a programming assignment implementing Quicksort to sort an array of 21 integers while counting the number of exchanges. This task helped reinforce my understanding of recursion, in-place sorting, and efficiency considerations. The output demonstrated how Quicksort outperforms Insertion Sort significantly, requiring only 24 exchanges compared to Insertion Sort's 114 (Cormen et al., 2009). This concrete implementation bridged the gap between theory and practice, allowing me to better internalize sorting algorithm efficiency.

Conclusion:

This week's focus on sorting algorithms was both theoretically rich and practically challenging. I now feel confident in not only explaining how sorting algorithms work but also implementing them and analyzing their efficiency. The hands-on experience with Quicksort especially clarified how algorithm design choices, such as pivot selection, influence performance and memory usage.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. https://mitpress.mit.edu/9780262533058/introduction-to-algorithms/

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley. https://www.amazon.com/Algorithms-4th-Robert-Sedgewick/dp/032157351X

Wordcount: 384