Introduction

In this coursework, we delve into the performance of two classic sorting algorithms: BubbleSort and MergeSort. These algorithms serve as fundamental building blocks in computer science, each with unique characteristics that make them suitable for different scenarios.

Algorithm Complexity Analysis

BubbleSort is a simple comparison-based algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The algorithm continues to pass through the list until no swaps are needed, indicating that the list is sorted. BubbleSort has a time complexity of O(n^2) in the average and worst-case scenarios, and a space complexity of O(1), as it only requires a single additional memory space for the comparison.

MergeSort, on the other hand, is a divide-and-conquer algorithm that divides the input array into two halves, calls itself for the two halves, and then merges the sorted halves. The time complexity of MergeSort is O(n log n) in all cases, and its space complexity is O(n) due to the temporary arrays used during the merge process. MergeSort's efficiency in handling large datasets stems from its logarithmic growth rate, making it superior to BubbleSort for sizable lists.

Performance Test Results

The performance of these sorting algorithms was tested using files with varying numbers of card elements: sort10.txt, sort100.txt, and sort10000.txt. The results, as tabulated in the sortComparison.csv file, reveal significant differences in execution times.

Chart: BubbleSort vs. MergeSort Performance

Chart Description: This 2D line plot illustrates the execution times of BubbleSort and MergeSort across three different dataset sizes. The x-axis represents the number of elements (10, 100, 10000), and the y-axis represents the execution time in milliseconds. The blue line represents BubbleSort, and the red line represents MergeSort. As the graph shows, the performance gap between the two algorithms widens with increasing dataset sizes.