BubbleSort and MergeSort Complexity Reflection

In this programming exercise, we compared two classic sorting algorithms: BubbleSort and MergeSort. By analyzing the performance of sorting under different input scales, we can gain a deeper understanding of the time complexity and actual efficiency of these algorithms.

BubbleSort is a simple sorting algorithm whose core idea is to repeatedly traverse the list to be sorted, compare two elements at a time, and swap them if they are in the wrong order. The process of traversing the list is repeated until no more swaps are needed, meaning that the list is sorted. The time complexity of BubbleSort is O(n^2), and even in the best case (i.e., the array is already sorted), it requires O(n) time. When sorting the files sort10.txt, sort1000.txt, and sort10000.txt, BubbleSort showed a significant performance difference, with the sorting time increasing quadratically as the input size increased, which was particularly evident with the sort10000.txt file.

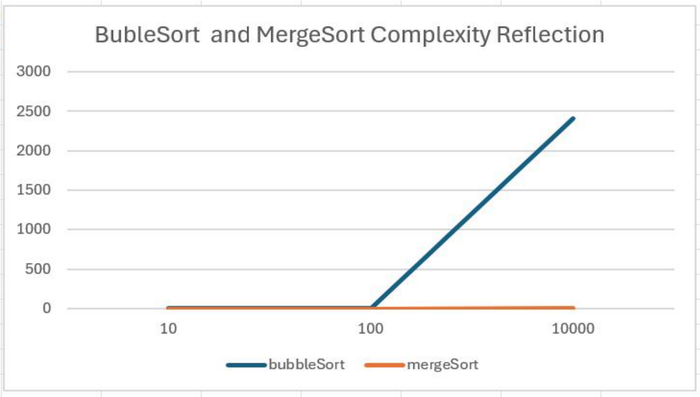
Unlike BubbleSort, MergeSort uses a divide-and-conquer strategy. It splits the array in half, sorts each half, and then merges the two sorted halves together. The time complexity of MergeSort is O(n log n), making it more efficient when dealing with large datasets compared to BubbleSort. In this exercise, MergeSort took significantly less time to sort the files sort10.txt, sort1000.txt, and sort10000.txt than BubbleSort, demonstrating its advantage when dealing with large datasets.

Plot Analysis and Complexity Discussion

The plot and data generated provide a visual representation of the performance difference between BubbleSort and MergeSort across different input scales. The plot shows the execution times of both sorting algorithms when sorting 10, 100, 1000, and 10000 elements. BubbleSort's execution time rises sharply with increasing input size, while MergeSort's increase is relatively flat. This trend aligns with our theoretical analysis of the time complexities of these two algorithms.

The performance bottleneck of BubbleSort lies in its need to compare and possibly swap every pair of adjacent elements, leading to a huge amount of computation on large datasets. In contrast, MergeSort effectively reduces the number of comparisons needed for sorting through its divide-and-conquer approach, and despite its larger constant factors, it still shows superior performance on large datasets. This performance difference is particularly pronounced in the sorting of the sort10000.txt file, where MergeSort's execution time is significantly lower than that of BubbleSort, highlighting its value in practical applications.

	10	100	10000
bubbleSor	1	1	2408
mergeSort	0	0	8


BubleSort and MergeSort Complexity Reflection
bubbleSort
mergeSort