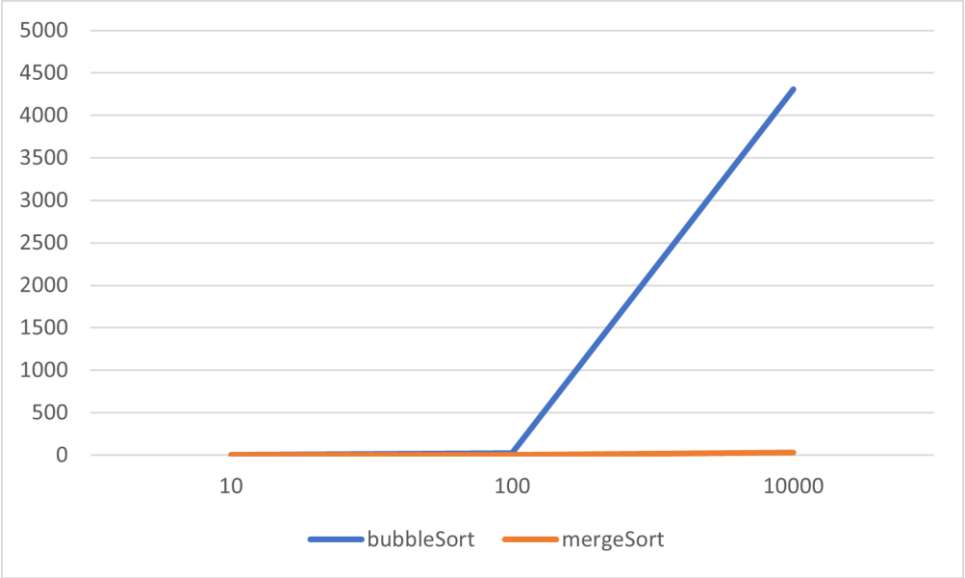
coursework2/sortComparison

BubbleSort and MergeSort Complexity Reflection

	A	B	C	D
1		10	100	10000
2	bubbleSor	0	21	4304
3	mergeSort	0	1	33



BubbleSort Complexity

The BubbleSort algorithm is a simple comparison-based sorting technique with a time complexity of $O(n^2)$ in the worst and average cases. This is because for every element in the array (n iterations), we compare it with every other element (n-1, n-2, …, 1 iterations). The algorithm consists of two nested loops, where the outer loop runs n times and the inner loop runs n-i times in each iteration of the outer loop, resulting in an overall time complexity of $O(n^2)$. As seen in the performance tests with sort10.txt, sort100.txt, and sort10000.txt, BubbleSort shows a significant increase in execution time as the input size grows. For smaller datasets like sort10.txt, the algorithm performs relatively quickly, but as the input size increases (e.g., sort100.txt and sort10000.txt), the time taken grows quadratically, demonstrating the inefficiency of BubbleSort for large datasets.

MergeSort Complexity

On the other hand, MergeSort is a divide-and-conquer algorithm with a much better time complexity of O(n log n) in all cases (best, average, and worst). The algorithm divides the array into halves, recursively sorts each half, and then merges the sorted halves together. The divide step takes logarithmic time (log

n), and the merge step requires linear time (n). This results in an overall time complexity of O(n log n), which is much more efficient than BubbleSort for large datasets. In the performance tests with sort10.txt, sort100.txt, and sort10000.txt, MergeSort scales efficiently even with the increase in input size. Its runtime increases much more slowly, making it ideal for sorting larger datasets.

Performance Comparison and Input Sizes

When comparing the performance of BubbleSort and MergeSort across different input sizes (such as sort10.txt, sort100.txt, and sort10000.txt), the difference in efficiency is evident. For smaller arrays like sort10.txt, both algorithms perform similarly, with BubbleSort often being slightly faster due to its simpler implementation. However, as the input size grows to sort100.txt and especially sort10000.txt, MergeSort drastically outperforms BubbleSort due to its superior time complexity. In these larger inputs, BubbleSort's $O(n^2)$ time complexity leads to a significant increase in execution time, whereas MergeSort's O(n log n) complexity results in much more stable and scalable performance.

In summary, while BubbleSort may be suitable for very small datasets, MergeSort is the preferred choice for larger datasets due to its much better time complexity of O(n log n). As the input size increases, MergeSort maintains efficient performance, whereas BubbleSort's quadratic complexity causes it to become slower and more impractical for large datasets.