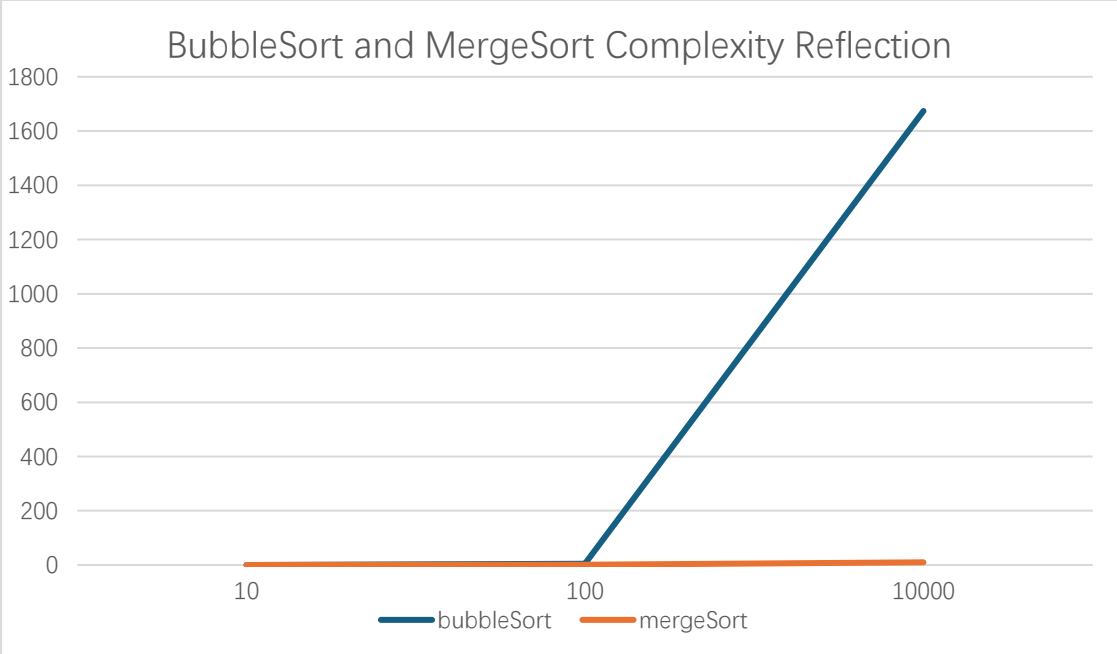
BubbleSort and MergeSort Complexity Reflection

The experimental results clearly demonstrate the significant performance difference between BubbleSort and MergeSort algorithms when sorting card sequences of varying sizes (10, 100, and 10,000 elements). For small inputs (n=10, n=100), both algorithms perform similarly, completing the sort in negligible time. However, as the input size grows, the performance gap becomes increasingly pronounced.

BubbleSort shows a dramatic increase in execution time as the input size grows, taking 0ms for 10 elements, 5ms for 100 elements, and jumping to 1674ms for 10,000 elements. This aligns with BubbleSort's theoretical time complexity of $O(n^2)$, where n is the input size. The quadratic growth is evident in the experimental results, as the execution time grows exponentially with the input size. This is because BubbleSort performs n-1 passes through the array, and in each pass, it makes up to n-1 comparisons and swaps, leading to poor performance on larger datasets.

In contrast, MergeSort demonstrates superior performance, especially for larger inputs, with execution times of 0ms, 1ms, and 10ms for 10, 100, and 10,000 elements respectively. This efficiency reflects MergeSort's theoretical time complexity of $O(n \log n)$. The algorithm's divide-and-conquer approach, which recursively splits the input into smaller sublists and merges them in sorted order, proves to be significantly more efficient than BubbleSort's repeated comparison-and-swap approach. The experimental results validate the theoretical advantage of MergeSort, showing that it scales much better with increased input sizes, making it a more suitable choice for sorting large datasets.



	10	100	10000
bubbleSort	0	5	1674
mergeSort	0	1	10