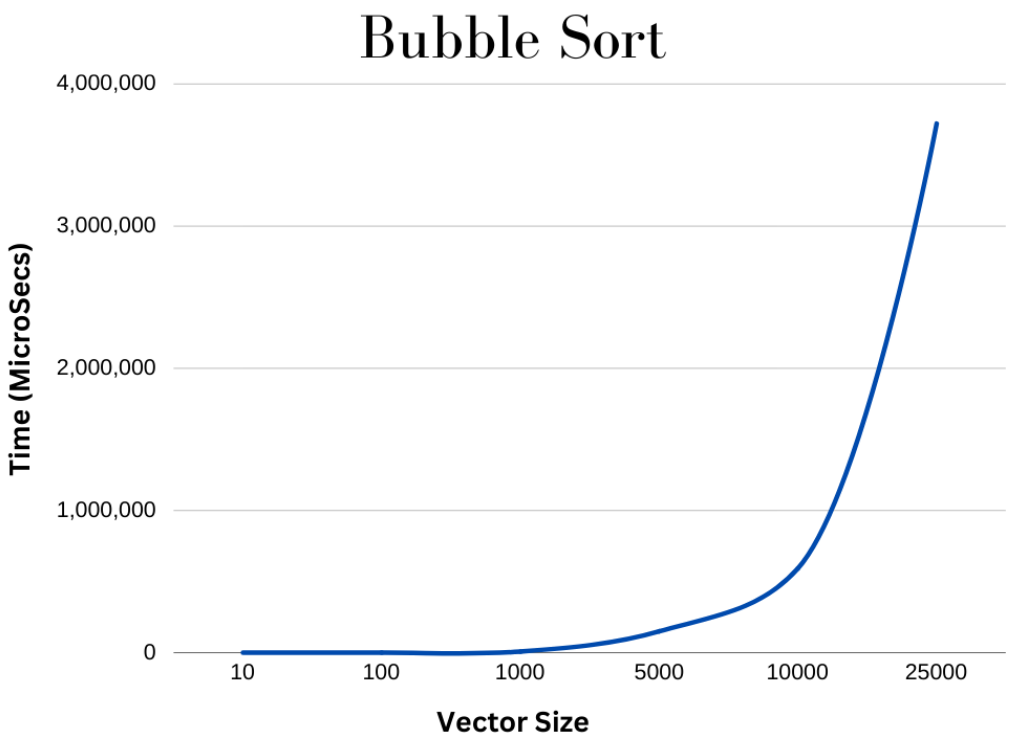
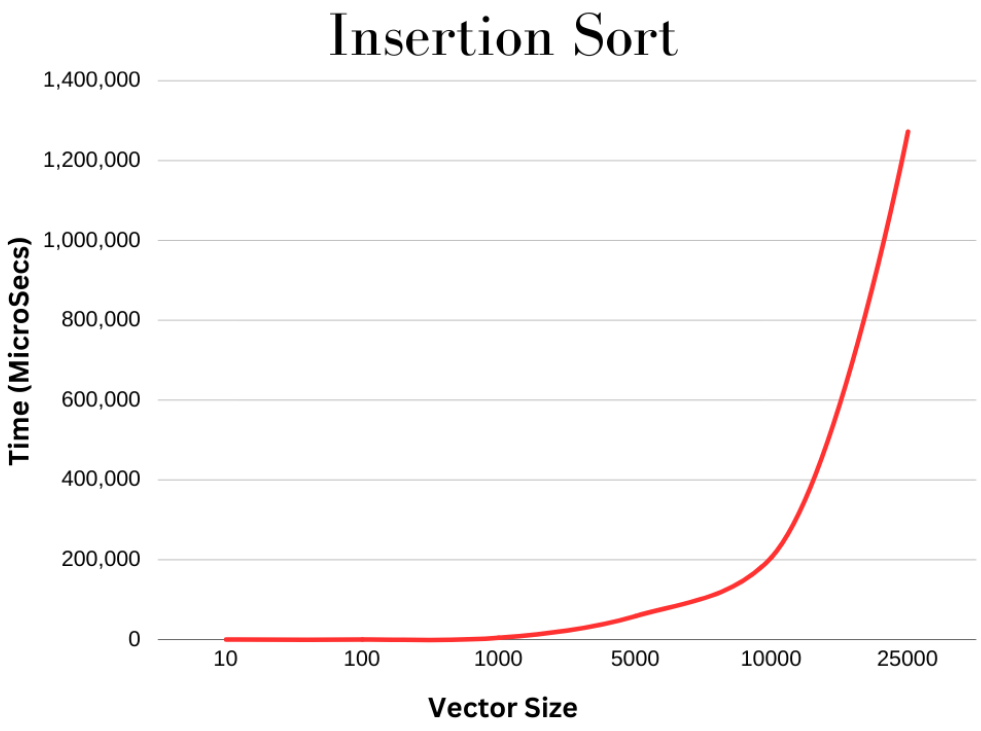
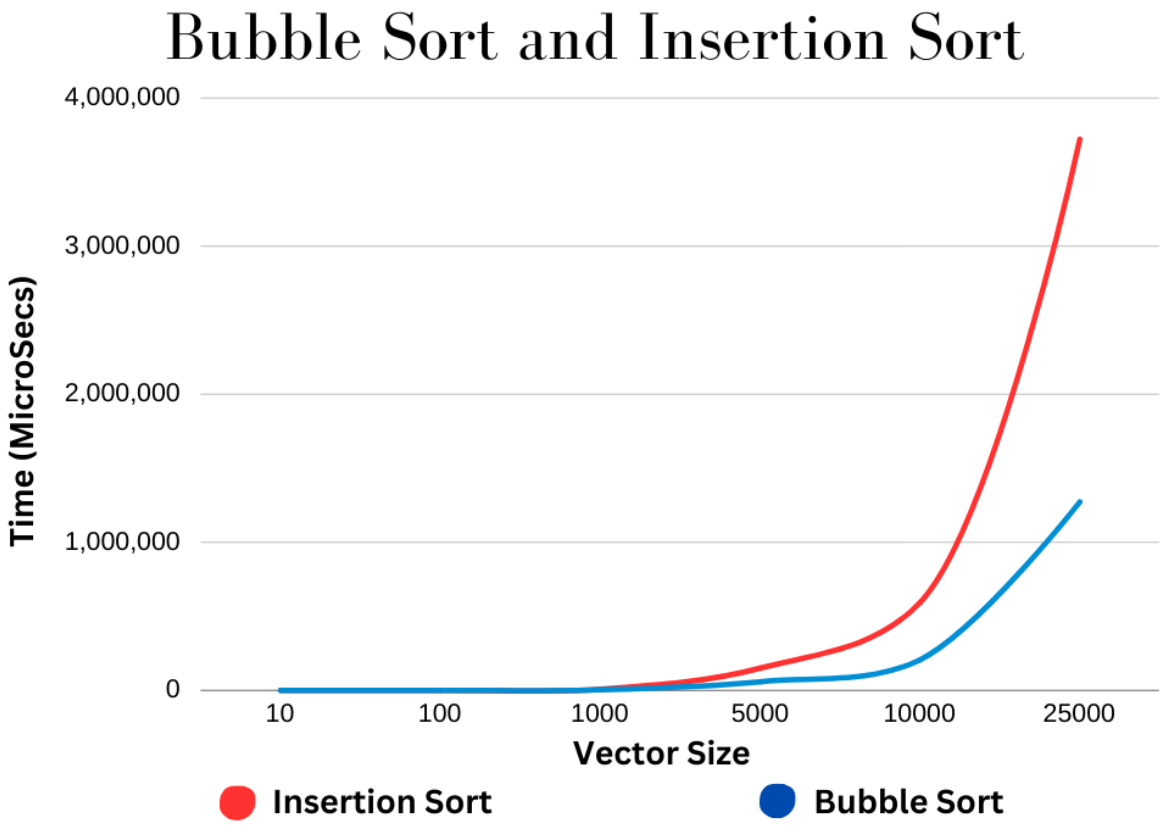
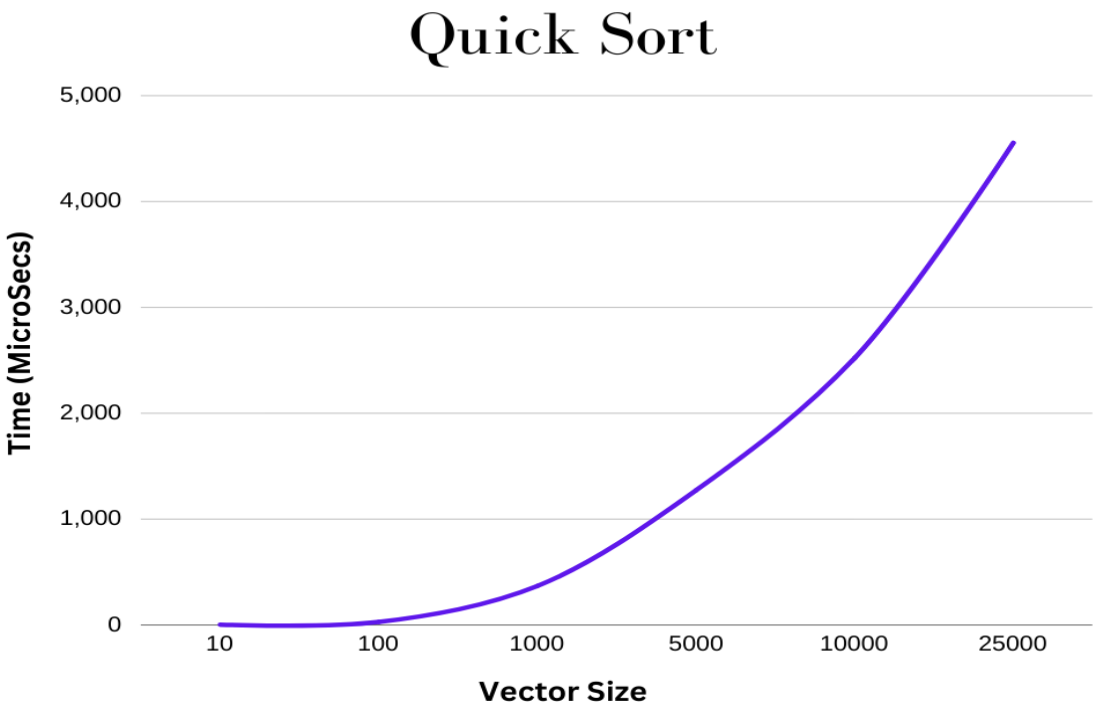
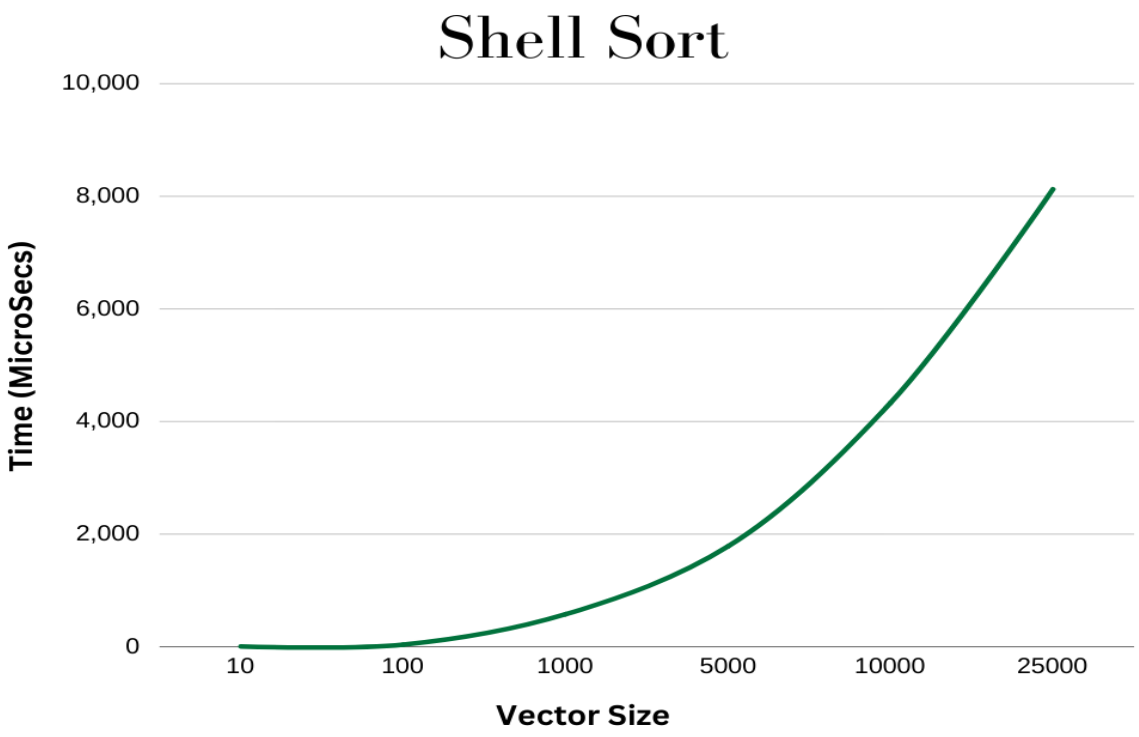
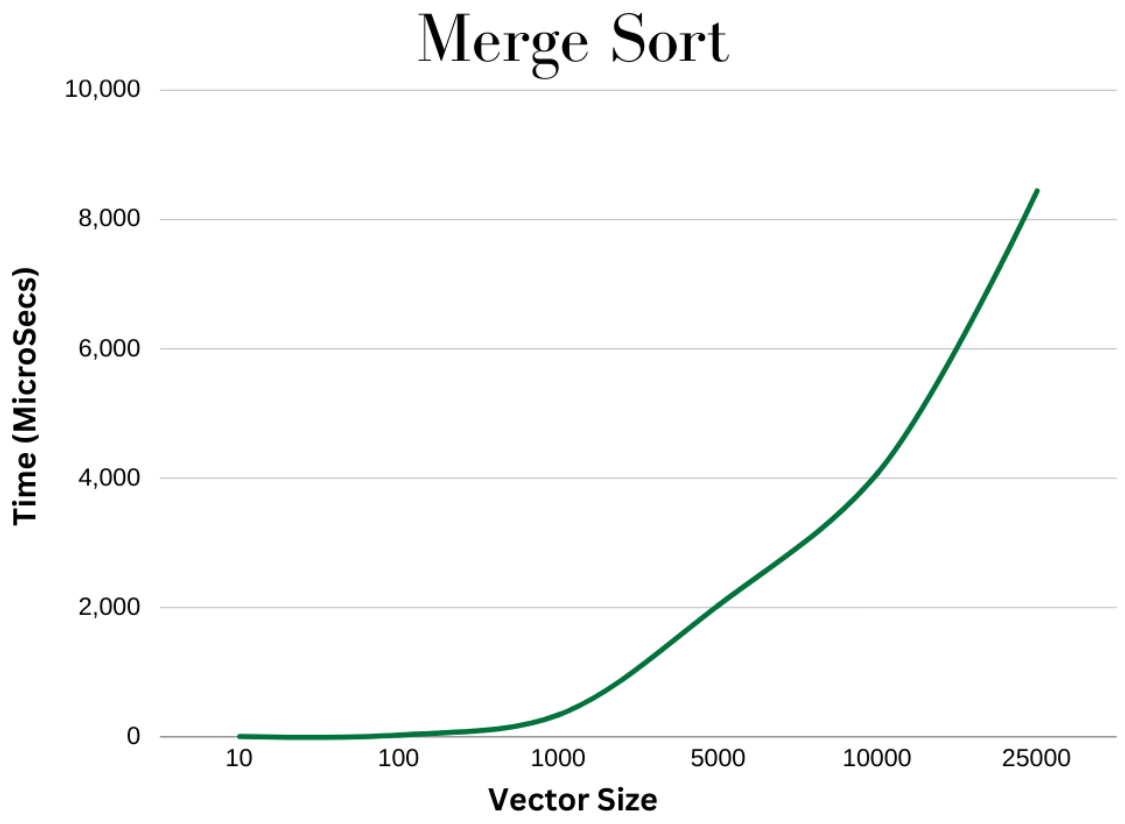

Bubble Sort
4,000,000
3,000,000
2,000,000
1,000,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size

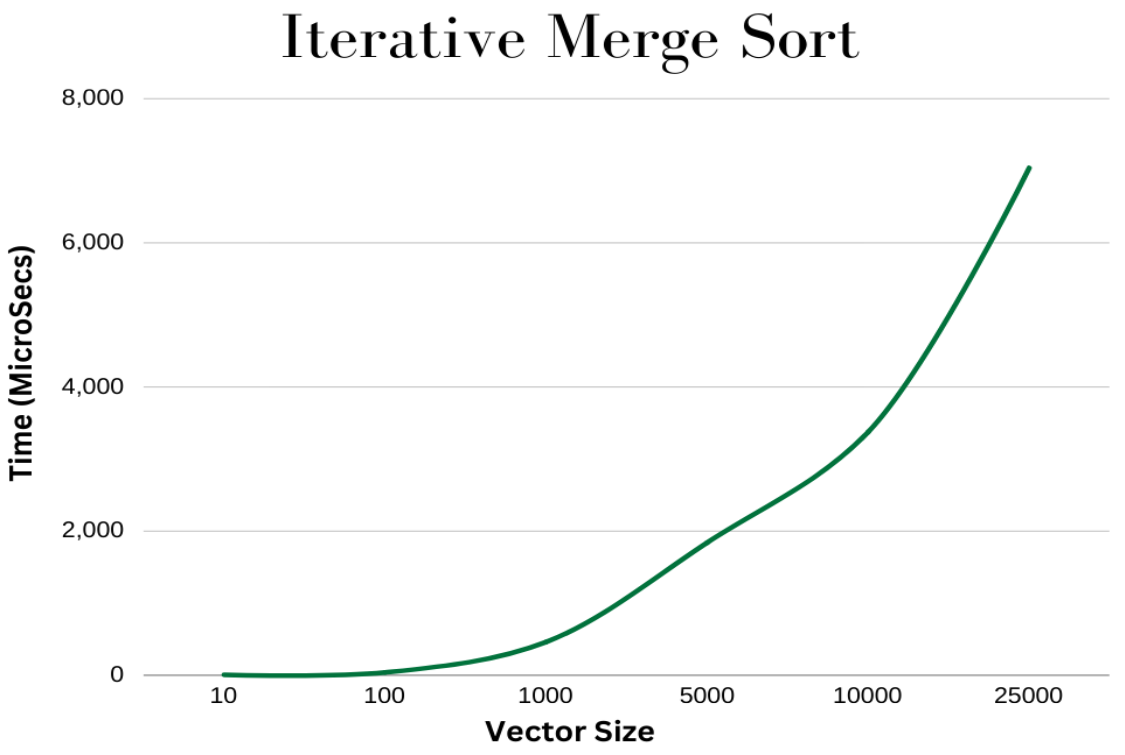

Insertion Sort
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size

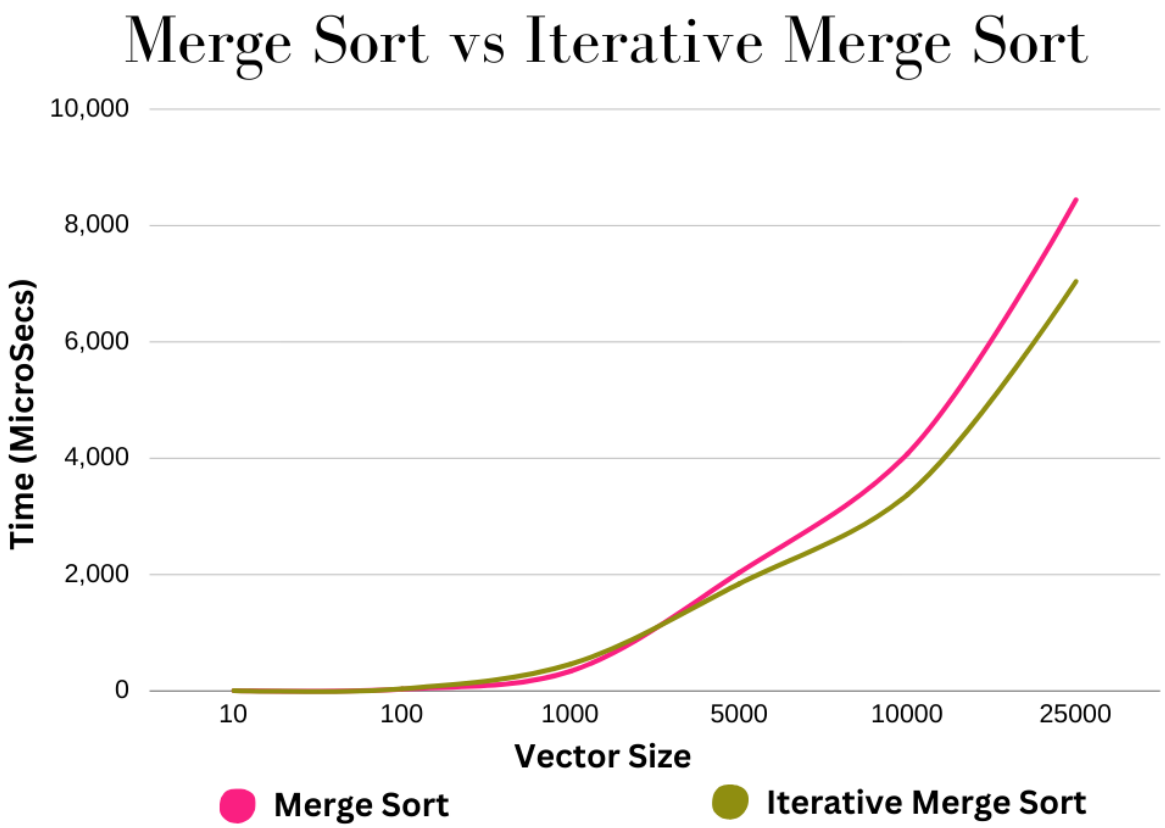

Bubble Sort and Insertion Sort
4,000,000
3,000,000
2,000,000
1,000,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size
Insertion Sort
Bubble Sort

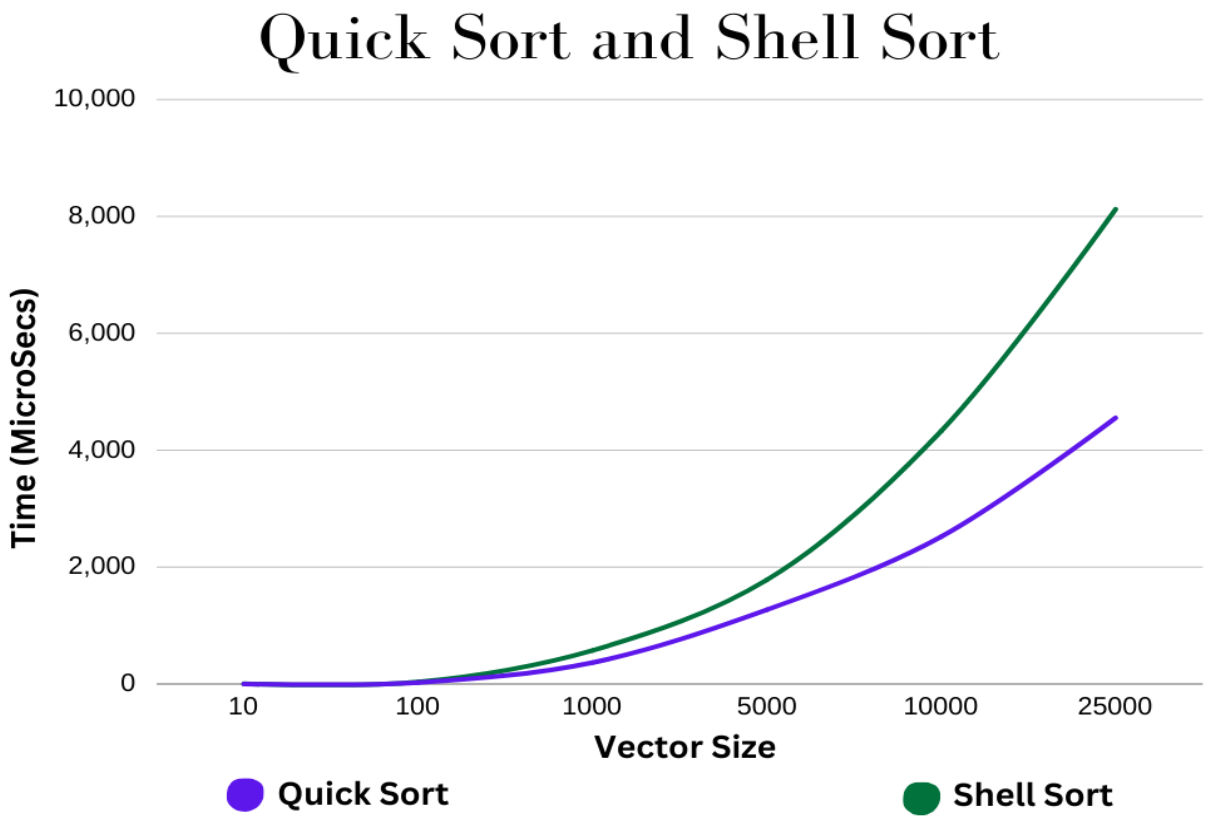

Quick Sort
5,000
4,000
3,000
2,000
1,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size

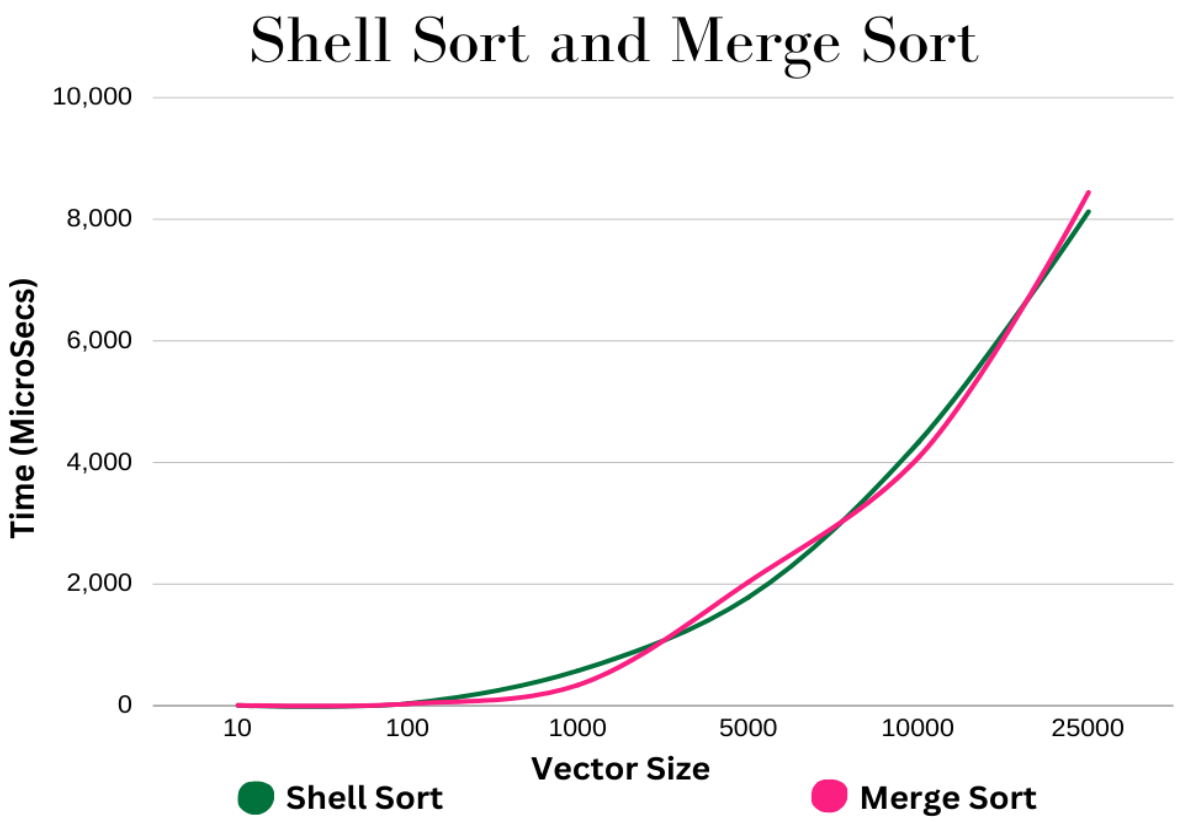

Shell Sort
10,000
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size

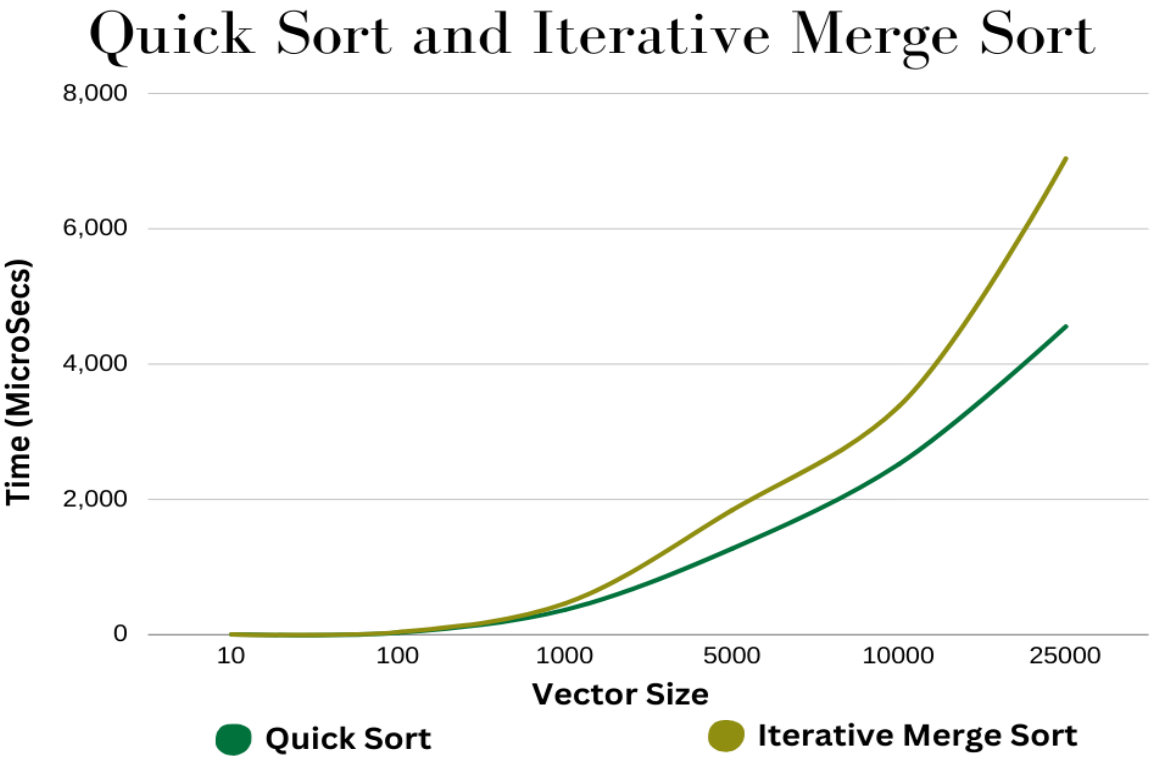

Merge Sort
10,000
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size

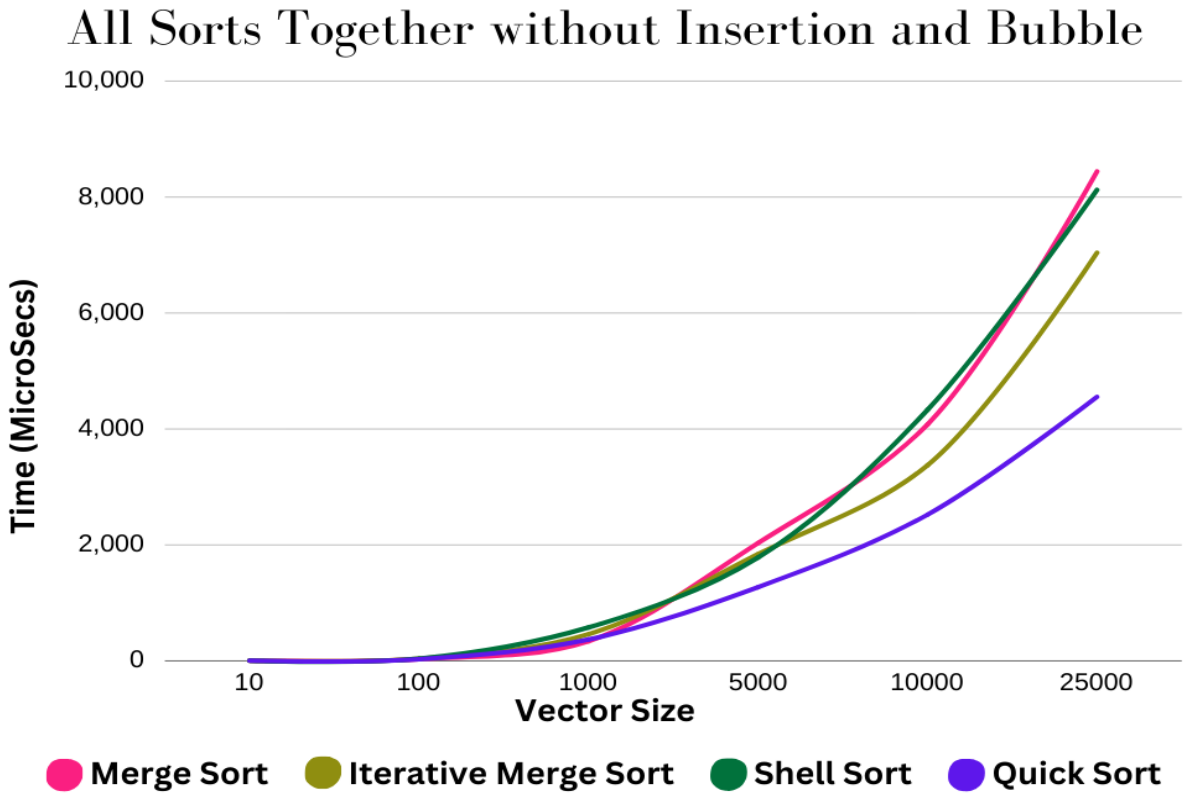

Iterative Merge Sort
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size

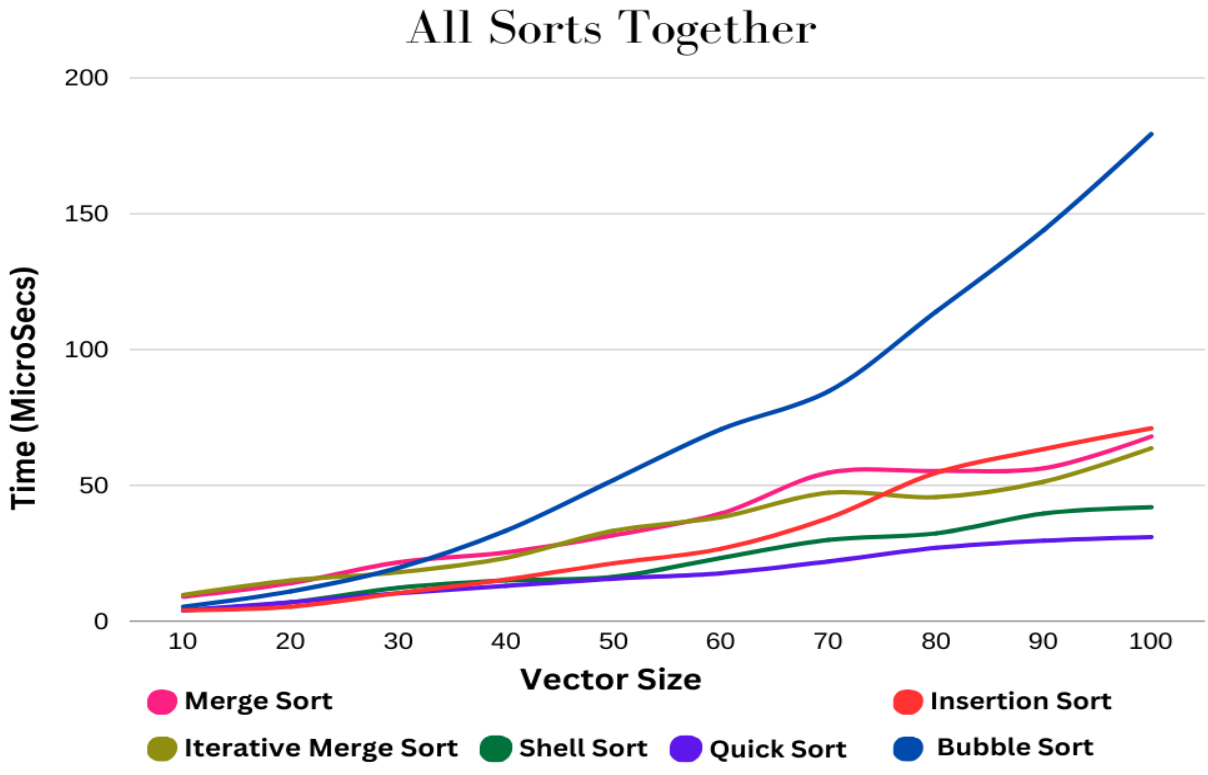

Merge Sort vs Iterative Merge Sort
10,000
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size
Merge Sort
Iterative Merge Sort


Quick Sort and Shell Sort
10,000
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size
Quick Sort
Shell Sort


Shell Sort and Merge Sort
10,000
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size
Shell Sort
Merge Sort


Quick Sort and Iterative Merge Sort
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size
Quick Sort
Iterative Merge Sort


All Sorts Together without Insertion and Bubble
10,000
8,000
6,000
4,000
2,000
0
Time (MicroSecs)
10
100
1000
5000
10000
25000
Vector Size
Merge Sort
Iterative Merge Sort
Shell Sort
Quick Sort


All Sorts Together
200
150
100
50
0
Time (MicroSecs)
10
20
30
40
50
60
70
80
90
100
Vector Size
Merge Sort
Insertion Sort
Iterative Merge Sort
Shell Sort
Quick Sort
Bubble Sort

Report

Table BigO of Sorted Algorithm

Sort Algorithm	Time Complexity (Average)	Space Complexity (Worst)
Bubble	Θ(n^2)	O(1)
Insertion	Θ(n^2)	O(1)
Merge	Θ(n log(n))	O(n)
Iterative Merge	Θ(n log(n))	O(n)
Quick	Θ(n log(n))	O(log n)
Shell	Θ(n (log(n)) ^2)	O(1)

Explanation why iterative Merge is Θ(n log(n)) in time complexity and O(n) in space complexity in my code:
Time complexity
1.The outer loop runs log(n) times since i is doubled in each iteration until it surpasses n.
2.Because each piece is processed only once, the inner loop executes in linear time O(n).
3.In the merging stage, the temporary vector is copied back to the original vector, which takes O(n) time.
The total time complexity is therefore O(log n) * O(n) * O(n) = O(n log n).

Space complexity
1. The function uses a temp_vec of size vec_size (where vec_size is the number of elements to be sorted). As a result, the needed space for this vector is O(n).
2. Other variables in the function (such as loop counters and indices) are constant in terms of input size and add little to space complexity.
As a result, the space utilized by the temp_vec is the main component in the space complexity, making the overall space complexity O(n).

Insertion sort is better than Bubble Sort
Insertion sort is often preferred over bubble sort due to its adaptive nature, efficient performance on small data sets, simple implementation, and suitability for online sorting. While both are stable sorting algorithms, insertion sort's advantages make it a more practical choice in many scenarios.

Iterative merge sort is better than merge sort
For numerous reasons, iterative merge sort is typically regarded as more efficient than classic recursive merge sort. It eliminates the expense of recursive function calls, which makes it more memory-efficient, and its iterative structure can lead to higher cache locality, which improves overall speed, especially for big data sets.

Quick Sort is better than shell sort
Quicksort outperforms shell sort on average, however shell sort is more efficient when the provided data is already/almost sorted. Quicksort requires stack calls, whereas shell sort does not.

Shell sort and Iterative merge sort

Shell sort is a more memory-efficient in-place sorting algorithm with a time complexity of O(n) to O(n2), making it suitable for moderate-sized datasets. For bigger datasets, Iterative Merge Sort, having a time complexity of O(n log n), is typically more efficient than Shell sort, but it requires additional memory proportional to the input size. Shell sort is easier to construct and in-place, making it suited for memory-constrained applications. The decision between them is determined by criteria such as dataset size, memory constraints, and implementation simplicity.

Quicksort outperforms all other sort algorithms for this project in terms of performance.