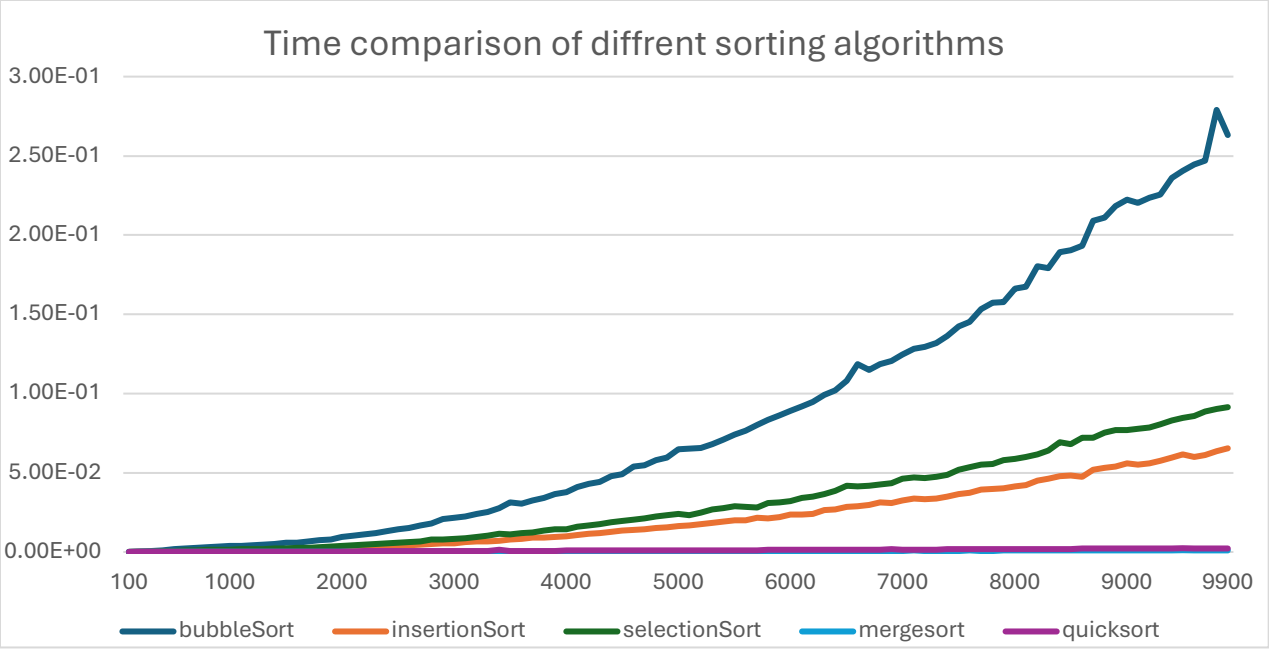
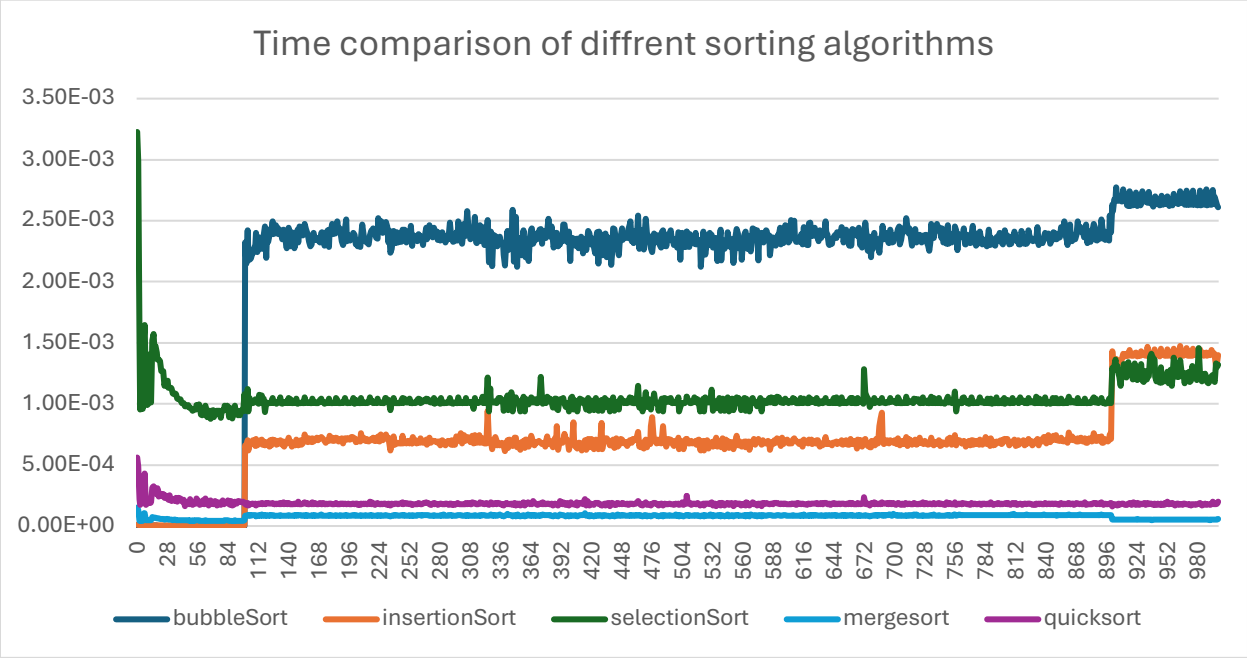

Time comparison of diffrent sorting algorithms
3.00E-01
2.50E-01
2.00E-01
1.50E-01
1.00E-01
5.00E-02
0.00E+00
100
1000
2000
3000
4000
5000
6000
7000
8000
9000
9900
bubbleSort
insertionSort
selectionSort
mergesort
quicksort


Time comparison of diffrent sorting algorithms
3.50E-03
3.00E-03
2.50E-03
2.00E-03
1.50E-03
1.00E-03
5.00E-04
0.00E+00
0
28
56
84
112
140
168
196
224
252
280
308
336
364
392
420
448
476
504
532
560
588
616
644
672
700
728
756
784
812
840
868
896
924
952
980
bubbleSort
insertionSort
selectionSort
mergesort
quicksort

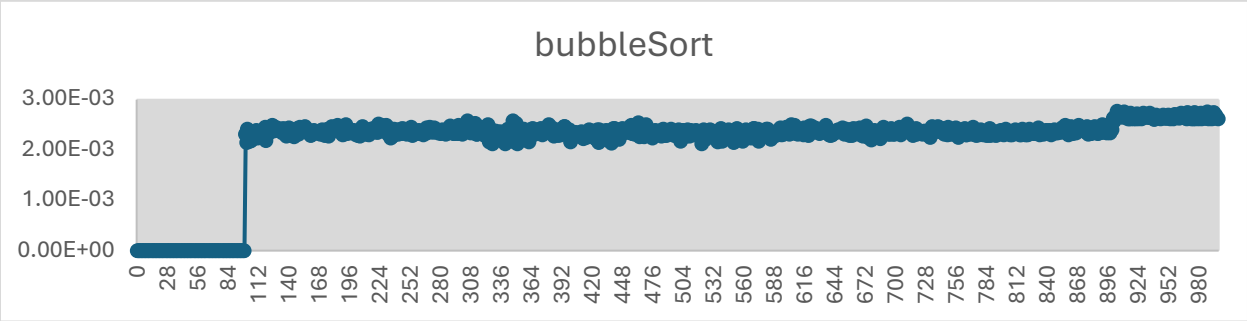
Sorting Algorithms Analysis Report

Overview

This report analyzes the performance of five sorting algorithms: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, and Quick Sort. The analysis is based on their performance with a specific data arrangement where:

- First 100 elements are sorted
- Middle 800 elements are random
- Last 100 elements are sorted in reverse order

1. Bubble Sort Analysis



Performance Characteristics

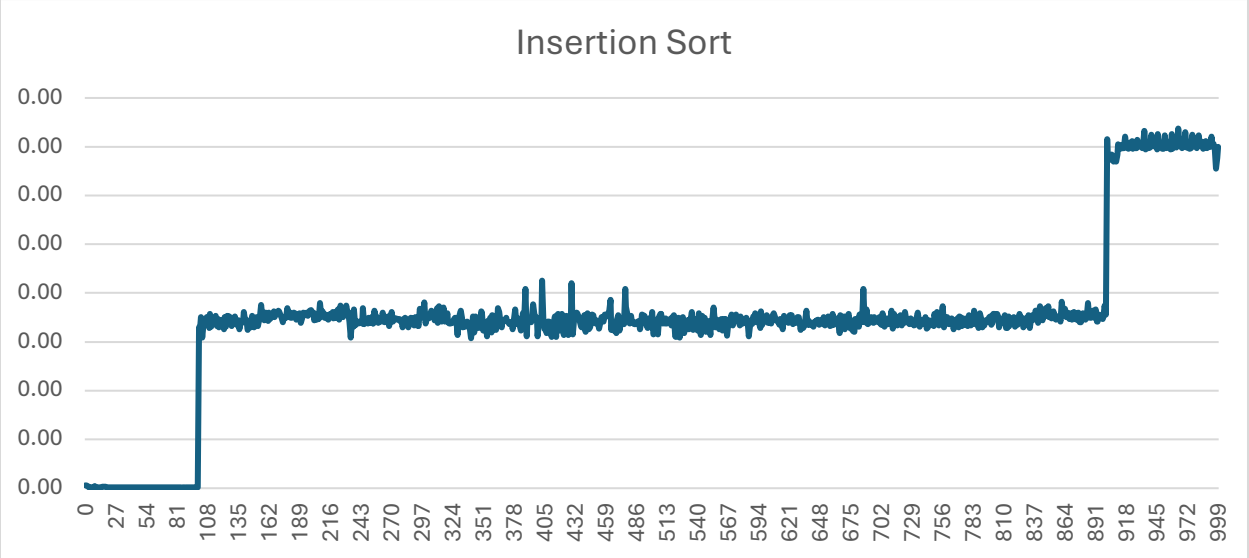
- Time Complexity:
  - Best Case: O(n) - when array is already sorted
  - Average Case: $O(n^2)$
  - Worst Case: $O(n^2)$
- Space Complexity: O(1)

Observed Performance

From the graphs, Bubble Sort shows:

- Highest overall execution time among all algorithms
- Quadratic growth pattern as input size increases
- Performance degrades significantly with larger datasets
- Shows consistent poor performance even with partially sorted data
- fastest with sorted array and slowest with unsorted or reversed array

2. Insertion Sort Analysis



Performance Characteristics

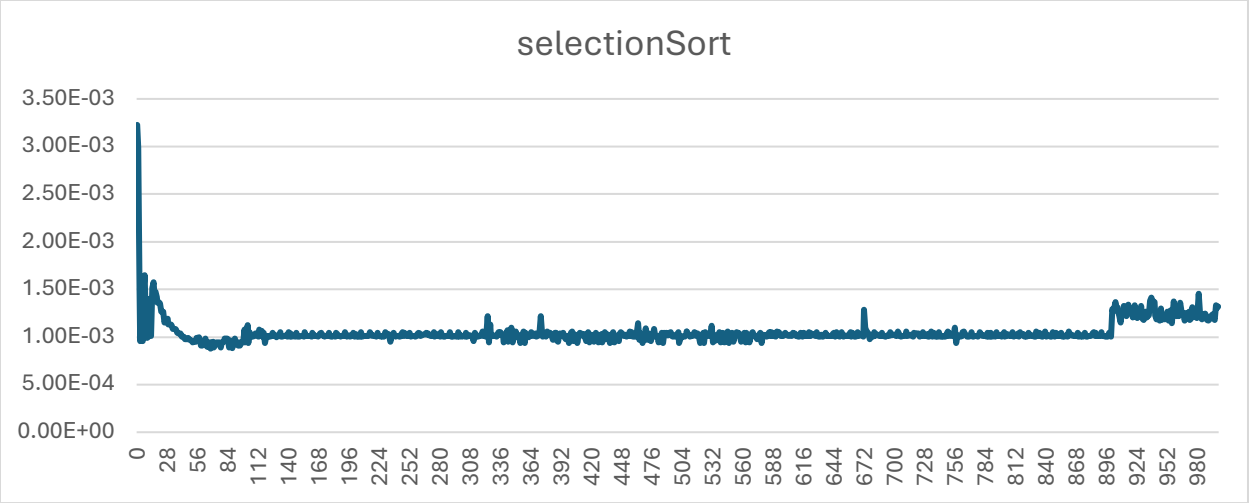
- Time Complexity:
  - Best Case: O(n) - when array is already sorted
  - Average Case: $O(n^2)$
  - Worst Case: $O(n^2)$
- Space Complexity: O(1)

Observed Performance

The graphs indicate:

- Better performance than Bubble Sort
- Benefits from partially sorted sequences
- Shows stepped pattern in execution time
-

3. Selection Sort Analysis



Performance Characteristics

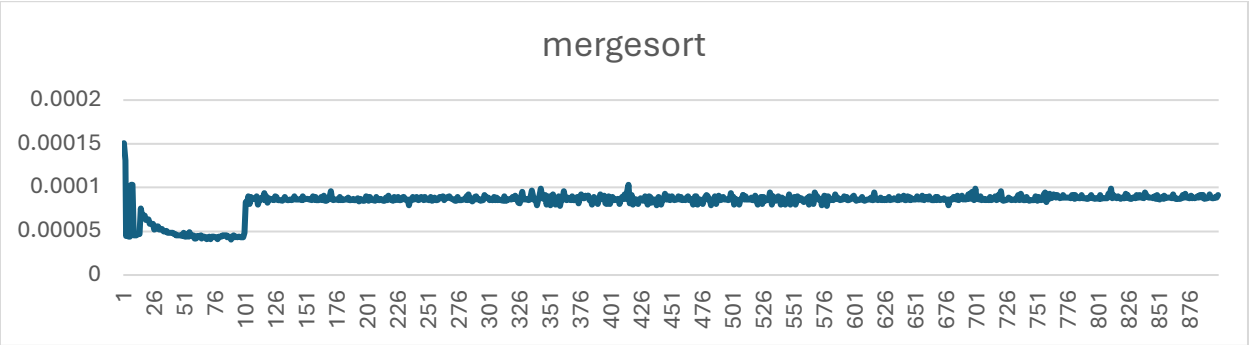
- Time Complexity:
  - Best Case: $O(n^2)$
  - Average Case: $O(n^2)$
  - Worst Case: $O(n^2)$
- Space Complexity: O(1)

Observed Performance

From the data:

- More consistent performance than Bubble and Insertion Sort
- Less affected by initial data arrangement
- Shows steady increase in time with input size
-

4. Merge Sort Analysis



Performance Characteristics

- Time Complexity:
  - Best Case: O(n log n)
  - Average Case: O(n log n)
  - Worst Case: O(n log n)
- Space Complexity: O(n)

Observed Performance

The graphs show:

- Most consistent performance among all algorithms
- Logarithmic growth pattern clearly visible
- Minimal variation in execution time
- Not significantly affected by input arrangement

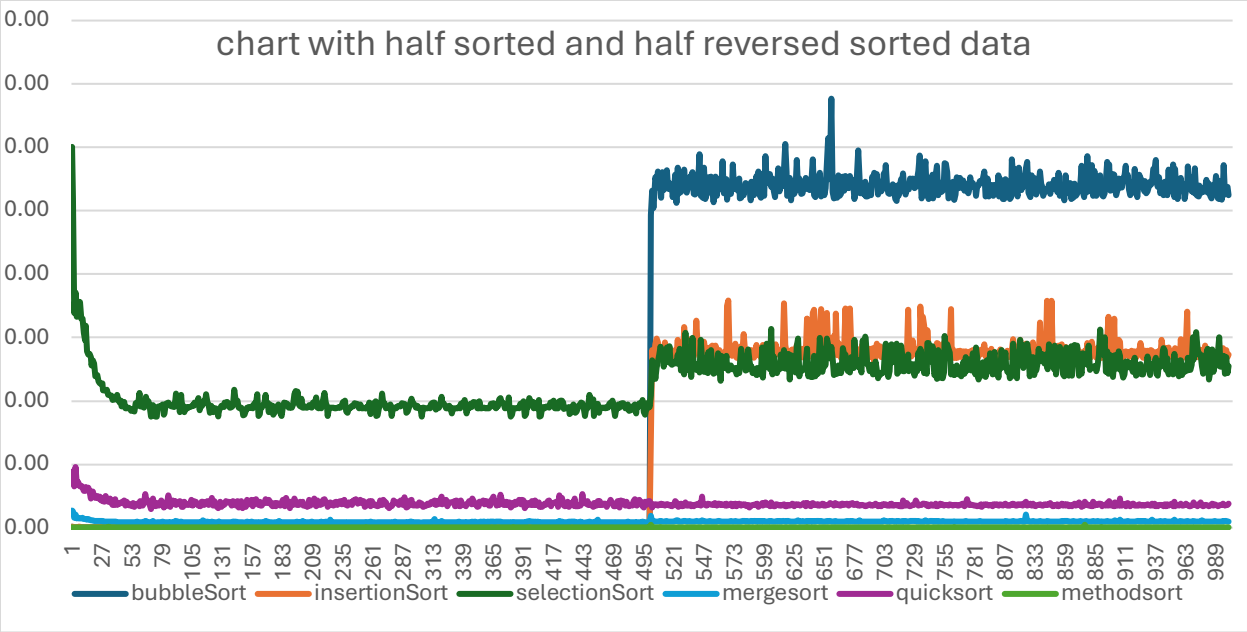
5. Quick Sort Analysis

Performance Characteristics
- Time Complexity:
  - Best Case: O(n log n)
  - Average Case: O(n log n)
  - Worst Case: $O(n^2)$
- Space Complexity: O(log n)

Observed Performance
From the analysis:
- Best overall performance in terms of execution time
- Very stable performance across different input sizes
- Minimal impact from partial sorting
- Maintains consistently low execution time
- Shows best practical performance among all algorithms

Comparative Analysis

Performance Ranking (Best to Worst):
1. Quick Sort
2. Merge Sort
3. Insertion Sort
4. Selection Sort
5. Bubble Sort



Key Observations:

1. Algorithmic Efficiency:
   - Divide-and-conquer algorithms (Quick Sort, Merge Sort) significantly outperform simple comparison sorts
   - Simple algorithms show quadratic growth with input size
   - Advanced algorithms maintain logarithmic growth patterns

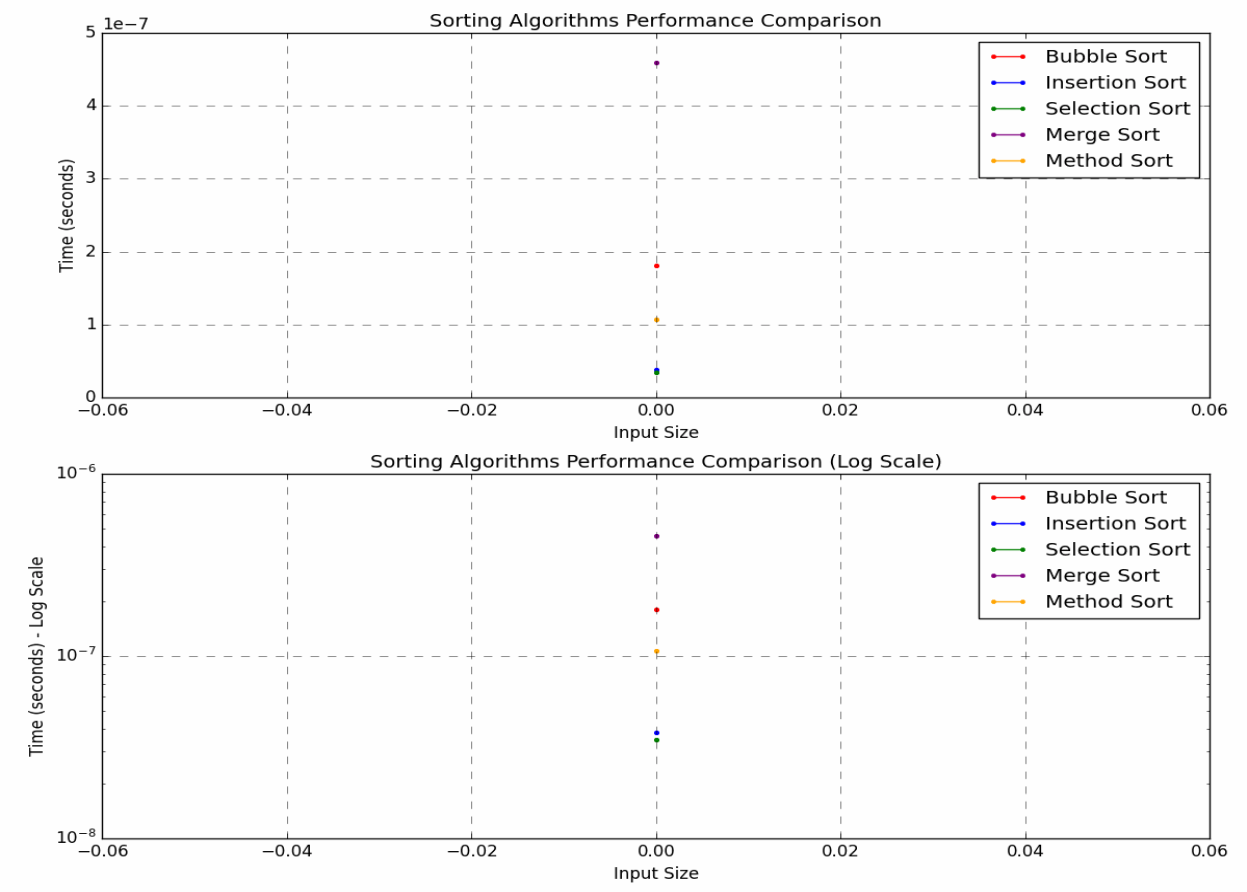
2. Impact of Data Arrangement:
   - Bubble Sort most affected by reverse-sorted section
   - Insertion Sort shows efficiency with pre-sorted sections
   - Merge Sort and Quick Sort maintain consistent performance
   - Selection Sort shows uniform but poor performance

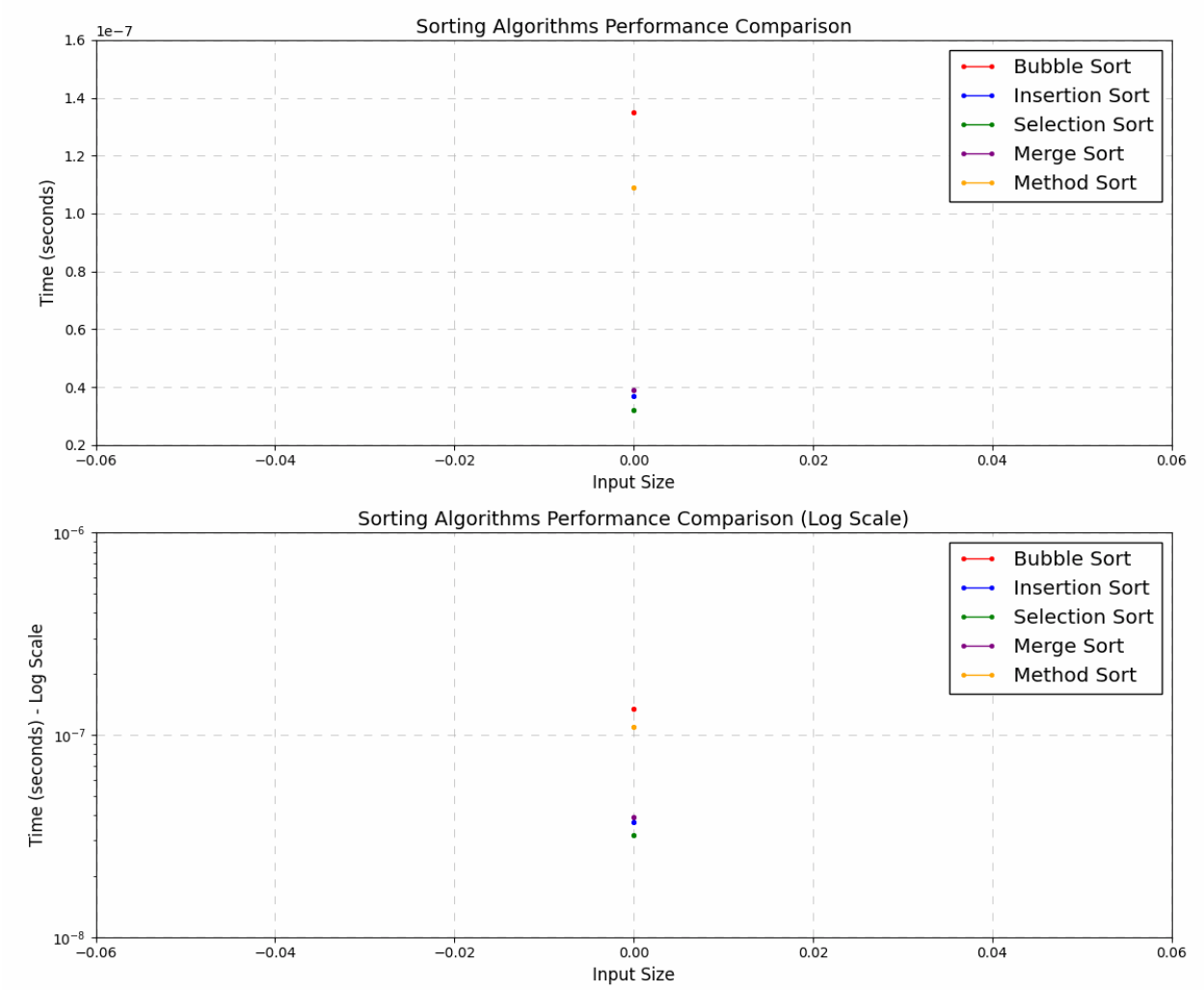
3. Scalability:
   - Quick Sort and Merge Sort scale well with larger inputs
   - Simple sorting algorithms show poor scalability
   - Performance gap widens significantly with larger datasets

Conclusion

The analysis clearly demonstrates the superiority of advanced sorting algorithms (Quick Sort and Merge Sort) over simple comparison sorts. The specific data arrangement in this study (sorted-random-reverse) highlights the importance of choosing the right algorithm based on data characteristics and size.

Adding data


Sorting Algorithms Performance Comparison
1e−7
Time (seconds)
Input Size
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Method Sort
Sorting Algorithms Performance Comparison (Log Scale)
Time (seconds) - Log Scale
Input Size
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Method Sort


Sorting Algorithms Performance Comparison
1e−7
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Time (seconds)
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
Input Size
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Method Sort
Sorting Algorithms Performance Comparison (Log Scale)
10^{-6}
10^{-7}
10^{-8}
Time (seconds) - Log Scale
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
Input Size
Bubble Sort
Insertion Sort
Selection Sort
Merge Sort
Method Sort