Sorting Algorithm Performance Comparison

Rahul Karru
Roll No: 242IS024

Machine Used

- **Device Name**: MacBook Air 2020 (M1)
- **Processor**: Apple M1 (8-core CPU)
- **RAM**: 16 GB (15.7 GB usable)
- **Operating System**: macOS 12.5 Monterey
- **Compiler**: GCC
- **Programming Language**: C
- **IDE**: VSCode (Terminal: Bash)
- **Graphics and GUI Tools**: GTK (from Homebrew), gnuplot

Algorithms Under Evaluation

- Quick Sort (with three pivot strategies: first, last, and median)
- Radix Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Heap Sort
- Shell Sort

Time Complexity Comparison

Algorithm	Best Case	Average Case	Worst Case
Quick Sort (First Pivot)	O(n log n)	O(n log n)	$O(n^2)$
Quick Sort (Last Pivot)	O(n log n)	O(n log n)	$O(n^2)$
Quick Sort (Median Pivot)	O(n log n)	O(n log n)	O(n log n)
Radix Sort	O(nk)	O(nk)	O(nk)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Heap Sort	O(n log n)	O(n log n)	O(n log n)
Shell Sort	O(n log n)	$O(n \log^2 n)$	$O(n^2)$

Space Complexity Comparison

Algorithm	Space Complexity
Quick Sort (All Pivots)	O(log n)
Radix Sort	O(n + k)
Insertion Sort	O(1)
Bubble Sort	O(1)
Merge Sort	O(n)
Heap Sort	O(1)
Shell Sort	O(1)

Graphs and UI Discussion

Performance Graphs

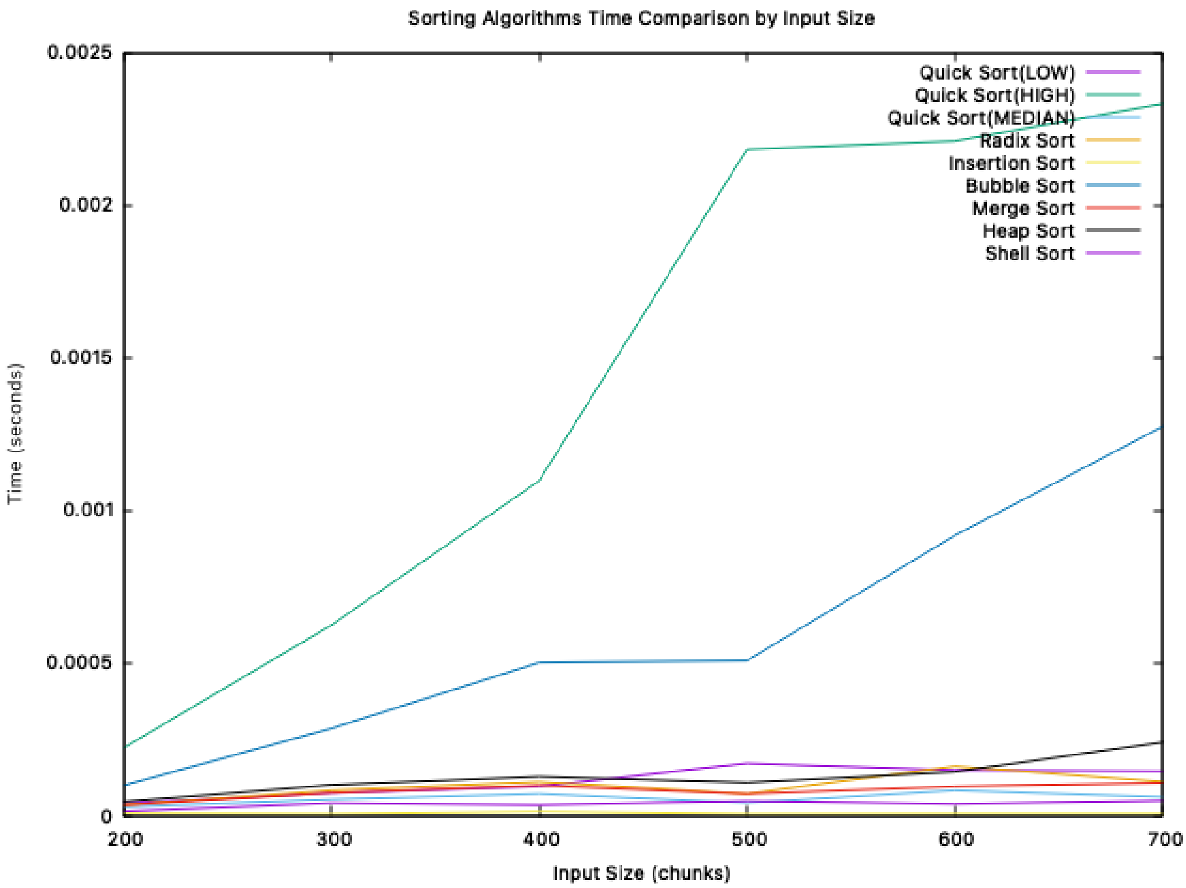

Figure 1: Performance Comparison of Sorting Algorithms: Time for Different Input Sizes

Input Size	Quick Sort (LOW)	Quick Sort (HIGH)	Quick Sort (MEDIAN)	Radix Sort	Insertion Sort	Bubble Sort	Merge Sort	Heap Sort	Shell Sort
200	0.000042	0.000222	0.000029	0.000036	0.000008	0.000099	0.000035	0.000101	0.000015
300	0.000075	0.000624	0.000054	0.000084	0.000006	0.000286	0.000076	0.000129	0.000036
400	0.000096	0.001097	0.000073	0.000111	0.000014	0.000502	0.000099	0.000109	0.000049
500	0.000172	0.002182	0.000043	0.000073	0.000006	0.000508	0.000073	0.000144	0.000039
600	0.000149	0.002209	0.000084	0.000162	0.000007	0.000918	0.000096	0.000241	0.000050
700	0.000146	0.002331	0.000062	0.000112	0.000007	0.001275	0.000108	0.000241	0.000050

Table 4: Sorting Times for Different Algorithms at Various Input Sizes

User Interface (UI)

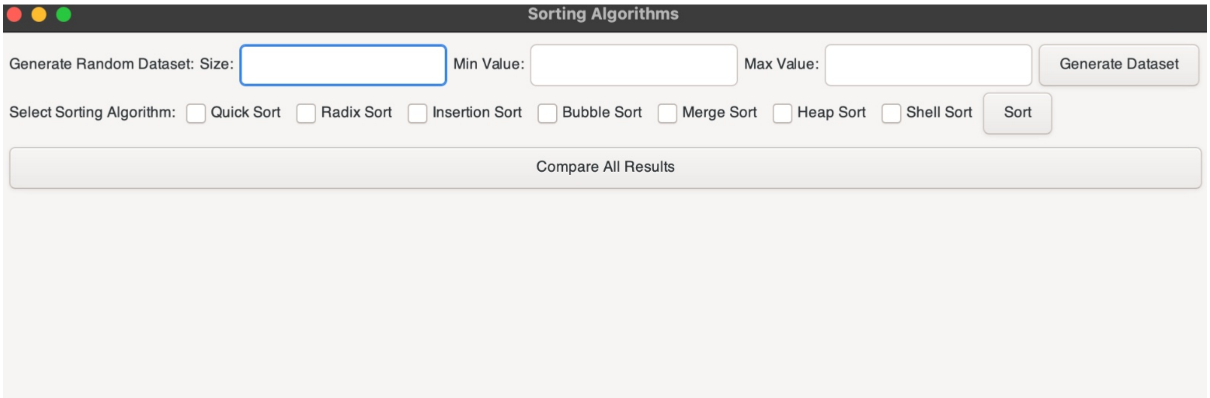

Figure 2: User Interface for Sorting Algorithm Performance Evaluation

- **Input Fields**: Users can specify the dataset size and the range of random numbers to generate, allowing for flexible experimentation with dataset parameters.
- **Generate Button**: Generates a random list of numbers based on user input criteria.
- **Sorting Options**: Users can select one or multiple algorithms to run, facilitating side-by-side performance analysis.
- **Sort Button**: Executes the sorting algorithms on the generated dataset and stores the results for performance evaluation.
- **Plot Button**: Generates a performance graph of sorting times for selected algorithms.
- **User-Friendly Design**: The intuitive layout is designed for ease of use, enabling users to configure input parameters, execute sorting, and analyze performance graphs effortlessly.

Conclusion

The performance evaluation on the MacBook Air M1 system reveals the following insights:

- **Quick Sort (Median Pivot)** consistently demonstrates superior performance. Its execution time increases minimally with larger datasets, making it the most reliable strategy.
- **Quick Sort (First and Last Pivots)** tend to underperform with larger datasets due to their susceptibility to worst-case scenarios.
- **Radix Sort** shows linear time complexity, performing well with integer data but limited by memory overhead.
- **Merge Sort** and **Heap Sort** offer predictable performance and are recommended for scenarios where stability or memory efficiency is crucial.
- **Insertion Sort**, **Bubble Sort**, and **Shell Sort** are less suitable for large datasets due to their time complexity, with Shell Sort providing some improvement over the others.
- **Heap Sort** provides competitive performance with Merge Sort but has the advantage
- **Heap Sort** provides competitive performance with Merge Sort but has the advantage of lower memory usage, making it preferable in memory-constrained environments.