Kailyn Lau

7 November 2024

Assignment 7: Sorting Algorithms

Big-O Complexity and Empirical Analysis of Sorting Algorithms

In this experiment, we measured the performance of four common sorting algorithms (Selection Sort, Insertion Sort, Merge Sort, and Quick Sort) on input sizes ranging from 10,000 to 150,000 elements. The goal was to empirically analyze the execution times of each sort and compare these with the expected theoretical Big-O complexities of each algorithm. This will help us validate our understanding of how the algorithms scale as the input size grows.

Introduction:

Sorting algorithms are fundamental in computer science. This report investigates the performance of four widely studied sorting algorithms: Selection Sort, Insertion Sort, Merge Sort, and Quick Sort.

Selection Sort is a simple comparison-based algorithm that repeatedly selects the smallest element from the unsorted elements and moves it to its correct position. It has a relatively poor performance with a time complexity of $O(n^2)$, making it inefficient for larger datasets, although it takes up very little space and is simple to implement. Insertion Sort is similar, building the sorted array one element at a time by repeatedly inserting each new element into its correct position within the already sorted elements. It also has a worst-case time complexity of $O(n^2)$, but it performs well on small or nearly sorted datasets.

Merge Sort is a divide-and conquer algorithm that recursively splits the array into smaller sub-arrays, sorts them, and then merges them back together. This algorithm has a more efficient time complexity of O(n log n) in all cases, making it well-suited for large datasets. Quick Sort also follows a divide-and-conquer approach and makes use of a pivot element. With Quick Sort, the array is partitioned into two sub-arrays, one with elements smaller than the pivot and one with larger elements, and then the sub-arrays are also recursively sorted. Its average case time complexity is O(n log n), although worst-case complexity (often through poor pivot selection) is $O(n^2)$.

Methodology:

We implemented each of the four sorting algorithms and timed their execution on datasets of increasing size, from 10,000 to 150,000 elements. The data was collected and measured in seconds, and the results are presented below. We then analyzed how the execution times correspond to the known theoretical time complexities for each sorting algorithm:

Selection Sort: $O(n^2)$

Insertion Sort: $O(n^2)$

Merge Sort: O(n log n)

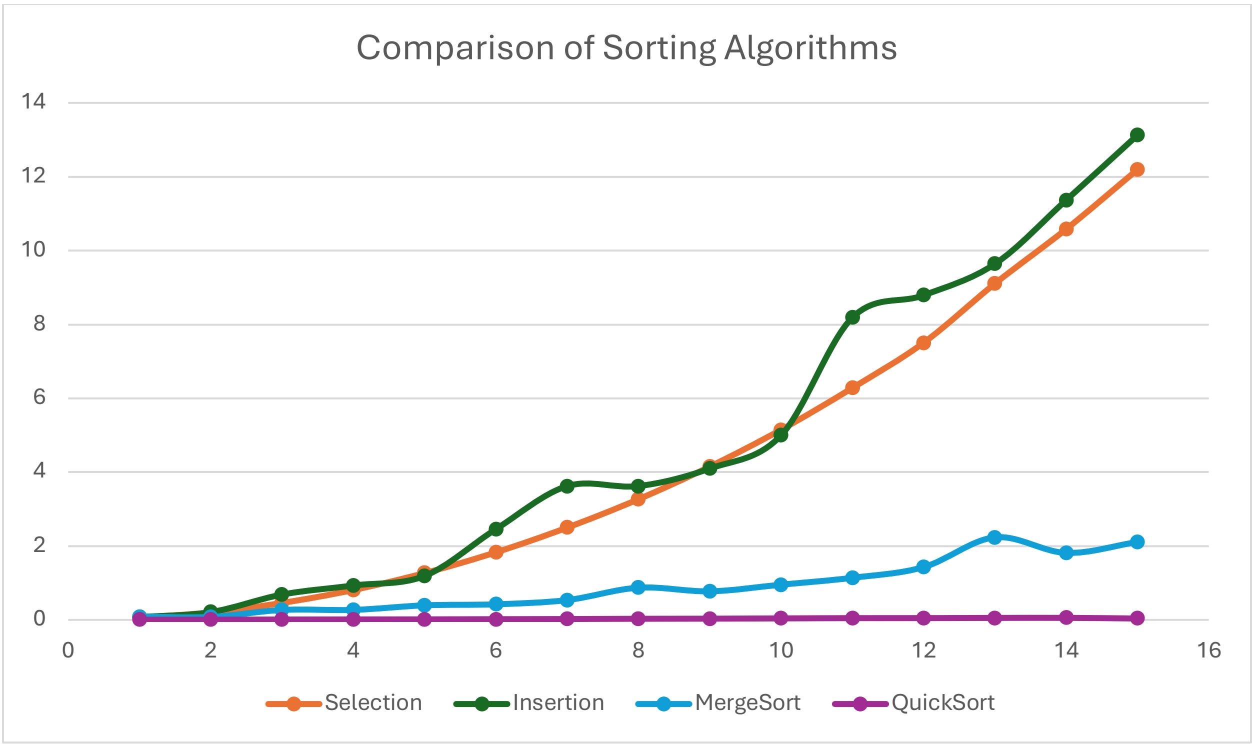
Quick Sort: O(n log n) (average case)

Results:

The following table summarizes the execution times (in seconds) for each sorting algorithm at various input sizes:

DATA	Selection	Insertion	MergeSort	QuickSort
10000	0.05987908	0.07450442	0.08721108	0.003661
20000	0.20242804	0.21013179	0.07983404	0.00663179
30000	0.45897242	0.68535596	0.26882033	0.00951113
40000	0.80888917	0.93105033	0.27075308	0.01289417
50000	1.26729613	1.19020133	0.39374033	0.01442496
60000	1.82924458	2.45731854	0.42196579	0.01901679
70000	2.49917767	3.62238629	0.53632033	0.023251
80000	3.26765454	3.6231045	0.871943	0.03002746
90000	4.15058671	4.09798313	0.77338792	0.03289367
100000	5.14929638	4.99799983	0.95343292	0.04046271
110000	6.28109233	8.18665075	1.13939729	0.04753279
120000	7.50499946	8.78996917	1.42938383	0.04823221
130000	9.1068895	9.63997583	2.23501692	0.0531955
140000	10.5837369	11.3681446	1.81440904	0.05484288
150000	12.1958314	13.128788	2.11002808	0.037449

The data has been plotted below to visualize the trends for each algorithm:



We can see the following trends from the data portrayed above:

- Selection Sort: The execution time increases quadratically as input size increases. This aligns with its $O(n^2)$ complexity.
- Insertion Sort: Similarly, Insertion Sort shows a quadratic increase in time, consistent with its $O(n^2)$ complexity.
- Merge Sort: Merge Sort exhibits a near-constant growth, although not quadratic in nature. It aligns more with the expected O(n log n) complexity.
- Quick Sort: Quick Sort also follows a near-constant growth (better seen in the written instead of graphical data), though with slight fluctuations due to the pivot selection in the algorithm. It behaves closer to O(n log n), confirming its average-case complexity.

Analysis:

The results are consistent with the theoretical complexities of the sorting algorithms. Selection Sort and Insertion Sort exhibit quadratic growth, while Merge Sort and Quick Sort show linearithmic (combination of “linear” and “logarithmic”, showcasing *n log n* growth) growth. These findings confirm that the algorithms behave as expected in terms of time complexity.

The worst-case running time for Selection Sort and Insertion Sort is $O(n^2)$, which we observed in our experiments as the execution times increased significantly with larger input sizes. Merge Sort and Quick Sort both have a worst-case time of O(n log n), and the data supports this, with their times growing more slowly compared to the quadratic algorithms.

One surprising result was that Insertion Sort's execution time grew significantly faster than expected for larger datasets, likely due to its inefficiency with larger inputs. Quick Sort showed relatively consistent performance, suggesting that the average-case behavior is indeed closer to O(n log n). There are a couple of outliers in the collected data (none of the graphs showed perfect growth), which could have been minimized through more thorough testing, including multiple trials, different code implementations for each sort, increasing data sizes and/or distributions, etc.

Conclusion:

The results of our experiment confirm the expected theoretical time complexities of the sorting algorithms. While Selection Sort and Insertion Sort showed quadratic growth, Merge Sort and Quick Sort demonstrated linearithmic growth. These observations align with the Big-O complexities discussed in class, and the experiment provides a concrete example of how algorithm performance scales with input size.