Bethaney Mallory-Smothers

Dr. Parra-Rodriguez

Introduction to Algorithms (CSC 3130-02)

6 February 2025

# Sorting Algorithms

## 1. Merge Sort

[1, 25, 31, 16] and [-3, 0, 16, 27]

The second list is already sorted, wo we will focus on the first list.

[1, 25, 31, 16]

[1, 25]  [16, 31]

[1, 16]   [25, 31]

Merged first list: [1, 16, 25, 31]

Now, we can merge the first list with the second list.

[1, 25, 31, 16]    [-3, 0, 16, 27]

Compare 1 and –3: [-3]

Compare 1 and 0: [-3,0]

Compare 1 and 16: [-3, 0, 1]

Compare 16 and 16: [-3, 0, 1, 16]

Compare 25 and 16: [-3, 0, 1, 16, 16]

Compare 25 and 27: [-3, 0, 1, 16, 16, 25]

Compare 31 and 27: [-3, 0, 1, 16, 16, 25, 27]

Put 31 at the end:     [-3, 0, 1, 16, 16, 25, 27, 31]

**The final merged list: [-3, 0, 1, 16, 16, 25, 27, 31]**

2. **Insertion Sort**

[-1, -5, 67, -10, 21, 8, 4, 1]

-1 is already sorted

Insert -5: [**-5,** -1, 67, -10, 21, 8, 4, 1]

Insert 67: [-5, -1, **67,** -10, 21, 8, 4, 1]

Insert -10: [**-10,** -5, -1, 67, 21, 8, 4, 1]

Insert 21: [-10, -5, -1, **21,** 67, 8, 4, 1]

Insert 8: [-10, -5, -1, **8,** 21, 67, 4, 1]

Insert 4: [-10, -5, -1, **4,** 8, 21, 67, 1]

Insert 1: [-10, -5, -1, **1,** 4, 8, 21, 67]

**The final insertion sorted list: [-10, -5, -1, 1, 4, 8, 21, 67]**

3. **Quick Sort**

[-5, 42, 6, 19, 11, 25, 26, -3]

The pivot I am choosing will be 19.

**Left (<= 19):** [-5, 6, 11, -3] **Pivot:** [19] **Right(> 19):** [ 42, 25, 26]

**Now, sort the "Left"**

[-5, 6, 11, -3]

The pivot I am choosing will be 6.

**Left (<= 6):** [-5, -3] **Pivot:** [6] **Right (> 6):** [ 11]

Sorted: [-5, -3, 6, 11]

**Now, sort the "Right"**

[ 42, 25, 26]

The pivot I am choosing will be 25.

**Left:** [] **Pivot:** [25]  **Right (> 25):** [ 42, 26]

[25, 42, 26]

The pivot I am choosing will be 26.

**Left:** [25] **Pivot:** [26]  **Right:** [42]

[25, 26, 42]

**The final quick sorted list: [-5, -3, 6, 11, 25, 26, 42]**

## 4. Shell Sort

[15, 14, -6, 10, 1, 15, -6, 0]

First we need to define the gaps. There are 8 terms so the gaps will be [4, 2, 1].

**Gap = 4**

Compare (15, 1): Swap - [1, 14, -6, 10, 15, 15, -6, 0]

Compare (14, 15): No Swap - [1, 14, -6, 10, 15, 15, -6, 0]

Compare (-6, -6) No Swap - [1, 14, -6, 10, 15, 15, -6, 0]

Compare (10, 0) Swap - [1, 14, -6, 0, 15, 15, -6, 10]

**Gap = 2**

[1, 14, -6, 0, 15, 15, -6, 10]

Compare (1, -6): Swap - [-6, 14, 1, 0, 15, 15, -6, 10]

Compare (14, 0): Swap - [-6, 0, 1, 14, 15, 15, -6, 10]

Compare (1, 15): No Swap

Compare (14, -6): Swap - [-6, 0, 1, -6, 15, 15, 14, 10]

Compare (15, 10): Swap - [-6, 0, 1, -6, 15, 10,14, 15]

**Gap = 1**

Here, we are going to use insertion sort.

[-6, 0, 1, -6, 15, 10, 14, 15]

Swap (-6, -6): [**-6, -6,** 1, 0, 15, 10, 14, 15]

Swap (1, 0): [-6, -6, **0, 1,** 15, 10, 14, 15]

Swap (15, 10): [-6, -6, 0, 1, **10, 15,** 14, 15]

Swap (15, 14): [-6, -6, 0, 1, 10, **14, 15,** 15]

**The final shell sorted list: [-6, -6, 0, 1, 10, 14, 15, 15]**

## 5. Ranking of Algorithms

**Merge Sort** – Merge Sort is most reliable, running at O(n log n) no matter what.

**Quick Sort** – Quick Sort performs at O(n log n), which makes it very efficient. However, in the worst case, it can slow down to O($n^2$).

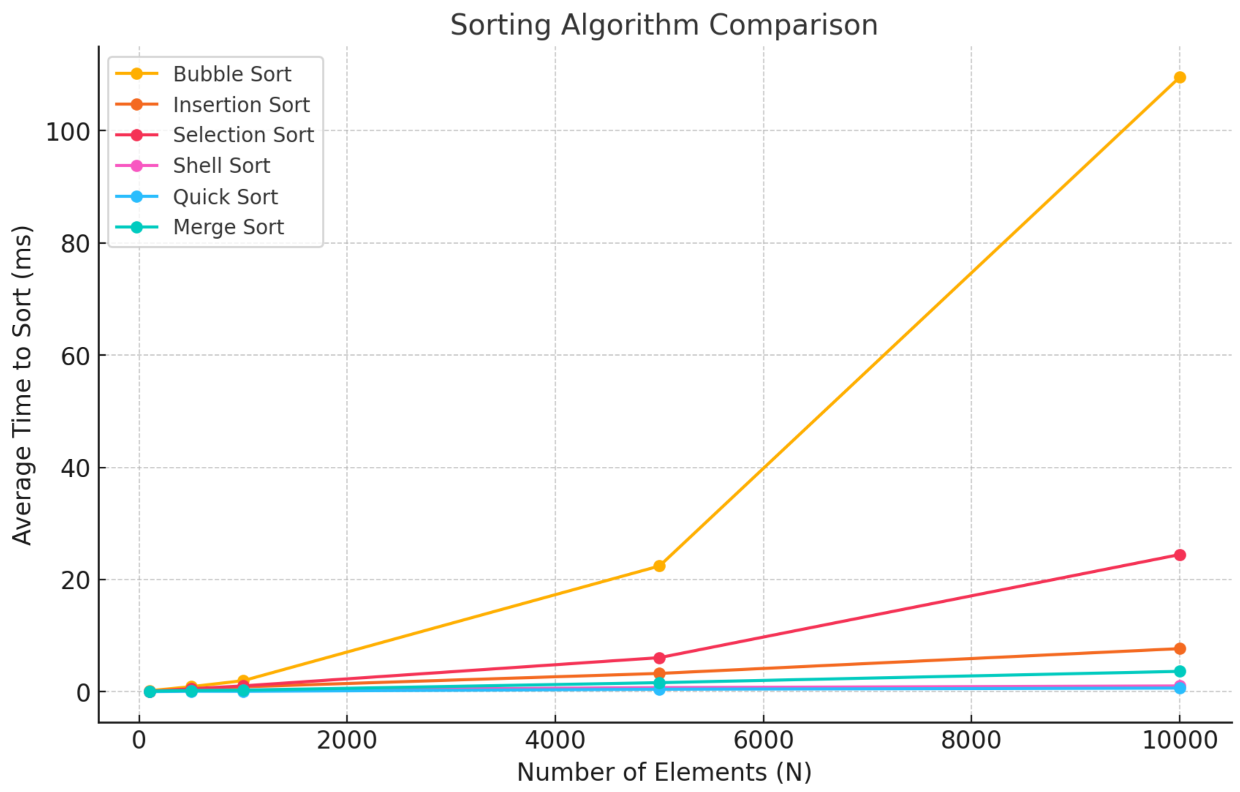
**Shell Sort** – The performance of Shell Sort can vary depending on the gap sequence used, but it can approach O(n log n) with an optimal sequence. However, for many sequences, it runs closer to O($n^2$), so it is not as fast as Merge or Quick Sort.

**Insertion Sort** – Insertion Sort is simple and efficient for small or nearly sorted lists, but its performance degrades to O($n^2$) for larger unsorted datasets, making it slower compared to others.

**Selection Sort** – Similar to Insertion Sort, Selection Sort has an O($n^2$) time complexity. It is not as efficient because it repeatedly selects the minimum element and places it in the correct position, which ends in more data movement compared to Insertion Sort.

**Bubble Sort** – Bubble Sort is the slowest of these algorithms. While its best case is O(n) if the list is already sorted, its time complexity is O($n^2$), making it inefficient for most use cases.

## 9. Sorting Algorithm Performance Comparison



## 10. Summary

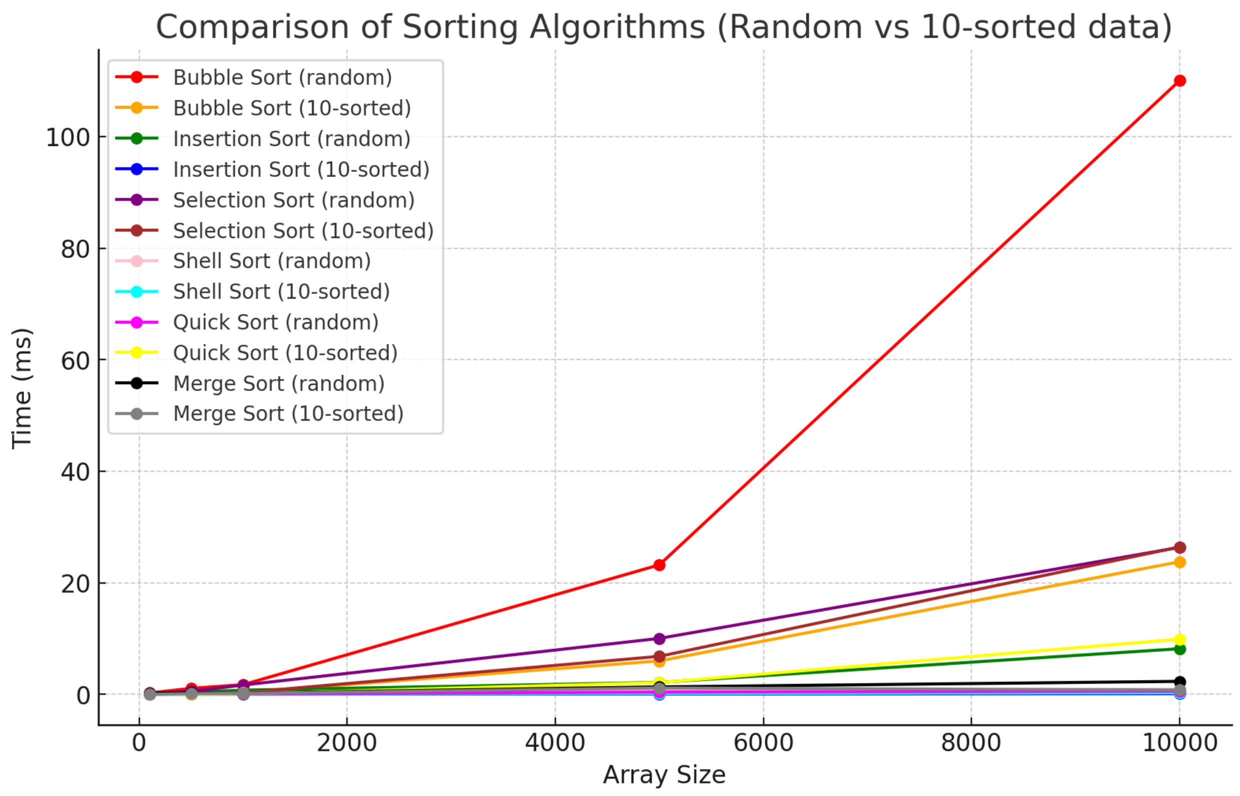
From my experiments, I noticed differences in how each sorting algorithm performed as the input size increased. As expected, Bubble Sort, Insertion Sort, and Selection Sort—all with $O(n^2)$ time complexity—became much slower as the arrays got larger. Bubble Sort was the slowest overall, while Insertion Sort handled smaller inputs better.

The more efficient algorithms, which are, Shell Sort, Quick Sort, and Merge Sort, performed much better since they run in O(n log n) time. Quick Sort was the fastest overall, especially for large arrays, while Merge Sort was slightly slower due to extra memory usage. Shell Sort fell somewhere in between. It was not as fast as Quick or Merge Sort but still outperformed the $O(n^2)$ sorts.

Overall, the actual performance mostly matched what I expected from theory, but there were a few surprises. Insertion Sort did better than I thought, mostly because it works well on partially sorted data. Also, Quick Sort slightly did better than Merge Sort, which makes sense since it tends to have a smaller constant factor in practice. This experiment showed that while Big-O notation is a great way to compare algorithms, real-world performance also depends on factors like memory usage and implementation details.

## 12. Summary & Chart

When sorting 10-sorted data, all the algorithms performed better compared to sorting completely random data. The biggest improvements were seen in Insertion Sort and Shell Sort since they work efficiently when elements are already close to their correct positions. Bubble Sort also performed better, but it was still the slowest overall. Meanwhile, Quick Sort and Merge Sort did not see much change since their divide-and-conquer strategy is already optimized for various input types. Although the ranking of the algorithms stayed mostly the same, the performance gap between the slower and faster ones was smaller.


Comparison of Sorting Algorithms (Random vs 10-sorted data)
Bubble Sort (random)
Bubble Sort (10-sorted)
Insertion Sort (random)
Insertion Sort (10-sorted)
Selection Sort (random)
Selection Sort (10-sorted)
Shell Sort (random)
Shell Sort (10-sorted)
Quick Sort (random)
Quick Sort (10-sorted)
Merge Sort (random)
Merge Sort (10-sorted)
Time (ms)
Array Size