Amalia Karaman Sorting Algorithms

1. Merging [1, 25, 31, 16] and [-3, 0, 16, 27] assuming sorted:

Merge:
L = [1, 25, 31, 16], R = [-3, 0, 16, 27]
1 and -3 → add -3
1 and 0 → add 0
1 and 16 → add 1
25 and 16 → add 16
25 and 27 → add 25
31 and 27 → add 27
Remaining: 31, 16
Final merged list: [-3, 0, 1, 16, 25, 27, 31, 16]

2. Insertion Sort: [-1, -5, 67, -10, 21, 8, 4, 1]
Insert -5 → [-5, -1, 67, -10, 21, 8, 4, 1]
Insert 67 → in place already
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]
Final: [-10, -5, -1, 1, 4, 8, 21, 67]

3. QuickSort: [-5, 42, 6, 19, 11, 25, 26, -3]
Pivot = -3 (last element)
Partition:
Left: [-5], Right: [42, 6, 19, 11, 25, 26]

Sort right side (pivot = 26)
partition
Left: [6, 19, 11, 25], Right: [42]
Sort [6, 19, 11, 25] (pivot = 25)
partition
Left: [6, 19, 11], Right: []
Sort [6, 19, 11] (pivot = 11)
partition
Left: [6], Right: [19]

Final Merge:
[6, 11, 19] + 25 + 26 + 42

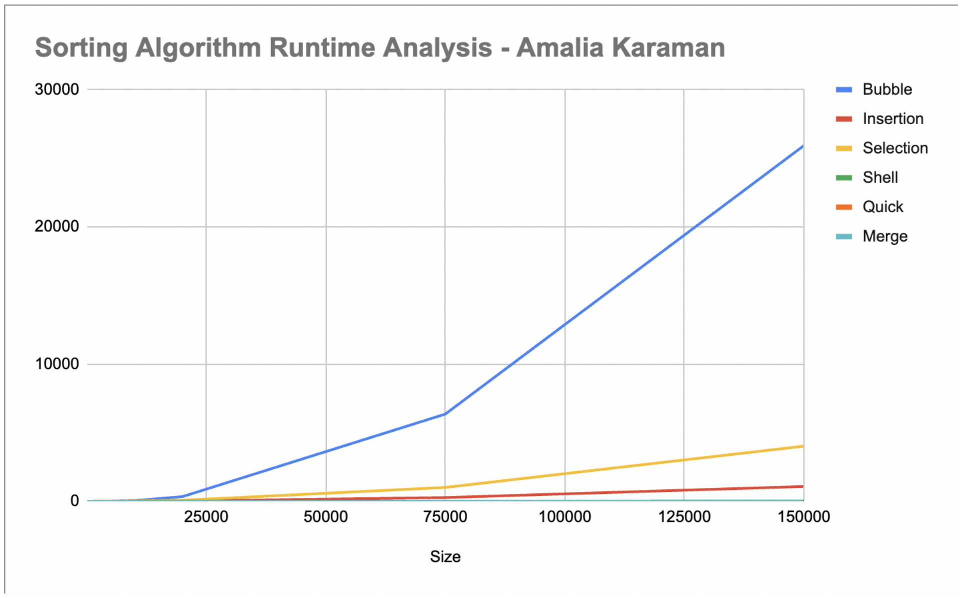
Final list: [-5, -3, 6, 11, 19, 25, 26, 42]

4. Shell Sort: [15, 14, -6, 10, 1, 15, -6, 0]
Gap = 4 to [1, 14, -6, 10, 15, 15, -6, 0]
Gap = 2 to [-6, 0, 1, 14, 15, 15, -6, 10]
Gap = 1 to [-6, -6, 0, 1, 10, 14, 15, 15]

Final sorted list: [-6, -6, 0, 1, 10, 14, 15, 15]

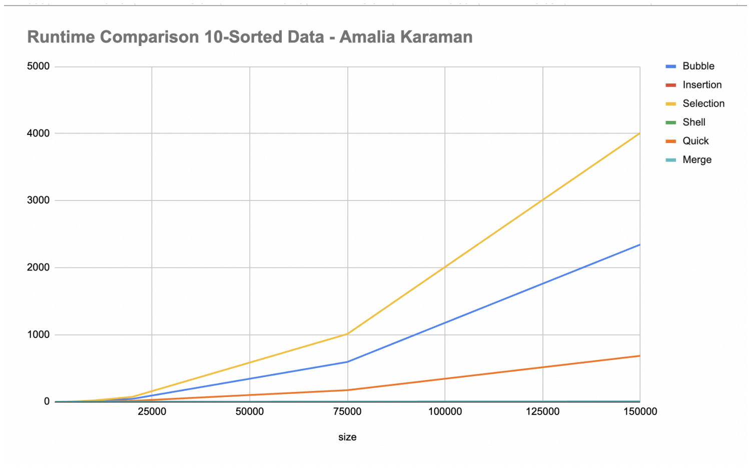
5. Ranking by Time Complexity Fastest to slowest:
1. QuickSort: O(n log n) average. Fastest on large inputs due to partitioning.
2. MergeSort: O(n log n) always. Stable using divide and conquer.
3. ShellSort: About O(n log^2 n). Faster than basic sorts due to gap-based comparisons.
4. InsertionSort: O(n^2), but near O(n) on sorted data. Fastest on 10-sorted.
5. SelectionSort : O(n^2). Always does n^2 comparisons regardless of order.
6. BubbleSort: O(n^2). Swaps every adjacent pair repeatedly, slowest overall.

9.



Graph shows QuickSort and MergeSort as fastest on random data. ShellSort held up well in mid-range. Bubble, Selection, and Insertion were slowest.

10. QuickSort was fastest overall, especially on the large inputs but MergeSort was consistent and close behind Quicksort. ShellSort performed well in the middle, faster than the basic sorts, bubble and selection. Bubble and Selection were the slowest, as expected due to O(n^2) time. My predictions matched except QuickSort outperformed MergeSort in my run.

12. 

My data shows that InsertionSort outperformed all other algorithms on 10-sorted data. ShellSort and MergeSort followed closely. QuickSort was slightly slower than expected due to its recursive behavior. Bubble and SelectionSort showed improved performance but were still the slowest. InsertionSort was the fastest overall because it performs near O(n) on nearly sorted input. ShellSort also benefited from partial order and ran much faster than it did on random data. MergeSort stayed consistent with solid O(n log n) performance. QuickSort was good, but slower than expected due to unnecessary recursion when the input is nearly sorted. Selection and BubbleSort were still the slowest but they improved slightly compared to the random case.