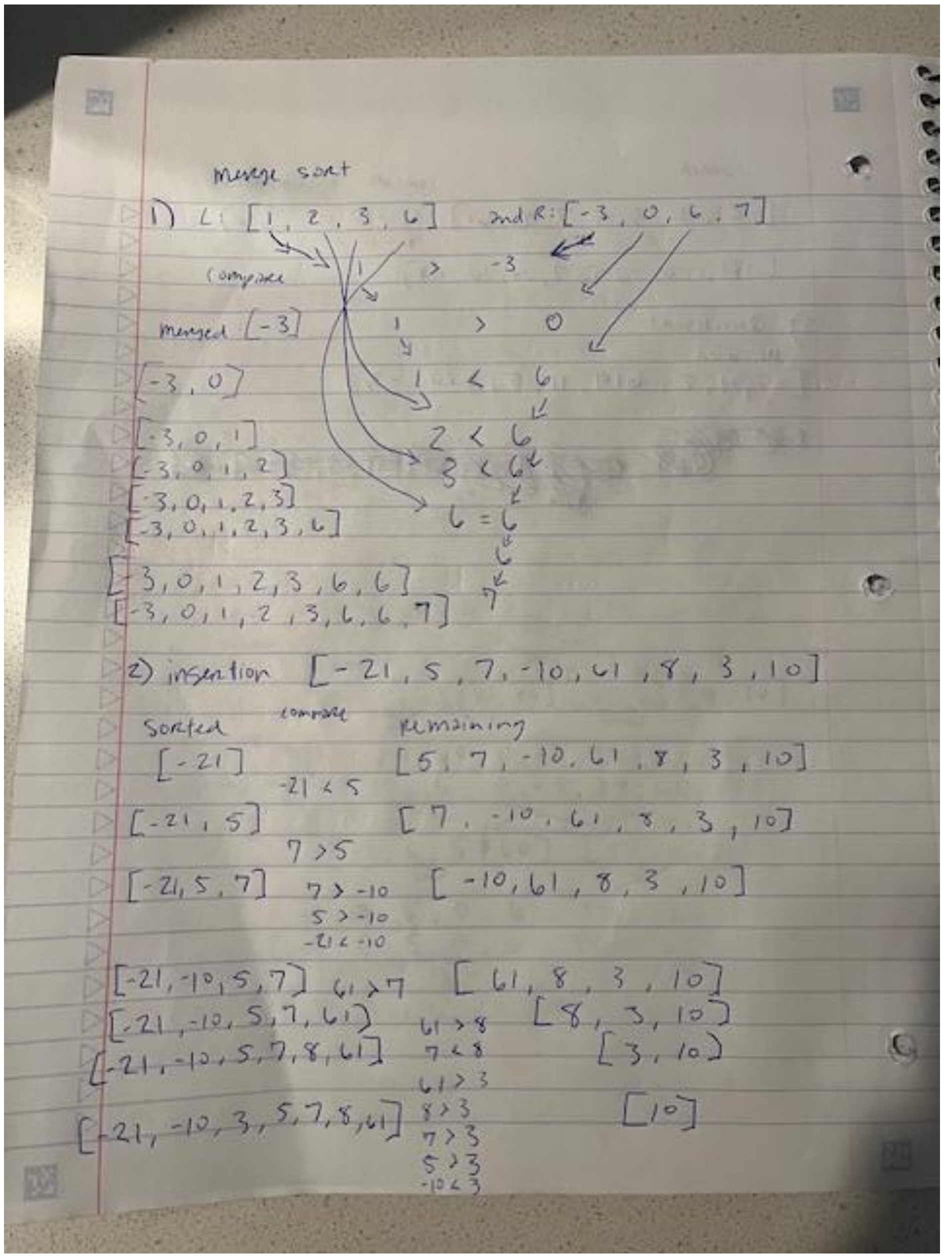
Christina Clements

Sorting Algorithms and Performance



Merge sort

1) L: [1, 2, 3, 6] and R: [-3, 0, 6, 7]

compare

1 > -3

merged [-3]

1 > 0

[-3, 0]

1 < 6

[-3, 0, 1]

2 < 6

[-3, 0, 1, 2]

3 < 6

[-3, 0, 1, 2, 3]

6 = 6

[-3, 0, 1, 2, 3, 6]

6

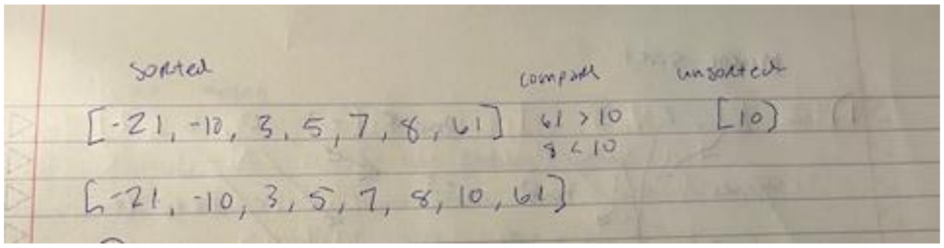
[-3, 0, 1, 2, 3, 6, 6]

7

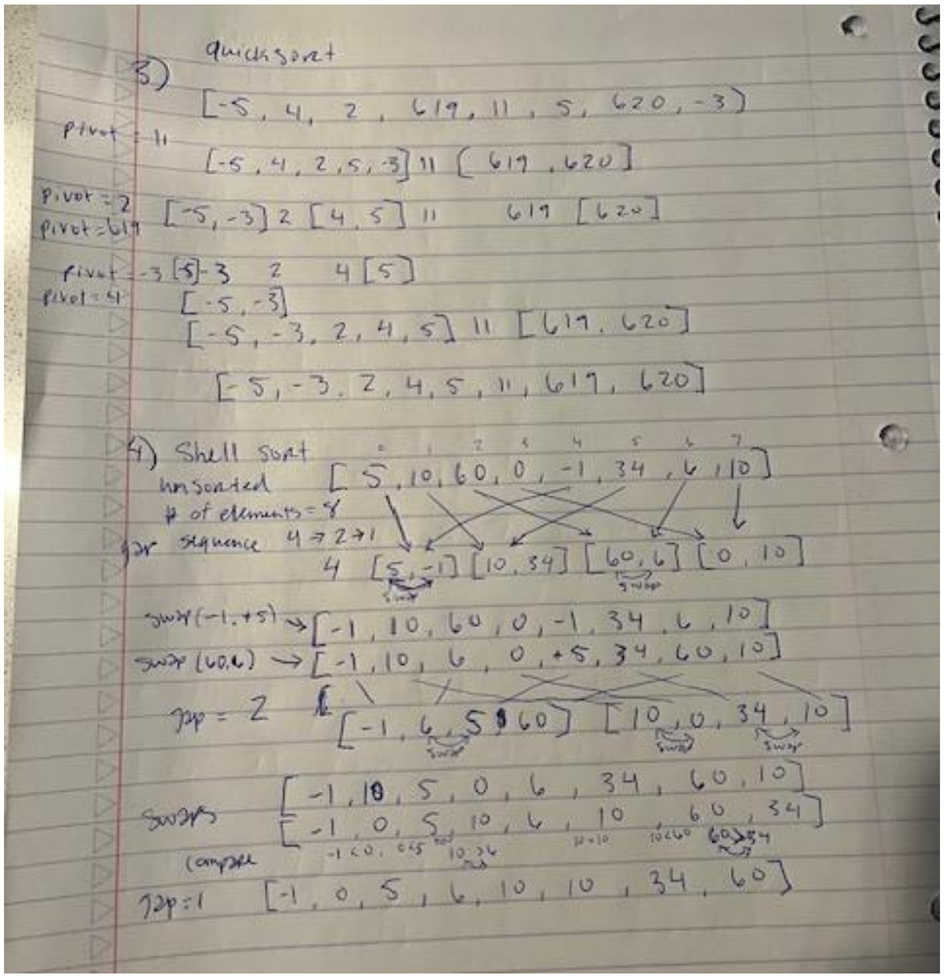
[-3, 0, 1, 2, 3, 6, 6, 7]

2) insertion [-21, 5, 7, -10, 61, 8, 3, 10]

Sorted	compare	remaining
[-21]		[5, 7, -10, 61, 8, 3, 10]
	-21 < 5	
[-21, 5]		[7, -10, 61, 8, 3, 10]
	7 > 5	
[-21, 5, 7]	7 > -10 5 > -10 -21 < -10	[-10, 61, 8, 3, 10]
[-21, -10, 5, 7]	61 > 7	[61, 8, 3, 10]
[-21, -10, 5, 7, 61]	61 > 8 7 < 8	[8, 3, 10]
[-21, -10, 5, 7, 8, 61]	61 > 3 8 > 3 7 > 3 5 > 3 -10 < 3	[3, 10]
[-21, -10, 3, 5, 7, 8, 61]		[10]



Sorted	compare	unsorted
[-21, -10, 3, 5, 7, 8, 61]	61 > 10 8 < 10	[10]
[-21, -10, 3, 5, 7, 8, 10, 61]		



3) quicksort

[-5, 4, 2, 619, 11, 5, 620, -3)

pivot = 11

[-5, 4, 2, 5, -3] 11 [619, 620]

pivot = 2 [-5, -3] 2 [4, 5] 11 619 [620]

pivot = 619

pivot = -3 [-5] -3 2 4 [5]

pivot = 4

[-5, -3]

[-5, -3, 2, 4, 5] 11 [619, 620]

[-5, -3, 2, 4, 5, 11, 619, 620]

4) Shell sort

unsorted [5, 10, 60, 0, -1, 34, 6, 10]

0 1 2 3 4 5 6 7

# of elements = 8

gap sequence 4 → 2 → 1

4 [5, -1] [10, 34] [60, 6] [0, 10]

swap swap

swap(-1, +5) → [-1, 10, 60, 0, -1, 34, 6, 10]

swap (60, 6) → [-1, 10, 6, 0, +5, 34, 60, 10]

gap = 2 [-1, 6, 5 60] [10, 0, 34, 10]

swap swap swap

swaps [-1, 10, 5, 0, 6, 34, 60, 10]

[-1, 0, 5, 10, 6, 10, 60, 34]

compare -1 < 0, 0 < 5 10 > 6 10 = 10 10 < 60 60 > 34

gap = 1 [-1, 0, 5, 6, 10, 10, 34, 60]

5. Fastest to Slowest Ranking (where n is the number of elements)

Merge Sort (tie)

Ω(n log n) Θ(n log n) O(n log n)

Quick Sort (tie)

Ω(n log n) Θ(n log n) O(n^2)

These two algorithms are ranked as the fastest because they both have the same average sorting complexity on n log n. Merge is placed first because it is true for all cases, where quick sort's worst case is n^2.

Shell Sort

Ω(n log n) O(n^2)

The performance of this algorithm heavily depends on gap sequence choice, but it does have a faster best case than the other three below.

Insertion Sort (tie)

Ω(n) Θ(n^2) O(n^2)

Bubble Sort (tie)

Ω(n) Θ(n^2) O(n^2)

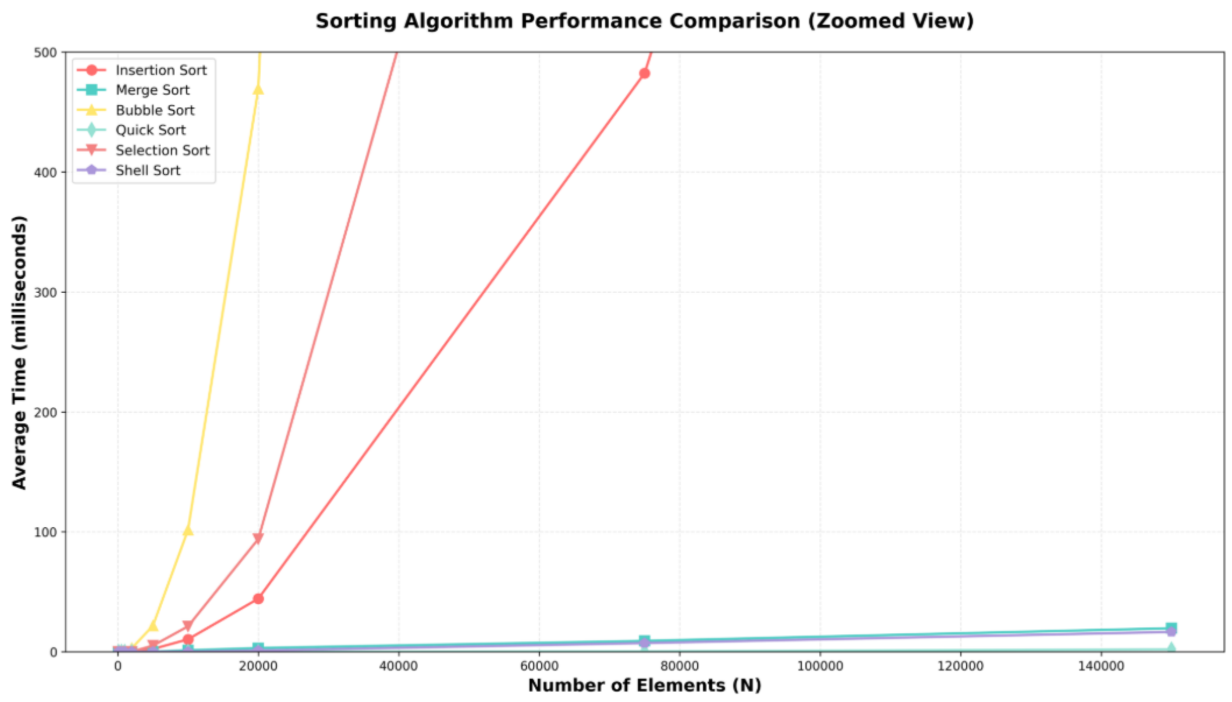
Selection Sort

Ω(n^2) Θ(n^2) O(n^2)

These three are all tied for last because on average, their average time complexity is n^2. Bubble sort and insertion can achieve O(n) best case on sorted data, so that is why they are ahead of selection sort.

6-8 in code on github.

9.



10. The results of our testing confirm our predictions that were based on the asymptotic complexity of each algorithm. My first predicted ranking was merge/quick sort (tie), shell sort, insertion/bubble sort (tie) and selection sort. The results are listed below.

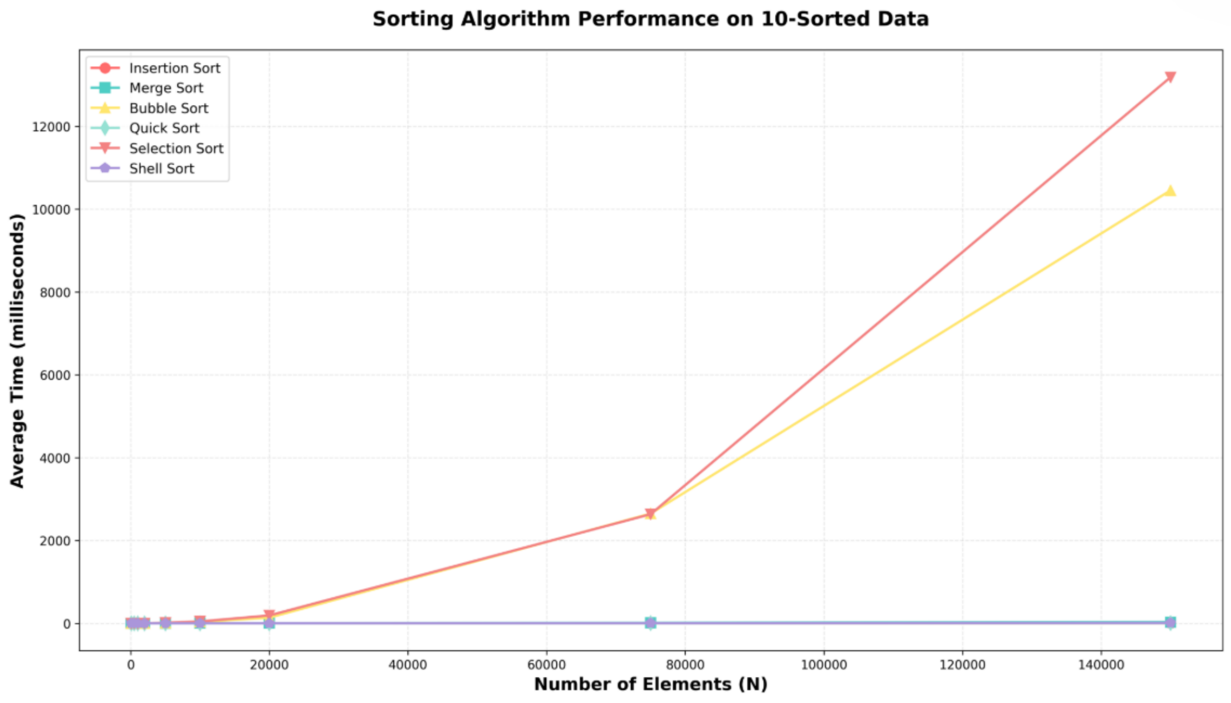
Actual experimental ranking (at N=150,000):

1. Quick Sort (2.0 ms)
2. Shell Sort (16.55 ms)
3. Merge Sort (19.6 ms)
4. Insertion Sort (1,854.65 ms)
5. Selection Sort (4,899.8 ms)
6. Bubble Sort (30,722.35 ms)

   There is a huge gap between the performance of the top three and the bottom three algorithms, but this was expected due to the top portion having an average time complexity of O(n log n) and the bottom portion having O(n^2). The gap increased exponentially as N increased demonstrating the importance for lower time complexity. I predicted the two general ranking sets, but not the exact placement of each algorithm. Quick sort was the fastest, then shell sort, followed by merge sort. Quick sort was by far the fastest, which I did predict. I did not expect it to perform significantly better than merge sort, since I had them tied originally. Shell sort was also faster than I predicted. Bubble sort was by far the slowest, which I had not predicted. One of the reasons for this is the amount of swaps and comparisons are much higher than for insertion or selection sort.

11. in code, Tester class

12.



Random Data Ranking (N=150,000):

1. Quick Sort (2.0 ms)
2. Shell Sort (16.55 ms)
3. Merge Sort (19.6 ms)
4. Insertion Sort (1,854.65 ms)
5. Selection Sort (4,899.8 ms)
6. Bubble Sort (30,722.35 ms)

10-Sorted Data Ranking (N=150,000):

1. **Insertion Sort (2.45 ms)**
2. Quick Sort (4.55 ms)
3. Shell Sort (9.8 ms)
4. Merge Sort (29.45 ms)
5. **Bubble Sort (10,454.35 ms)**
6. **Selection Sort (13,185.65 ms)**

Insertion sort moved up the list because it performs optimally when elements are close to their sorted position, like they are with 10-sorted data. Bubble sort also improved in performance because there were fewer swaps needed when the elements are nearly sorted. Shell sort performed better because it is designed for partially sorted data. Both selection and merge sort are non-adaptive, which means it makes the same amount of comparisons regardless of ordering. Quick sort is not really affected by partially sorted data either. The time differences could be due to CPU performance or the amount of comparisons and swaps needed.