Assignment 2 - Sorting

Mayur Patel

1. merging [1, 2, 3, 6] and [-3, 0, 6, 7]

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

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

[-21, 5, 7, -10, 61, 8, 3, 10] 1st pass
sorted

[-21, 5, 7, -10, 61, 8, 3, 10] 2nd pass
sorted →

[-21, -10, 5, 7, 61, 8, 3, 10] 3rd pass
sorted moved -10

[-21, -10, 5, 7, 61, 8, 3, 10] 4th pass
sorted →

[-21, -10, 5, 7, 8, 61, 3, 10] 5th pass
sorted → moved 8

[-21, -10, 3, 5, 7, 8, 61, 10] 6th pass
sorted → moved 3

[-21, -10, 3, 5, 7, 8, 10, 61] 7th pass
completely sorted moved 10

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

pivot = last element   less: left   greater: right

[-5, -3, | 2, 619, 11, 5, 620, 4] partition 1

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

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

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

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

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

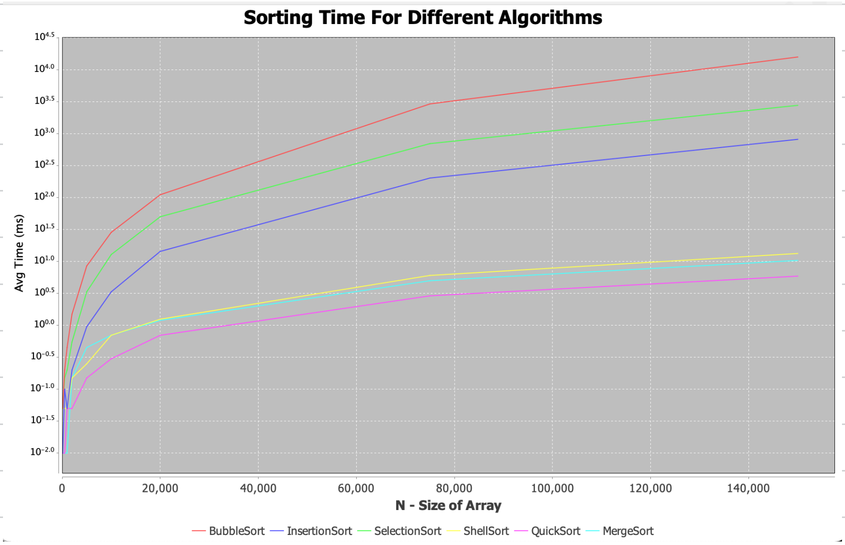
gap = 2 [-1, 10, 6, 0, 5, 34, 60, 10] swap

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

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

5. 1. Merge Sort: O(n log n) time complexity; most consistant
2. Quick Sort: avg O(n log n) time complexity; less consistant.
3. Shell Sort: Improves on insertion sort; must be faster
4. Insertion Sort: Doesn't go through full arr each time.
5. Selection Sort: Full pass of arr each time.
6. Bubble Sort: most swaps, sometimes multiple per pass.

9.

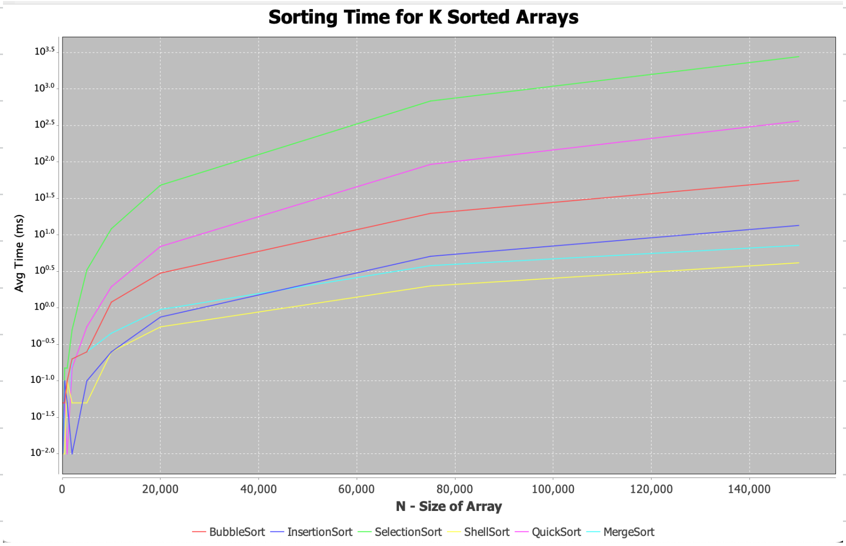


Made in Java with JFreeChart.

Biggest issue I had was the numbers scaled too fast and made the O(n²) sorting algorithms the only ones to show, which really showed how slow they are. I tried fixing it by using a log scale instead of linear.

10. The results almost matched the rankings I thought of, only with Quicksort beating Mergesort. I saw that even if algorithms have the same time complexity, their runtimes can vary a lot. I thought Quicksort would be slower since its slowest is O(n²) but with large datasets, it beat the competition.

12.



Made in Java

Overall almost every algorithm improved with k sorted data. Only exceptions are Selection Sort which didn't change at all and Quicksort which did noticably worse. This is mainly because the partitions are much more uneven when the data is partially sorted and with the pivot I chose as the last element in the array. If I implement using median of 3, then it would probably improve.