Text Problems

1 (text) [2 points] Show the step-by-step process of merging two sorted lists in using the mergesort algorithm. The sorted lists are: [1, 25, 31, 16] and [-3, 0, 16, 27]

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

1: 1 < -3 => [-3]

2: 0 < 1 => [-3, 0]

3: 1 < 16 => [-3, 0, 1]

4: 16 = 16 => [-3, 0, 1, 16]

5: 25 > 16 => [-3, 0, 1, 16, 16]

6: 25 < 27 => [-3, 0, 1, 16, 16, 25]

7: 27 < 31 => [-3, 0, 1, 16, 16, 25, 27]

8: 31 compare with empty

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

sorted

2 (text) [2 points] Show the step-by-step process of sorting a list using the insertion sort algorithm. The unsorted list is: [-1, -5, 67, -10, 21, 8, 4, 1]

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

1: -1 > -5

⇒ [-5, -1, 67, -10, 21, 8, 4, 1]

2: 67 > -1, stay

3: int temp = -10, compare to the left side 67 > -10 swap, continue to compare ⇒ -1 > -10 and -5 > -10 ⇒ swap 2 times

⇒ [-10, -5, -1, 67, 21, 8, 4, 1]

4: int temp = 21, Compare: 21 < 67 ⇒ swap, Keep comparing 21 > -1 ⇒ stay, since -1 > than all nums on left

⇒ [-10, -5, -1, 21, 67, 8, 4, 1]

5: int temp = 8, Compare: 67 > 8 ⇒ swap, Keep checking, 21 > 8 ⇒ swap -1 < 8, stay.

⇒ [-10, -5, -1, 8, 21, 67, 4, 1]

6: int temp = 4, Compare: 67 > 4 ⇒ swap, 21 > 4 ⇒ swap 8 > 4 ⇒ swap, -1 < 4 ⇒ stay.

⇒ [-10, -5, -1, 4, 8, 21, 67, 1]

7: int temp: 1, Compare: 67 > 1 ⇒ swap, 21 > 1 ⇒ swap, 8 > 1 ⇒ swap 4 > 1 ⇒ swap, -1 < 1 ⇒ stay.

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

3 (text) [2 points] Show the step-by-step process of sorting a list using the quicksort sort algorithm. The unsorted list is: [-5, 42, 6, 19, 11, 25, 26, -3]

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

Using Median of three Partitioning, Center = (L+R)/2 = (0+7)/2 = 3

A[Center] = 19

Pivot = median of (-5, 19, -3) ⇒ -3

[-5, 42, 6, 19, 11, 25, 26, -3] (i, j, Pivot)

1: -5 < -3

⇒ [-5, 42, 6, 19, 11, 25, 26, -3]

2: Swap 42 and -3

⇒ [-5, -3, 6, 19, 11, 25, 26, 42]

Left Sublist: [-5] Sorted ✓

Right sub-list: [6, 19, 11, 25, 26, 42] (indices 0 1 2 3 4 5)

Pivot = Median (6, 11, 42) = 11

center = (0+5)/2 = 2

3: Swap 11 and 42

[6, 19, 42, 25, 26, 11]

4: 19 > 11,

⇒ [6, 19, 42, 25, 26, 11]

5: 6 < 11

⇒ [6, 19, 42, 25, 26, 11]

6: i > j, swap 11 and 19

⇒ [6, 11, 42, 25, 26, 19] (Pivot)

[6] Sorted

[42, 25, 26, 19] (indices 0 1 2 3)

Center = (0+3)/2 = 1

Pivot = (42, 25, 19) = 25

7: swap 19 and 25

⇒ [42, 19, 26, 25]

8: 42 > 25

9: 19 < 25

⇒ [42, 19, 26, 25]

10: Swap 42 and 19

[19, 42, 26, 25]

11: Swap 42 and 25

⇒ [19, 25, 26, 42]

Sorted

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

4 (text) [2 points] Show the step-by-step process of sorting a list using the shell sort algorithm. The unsorted list is: [15, 14, -6, 10, 1, 15, -6, 0]

h = 8/2 = 4 [15, 14, -6, 10, 1, 15, -6, 0]

1: 15 > 1 => swap => [1, 14, -6, 10, 15, 15, -6, 0]

2: 14 < 15 => stay => [1, 14, -6, 10, 15, 15, -6, 0]

3: -6 = -6 => no swap => [1, 14, -6, 10, 15, 15, -6, 0]

4: 10 > 0 => swap => [1, 14, -6, 0, 15, 15, -6, 10]

h = 2 [1, 14, -6, 0, 15, 15, -6, 10]

Sublist = [1, -6, 15, -6,] & [14, 0, 15, 10]

1st iteration: [-6, 14, -6, 0, 1, 15, 15, 10]

2nd iteration: [-6, 0, -6, 10, 1, 14, 15, 15]

h = 1

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

swap swap

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

Sorted ✓

5 (text) [4 points] Rank the six sorting algorithms in order of how you expect their runtimes to compare (fastest to slowest). Your ranking should be based on the asymptotic analysis of the algorithms. You may predict a tie if you think that multiple algorithms have the same runtime. Provide a brief justification for your ranking.

My ranking for 6 sorting algorithm are:
(From fastest to slowest)

1. Quicksort with the time complexity on average $O(n \cdot \log n)$ and worst case could be $O(n^2)$ worst case.
- the reason why Quick sort is at first place even when Merge Sort is also $O(n \cdot \log n)$ is because of the pivot selection could make the sorting timing become inconsistence However, if the code is implemeated correctly and pivot can divide the array into similar size sub-list, Quick sort is very fast.

2. Merge Sort with the time complexity on average $O(\log n)$ (always) This sorting algorithm runs consistently in both best case and worst case, regarding of input order (nearly sorted or completely unsorted) since it divides the array in half every iteration. It's slightly slower than due to extra memory usage.

3. Shell Sort - $O(n \log n)$ best case, $O(n^2)$ in worst case.
It depends on the gap (size/2)

4. Insertion Sort - $O(n)$ best case, $O(n^2)$ worst case.
It's only $O(n)$ when the array is nearly sorted, but otherwise, it will still needs to go through $n-1$ comparisons.

5. Selection Sort - Alway $O(n^2)$ since it makes $n^2/2$ comparisons, even when the list is sorted, it will still has to loop through the whole array.

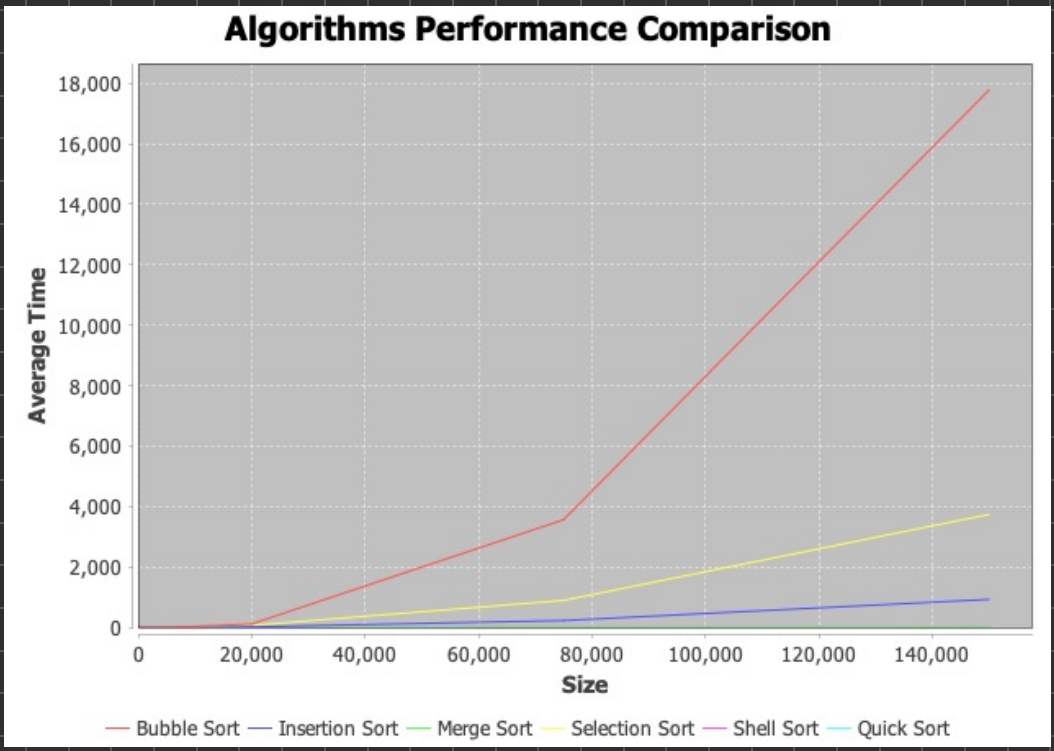
6. Bubble Sort - $O(n)$ best case, $O(n^2)$ worst case. I think it's the slowest algorithm since it repeatedly swaps adjacent elements.

Maybe Tie: Quich and Merge sort since they both $O(n \log n)$.
Insertion and Selection Sort since the only time Insertion is $O(n)$ is when it nearly sorted.

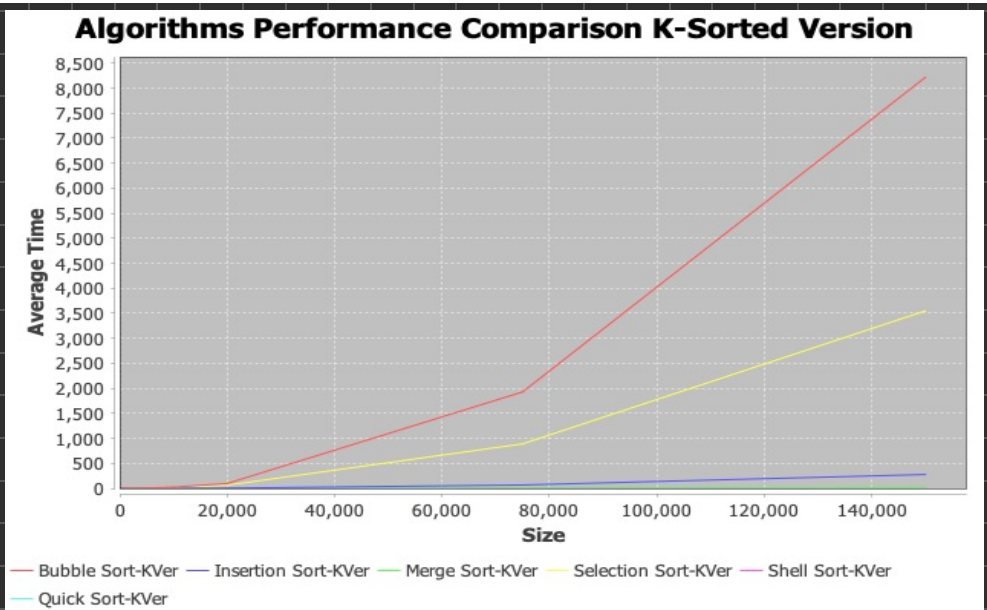
10 (text) [7 points] Write a short summary of what you observed in your experiments. What did you notice about the relative performance of the different algorithms? Did the actual performance always match up to the ranking you predicted in part (1) based on the asymptotic analysis? Why or why not?

- My graph shows that Bubble Sort has the longest time run on average and next up is Selection Sort. Insertion Sort performs ways better than I expected, The best 3 are Shell sort, Quick Sort and merge Sort as I predicted since they all have $O(n \log n)$ in average case. However, it's hard to see if which one is faster on the graph since they all lay on top of each other and the slower sorting algorithms are making the graph looks zoomed out. The graph matched my prediction pretty well, except that I didn't expect Insertion to perform that well. The reason that my prediction was matched is because of the nested loop on the bubble sort, selection sort and insertion sort which make them horrible. The ones that use divide & Conquer sorted so quick because of how it breaks the length to half every iteration.

Bubble ≈ 18K ms, Selection ≈ 4K ms, Insertion ≈ 1K ms
Quick / merge / shell = Super fast < 10 ms



12 (text) [5 points] Make a graph comparing the runtimes of the six algorithms.
Write a short summary of what you observed in your experiments. Did the algorithms have the same ranking on 10-sorted data as on random data? Which algorithm(s) performed significantly differently? Why?
Note that: You will find the same issue as in problem 9. Once you get this issue, explain what it means.



It reduced the run time by alot. As we can see, it helps Bubble sort going from 18K to 8K which is roughly twice effienciency. The K-sort helps bubble and Selection sort to have fewer swaps. I think that the algorithms got faster b/c it reduced number of swaps and comparisons (since elements are now close to their final positions. Maybe lower the amount of recursive calls on some algorithms. However, it didn't do much to Shell / Merge / Quick Sort.