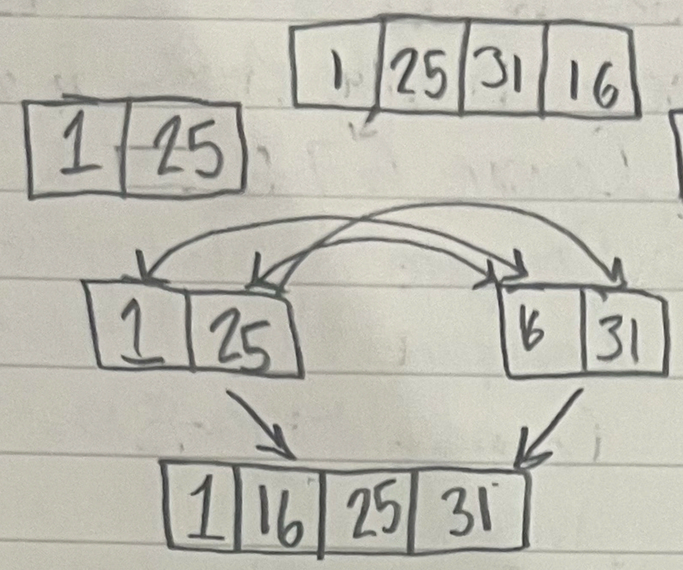
Assignment 2 — Kevin Gonzalez — date / /

1. Merge Sort - Assuming both of our list are sorted. Unless this is a typo, list 1 is not sorted



1. Split list
2. Swap 31 & 16 than merge.
3. compare elements.
   - 1 < 16 = 1
   - 25 > 16 = 16
   - 25 < 31 = 25

Finally add 31.

Now we can merge the two lists (conquer part of algorithm)

1. Compare 1 and -3 = -3
2. Compare 1 and 0 = 0
3. Compare 1 and 16 = 1
4. Compare 16 and 16 = 16 (from list 1)
5. Compare 25 and 16 = 16
6. Compare 25 and 27 = 25
7. Compare 31 and 27 = 27
8. Add last 31.

Assignment 2 cont... date / /

2. Insertion Sort: [-1, -5, 67, -10, 21, 8, 4, 1]

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

Assume sorted ↗

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

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

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

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

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

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

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

Iteration: 1. -5 < -1 ⇒ move -1 to right
insert -5 (the rest follow this, so I simplify)
2. Compare 67 > 1 ⇒ No change
3. Compare -10 < 67 ⇒ 67 right
Compare -10 < -1 ⇒ -1 right
Compare -10 < -5 ⇒ -5 right
insert -10
4. Compare 21 < 67 ⇒ 67 right
Compare 21 > -1 ⇒ insert 21
5. Compare 8 < 67 ⇒ 67 right
Compare 8 < 21 ⇒ 21 right
Compare 8 > -1 ⇒ insert 8
6. Compare 4 < 67 ⇒ 67 right
Compare 4 < 21 ⇒ 21 right
Compare 4 < 8 ⇒ 8 right
Compare 4 > -1 ⇒ insert 4
7. Compare 1 < 67 ⇒ 67 right
Compare 1 < 21 ⇒ 21 right
Compare 1 < 8 ⇒ 8 right
Compare 1 < 4 ⇒ 4 right
Compare 1 > -1 ⇒ 1 insert

Assignment 2 cont...

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

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

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

↓

remains same

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

Step 1: Choose last element as pivot.
Compare all elements to -3
-5 < -3 → -5 left
42 > -3 → 42 right
6 > -3 → 6 right
19 > -3 → 19 right
11 > -3 → 11 right
25 > -3 → 25 right
26 > -3 → 26 right

Step 2: Choose last element 26
42 > 26 → 42 right
6 < 26 → 6 left
19 < 26 → 19 left
11 < 26 → 11 left
25 < 26 → 25 left
26 < 26 → 26 left

Step 3: choose last element 25
26 < 25 → 6 stay left
19 < 25 → 19 stay left
11 < 25 → 11 stay left

Step 4: choose last element 11
19 > 11 → 19 right
6 < 11 → 6 left

Assignment 2 cont...

Shell Sort:

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

Step 1: Initial gap: $\frac{8}{2}$ ← length $= 4$
Compare and swap elements 4 positions apart

5 $ 1 ⇒ 1 < 15 → swap
14 $ 15 ⇒ 14 > 15 → no swap
-6 $ -6 ⇒ -6 == -6 → no swap
10 $ 0 ⇒ 0 < 10 → swap

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

Step 2: Repeat $\frac{4}{2} = 2$
Compare and swap elements 2 positions apart.

1 $ 6 ⇒ -6 < 1 → swap
14 $ 0 ⇒ 0 < 14 → swap
1 $ 15 ⇒ 1 > 15 → no swap
14 $ -6 ⇒ -6 < 14 → swap
15 $ 10 ⇒ 10 < 15 → swap

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

Step 3: Insertion sort $2/2 = 1$

-6 $ 0 ⇒ no swap
0 $ 1 ⇒ no swap
1 $ -6 ⇒ -6 < 1 → swap
1 $ 15 ⇒ no swap
15 $ 10 ⇒ 10 < 15 → swap
15 $ 14 ⇒ 14 < 15 → swap

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

Assignmen 2 Cont...

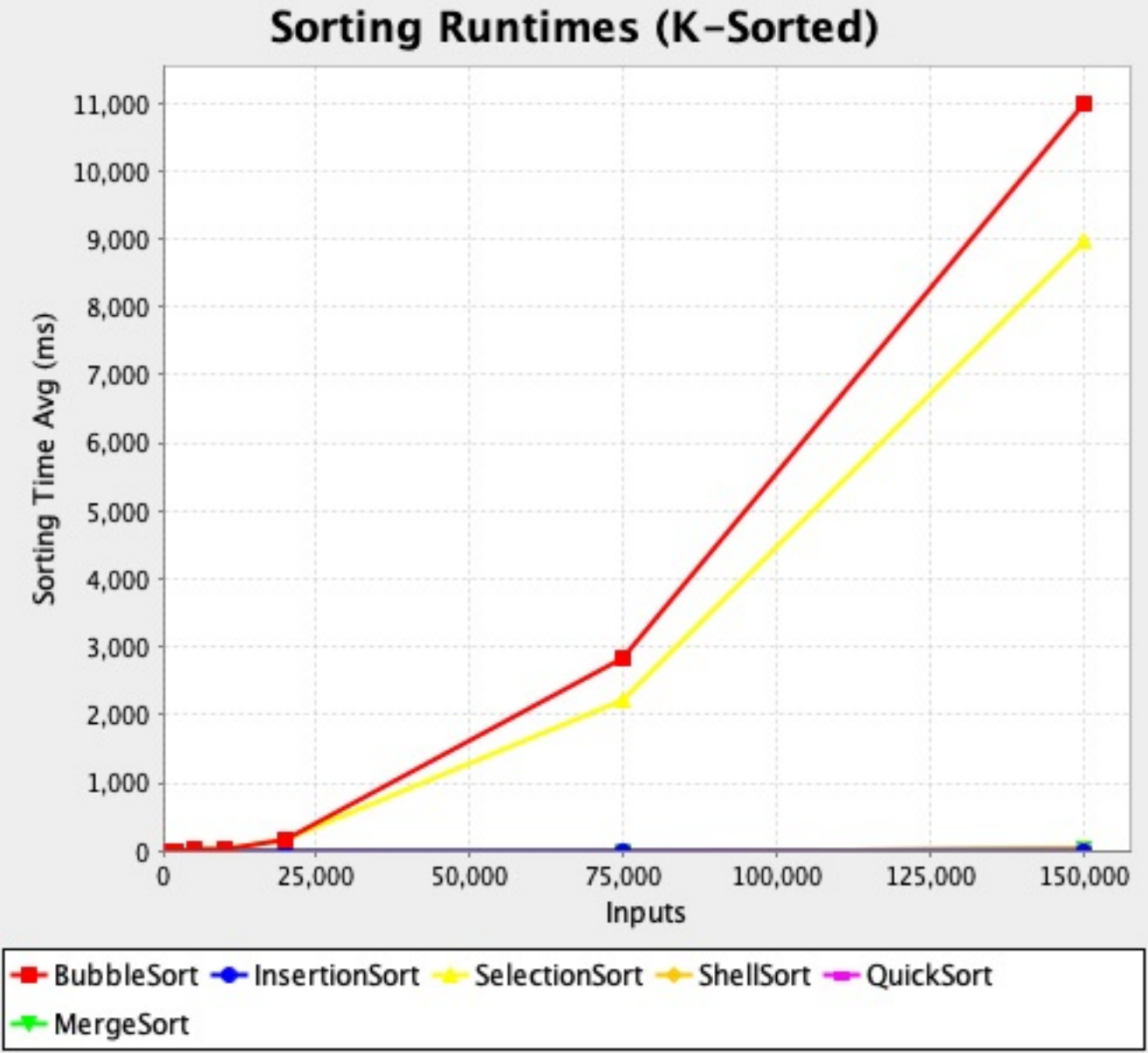
date / /

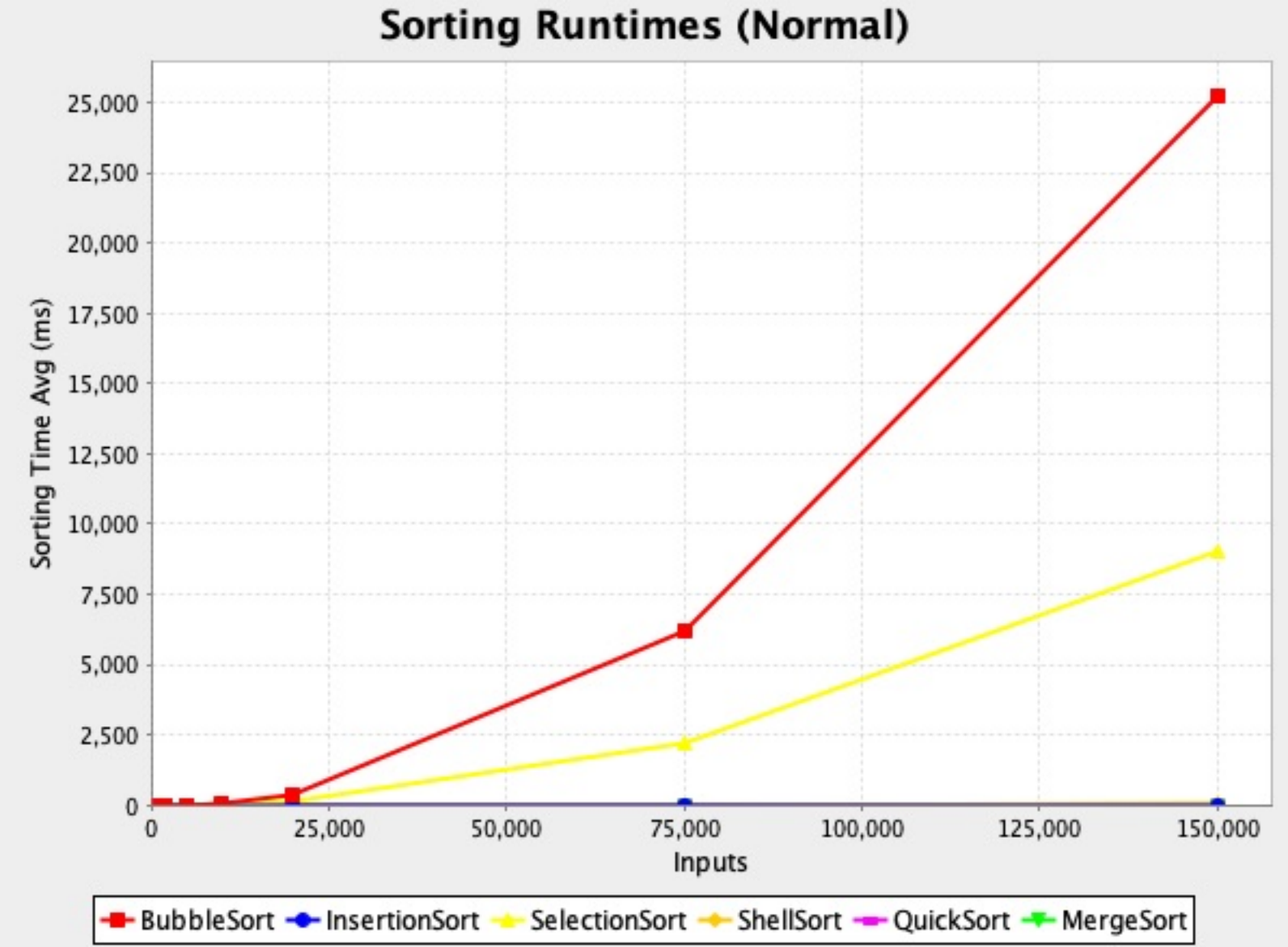
5.1. Merge sort
Shell sort
Quicksort
selection sort
insertion sort
Bubble sort

I put Merge sort on top due to its $\theta(n \log n)$ complexity that could help with large lists. Shell sort a close second only due to its $O(n^2)$ worst scenario. Otherwise, I would put it first because it could help with random lists.
Quicksort at tie with Shel sort because they share the same $O(n^2)$ complexity, but technically has a better best case than shellsort.
As for the last 3, its a tie because they would more often be slow unless you always have small lists.

10. Yes, as expected, the algorithms that ranked high managed to run in similar speed and matched my prediction. Additionally, I predicted that bubblesort and selection sort took insanely long when it came to giant lists. This would be because of their worst case iterate through the whole list one by one.

12. The algorithms for k sort was pretty much the same but faster with every algorithm. Something that caught me off-guard was how the performance gap between bubble sort and selection sort became significally smaller. My guess would be because k-sort begins already sorted by a bit.


Sorting Runtimes (K-Sorted)
Sorting Time Avg (ms)
11,000
10,000
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
0
25,000
50,000
75,000
100,000
125,000
150,000
Inputs
BubbleSort
InsertionSort
SelectionSort
ShellSort
QuickSort
MergeSort


Sorting Runtimes (Normal)
Sorting Time Avg (ms)
25,000
22,500
20,000
17,500
15,000
12,500
10,000
7,500
5,000
2,500
0
0
25,000
50,000
75,000
100,000
125,000
150,000
Inputs
BubbleSort
InsertionSort
SelectionSort
ShellSort
QuickSort
MergeSort