Programming assignment #3 data analysis:

After running the program and testing datasets of different sizes on both algorithms, I observed that quicksort consistently outperformed bubble sort in terms of speed. For smaller datasets (10 & 100 elements), both algorithms completed the sorting process almost instantly, with a runtime of 0 milliseconds. However, as the dataset size increased, the differences became more noticeable. At 1,000 elements, bubble sort took 36 milliseconds, while quicksort finished in just 1 millisecond. This gap widened significantly with larger datasets: at 5,000 elements, bubble sort took 937 milliseconds, compared to quicksort's 4 milliseconds. For 10,000 elements, bubble sort's runtime increased to 3,817 milliseconds, while quicksort only took 9 milliseconds. At 25,000 elements, bubble sort took 23,615 milliseconds (over 23 seconds), whereas quicksort completed the task in 25 milliseconds.

The trend continued with 50,000 elements (BS: 94,596 milliseconds, QS: 54 milliseconds) and 75,000 elements (BS: 213,322 milliseconds, QS: 82 milliseconds). Finally, for 100,000 elements, quicksort took only 117 milliseconds, while bubble sort became impractical due to its excessively long runtime.

The difference in runtime can be explained by the time complexity of each algorithm. bubble sort operates in O(n^2) time, meaning its runtime grows quadratically as the input size increases. This happens because it repeatedly compares and swaps adjacent elements, even when much of the list is already sorted. In contrast, quicksort has an O(n log n) average-case complexity due to its divide and conquer strategy, where it partitions the array into smaller subproblems and sorts them recursively. This greatly reduces the number of comparisons and swaps needed, making quicksort exponentially faster for large datasets.

Below are the graphed results, which clearly illustrate the exponential time complexity of bubble sort compared to the more efficient logarithmic time complexity of quicksort.

- Note: I'm aware it's hard to see the quicksort trend line I tried changing the scaling but that didn't help (not sure if there is a better way to do this, i'll include the google sheets document if you want to see the data table as well) I believe viewing the difference between sorting would be easier if I had done quicksort and insertion sort like the example.

(EXTRA CREDIT) I also did the memory usage for quicksort and merge sort and after running some tests, on average merge sort has higher memory usage than quicksort because merge sort's higher memory usage comes primarily from its non-in-place nature. During the merging process, it creates temporary arrays to hold the divided sub-arrays, effectively requiring extra storage proportional to the input size. This is in contrast to quicksort, which primarily operates in-place, where it rearranges elements within the original array with minimal extra memory allocation. While quicksort does utilize the call stack for recursive calls, resulting in a space

complexity of O(log n) on average, this is significantly less than merge sort's O(n) extra space. Therefore, the need for these temporary arrays during the merge step is what makes merge sort generally more memory-intensive than quicksort, especially for larger datasets. (input > 10,000 elements)

As a quick follow up I did encounter an issue on Mac OS / python where the memory used was not consistent with the input size of data. I later realized this is an issue on mac OS and python where it automatically allocates a certain baseline amount of memory for certain operations so it does not show up on tests sometimes. I recommend using larger datasets for a more significant difference anything below 10,000 elements will barely show any memory usage.