Cody Wu
133001517

PA5 Report

Introduction:

In this assignment, different types of sorting methods were explored: Bubble Sort, Heap Sort, Merge Sort, and QuickSort. All of these methods are different. Bubble swaps adjacent elements as it iterates through the list. Heap sort uses a heap to sort and extract the minimum value. Merge sort splits the array into smaller arrays to sort and then merges them together. Quicksort uses a pivot to split the array into smaller sections and sort each partition separately.

Theoretical Analysis:

Since bubble sort has to iterate through the list multiple times, it is inefficient for larger samples. Bubble sort has a time complexity of $O(n^2)$ for all cases. A known problem is that if the list is sorted, bubble sort has to iterate through anyway. However, this can be solved by some optimization adjustments.

Heap sort has a time complexity of O(n log n) for all cases. This is due to the fact that the heap is log n, and the sort iterates n times for all the elements. This is faster than bubble sort but can be harder to implement without the use of the queue class that was provided for this assignment.

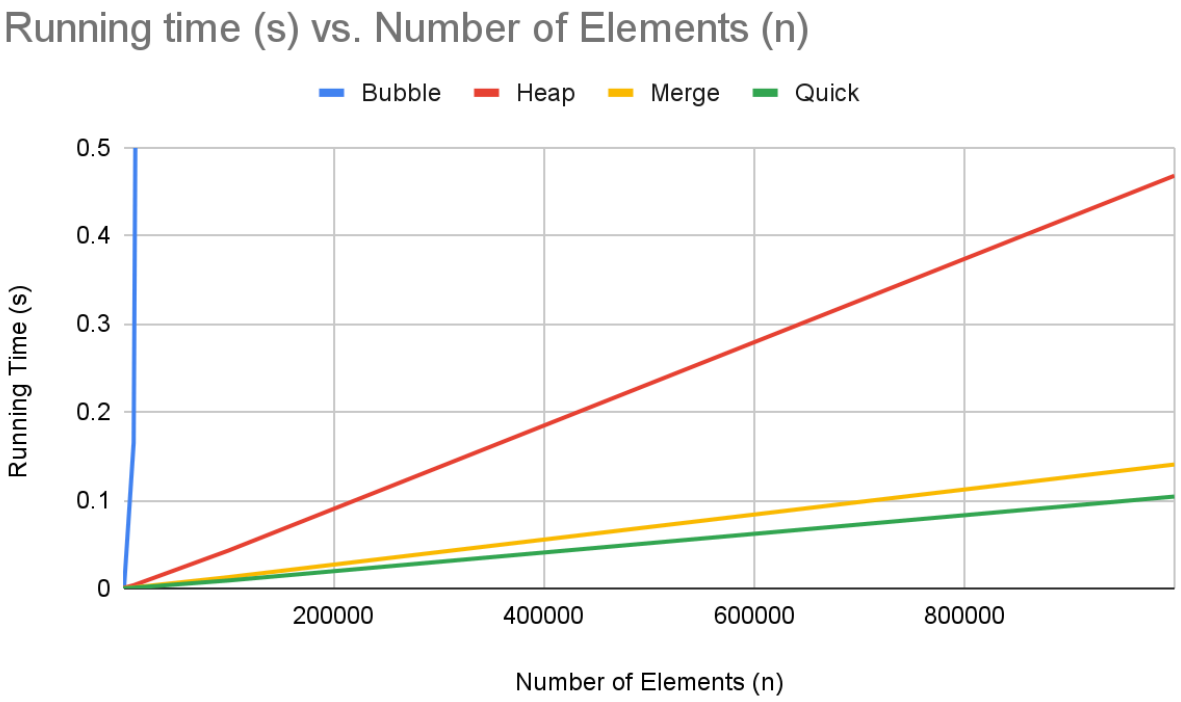
Merge sort has a time complexity of O(n) in the best case and O(n log n) for the average and worst case. Since the arrays get continuously split, it offers a complexity of n log n. In the best case where the array is already sorted, then merge sort has the best time complexity we have seen so far.

Quicksort using a simple random partition has a best and average case of O(n log n) and a worst case of $O(n^2)$. Depending on the partition, it is possible to have a higher or lower time complexity. This also follows the “divide and conquer” design that has an average time complexity of O(n log n) which can also be seen with merge sort. The divide and conquer design has to do with dividing the array and sorting from there which comes out to be O(n log n).

Experimental Setup:

Using randomly generated test inputs and an input size of 10 to 1000000, a better assessment can be concluded. The given test script was modified to change the amount of inputs while keeping its ability to track time and randomize inputs. It gives a wide range of test inputs as the random integer can be between 0 and 999,999. It also can show that certain sorting methods struggle as the number of inputs increases. Each experiment was done at least 5 times to get the most accurate results.

Experimental Results:



In the plot above, it can be seen that bubble sort takes significantly longer than the other elements. It jumps to a run time of 19.55 seconds at 100000 elements. It also performed the worst at every amount of elements tested. Heap sort was next as it was the worst out of the remaining three. Merge sort was barely slower than quicksort. Technically quicksort was the fastest. However, the difference was extremely marginal. The input type did not have a large effect on the time complexities. The theoretical analysis does agree with the experimental results shown with bubble sort. It was predicted to be the slowest and was. However, it was also predicted that heap sort should have the same run times as quick and merge sort. It can be seen

that the heap sort took longer than the others. A reason for this might be because of the heap itself. It has to access the memory (cache) which might cause it to take longer.