Kamran Awan
Byron Hoy
Data Structures & Algorithms I
12 December 2023

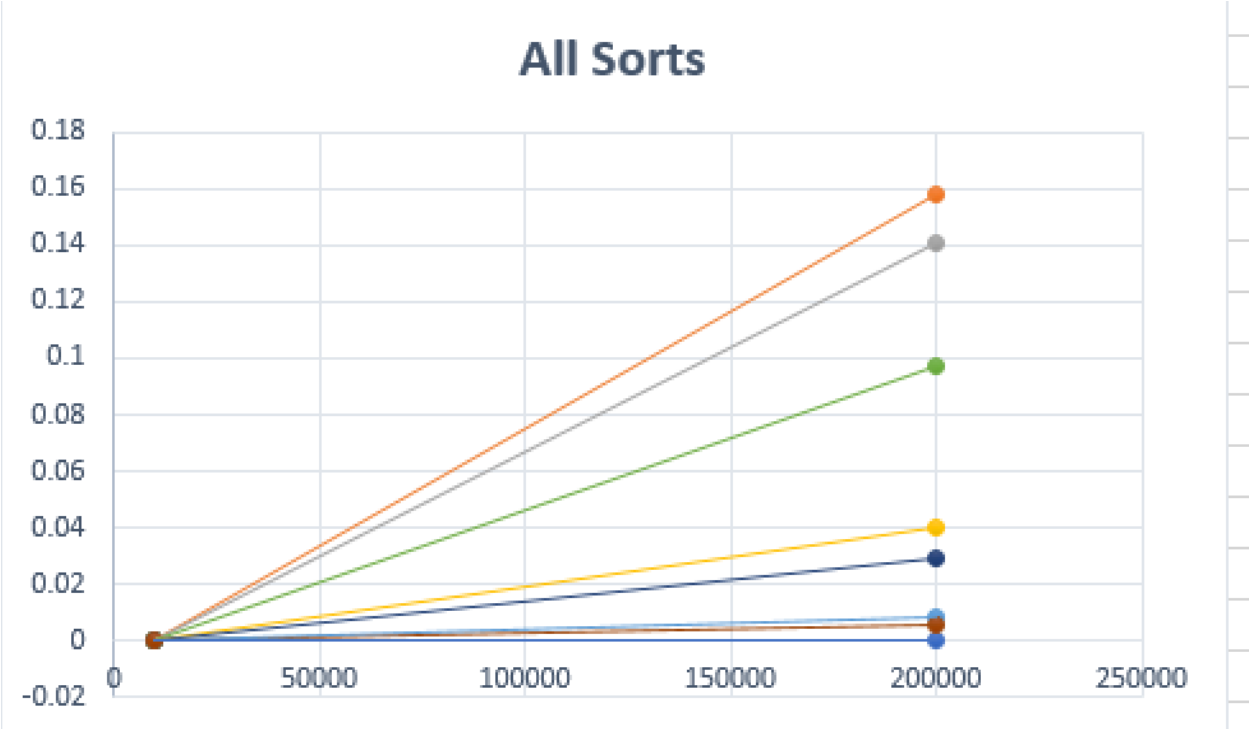
Final Project Report

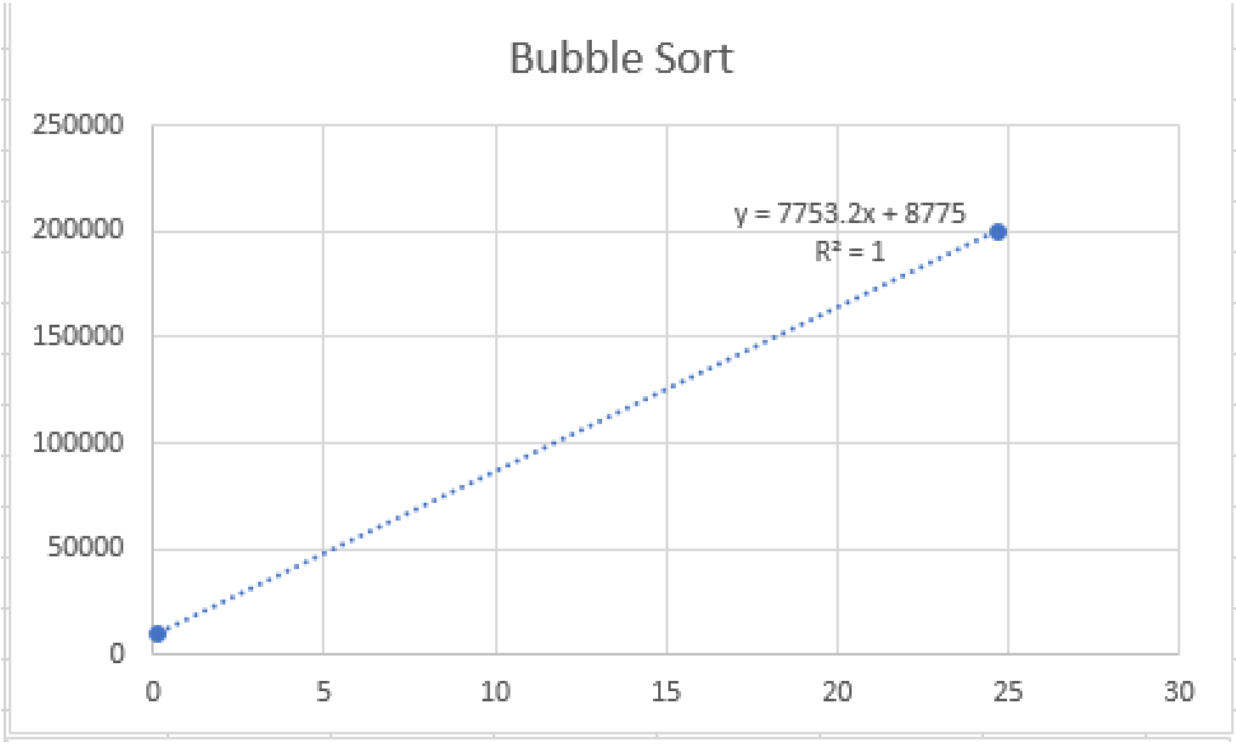
The sorting algorithms that were tested in this project were: bubble sort, selection sort, insertion sort, merge sort, shell sort, quick sort, and heap sort. The sorting algorithms were first tested using an array filled with 10,000 random numbers from 0-9 and then tested with 200,000 numbers to see the difference in time. All the sorting algorithms were able to sort the 10,000 numbers quickly with none of them taking over a second. The sorting algorithm that took the least amount of time to sort the 10,000 numbers was the heap sorting algorithm which only took 6 milliseconds. The one which took the longest to sort the 10,000 numbers was the bubble sort which took 158 milliseconds. This means that the heap sorting algorithm was over 26 times faster than the bubble sort algorithm. Therefore, using the fastest sorting algorithm can be very crucial for reducing the run time of a program. For the 200,000 numbers the algorithms that took a lot of time were the bubble sort, selection sort, insertion sort, and shell sort. The ones that did not take long were the merge sort, quick sort, and heap sort. The difference between the time taken by the bubble sort and the heap sort was almost 25 seconds which is a very big difference. This means that even though all the sorts are viable for a small number of numbers, when the numbers become too large only a few sorts can be used to avoid using up too much time.

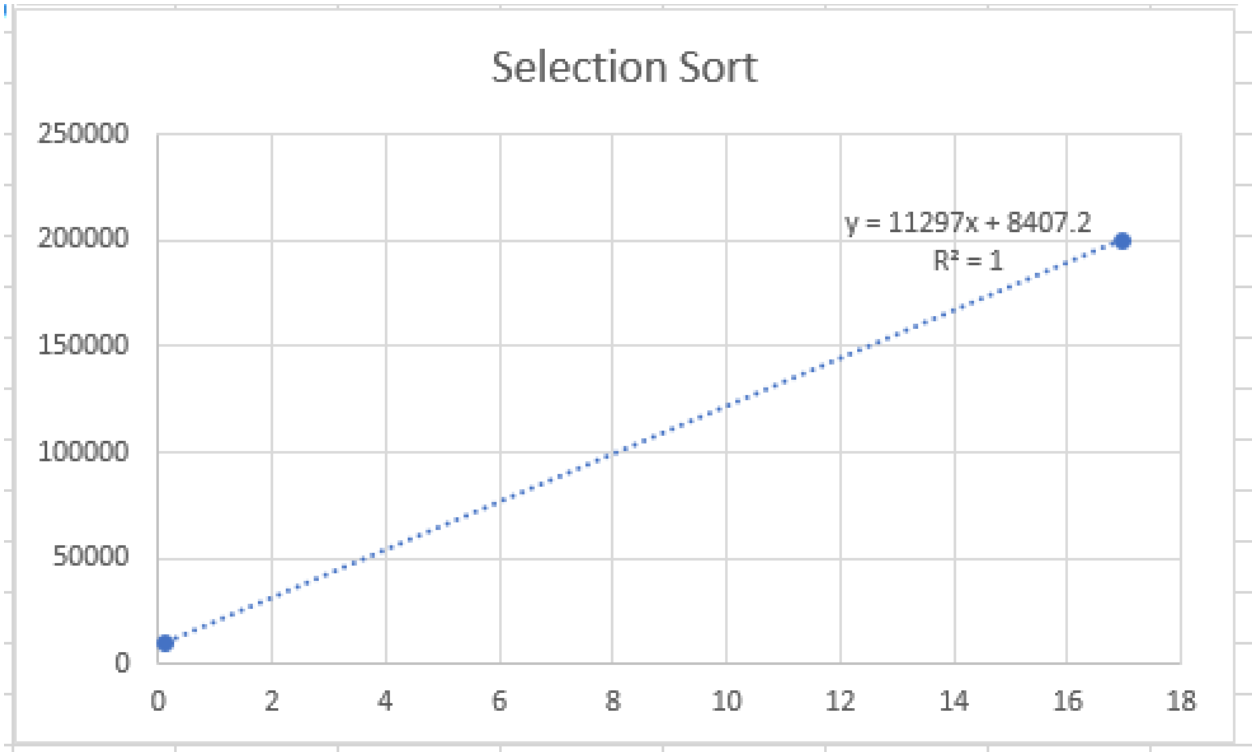
The strengths of bubble sort is that it can be written in a very few lines and it is very easy to understand and read. However, the weaknesses of this sort is that it iterates through the array too many times and takes too long to sort an array with many numbers. This means that even though this sort can be used for an array with a small number of numbers, it will not work well when the numbers exceed a certain amount. The big O of this sort is n^2 which is among the worst time complexities you can get in a sorting algorithm. The selection sort has the same strengths and weaknesses as the bubble sort algorithm. Even though the insertion sort algorithm performed much better than the bubble sort algorithm and the selection sort algorithm it still did not perform well enough to be used properly for a big array of numbers. The same goes for the shell sort. This means that the bubble sort, selection sort, insertion sort, and shell sort are not viable for a big array of numbers even though they are easy to write and understand. The merge sort, quick sort, and heap sort performed much better with a big O time of nlog(n). So the strengths of these algorithms is that they are able to sort big arrays of numbers much faster than the other ones. However, the weaknesses are that they are much more complicated and harder to understand.

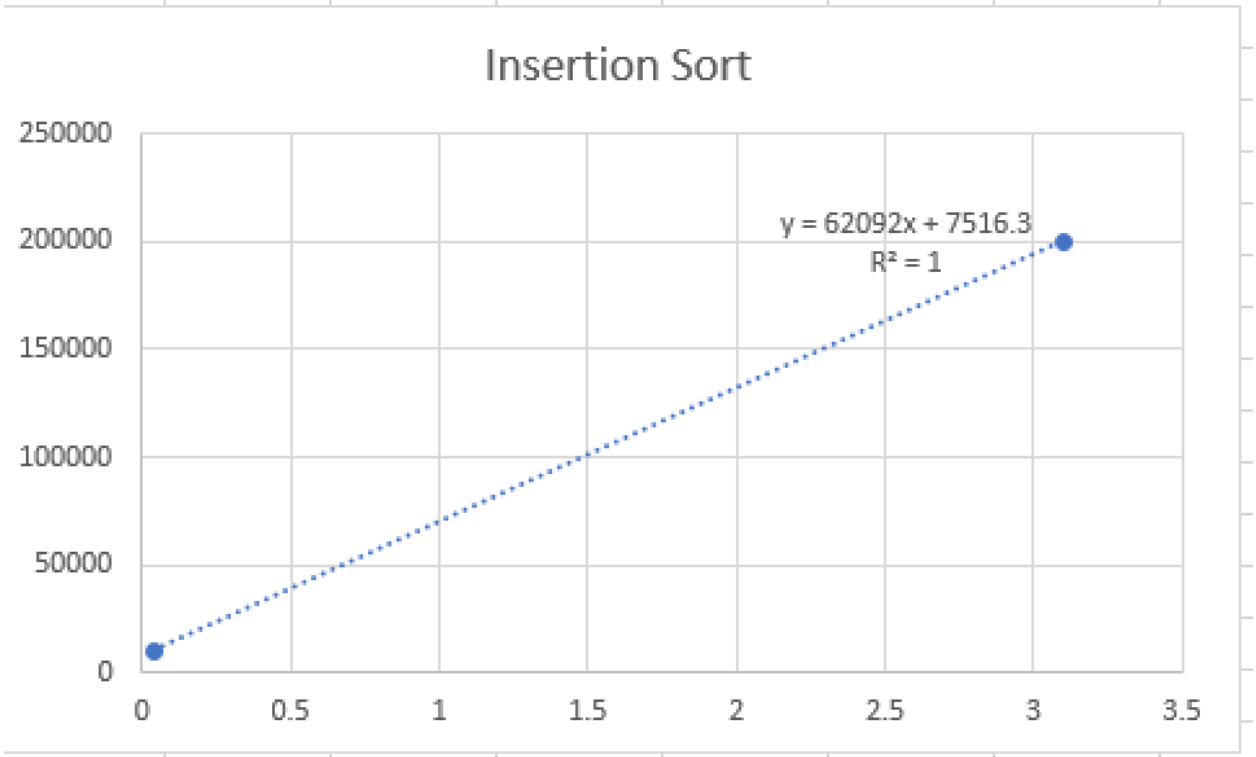
The bubble sort works by iterating through the array and then checking if each element after the current one is greater and switching them if it is. The selection sort iterates through the array and finds the smallest element in front of the current one and switches them. The insertion sort works by iterating through the array, removing the current element and placing it in its correct position. Merge sort divides the array into individual elements and then merges them to create a sorted array. The shell sorting algorithm is based on the insertion sorting algorithm. The shell sorting algorithm works by splitting the array into smaller arrays and then sorting them using an insertion sort. The quick sorting algorithm functions by using a pivot, the array is splitting with one side being less than the pivot and the other side greater than the pivot. This process repeats itself until all the elements are sorted, and then the elements are merged together

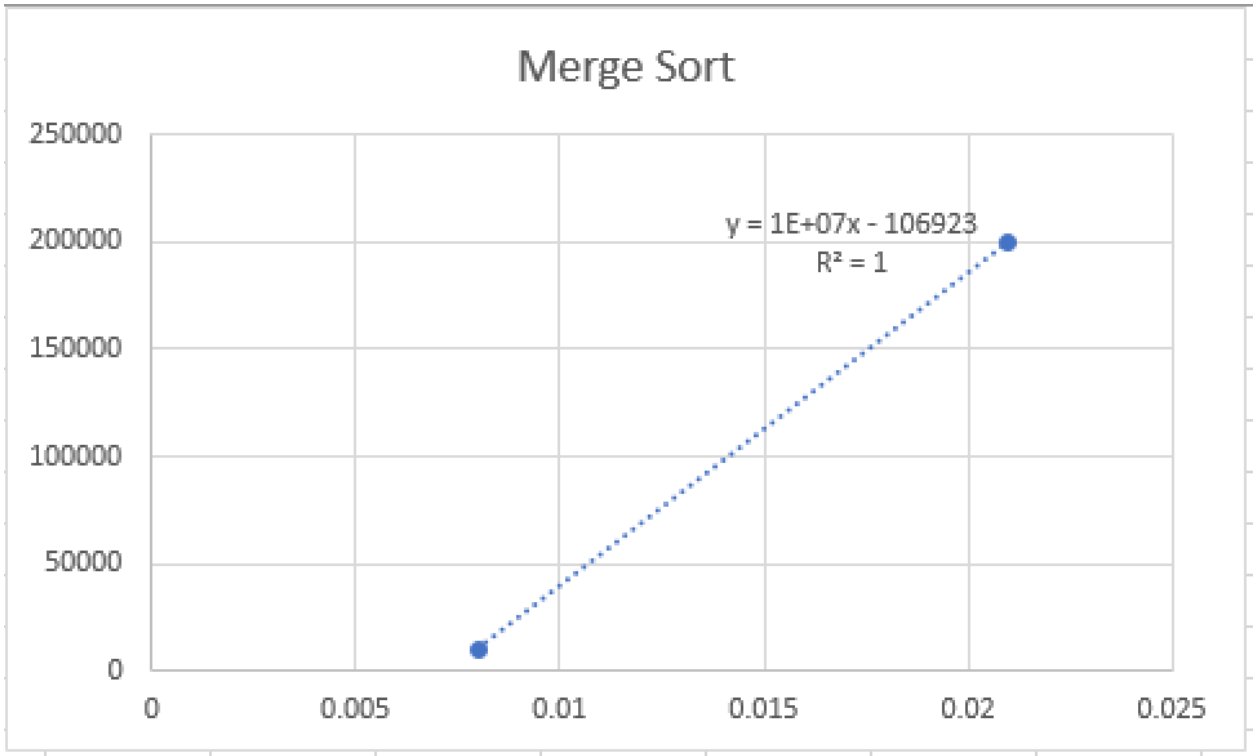
to create the sorted array. The heap sorting algorithm works by finding the largest element and placing it at the end each time it iterates.

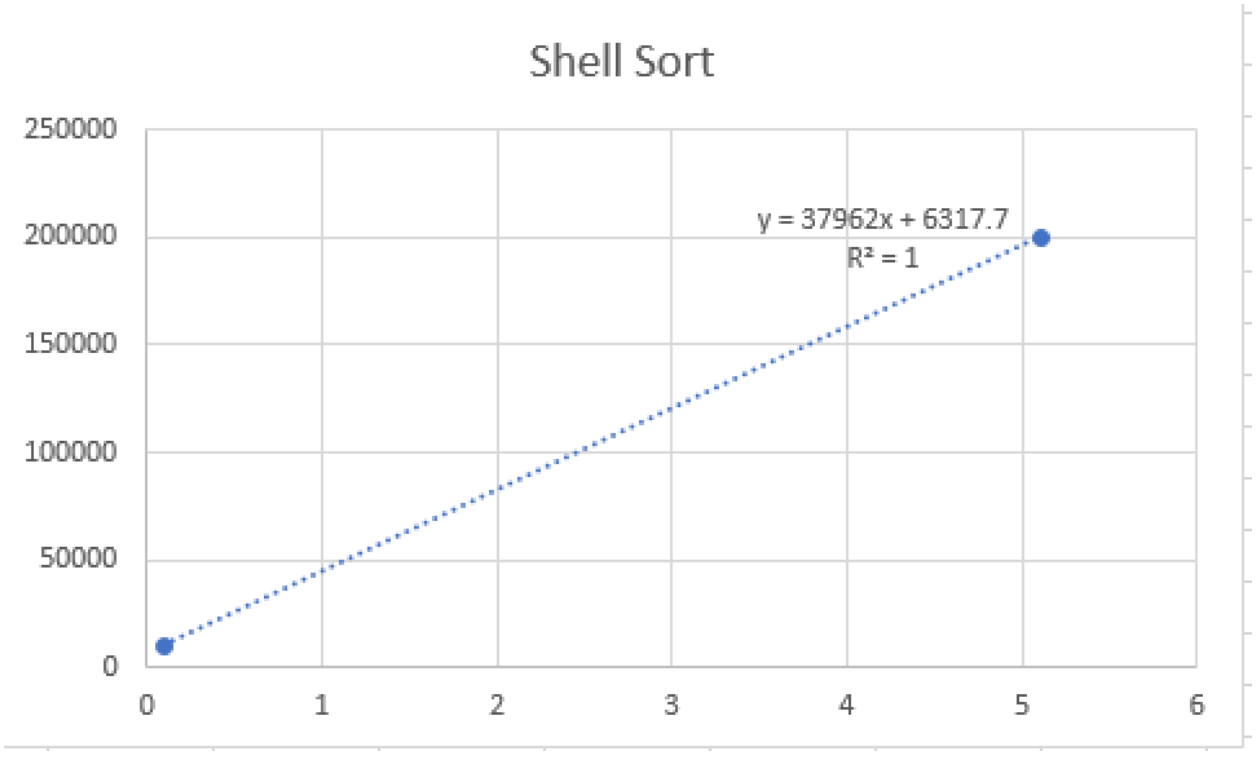


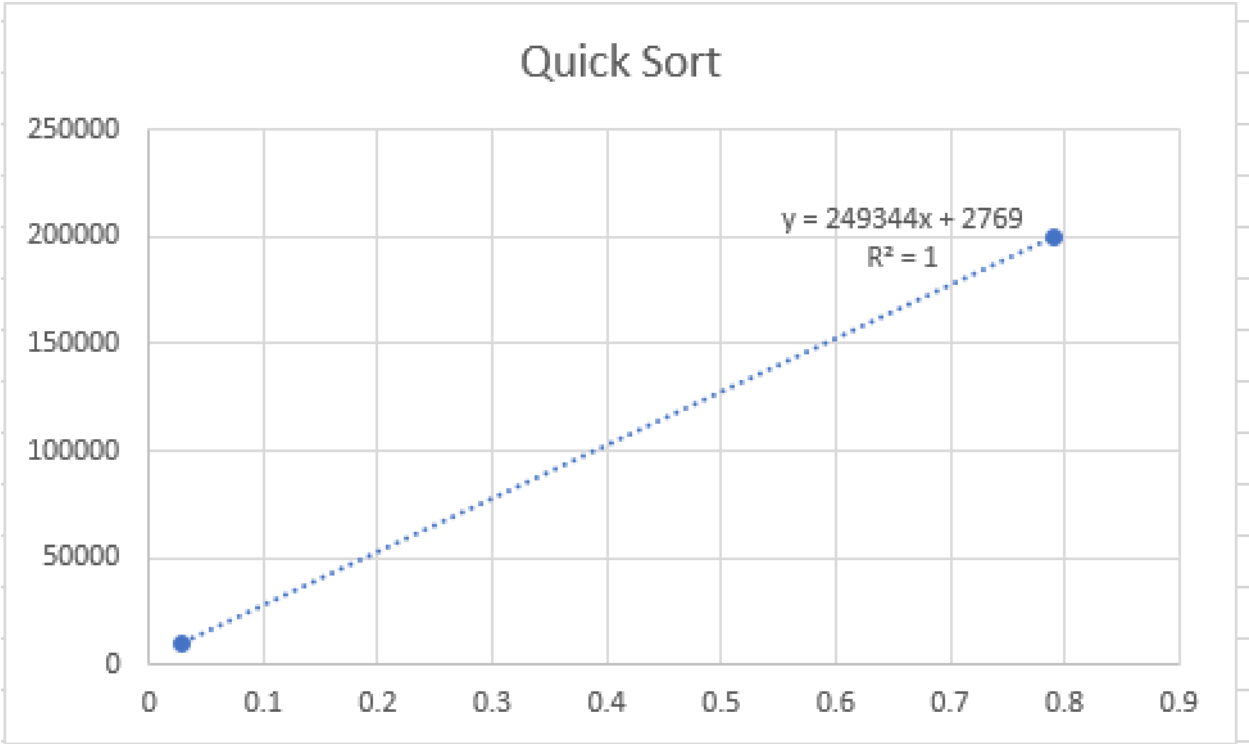


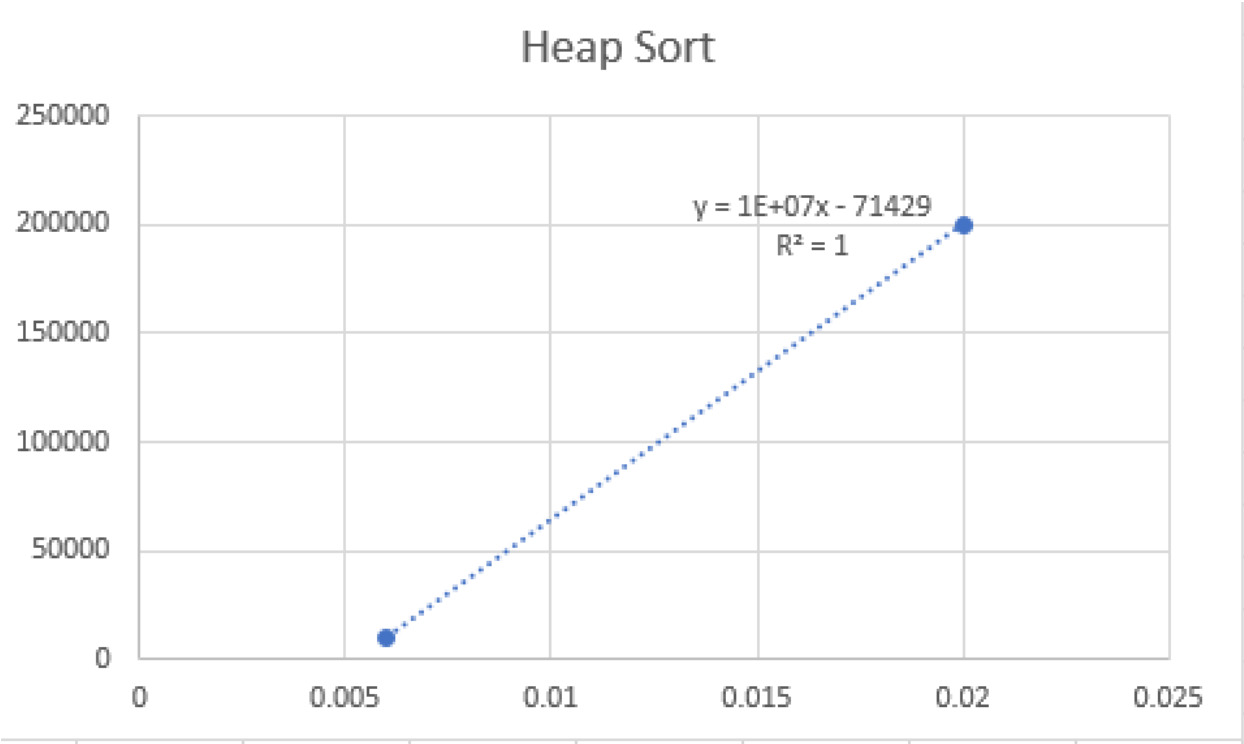

Selection Sort
250000
200000
150000
100000
50000
0
0
2
4
6
8
10
12
14
16
18
y = 11297x + 8407.2
R² = 1


Insertion Sort
250000
200000
150000
100000
50000
0
0
0.5
1
1.5
2
2.5
3
3.5
y = 62092x + 7516.3
R² = 1


Merge Sort
250000
200000
150000
100000
50000
0
0
0.005
0.01
0.015
0.02
0.025
y = 1E+07x - 106923
R² = 1


Shell Sort
250000
200000
150000
100000
50000
0
0
1
2
3
4
5
6
y = 37962x + 6317.7
R² = 1


Quick Sort
250000
200000
150000
100000
50000
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
y = 249344x + 2769
R² = 1


Heap Sort
250000
200000
150000
100000
50000
0
0
0.005
0.01
0.015
0.02
0.025
y = 1E+07x - 71429
R² = 1

sorts	first time	2nd time
bubble sort	0.158	24.664
	10000	200000
selection sort	0.141	16.96
	10000	200000
insertion sort	0.04	3.1
	10000	200000
merge sort	0.008	0.021
	10000	200000
shell sort	0.097	5.102
	10000	200000
quick sort	0.029	0.791
	10000	200000
heap sort	0.006	0.02
	10000	200000

Works Cited

https://www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm

https://www.geeksforgeeks.org/quick-sort/

https://www.geeksforgeeks.org/java-program-for-heap-sort/