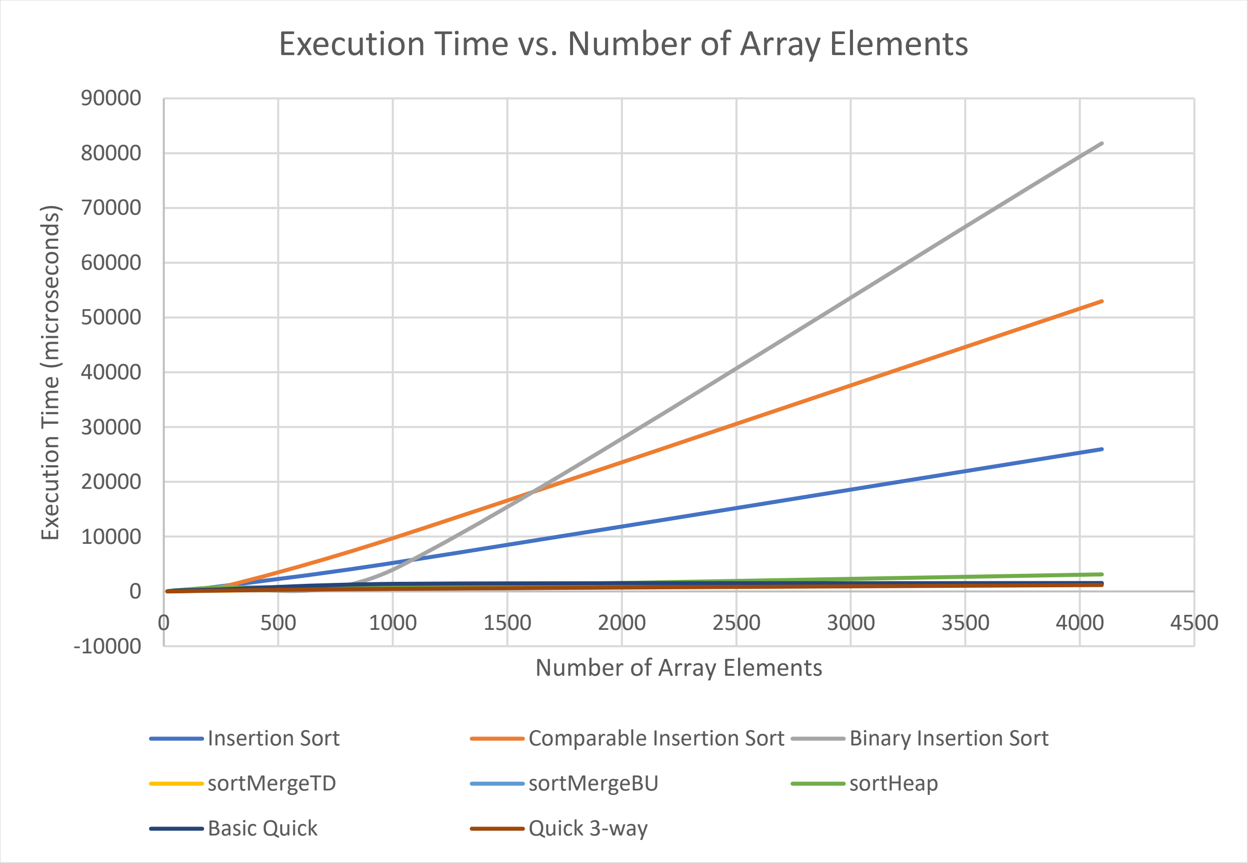
Experimental Analysis of the Implementations

3.1.1

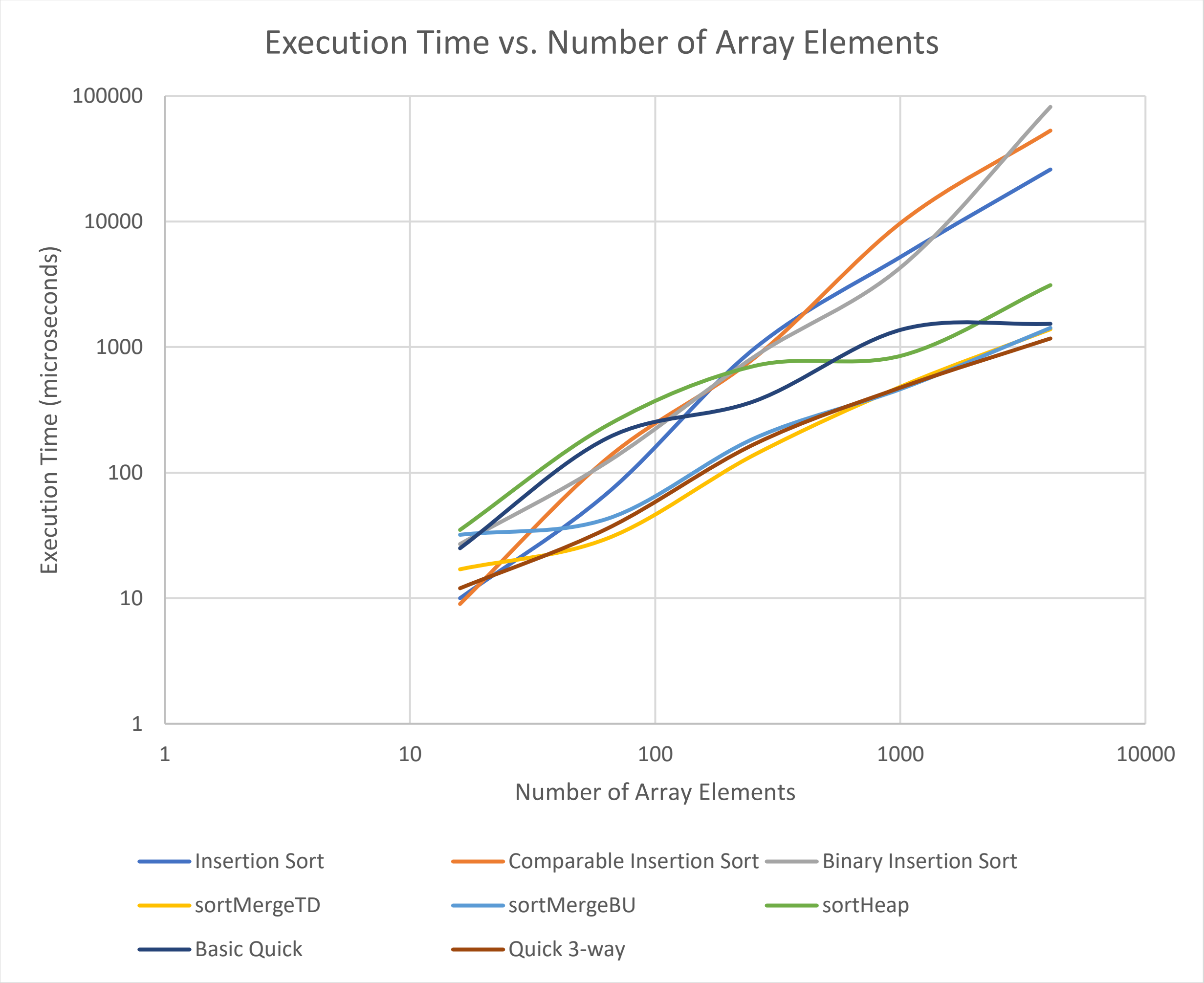
Data (time given in microseconds):

size	Insertion Sort	Comparable Insertion Sort	Binary Insertion Sort	sortMergeTD	sortMergeBU	sortHeap	Basic Quicksort	Quicksort 3-way
16	10	9	27	17	32	35	25	12
64	69	131	122	30	43	239	189	36
256	986	829	848	140	189	710	373	170
1024	5359	10040	4457	492	470	860	1387	483
4096	25962	52974	81794	1387	1421	3113	1532	1171

Normal scale graph:



Log-log scale graph:



3.1.2

The running time for the regular insertion sort without comparable is linear as the number of array elements to be sorted increases. The trendline equation given through excel is y = 6.4265x – 535.35 and this will give the approximation for execution time for x number of array elements. In general the Big-O running time is about O(N).

The running time for the comparable insertion sort is linear as the number of array elements to be sorted increases. The trendline equation given through excel is y = 13.202x – 1609.9 and this will give the approximation for execution time for x number of array elements. In general the Big-O running time is about O(N).

The running time for binary insertion sort is also approximately linear (except in the beginning) as the number of array elements to be sorted increases. The trendline equation given through excel is y = 20.483x – 4901.7 and this will give the approximation for execution time for x number of array elements. In general, the Big-O running time is about O(N). The performance of the “optimized” insertion sort is faster when the number of array elements is small but the running time worse than normal insertion sort and comparable insertion sort for large array elements.

The running time for the Top Down Merge Sort is linearithmic. The trendline equation given through excel is y = (1.3253)x^(0.8368) and this will give the approximation for execution time for x number of array elements. In general the big-O running time is about O(NlogN).

The running time for the Bottom Up Merge Sort is also linearithmic. The trendline equation given through excel is y = (3.27)x^(0.7198) and this will give the approximation for execution time for x number of array elements. In general the big-O running time is about O(NlogN).

The running time for Heap Sort is also linearithmic. The trendline equation given through excel is y = (7.2202)x^(0.7398) and this will give the approximation for execution time for x number of array elements. In general the big-O running time is about O(NlogN).

The running time for basic quick sort is linearithmic. The trendline equation given through excel is y = (5.4776)x^(0.7375) and this will given the approximate execution time for x number of array elements. In general, the big-O running time for the basic quicksort algorithm is about O(NlogN) however, in the worst case the big-O running time is quadratic.

The running time for 3-way quick sort is linearithmic. The trendline equation given through excel is y = (1.2054)x^(0.8482) and this will given the approximate execution time for x number of array elements. In general, the big-O running time for the 3-way quicksort algorithm is about O(NlogN) however, in the worst case the big-O running time is quadratic.

3.1.3

Going by my hypothesis and trendline equations of the graphs produced, here are the projected running times of each of sorting algorithms in microseconds:

Size	Insertion	CompareInsert	BinaryInsert	TD Merge	BU Merge	Heap Sort
16384	104756.426	214691.668	330691.772	4455.955	3532.541	9470.591
65536	420631.000	863596.372	1337472.188	14214.926	9581.864	26410.771

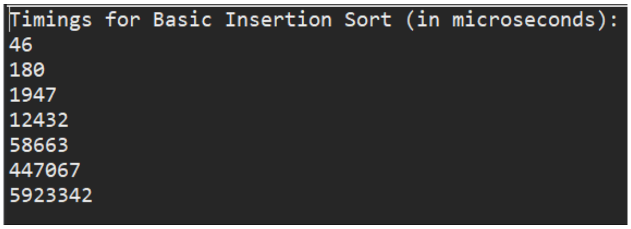
Size	Basic Quicksort	3-way QuickSort
16384	7026.273	4526.913
65536	19531.889	14671.329

3.1.4

Size	Insertion	CompareInsert	BinaryInsert	TD Merge	BU Merge	Heap Sort
16384	447067	552623	326913	7763	12104	14058
65536	5923342	8223970	5217321	79517	68716	22504

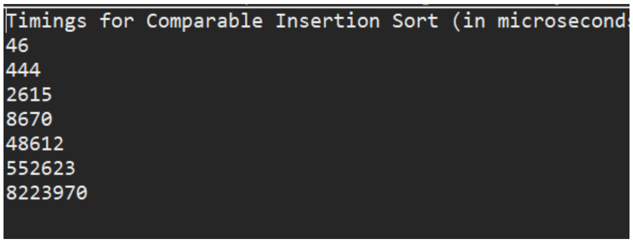
Size	Basic Quicksort	3-way QuickSort
16384	4515	5336
65536	15362	39377

Results of testing regular Insertion Sort:



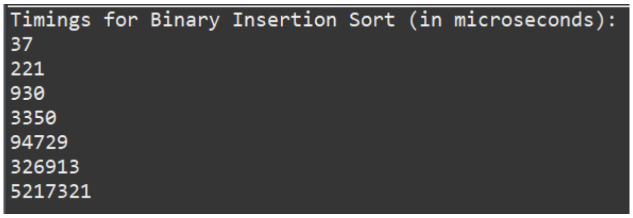
```
Timings for Basic Insertion Sort (in microseconds):
46
180
1947
12432
58663
447067
5923342
```

Results of testing comparable Insertion Sort:



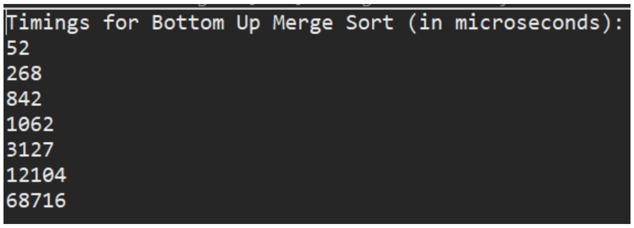
```
Timings for Comparable Insertion Sort (in microsecond
46
444
2615
8670
48612
552623
8223970
```

Results of testing binary Insertion Sort:



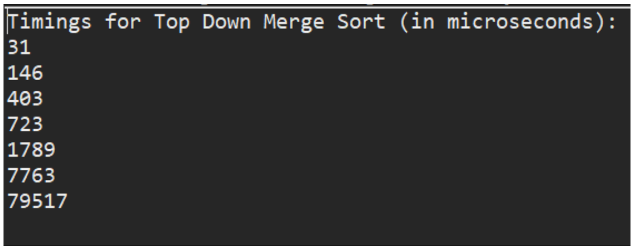
```
Timings for Binary Insertion Sort (in microseconds):
37
221
930
3350
94729
326913
5217321
```

Results of testing Bottom Up Merge Sort:



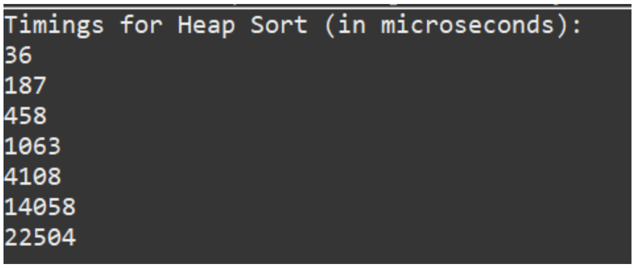
```
Timings for Bottom Up Merge Sort (in microseconds):
52
268
842
1062
3127
12104
68716
```

Results of testing Top Down Merge Sort:



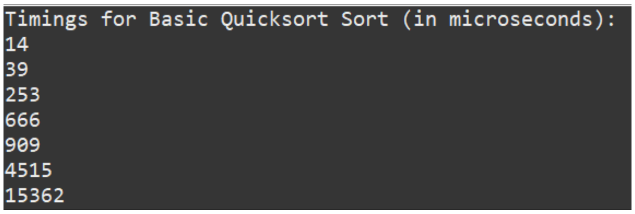
```
Timings for Top Down Merge Sort (in microseconds):
31
146
403
723
1789
7763
79517
```

Results of testing Heap Sort:



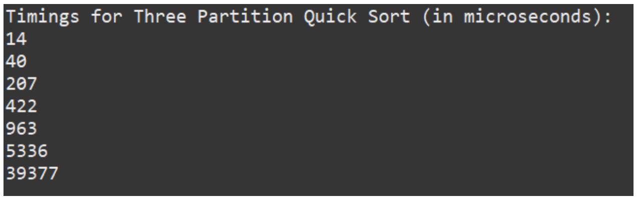
```
Timings for Heap Sort (in microseconds):
36
187
458
1063
4108
14058
22504
```

Results of testing Basic Quicksort:



```
Timings for Basic Quicksort Sort (in microseconds):
14
39
253
666
909
4515
15362
```

Results of testing 3-way Quicksort:



```
Timings for Three Partition Quick Sort (in microseconds):
14
40
207
422
963
5336
39377
```