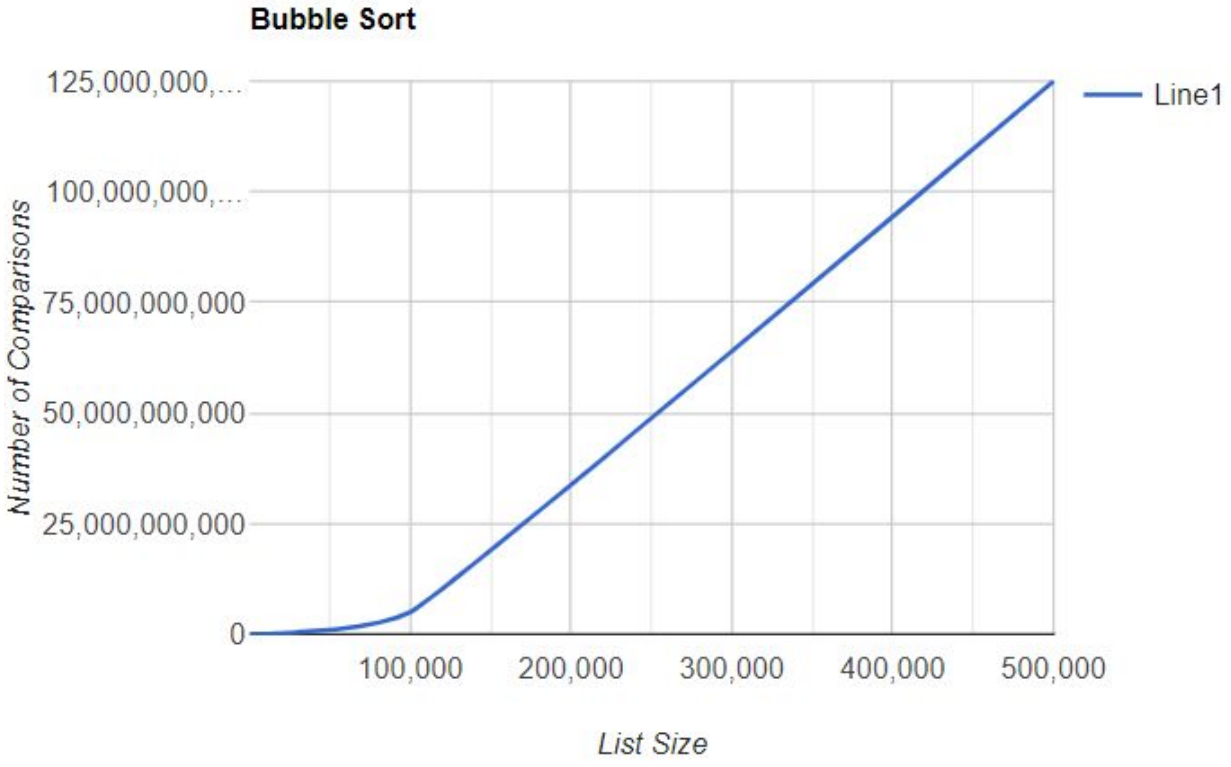
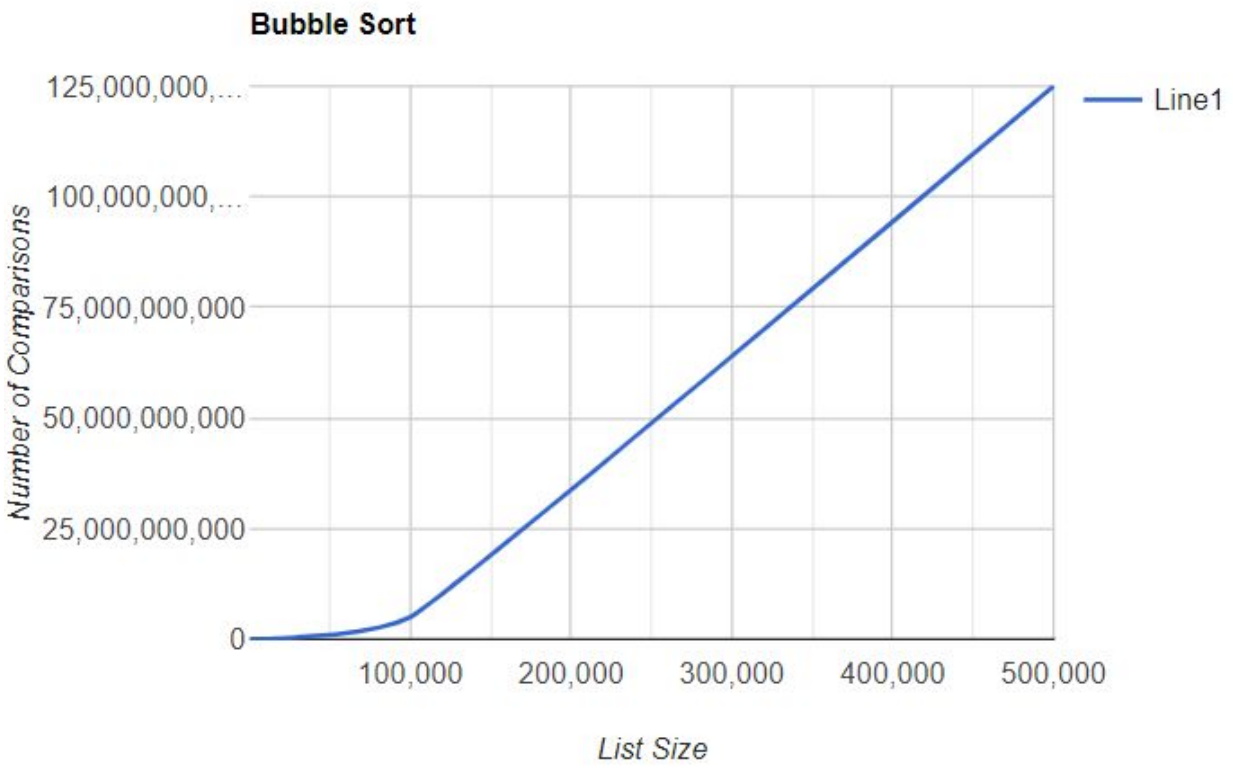
Nathan Kennedy
Jeremiah Lee
Claire Minahan

Lab 6

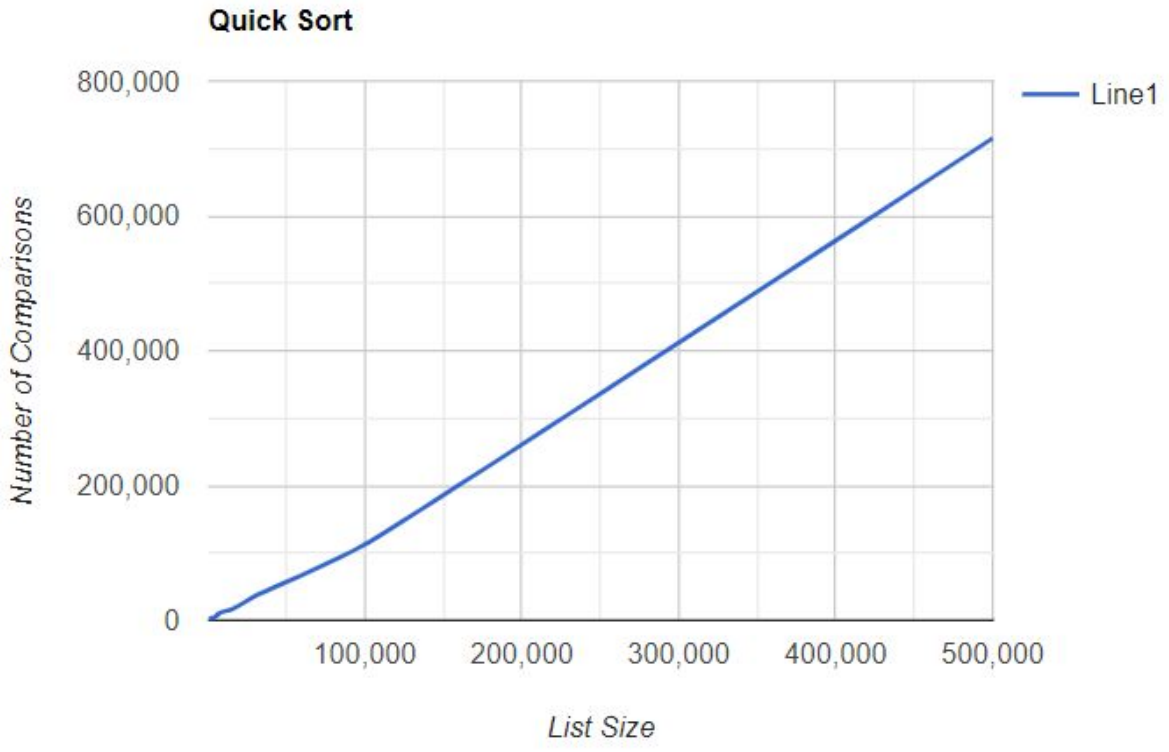
Algorithm Name: Bubble Sort	Type of List: Unsorted	Author: Nathan Kennedy
List Size	Comparisons	Time (seconds)
1,000 (observed)	499500	0.27431559562683105
2,000 (observed)	1999000	1.018017053604126
4,000 (observed)	7998000	2.878291130065918
8,000 (observed)	31996000	12.15920615196228
16,000 (observed)	127992000	47.972615003585815
32,000 (observed)	511984000	205.51697039604187
100,000 (observed)	4999950000	1485.4006497859955
500,000 (observed)	124999750000	28995.29119038582



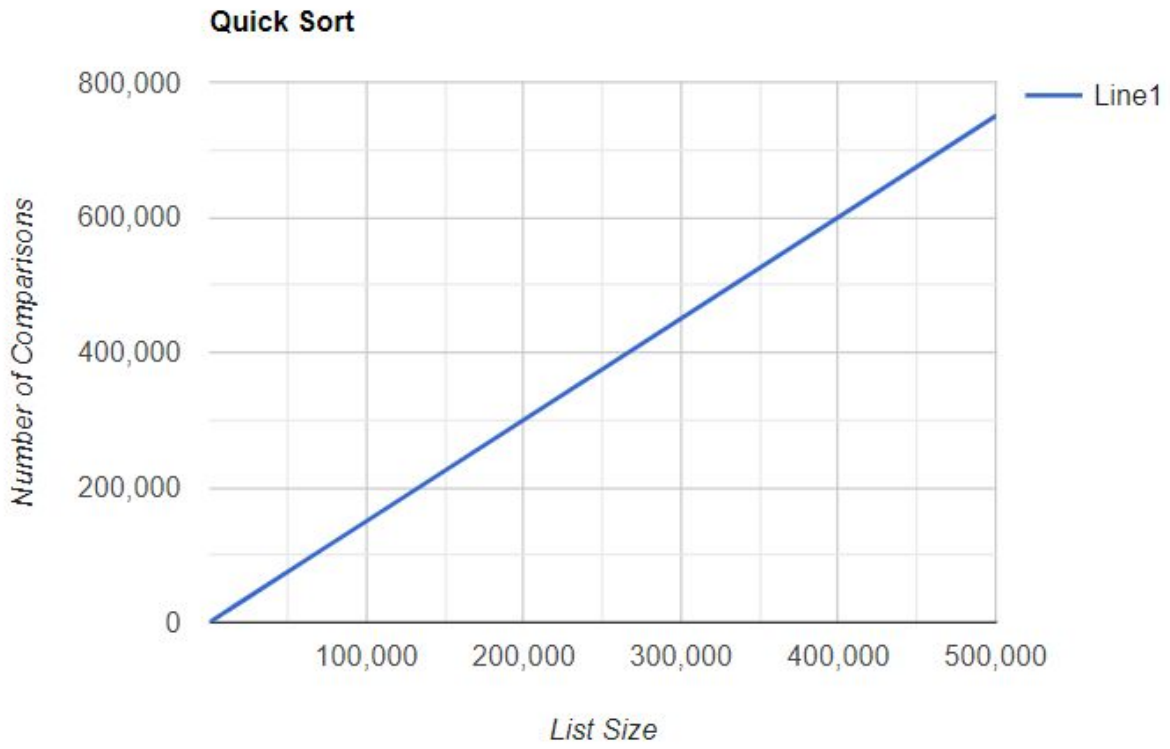
Algorithm Name: Bubble Sort	Type of List: Sorted	Author: Nathan Kennedy
List Size	Comparisons	Time (seconds)
1,000 (observed)	499500	0.14954924583435059
2,000 (observed)	1999000	0.5100846290588379
4,000 (observed)	7998000	1.703441858291626
8,000 (observed)	31996000	8.710205078125
16,000 (observed)	127992000	30.077494382858276
32,000 (observed)	511984000	125.35753154754639
100,000 (observed)	4999950000	621.3694348335266
500,000 (observed)	124999750000	18845.698927402496



Algorithm Name: Quicksort	Type of List: Unsorted	Author: Nathan Kennedy
List Size	Comparisons	Time (seconds)
1,000 (observed)	1818	0.0020294189453125
2,000 (observed)	3251	0.005982637405395508
4,000 (observed)	4199	0.013962268829345703
8,000 (observed)	11352	0.024945497512817383
16,000 (observed)	16986	0.05884099006652832
32,000 (observed)	38127	0.15358853340148926
100,000 (observed)	112304	0.5355660915374756
500,000 (observed)	715289	3.202446699142456



Algorithm Name: Quicksort	Type of List: Sorted	Author: Nathan Kennedy
List Size	Comparisons	Time (seconds)
1,000 (observed)	1499	0.0019936561584472656
2,000 (observed)	2999	0.0040013790130615234
4,000 (observed)	5999	0.008975982666015625
8,000 (observed)	11999	0.020937204360961914
16,000 (observed)	23999	0.05186295509338379
32,000 (observed)	47999	0.12366747856140137
100,000 (observed)	149999	0.4418175220489502
500,000 (observed)	749999	2.2988014221191406

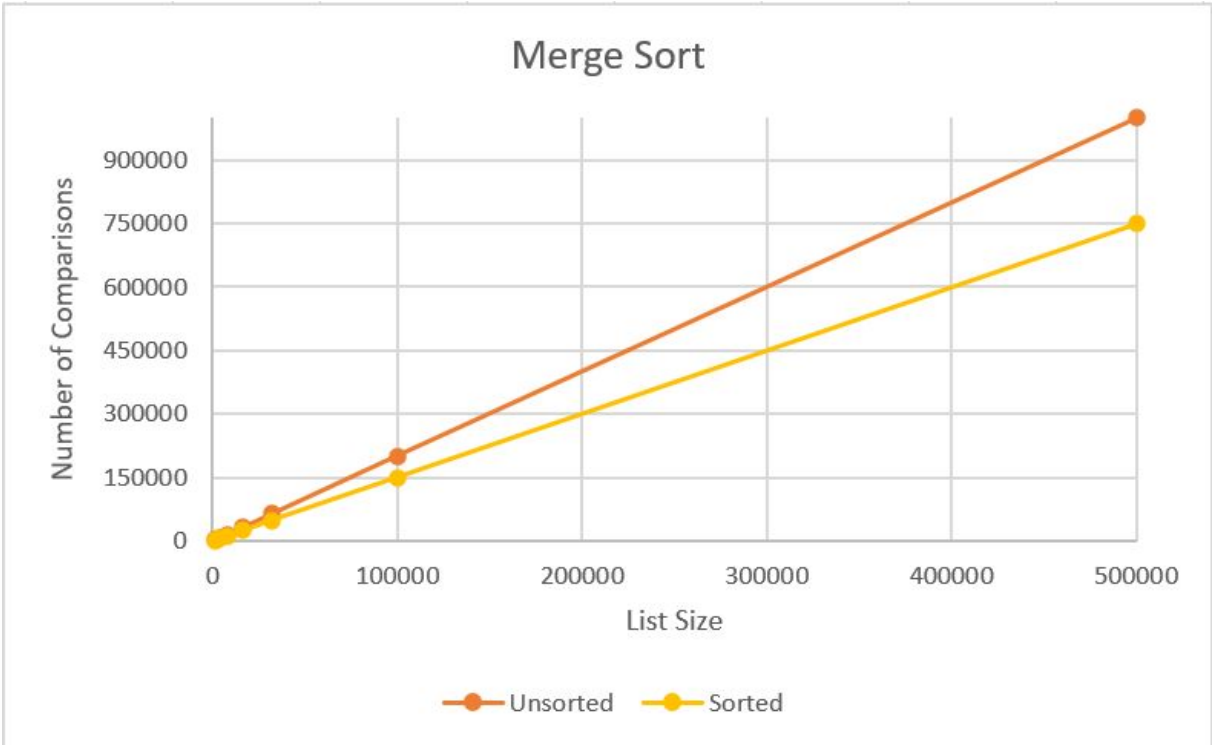


Algorithm Name: Selection Sort	Type of List: Unsorted	Author: Claire Minahan
List Size	Comparisons	Time (seconds)
1,000 (observed)	500,500	0.0807
2,000 (observed)	2,001,000	0.1805
4,000 (observed)	8,002,000	0.8856
8,000 (observed)	32,004,000	3.0997
16,000 (observed)	128,008,000	11.797
32,000 (observed)	512016000	52.1843
100,000 (observed)	5,000,050,000	590.5596
500,000 (observed)	125,000,250,500	29778.637

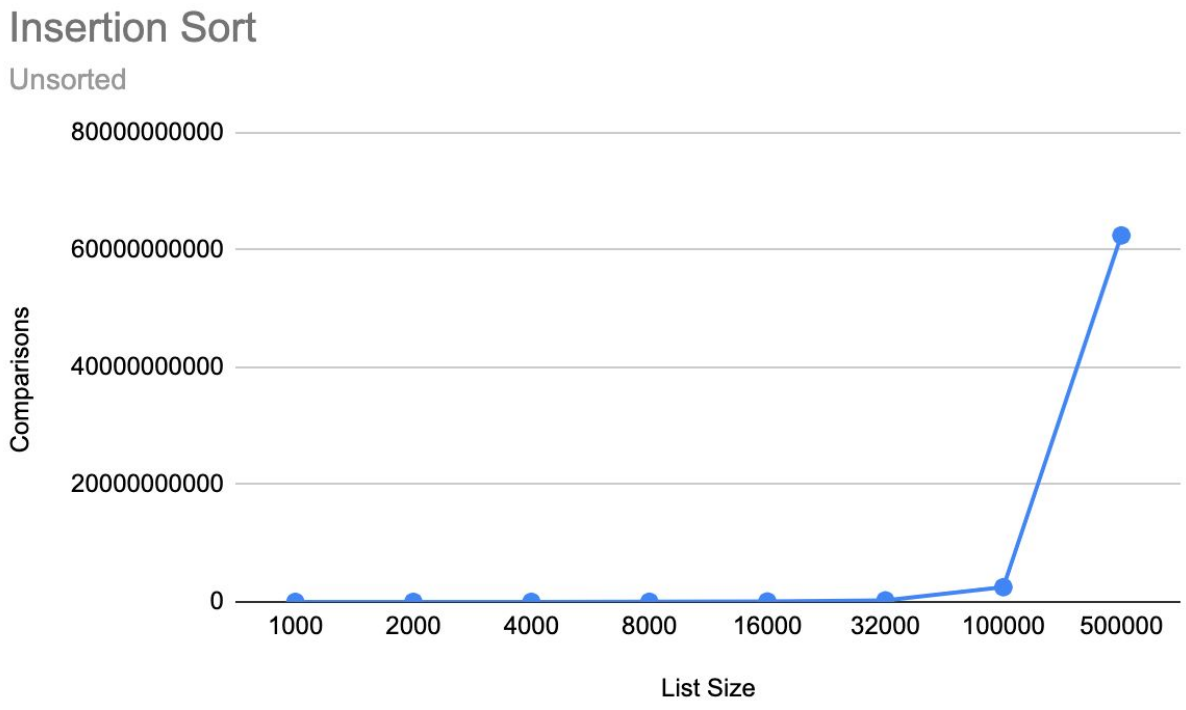
Algorithm Name: Selection Sort	Type of List: Sorted	Author: Claire Minahan
List Size	Comparisons	Time (seconds)
1,000 (observed)	500,500	0.0538
2,000 (observed)	2,001,000	0.2144
4,000 (observed)	8,002,000	0.9863
8,000 (observed)	32,004,000	4.3683
16,000 (observed)	128,008,000	19.1758
32,000 (observed)	512,016,000	68.0765
100,000 (observed)	5,000,050,000	669.735
500,000 (observed)	125,000,250,500	32293.368

Algorithm Name: Merge Sort	Type of List: Unsorted	Author: Claire Minahna
List Size	Comparisons	Time (seconds)
1,000 (observed)	1998	0.00398
2,000 (observed)	3999	0.00698
4,000 (observed)	7999	0.01297
8,000 (observed)	15999	0.0309
16,000 (observed)	31997	0.07977
32,000 (observed)	63999	0.15454
100,000 (observed)	199996	0.80981
500,000 (observed)	999995	3.5229

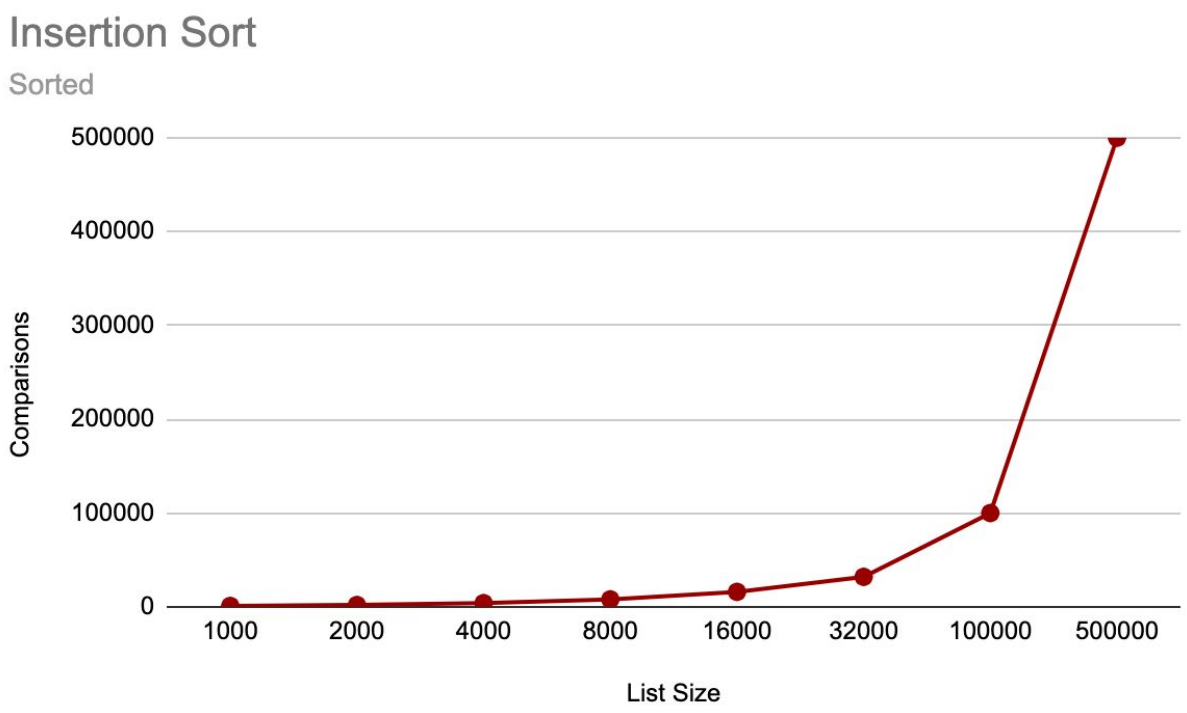
Algorithm Name: Merge Sort	Type of List: Sorted	Author: Claire Minahna
List Size	Comparisons	Time (seconds)
1,000 (observed)	1500	0.004
2,000 (observed)	3000	0.00501
4,000 (observed)	6000	0.0139
8,000 (observed)	12000	0.0219
16,000 (observed)	24000	0.0448
32,000 (observed)	48000	0.10105
100,000 (observed)	150000	0.39106
500,000 (observed)	750000	2.14535



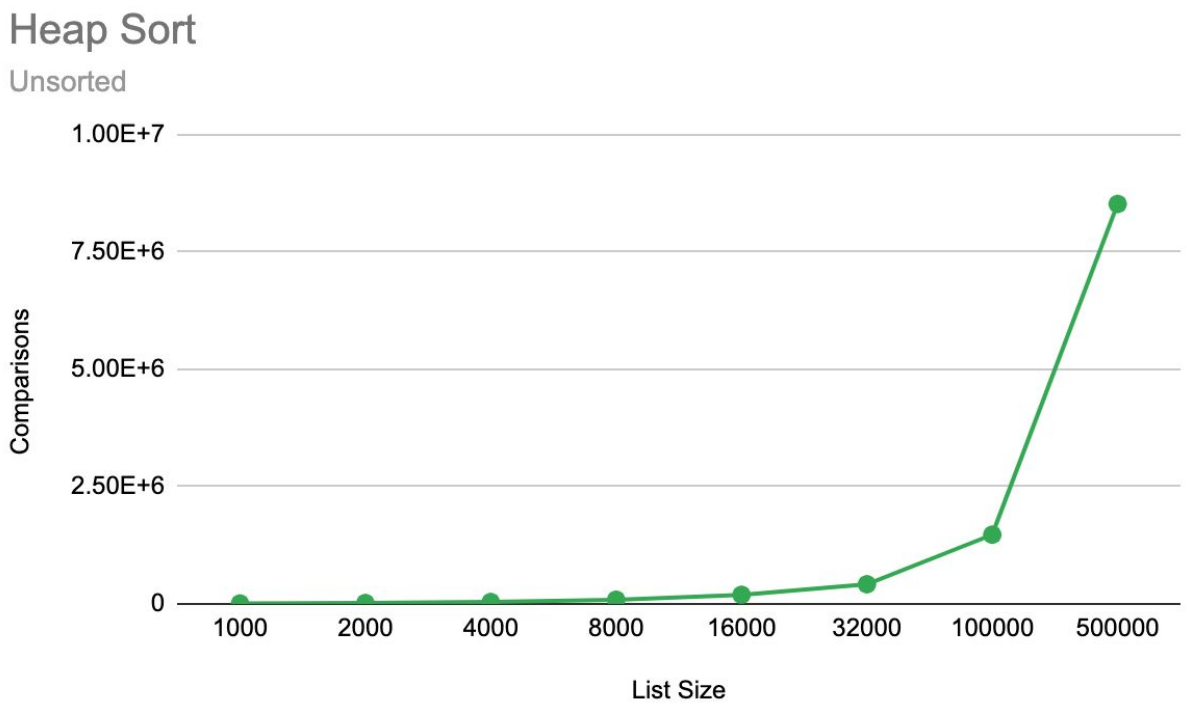
Algorithm Name: Insertion Sort	Type of List: Unsorted	Author: Jeremiah Lee
List Size	Comparisons	Time (seconds)
1,000 (observed)	251400	0.075602
2,000 (observed)	1001818	0.301637
4,000 (observed)	4036150	1.133543
8,000 (observed)	16036834	4.247107
16,000 (observed)	64285470	17.204220
32,000 (observed)	256607820	74.906428
100,000 (observed)	2502539371	791.068595
500,000 (observed)	62488851807	22876.394505



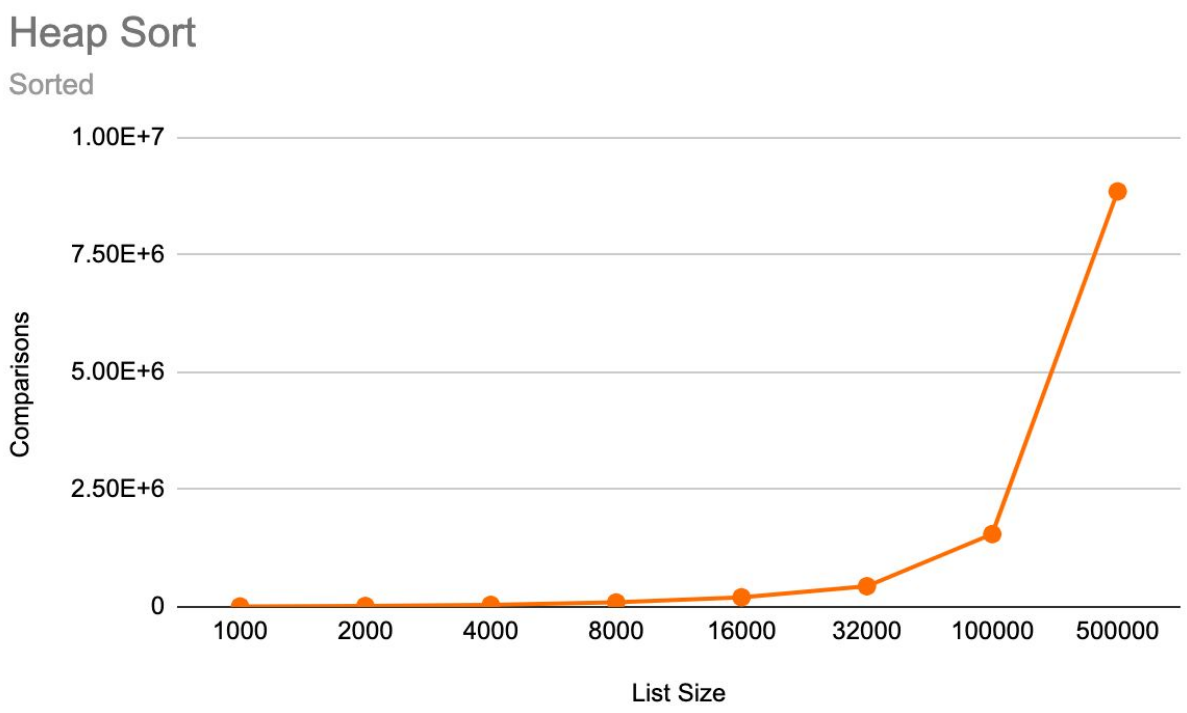
Algorithm Name: Insertion Sort	Type of List: Sorted	Author: Jeremiah Lee
List Size	Comparisons	Time (seconds)
1,000 (observed)	999	0.000194
2,000 (observed)	1999	0.000328
4,000 (observed)	3999	0.000675
8,000 (observed)	7999	0.001315
16,000 (observed)	15999	0.002884
32,000 (observed)	31999	0.009366
100,000 (observed)	99999	0.018769
500,000 (observed)	499999	0.081449



Algorithm Name: Heap Sort	Type of List: Unsorted	Author: Jeremiah Lee
List Size	Comparisons	Time (seconds)
1,000 (observed)	8043	0.005539
2,000 (observed)	18148	0.013429
4,000 (observed)	40260	0.029314
8,000 (observed)	88557	0.056266
16,000 (observed)	193074	0.122938
32,000 (observed)	418360	0.251803
100,000 (observed)	1474578	0.911743
500,000 (observed)	8524633	4.764575s



Algorithm Name: Heap Sort	Type of List: Sorted	Author: Jeremiah Lee
List Size	Comparisons	Time (seconds)
1,000 (observed)	8709	0.006302s
2,000 (observed)	19301	0.014014s
4,000 (observed)	43143	0.026393
8,000 (observed)	94639	0.055496
16,000 (observed)	204381	0.122235
32,000 (observed)	439525	0.256604s
100,000 (observed)	1550855	0.845023s
500,000 (observed)	8855701	4.440075s



Questions:

1. Which sort do you think is better? Why?

We believe that the best sort method is quicksort because it seemed to be the most efficient with both sorted and unsorted lists. Both comparisons ran 3.2 seconds or better.

2. Which sort is better when sorting a list that is already sorted (or mostly sorted)? Why?

The list that is better when sorting a list that is already (or mostly) sorted is heap sort because it takes a relatively small amount of time and doesn't take a large number of comparisons compared to the other sorting algorithms.

3. You probably found that insertion sort had about half as many comparisons as selection sort. Why? Why are the times for insertion sort not half what they are for selection sort?

The reason that insertion sort is about half as many comparisons as selection sort is because in selection sort, the algorithm goes through all the items in the sublist before making any changes, however, with insertion sort, once a comparison is made between two values, it switches them right away, and then proceeds to the next iteration.