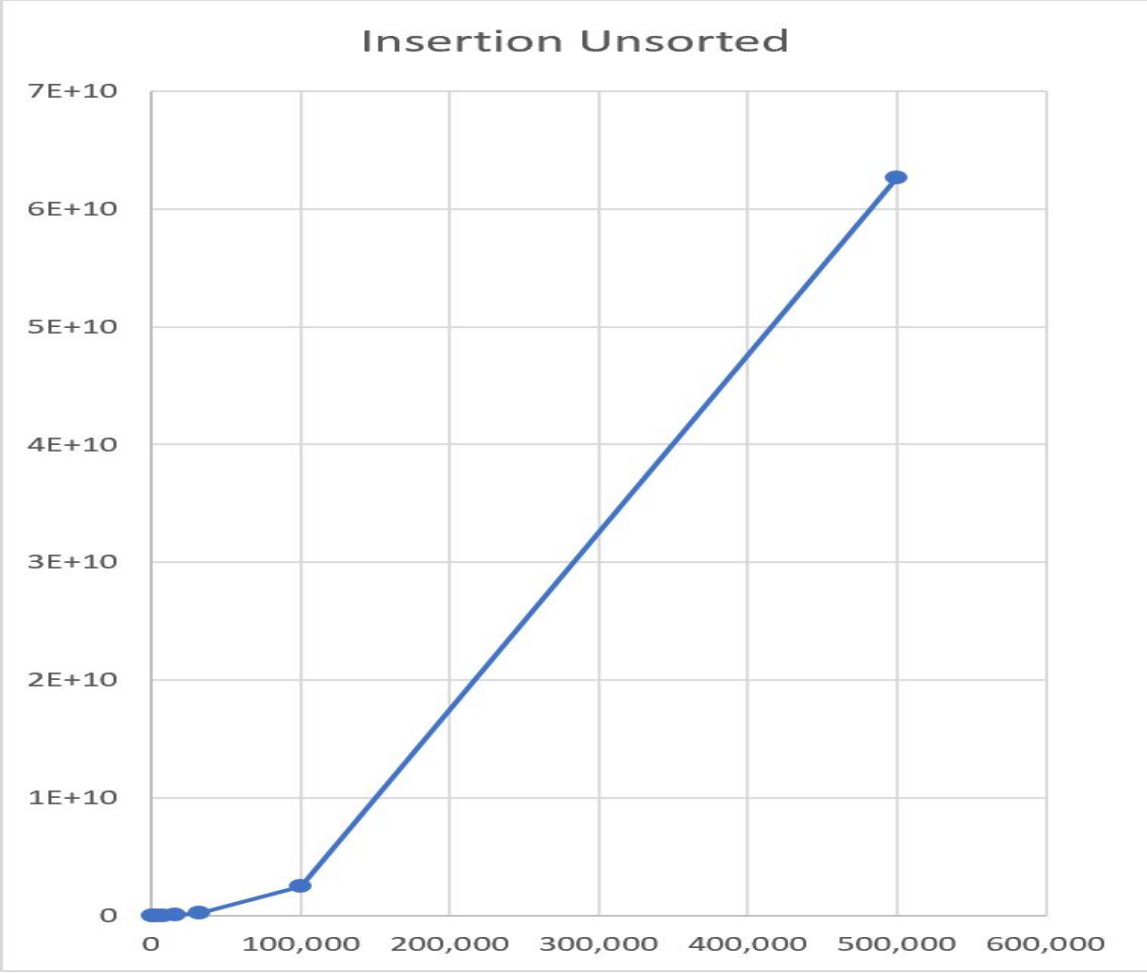
Sullivan Xiong
And
Meha Sharma

Comparing Sorting Algorithms
CPE202 — Lecturer Toshi
5/19/2019

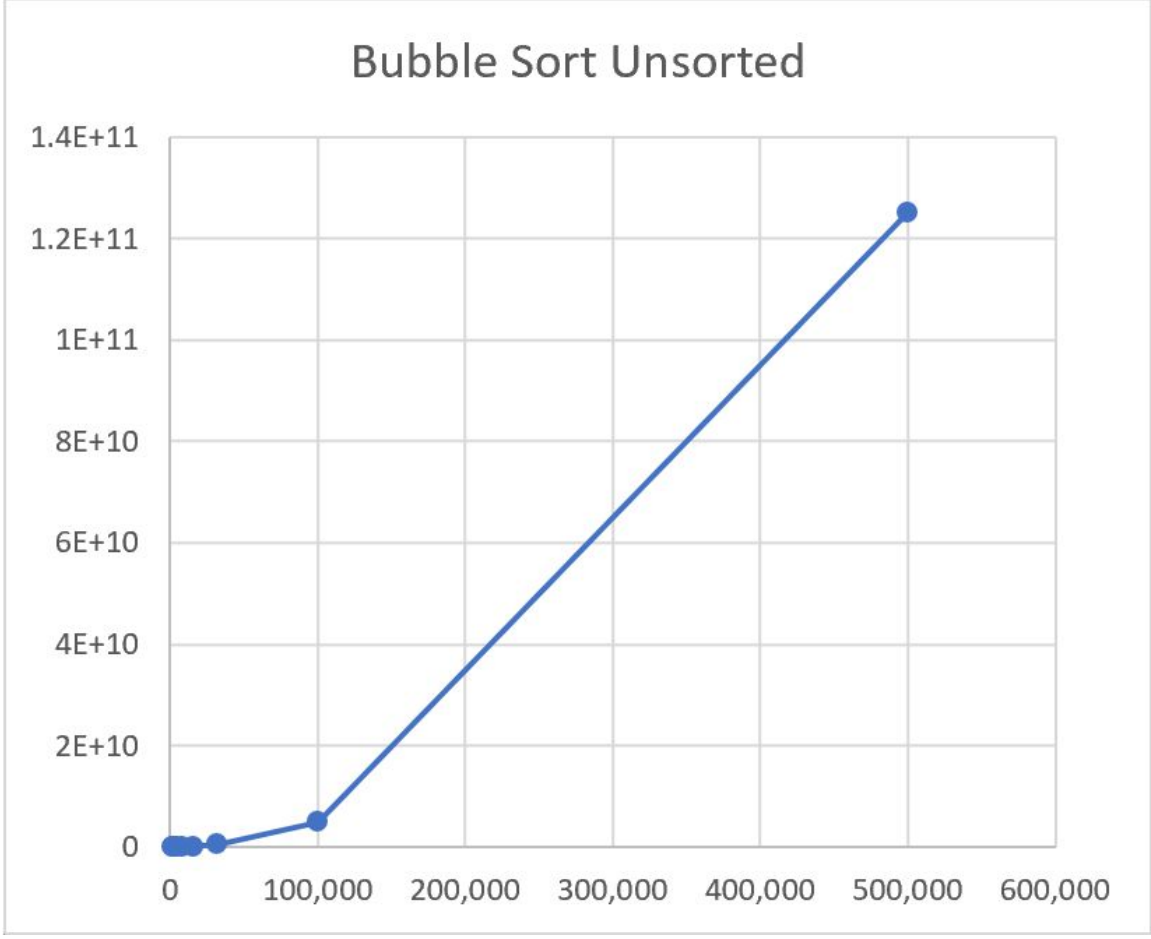
Table of Contents

Algorithm Name: Selection Sort **Type of list (sorted or unsorted)**: Unsorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	499500	0.100726843
2,000	1999000	0.177525759
4,000	7998000	0.695287943
8,000	31996000	2.842987537
16,000	127992000	13.24894118
32,000	511984000	52.55555224
100,000	4999950000	551.8558838
500,000	1.25E+11	13783.23895

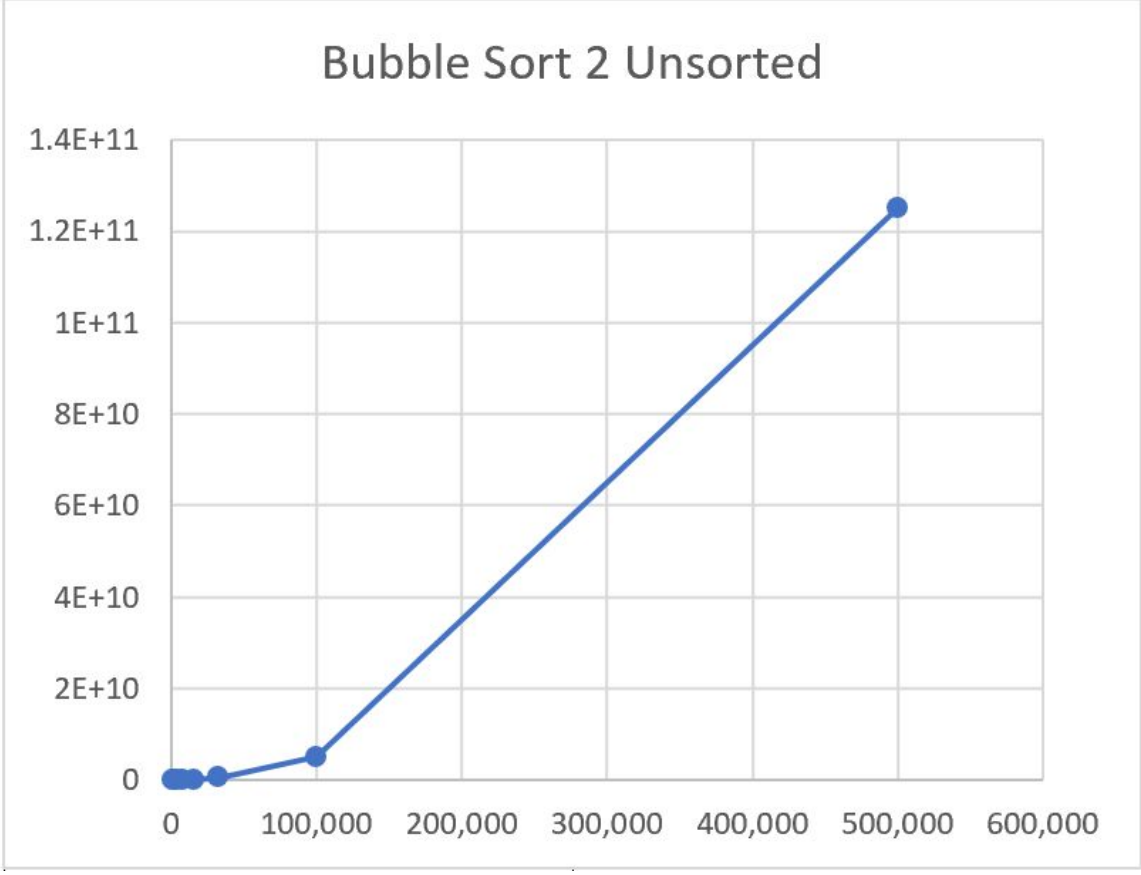
Algorithm Name: Insertion Sort **Type of list (sorted or unsorted)**: Unsorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	254537	0.106714725
2,000	1001444	0.365089893
4,000	3924671	1.202913761
8,000	15805755	7.329817772
16,000	64168190	25.53038287
32,000	255484302	127.7474024
100,000	2503767894	1104.433815
500,000	62604042310	26861.54812



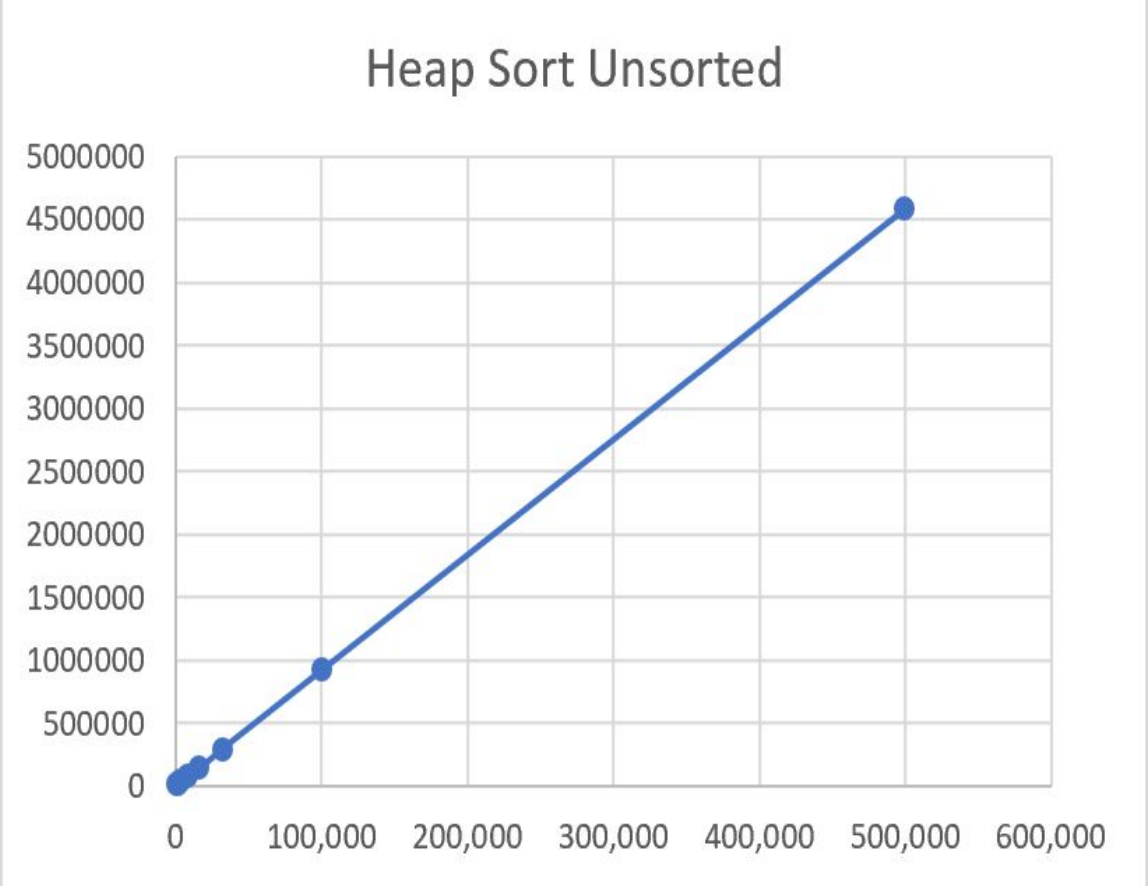
Algorithm Name: Bubble Sort **Type of list (sorted or unsorted)**: Unsorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	499500	0.130650759
2,000	1999000	0.602388859
4,000	7998000	1.55147934
8,000	31996000	8.487509727
16,000	127992000	28.08123088
32,000	511984000	145.7774777
100,000	4999950000	1360.640545
500,000	1.25E+11	34023.29184



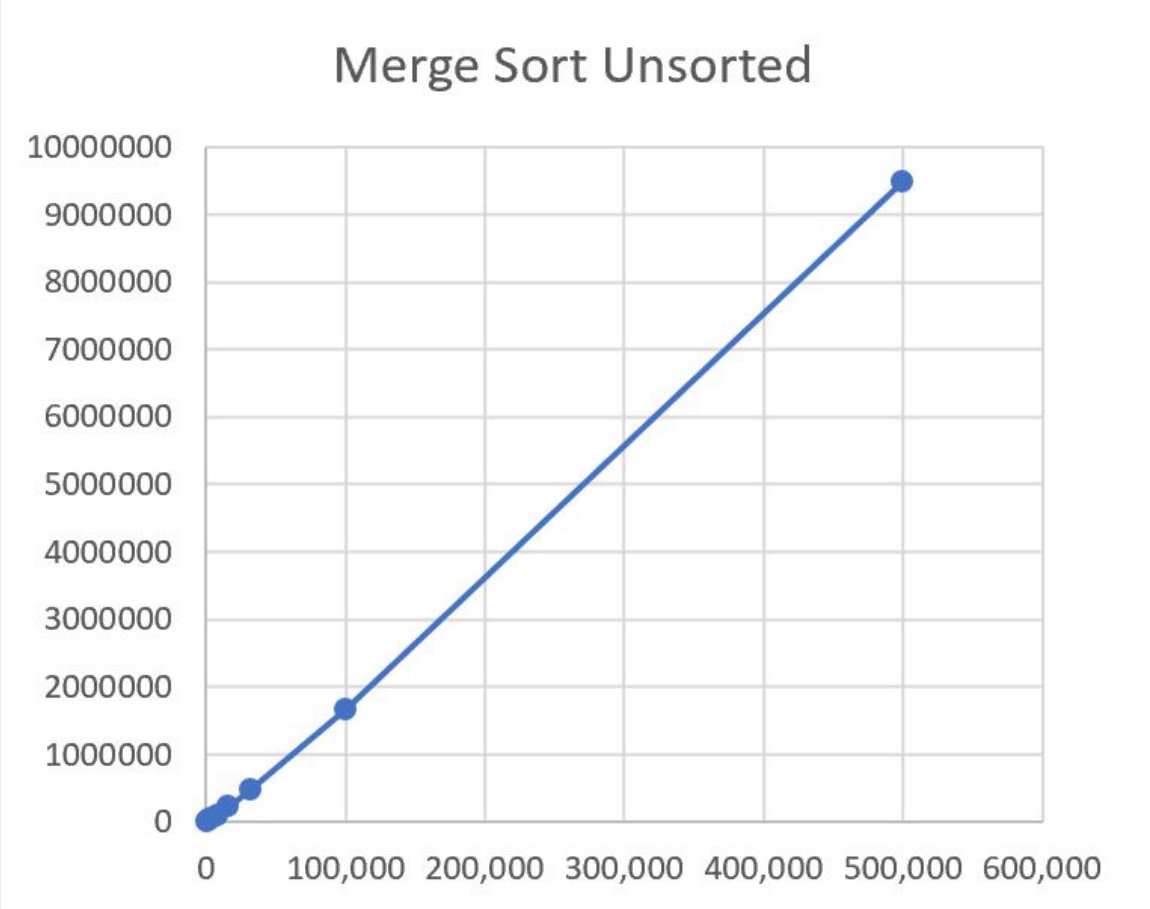
Algorithm Name: Bubble Sort 2 **Type of list (sorted or unsorted)**: Unsorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	499500	0.13463974
2,000	1999000	0.41385293
4,000	7998000	1.650394678
8,000	31996000	11.02213001
16,000	127992000	31.46189952
32,000	511984000	153.6349974
100,000	4999950000	1547.440754
500,000	1.25E+11	38567.18947



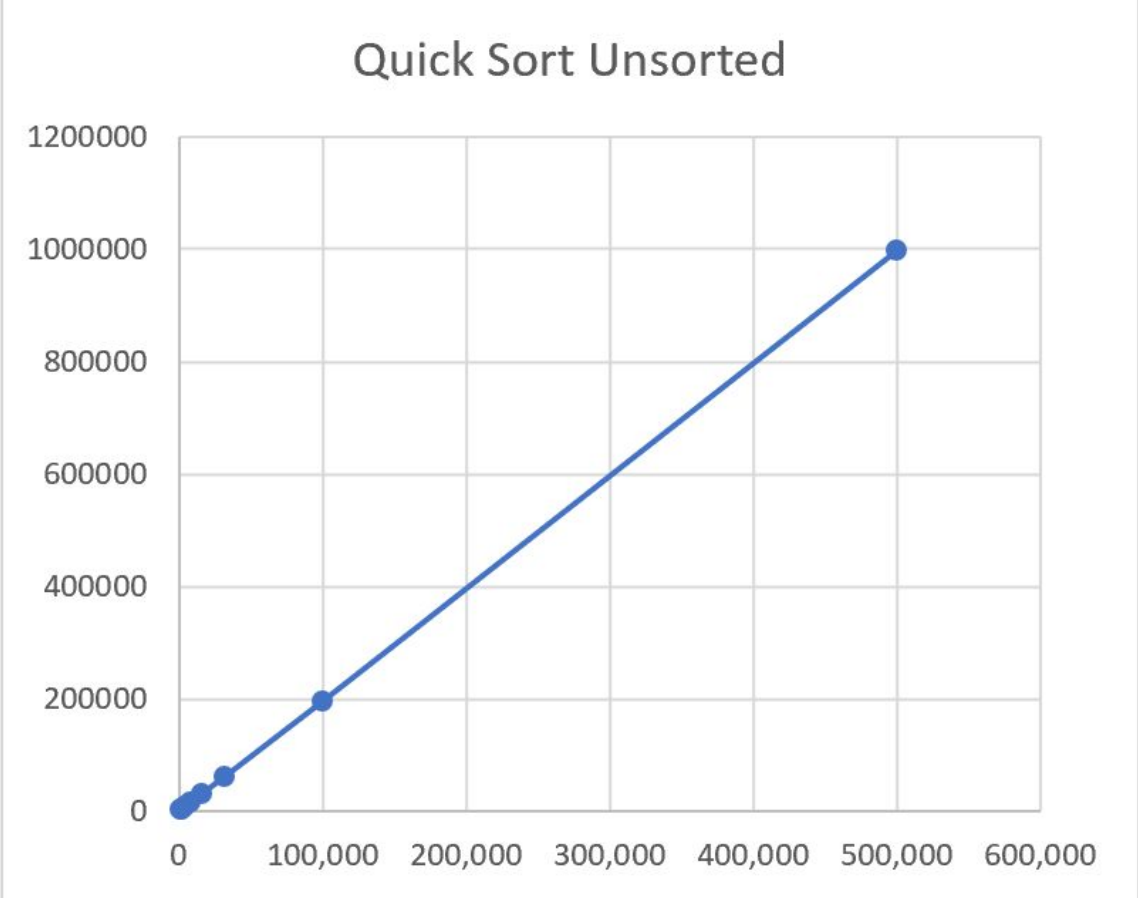
Algorithm Name: Heap Sort **Type of list (sorted or unsorted)**: Unsorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	9133	0.018948793
2,000	18378	0.028922081
4,000	36754	0.065820456
8,000	73320	0.127655029
16,000	147049	0.319850445
32,000	293736	0.627655029
100,000	918058	2.368775129
500,000	4589859	12.45573926



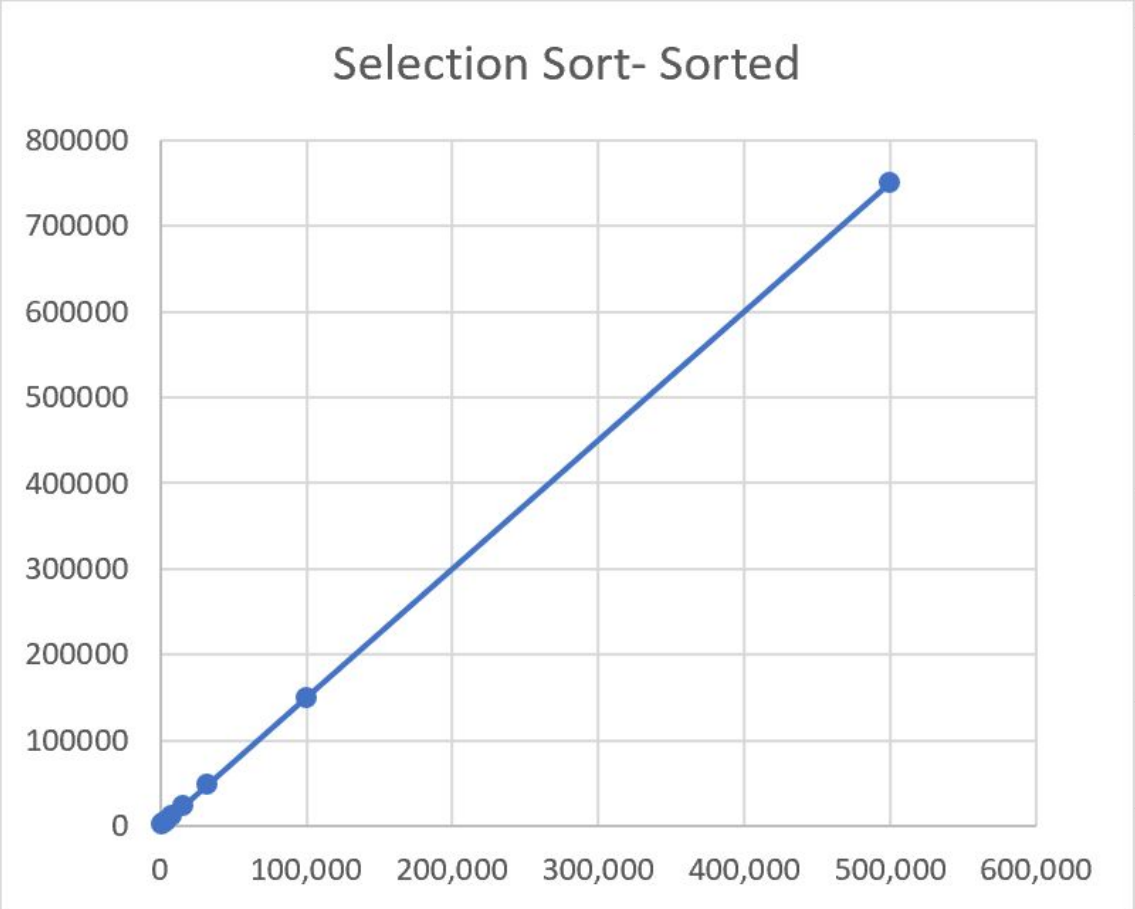
Algorithm Name: Merge Sort **Type of list (sorted or unsorted)**: Unsorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	9976	0.0039807
2,000	21952	0.007977724
4,000	47904	0.018950224
8,000	103808	0.035952091
16,000	223616	0.076794624
32,000	479232	0.169531583
100,000	1668928	0.624393992
500,000	9475712	4.560798407



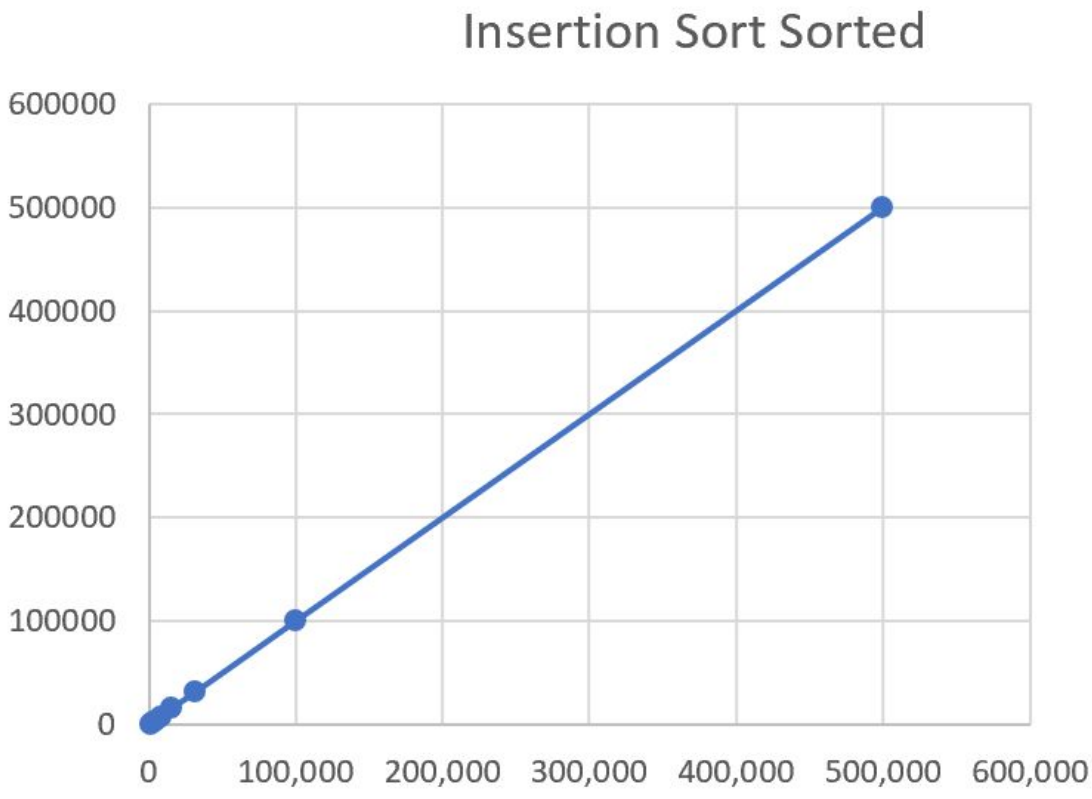
Algorithm Name: Quick Sort **Type of list (sorted or unsorted)**: Unsorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	1955	0.00498676
2,000	3921	0.006979942
4,000	7994	0.01600551
8,000	15410	0.0628314
16,000	30946	0.13364219
32,000	61431	0.151594161
100,000	196318	0.5575098
500,000	998307	3.179448307



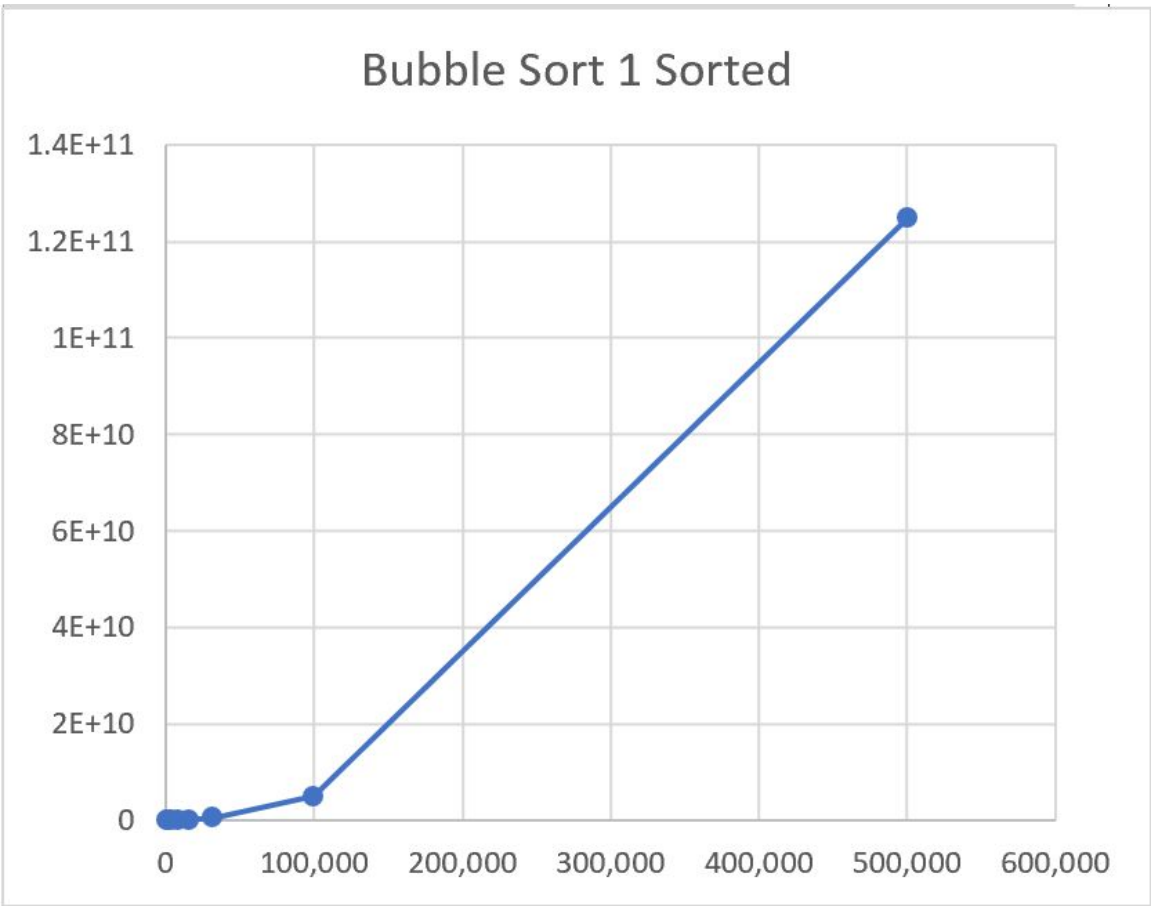
Algorithm Name: Selection Sort **Type of list (sorted or unsorted)**: Sorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	499500	0.044880629
2,000	1999000	0.261283398
4,000	7998000	1.317755699
8,000	31996000	4.2546556
16,000	127992000	21.27978373
32,000	511984000	121.7652142
100,000	4999950000	631.2235355
500,000	1.25E+11	16536.82662



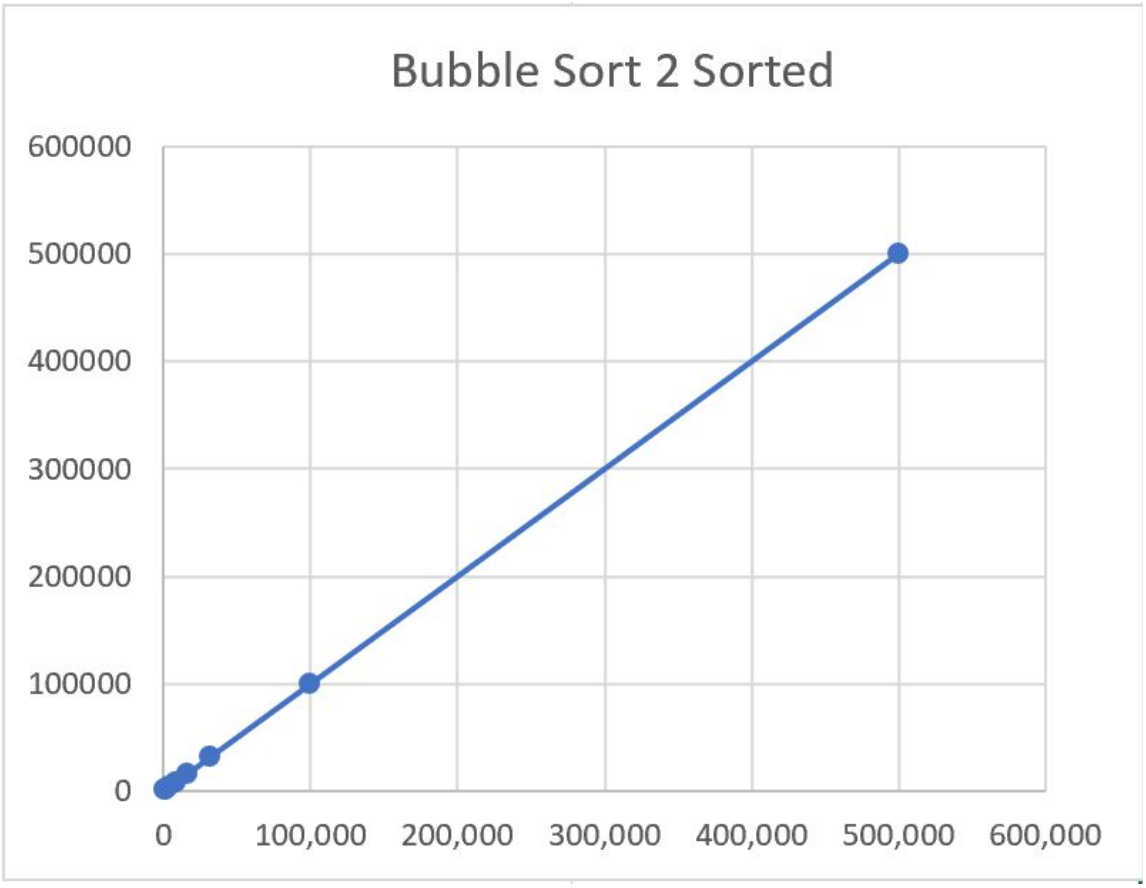
Algorithm Name: Insertion Sort **Type of list (sorted or unsorted)**: Sorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	1000	0
2,000	2000	0.000998259
4,000	4000	0
8,000	8000	0.001994848
16,000	16000	0.002992153
32,000	32000	0.008135319
100,000	100000	0.023546467
500,000	500000	0.099812327



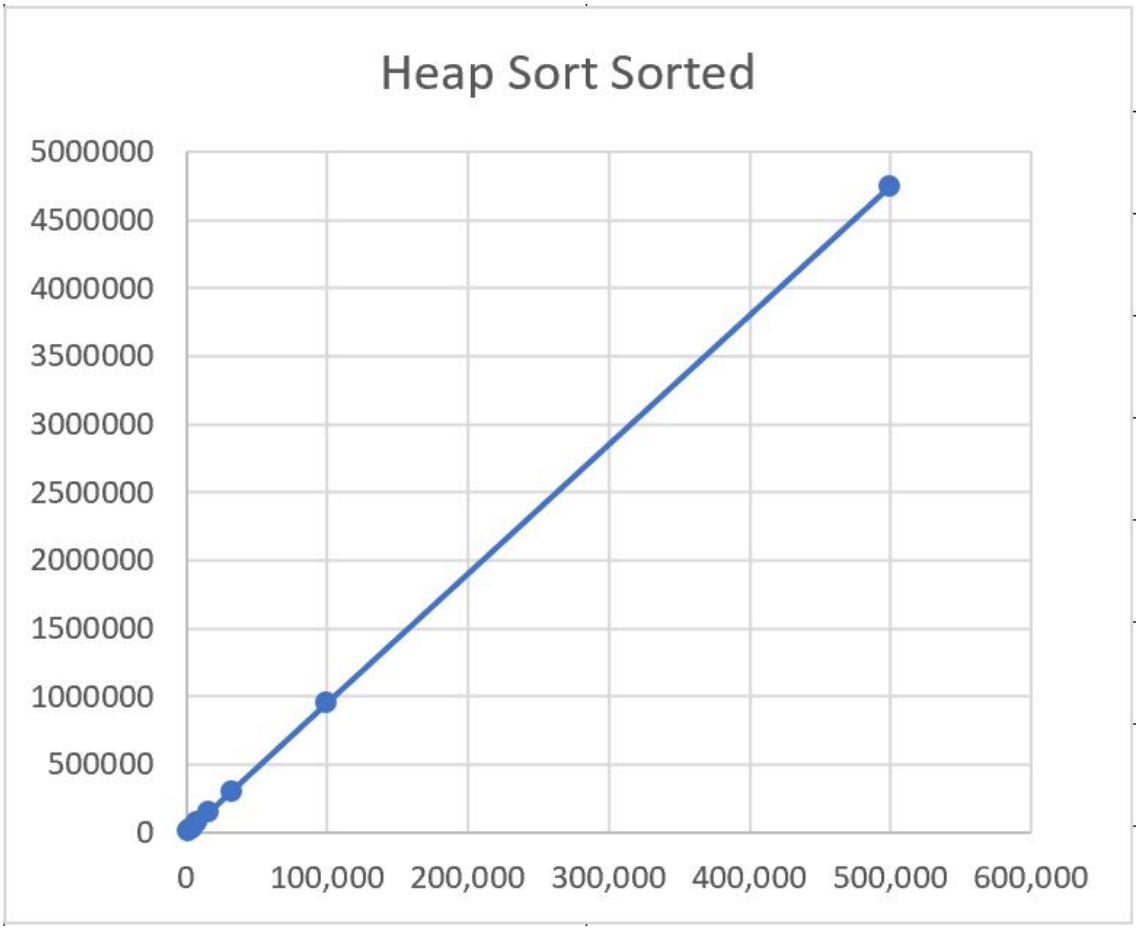
Algorithm Name: Bubble Sort **Type of list (sorted or unsorted)**: Sorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	499500	0.054882765
2,000	1999000	0.384944677
4,000	7998000	1.298999071
8,000	31996000	4.727323532
16,000	127992000	19.35224652
32,000	511984000	82.82352614
100,000	4999950000	264.896219
500,000	1.25E+11	6,605.99



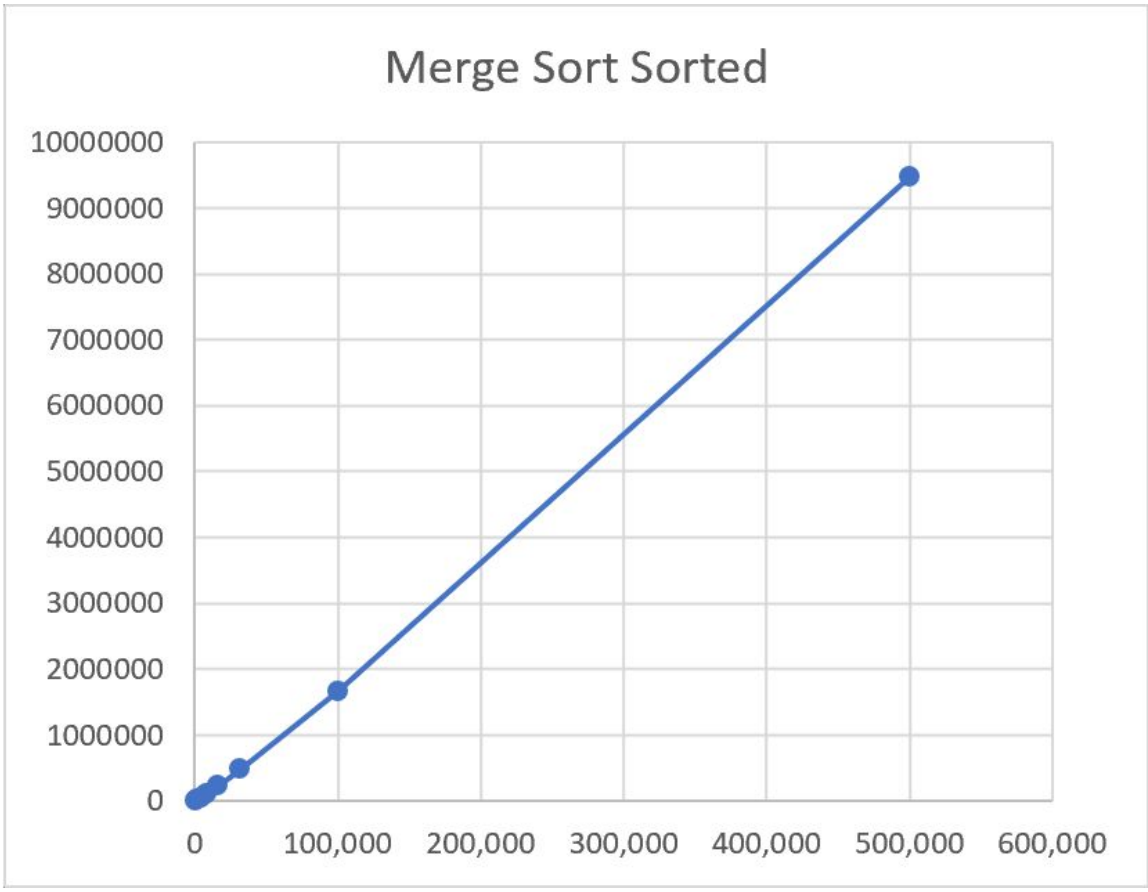
Algorithm Name: Bubble Sort 2 **Type of list (sorted or unsorted)**: Sorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	999	0
2,000	1999	0.001991749
4,000	3999	0.001005173
8,000	7999	0.000997305
16,000	15999	0.002991676
32,000	31999	0.005835533
100,000	99999	0.018264299
500,000	499999	0.090095116



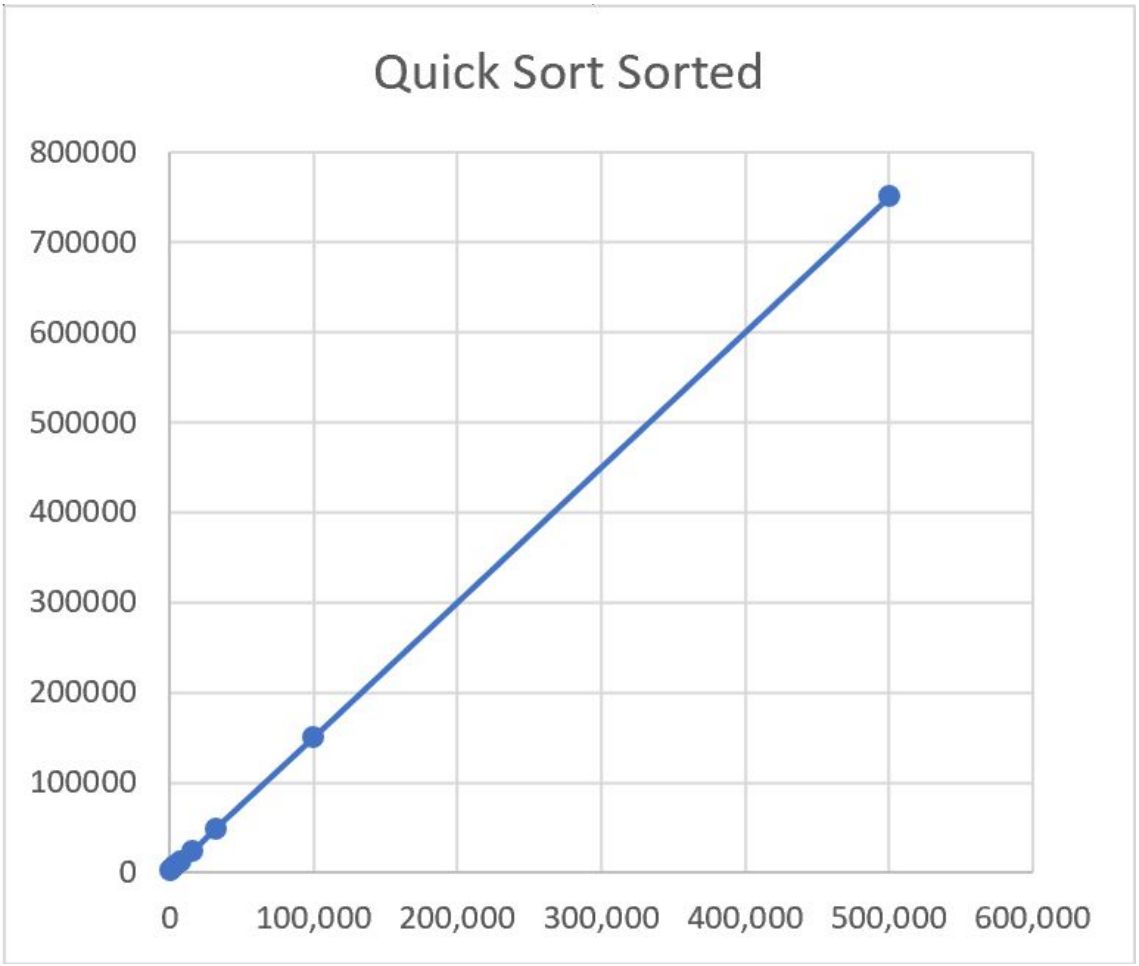
Algorithm Name: Heap Sort **Type of list (sorted or unsorted)**: Sorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	9487	0.010970116
2,000	18987	0.044883966
4,000	37987	0.049902916
8,000	75990	0.118682384
16,000	151987	0.252325535
32,000	303987	0.51361537
100,000	949987	1.66358279
500,000	4,749,987	8.533004693



Algorithm Name: Merge Sort **Type of list (sorted or unsorted)**: Sorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	9976	0.003044605
2,000	21952	0.005984545
4,000	47904	0.013013601
8,000	103808	0.033909559
16,000	223616	0.064824819
32,000	479232	0.142617702
100,000	1668928	0.472735404
500,000	9475712	2.891264677



Algorithm Name: Quick Sort **Type of list (sorted or unsorted)**: Sorted		
List Size	**Comparisons**	**Time (seconds)**
1,000	1500	0.001995325
2,000	3000	0.00398922
4,000	6000	0.008959055
8,000	12000	0.017952204
16,000	24000	0.040890217
32,000	48000	0.085768938
100,000	150000	0.285238743
500,000	750000	1.740348816



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

Meha and Sullivan came to the conclusion that Merge Sort is probably the best of out of all of these algorithms. It is the most consistent because it's stable, fast O(nlog(n)), and easy to implement. It also has the ability to be implemented in constant space, however in naive implementation it is O(n) space.

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

Insertion sort is the best when sorting a list that is already sorted because it only swaps IF the current unsorted is less than the last sorted, this means that the best case time complexity is O(n). It's also stable so it is better than bubble sort 2.

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 times for insertion sort isn't half what the time is for selection sort even though insertion has half the amount of comparisons is because swapping things takes a lot more time than searching for the largest item and then swapping once.