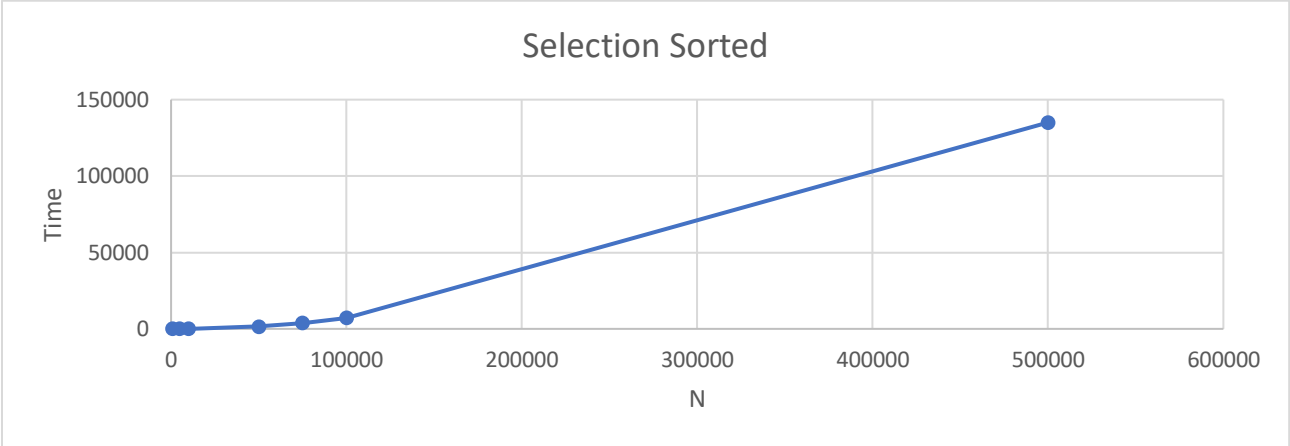
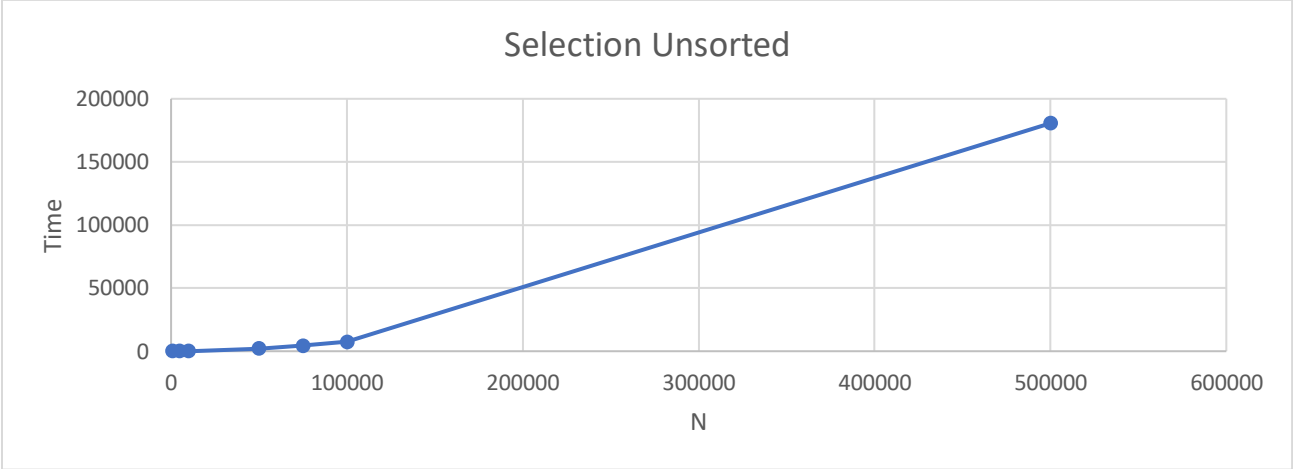
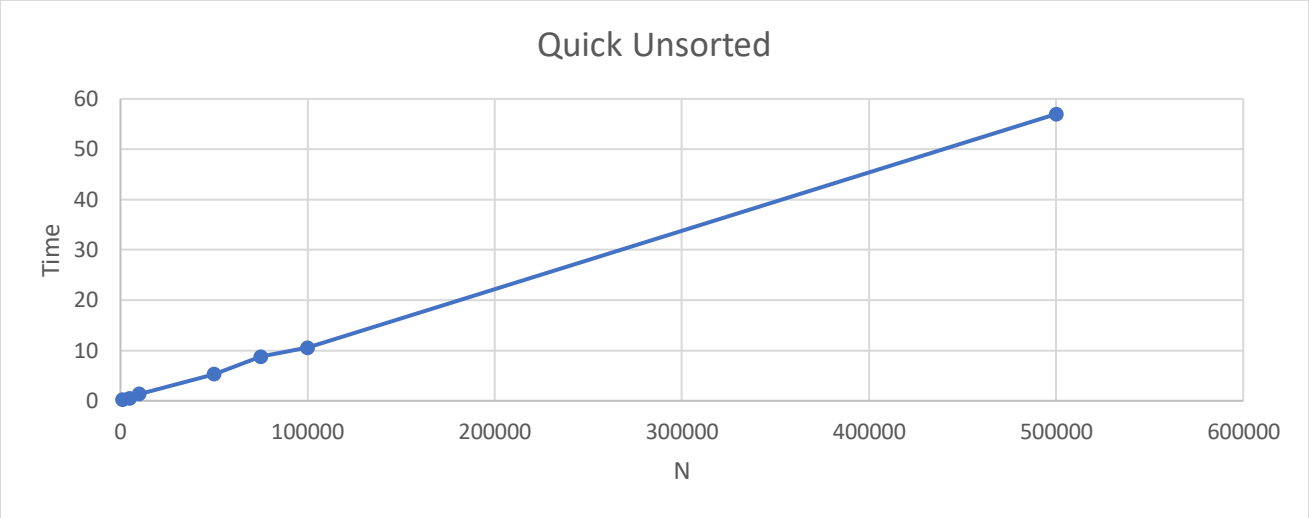
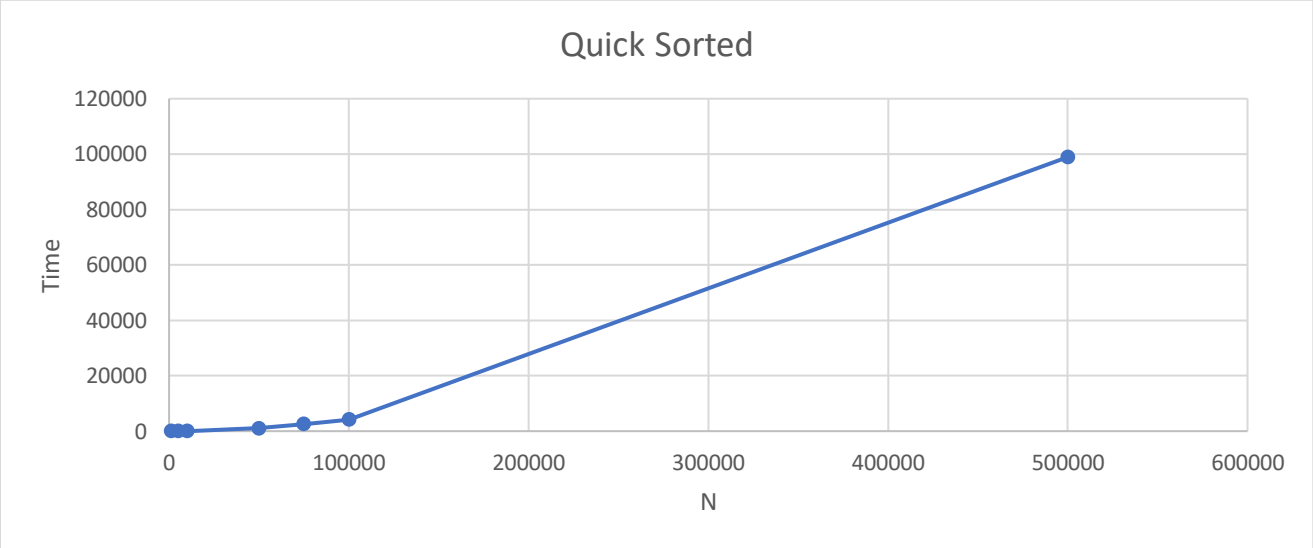




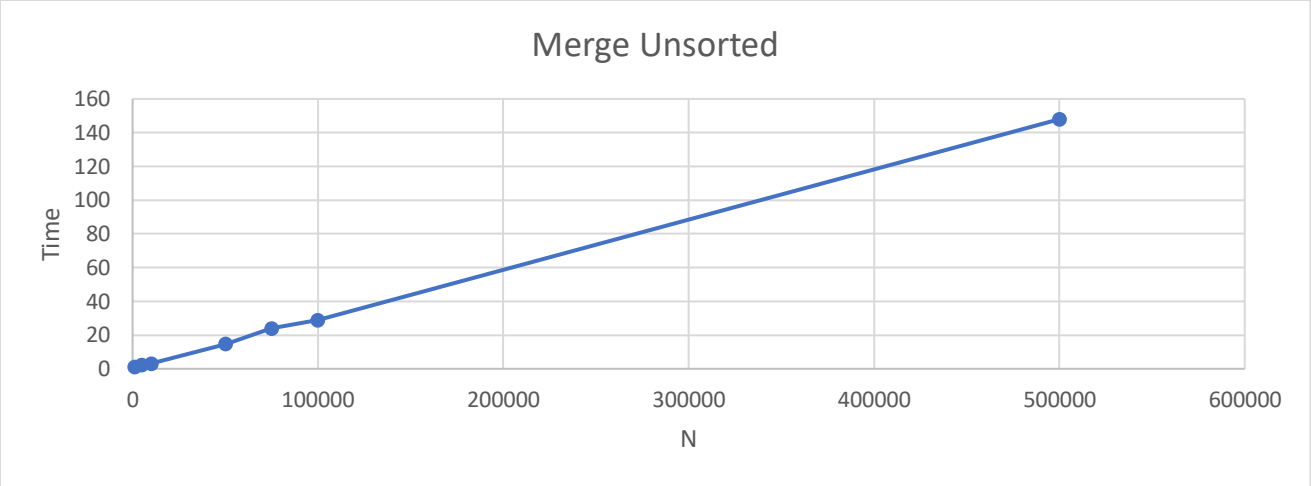
N (Unsorted)	Time (ms)
1000	2.3516
5000	22.9639
10000	88.4826
50000	2101.61
75000	4408.1
100000	7515.69
500000	180630

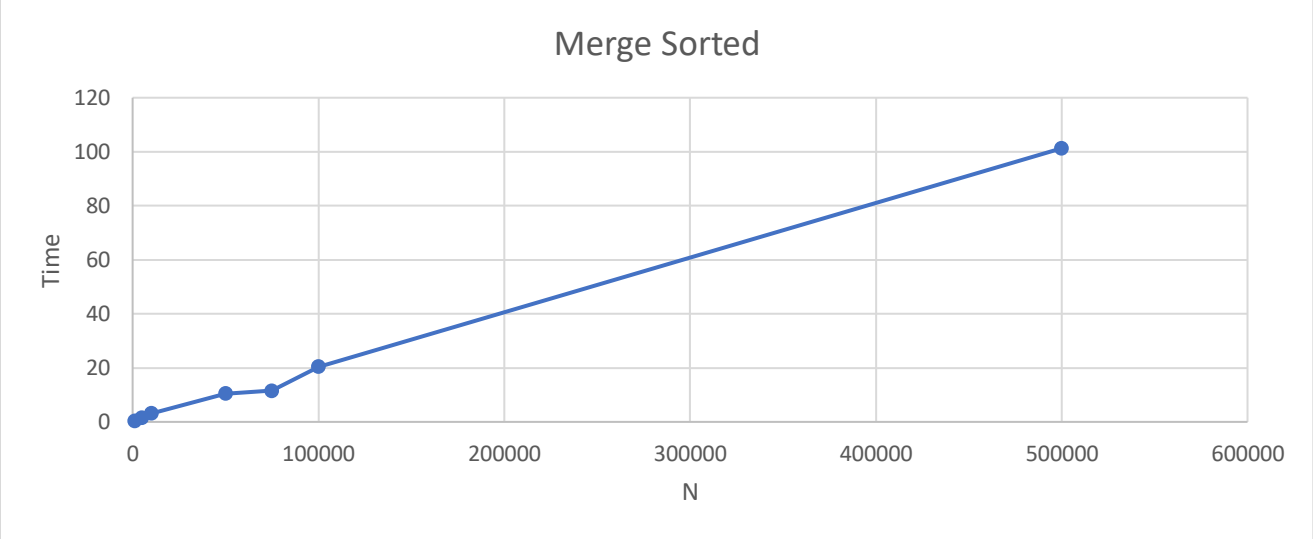
N (Sorted)	Time (ms)
1000	0.4491
5000	11.9513
10000	48.1069
50000	1624.62
75000	3744.49
100000	7285.14
500000	135030



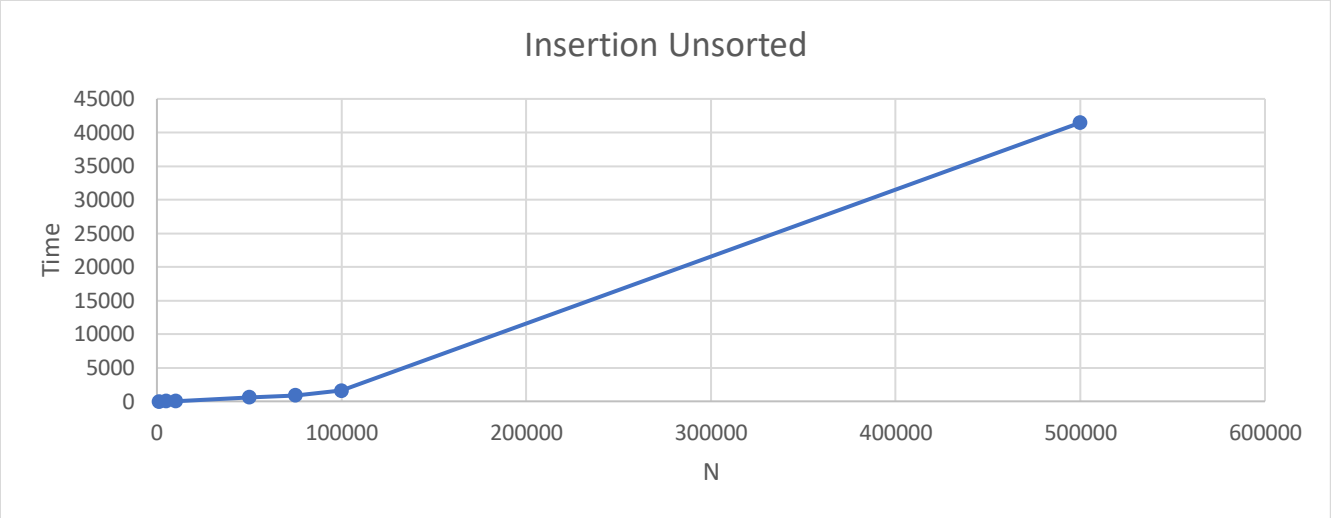


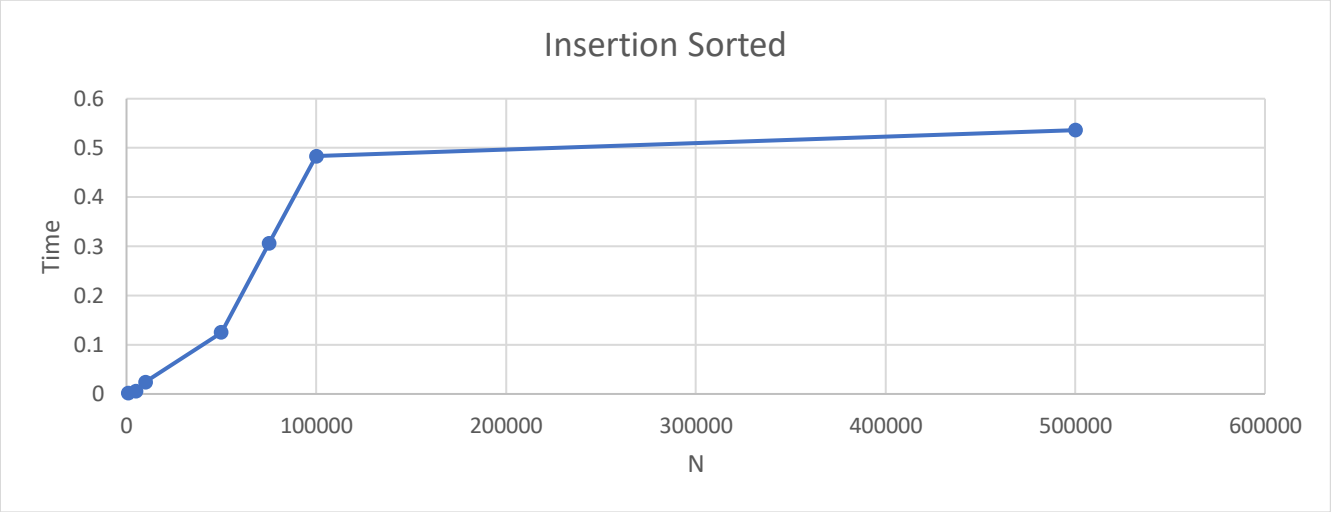
N (Unsorted)	Time (ms)
1000	0.2243
5000	0.5306
10000	1.3026
50000	5.2972
75000	8.7604
100000	10.5872
500000	56.9888
N (Sorted)	**Time (ms)**
1000	0.453
5000	13.8951
10000	44.5006
50000	1065.47
75000	2539.95
100000	4129.73
500000	98941.2



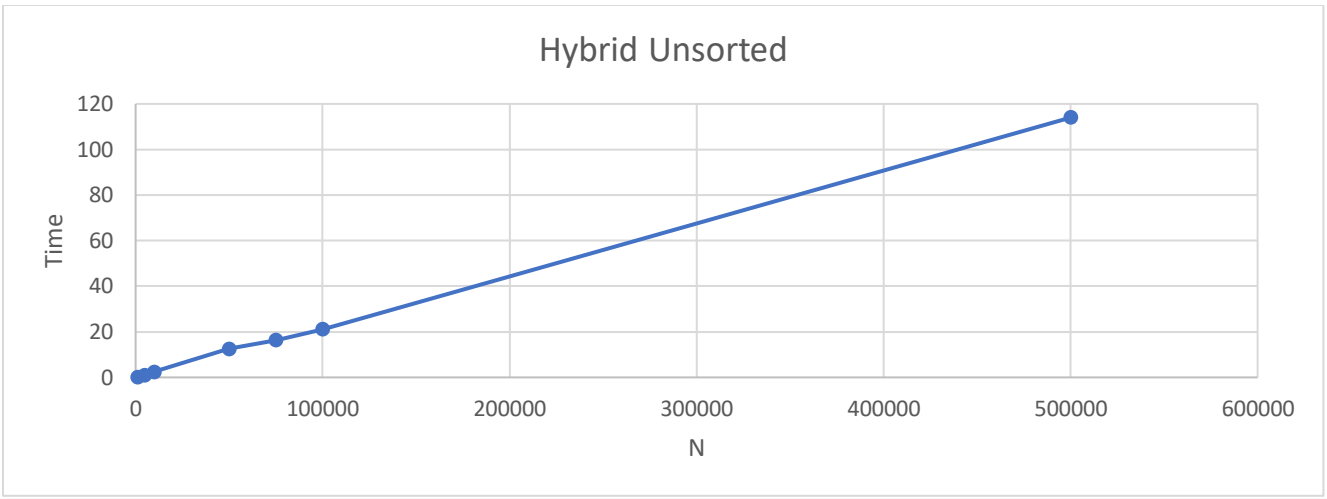


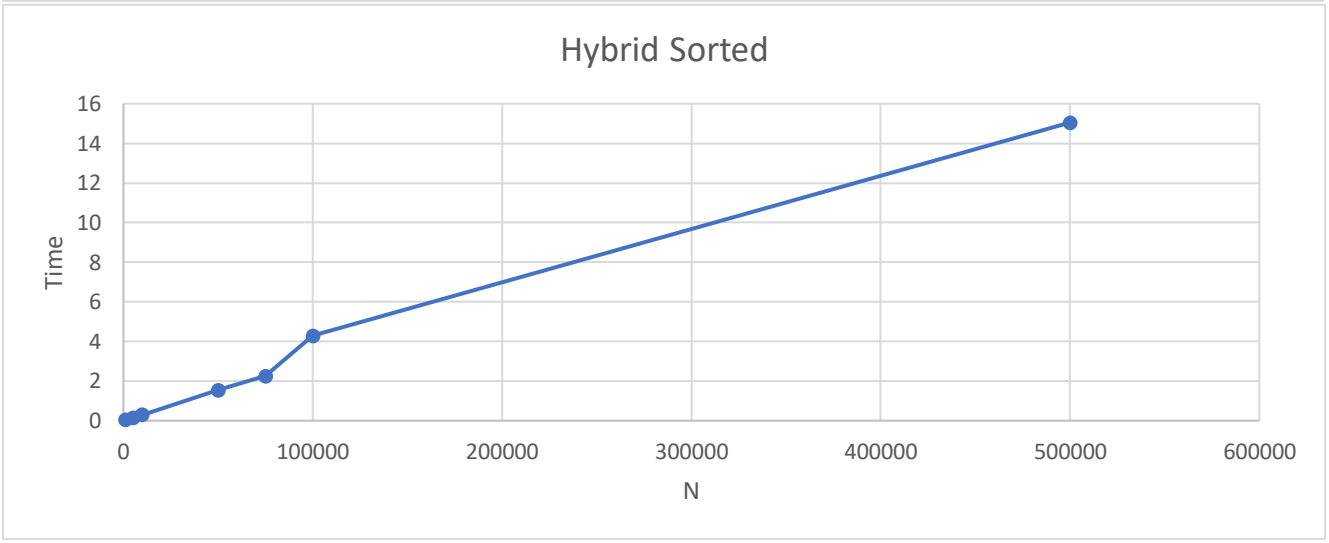
N (Unsorted)	Time (ms)
1000	0.9656
5000	2.2362
10000	2.9522
50000	14.5888
75000	23.8721
100000	28.8321
500000	147.952
N (Sorted)	**Time (ms)**
1000	0.3768
5000	1.576
10000	3.1514
50000	10.4602
75000	11.5275
100000	20.4298
500000	101.288





N (Unsorted)	Time (ms)
1000	0.6639
5000	11.6584
10000	17.1479
50000	578.064
75000	910.521
100000	1595.04
500000	41462.7
N (Sorted)	**Time (ms)**
1000	0.0019
5000	0.0056
10000	0.0242
50000	0.1246
75000	0.3055
100000	0.4835
500000	0.5361





N (Unsorted)	Time (ms)
1000	0.2152
5000	1.0547
10000	2.3568
50000	12.5356
75000	16.3878
100000	21.081
500000	113.998
N (Sorted)	**Time (ms)**
1000	0.0334
5000	0.1538
10000	0.3034
50000	1.5404
75000	2.2463
100000	4.2795
500000	15.0504

Hybrid Sort Motive

After running the performance test on all the required sorting algorithms, it was clear that most of the sorting algorithms acted poorly on an already sorted list, except for insertion sort.

So, I decided to combine the efficiency of insertion sort with sorted list with the fastest available sorting method for unsorted lists.

By first checking if the list is some how in a near sorted state or completely sorted then deciding which algorithm to use with favor to quick sort since it will be faster than insertion in its worst case.