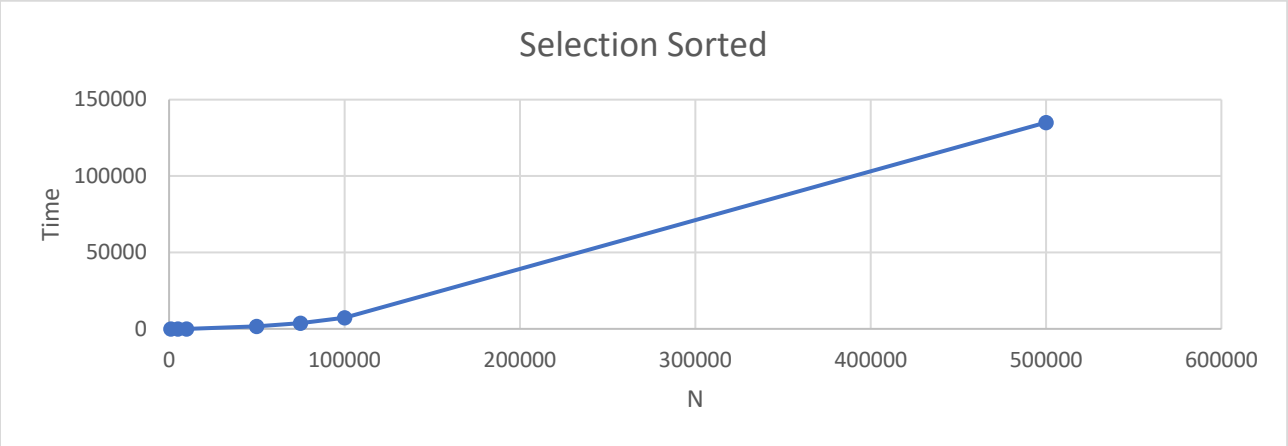
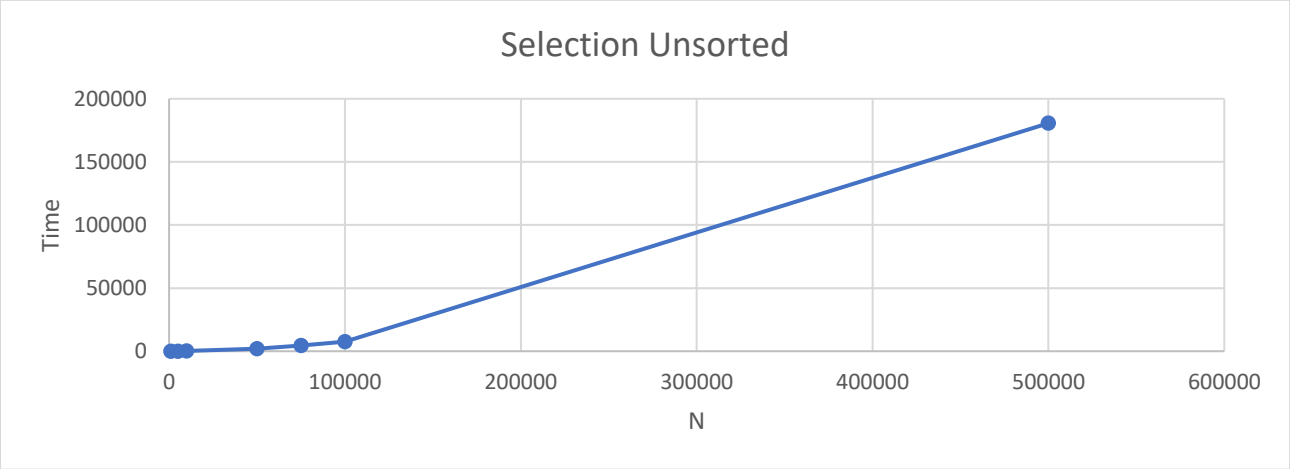
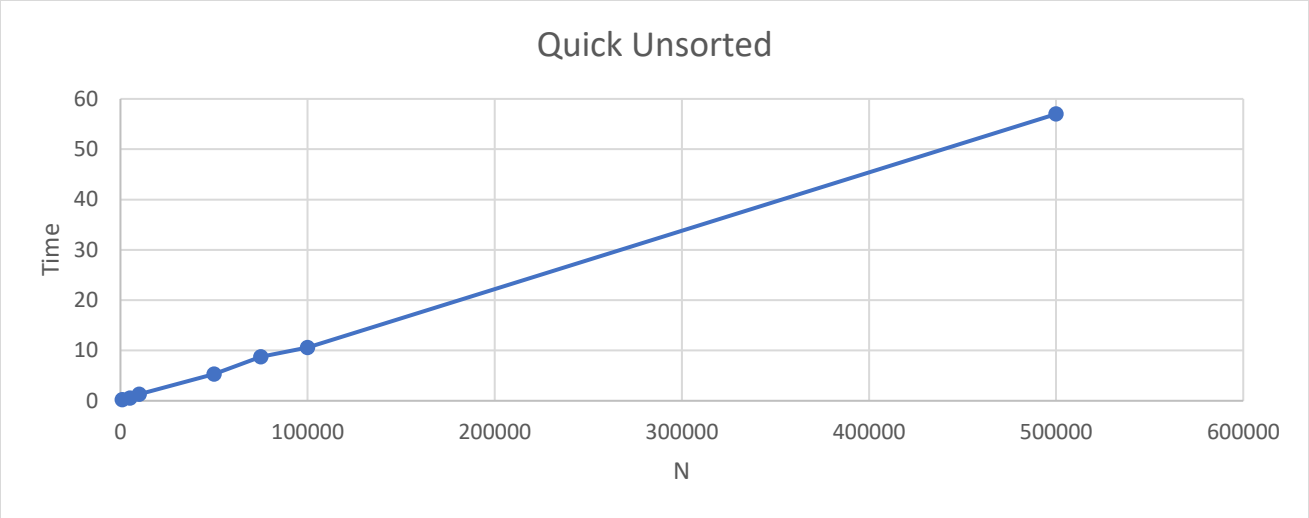
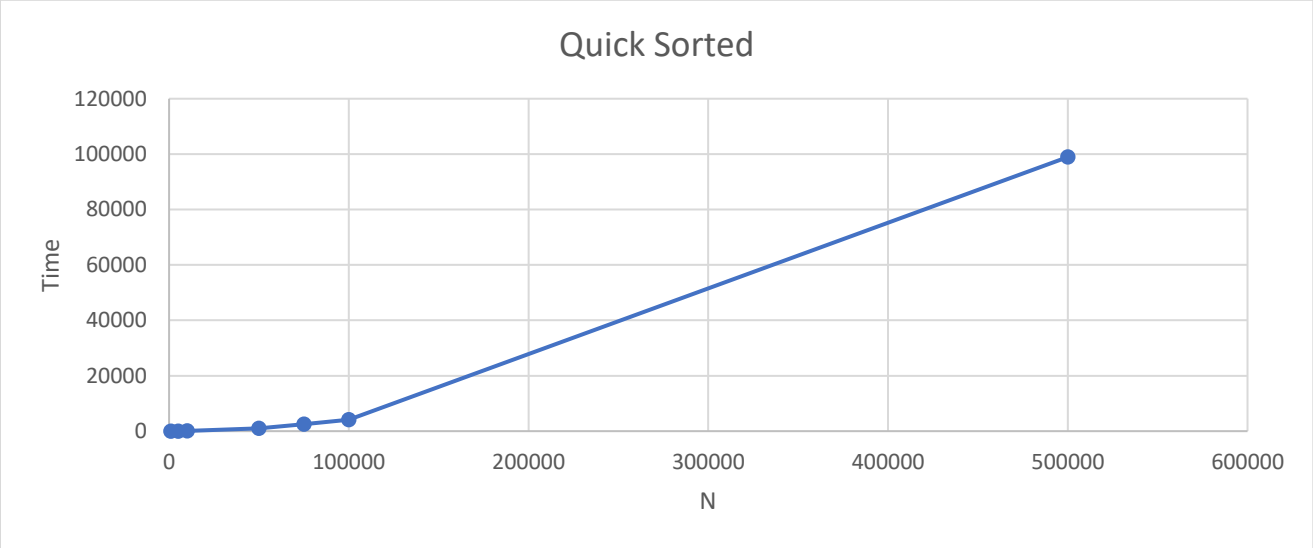


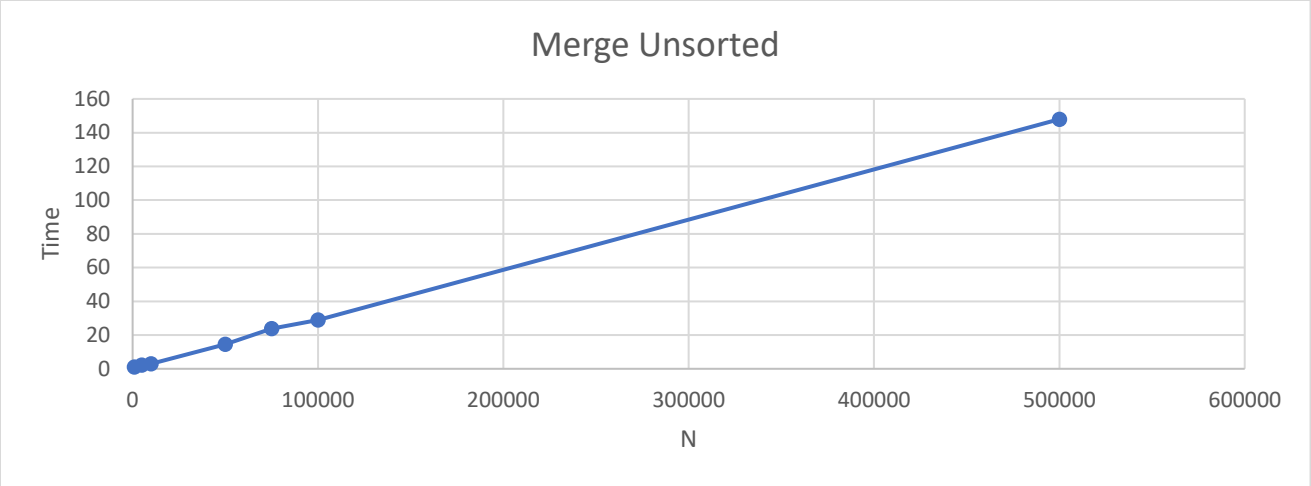


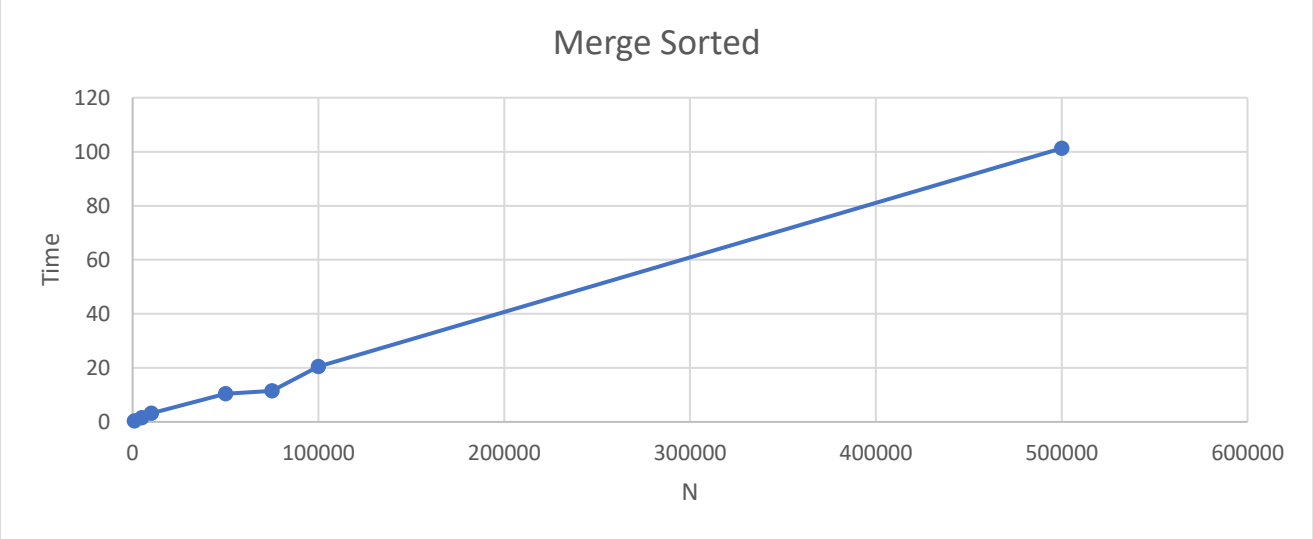
N (Unsorted)	Time (ms)
1000	2.3516
5000	22.9639
10000	88.4826
50000	2101.61
75000	4408.1
100000	7515.69
500000	180630
N (Sorted)	**Time (ms)**
1000	0.4491
5000	11.9513
10000	48.1069
50000	1624.62
75000	3744.49
100000	7285.14
500000	135030



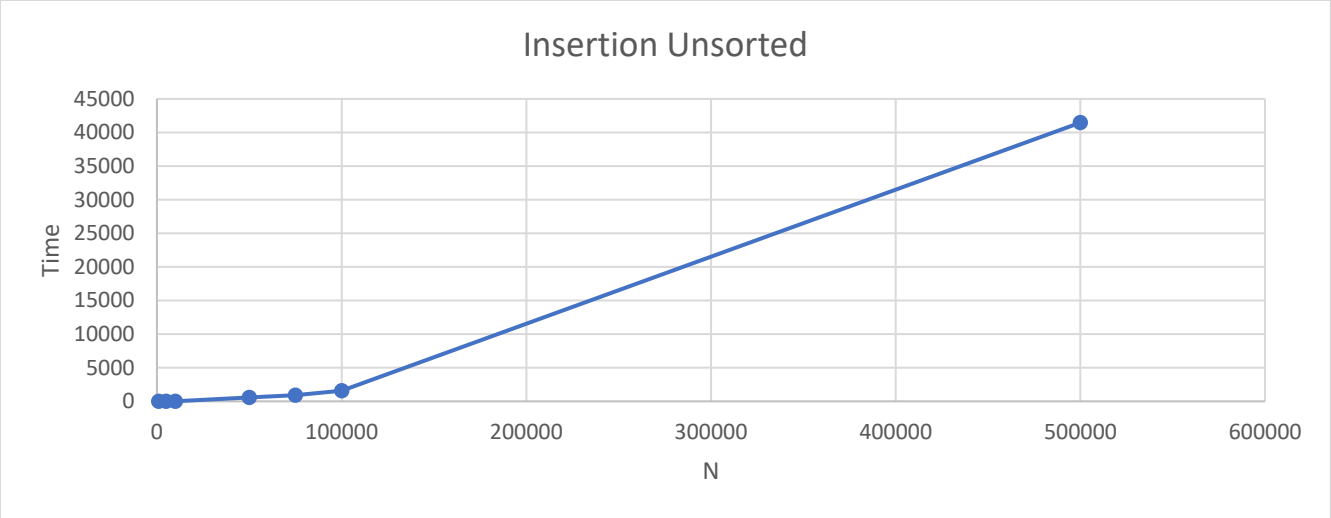


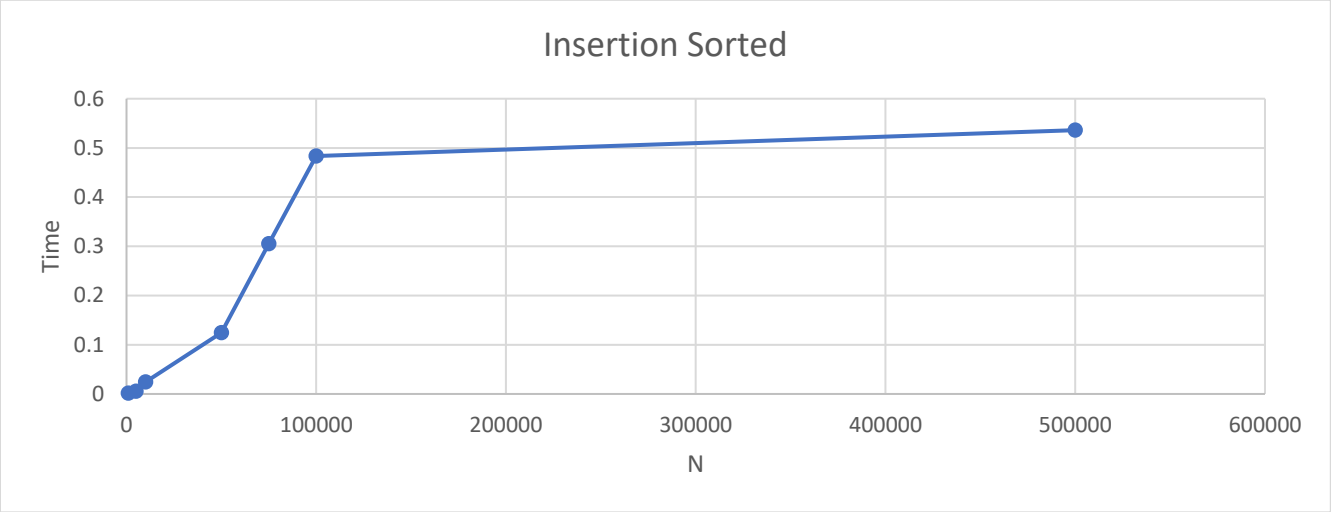
N (Unsorted)	Time (ms)
1000	0.2243
5000	0.5306
10000	1.3026
50000	5.2972
75000	8.7604
100000	10.5872
500000	56.9888
N (Sorted)	**Time (ms)**
1000	0.453
5000	13.8951
10000	44.5006
50000	1065.47
75000	2539.95
100000	4129.73
500000	98941.2



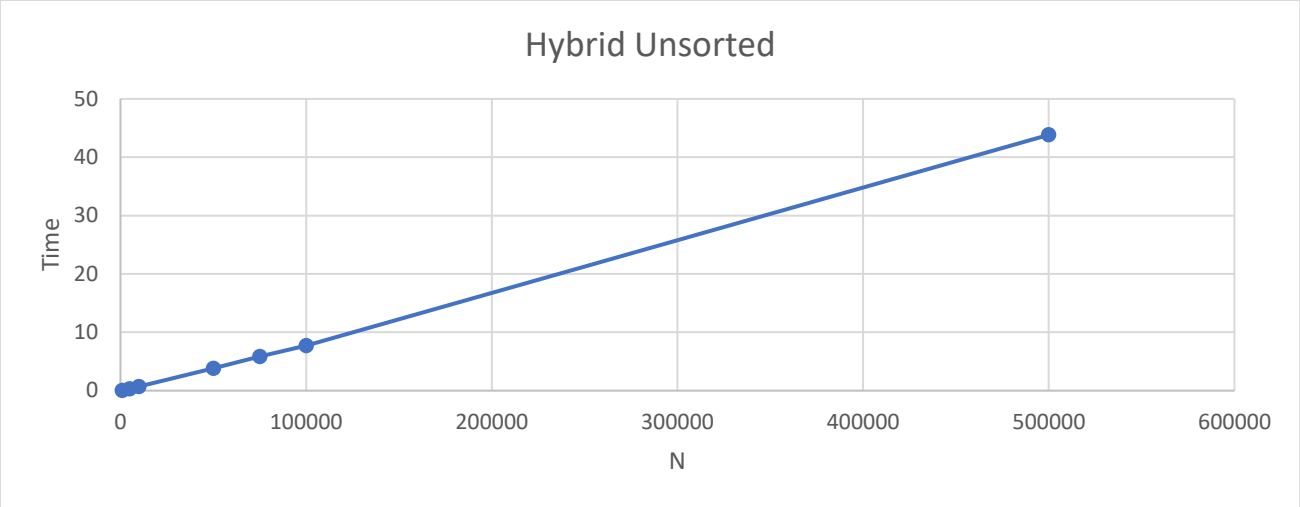


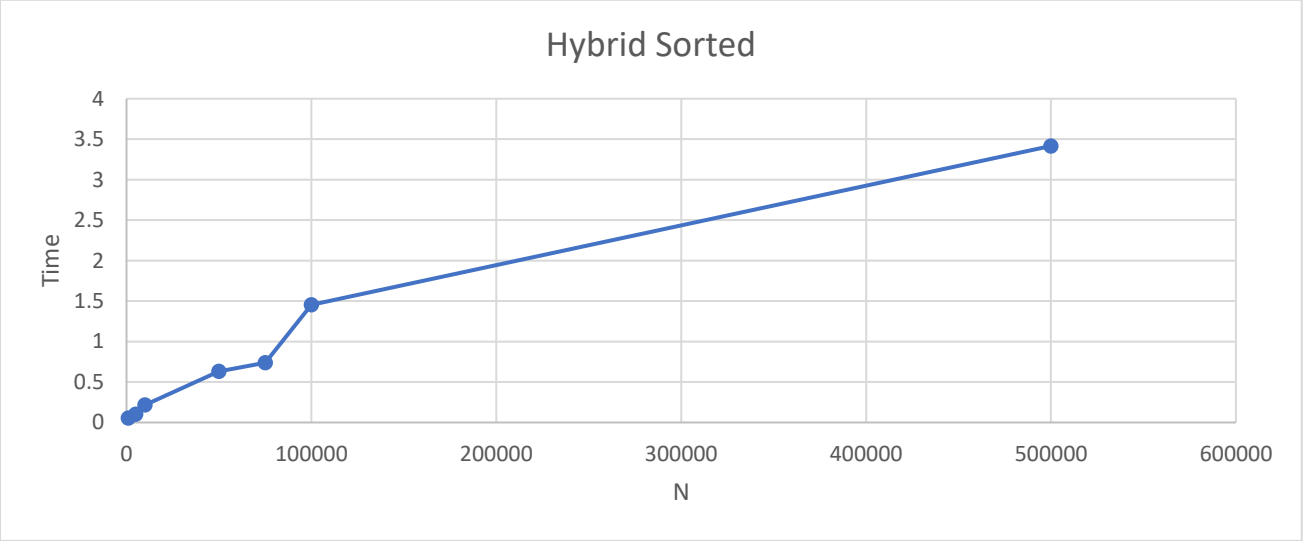
N (Unsorted)	Time (ms)
1000	0.9656
5000	2.2362
10000	2.9522
50000	14.5888
75000	23.8721
100000	28.8321
500000	147.952
N (Sorted)	**Time (ms)**
1000	0.3768
5000	1.576
10000	3.1514
50000	10.4602
75000	11.5275
100000	20.4298
500000	101.288



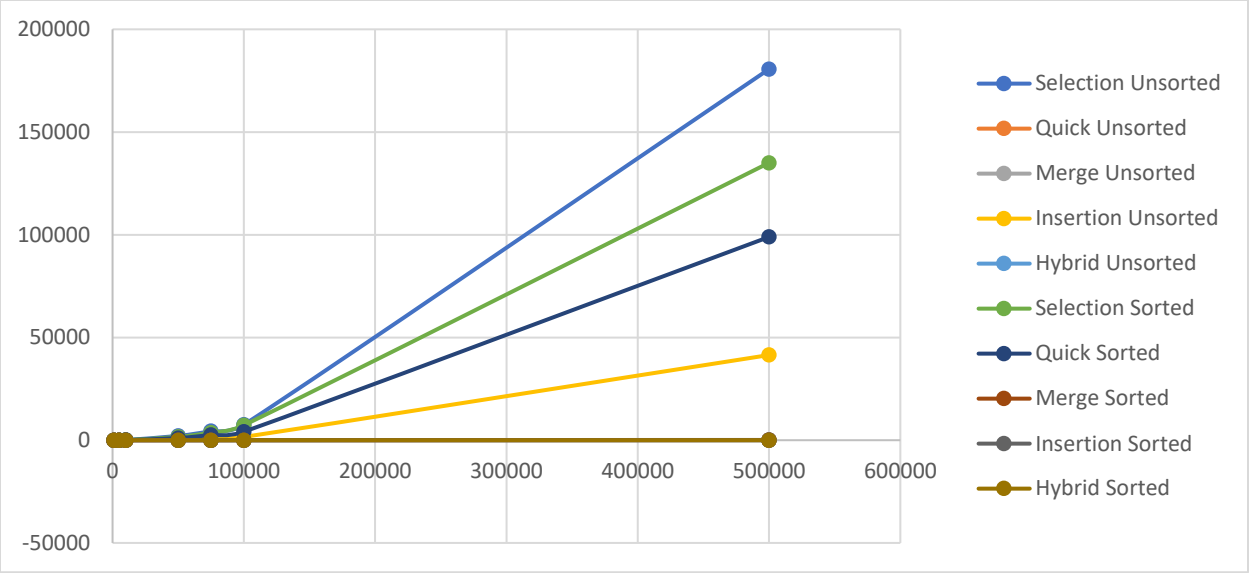


N (Unsorted)	Time (ms)
1000	0.6639
5000	11.6584
10000	17.1479
50000	578.064
75000	910.521
100000	1595.04
500000	41462.7
N (Sorted)	**Time (ms)**
1000	0.0019
5000	0.0056
10000	0.0242
50000	0.1246
75000	0.3055
100000	0.4835
500000	0.5361





N (Unsorted)	Time (ms)
1000	0.0643
5000	0.3203
10000	0.6772
50000	3.8289
75000	5.8581
100000	7.7218
500000	43.8386
N (Sorted)	**Time (ms)**
1000	0.053
5000	0.1
10000	0.2143
50000	0.6303
75000	0.7387
100000	1.452
500000	3.4165



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 already sorted lists with the fastest available sorting method for unsorted lists.

By implementing insertion sort on the list at first and if the lists after some sorting seems to be unsorted then switching to quick sort for the rest of the unsorted list.

Prepared by Kamel Mohsen

ID: 1162325