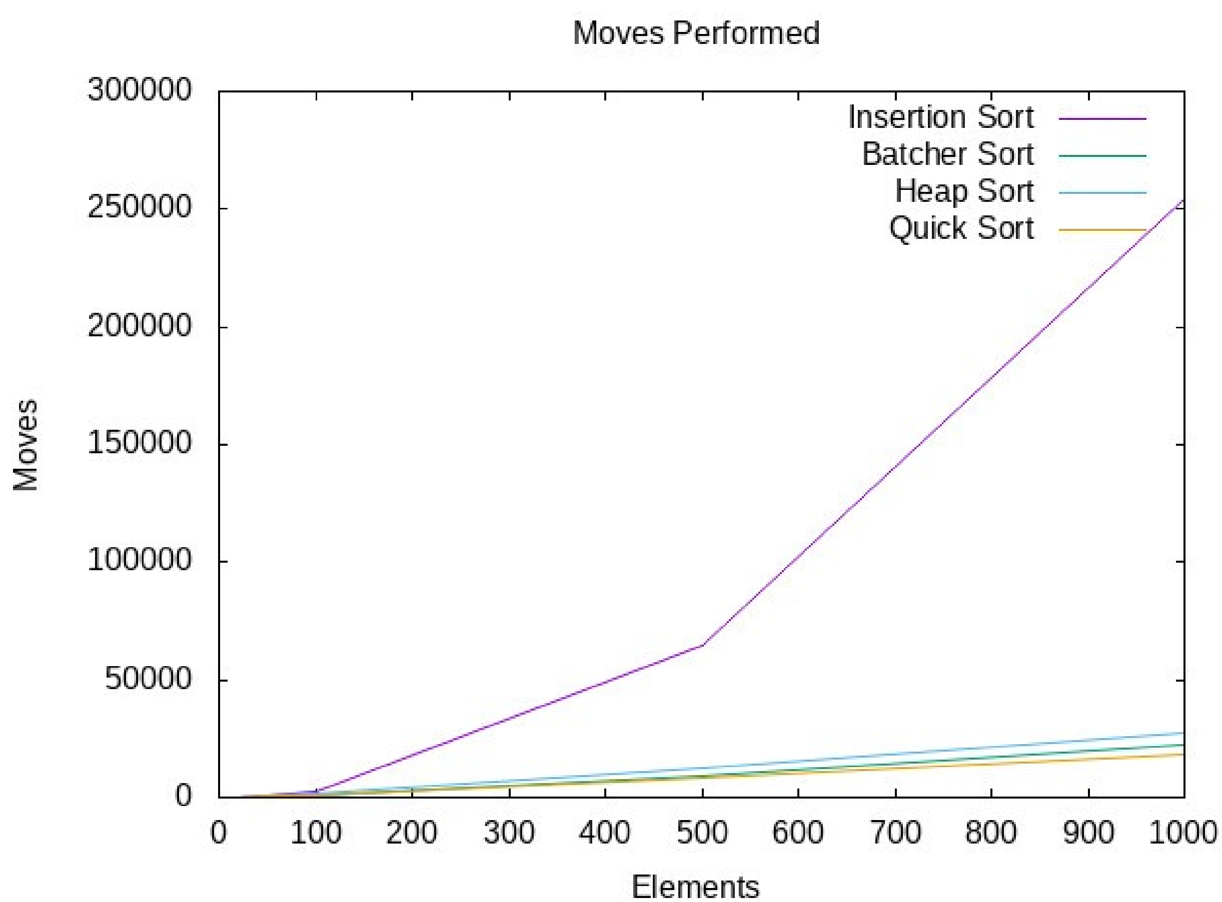
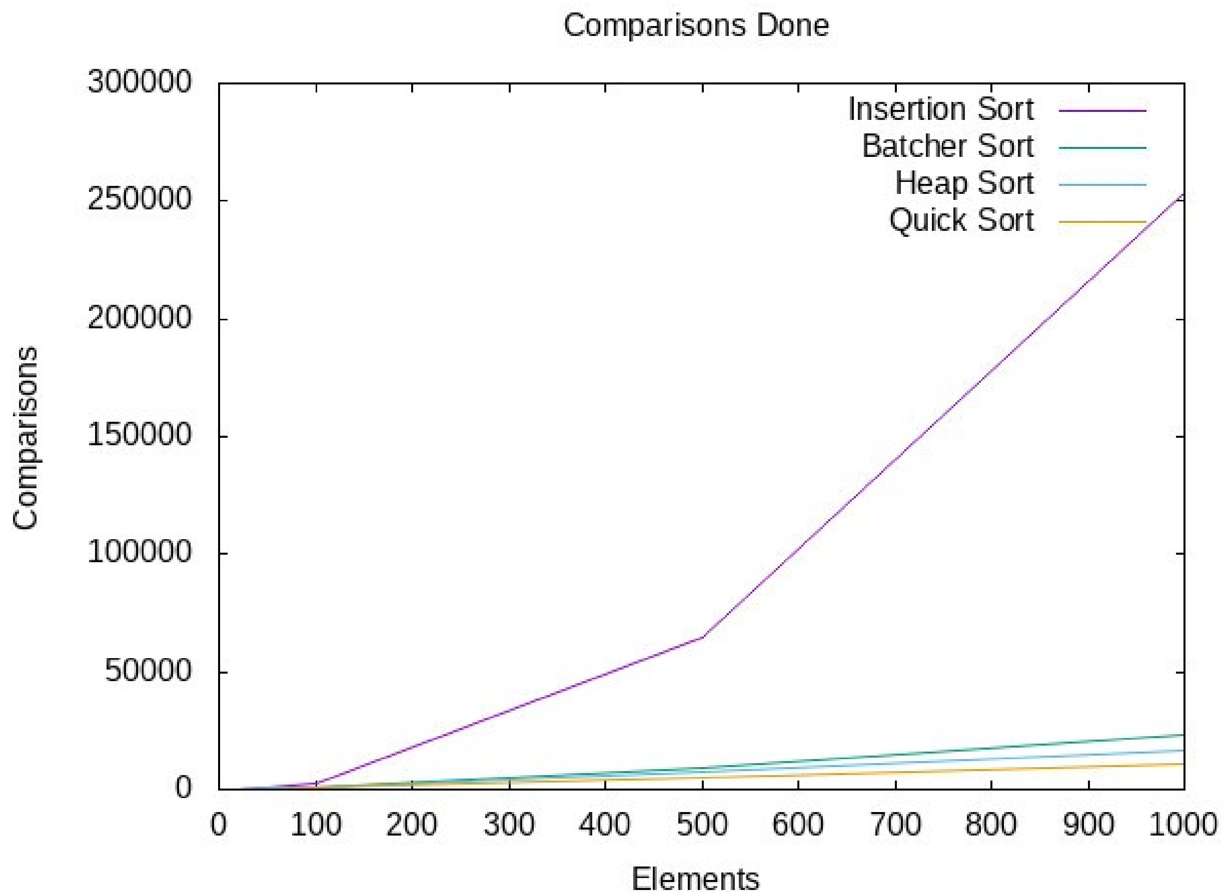
Writeup

From the graphs (and the data), it can be seen that quicksort is the most efficient sort, both in terms of number of comparisons, and the number of moves. This is followed by heap, batcher, and insertion, in that order.

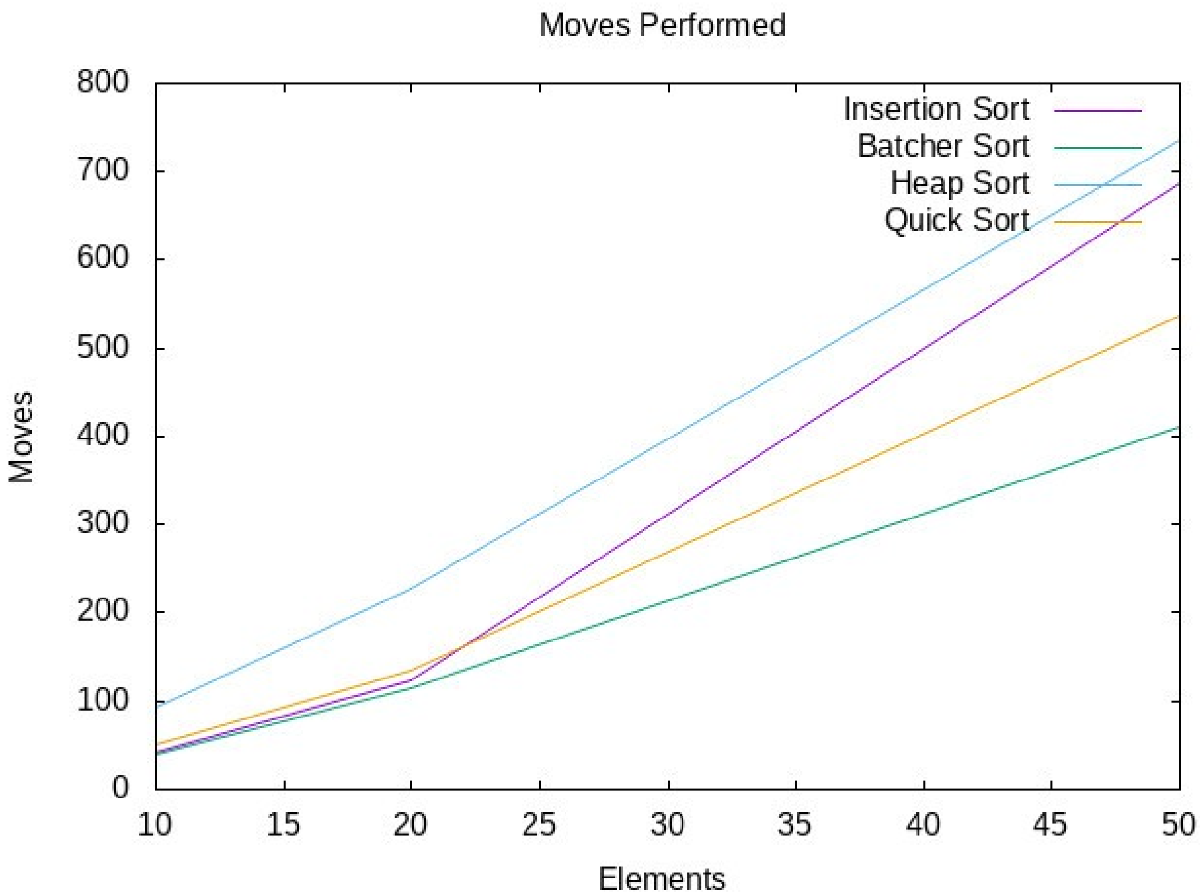
However, the number of moves taken by insertion sort are an order of magnitude (and more for higher n) higher than other sorts. So, the difference in the performance of other algorithms is masked by the outlier values of insertion sort. In the below graph for example, I took the highest n = 1000, and here too, it is difficult to differentiate between the performance of other algorithms.



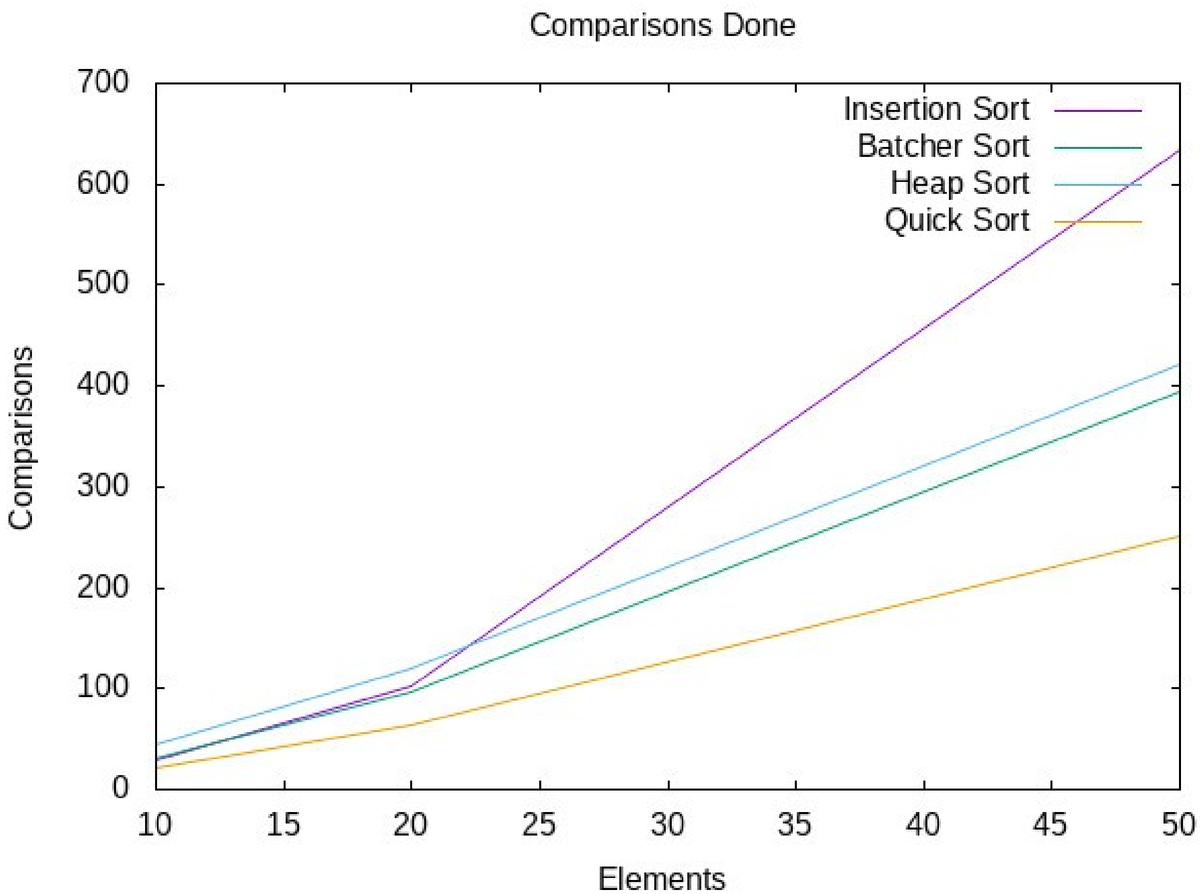


Higher n values skew the data even more.

Interestingly, insertion sort is quite efficient when n is small. For example, the below graphs plot the number of comparisons and moves for some n values <= 50.



Moves Performed
Moves
Elements
Insertion Sort
Batcher Sort
Heap Sort
Quick Sort
0
100
200
300
400
500
600
700
800
10
15
20
25
30
35
40
45
50



So, for about n = 20, insertion sort is almost as good as any other algorithm (and in some cases, better). Given its simplicity, it can be used for small values of n.

Since the graphs are skewed due to the outlier values of insertion sort, I have included the data for some higher values of n in tables below.

Number of Moves

n	Insertion Sort	Batcher Sort	Heap Sort	Quick Sort
10	41	39	93	51
20	124	114	228	135
50	687	411	735	537

100	2741	1209	1755	1053
500	64858	9072	12132	8280
1000	254769	22497	27225	18642
10000	24901706	371694	372558	256734

Number of Compares

n	Insertion Sort	Batcher Sort	Heap Sort	Quick Sort
10	29	31	44	22
20	102	97	120	63
50	635	395	421	252
100	2638	1077	1029	640
500	64354	9505	7422	4717
1000	253765	23499	16818	10531
10000	24891699	425695	235318	149913