Selection Sort

List Size	Comparisons	Time (seconds)
1,000 (observed)	499500	0.030
2,000 (observed)	1999000	0.130
4,000 (observed)	7998000	0.496
8,000 (observed)	31996000	2.021
16,000 (observed)	127992000	8.378
32,000 (observed)	511984000	32.039
100,000 (estimated)	5004254268	337.77
500,000 (estimated)	125166776600	8585.40
1,000,000 (estimated)	500771227800	34587.67
10,000,000 (estimated)	50111726780000	3541777.3

$\frac{16}{32} =$

$\frac{4}{x} = \frac{32}{100}$

Insertion Sort

List Size	Comparisons	Time (seconds)
1,000 (observed)	249607	0.030
2,000 (observed)	992890	0.110
4,000 (observed)	3999429	0.460
8,000 (observed)	16115116	1.888
16,000 (observed)	64321667	7.528
32,000 (observed)	257455590	30.297
100,000 (estimated)	2527214525	323.31
500,000 (estimated)	63517000000	8167.79
1,000,000 (estimated)	254650000000	32818.69
10,000,000 (estimated)	25659000000000	3331357.61

Merge Sort

List Size	Comparisons	Time (seconds)
1,000 (observed)	8703	0.000
2,000 (observed)	19409	0.010
4,000 (observed)	42774	0.010
8,000 (observed)	93591	0.020
16,000 (observed)	203267	0.037
32,000 (observed)	438603	0.085
100,000 (estimated)	1536618	0.533
500,000 (estimated)	8837121	2.695
1,000,000 (estimated)	18674232	5.271
10,000,000 (estimated)	220101127	75.321

1. Comparing the sorts at a general level, is one sort *always* better than the others?

Merge sort is always better than selection and insertion sort. At a list size of 10,000,000, merge sort took only 0.000004% of selection sort's comparisons and 0.000008% of insertion sort's comparisons, and only around 0.00002% of selection/insertion's time to complete.

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

Insertion is the best sort for a mostly/already sorted list. Best case for insertion sort is O(n), while best case for selection is O(n²) and merge is O(n log n).

3. You probably found that insertion sort has about half as many comparisons as selection sort. Why?

~~Selection sort~~ Insertion sort only scans the elements until it finds the one it needs (i.e. j+1). Meaning, insertion does not always have to ~~sca~~ scan all the elements. Selection sort, however, must scan all remaining elements to find j+1. So on average, insertion sort usually only executes about half as many comparisons as selection sort by design.

4. Given the above observation, why are the times for insertion sort not half what they are for selection sort? (For part of the answer, think about what insertion sort has to do more of compared to selection sort.)

Insertion sort takes more work and time modifying the list than selection sort does. Insertion sort has to push every later element one farther when inserting an element in place, whereas selection sort only switches two elements when sorting. So while selection sort takes more time scanning all the elements, it takes less time moving/sorting the elements.