Selection Sort

List Size	Comparisons	Time (seconds)
1,000 (observed)	**499500**	**0.065**
2,000 (observed)	**1999000**	**0.247**
4,000 (observed)	**7998000**	**1.015**
8,000 (observed)	**31996000**	**4.104**
16,000 (observed)	**127992000**	**16.959**
32,000 (observed)	**511984000**	**67.250**
100,000 (estimated)	**4999841608**	**656.738**
500,000 (estimated)	**124996040220**	**16428.500**
1,000,000 (estimated)	**499984160880**	**65714.000**
10,000,000 (estimated)	**49998416088000**	**6571400.000**

Insertion Sort

List Size	Comparisons	Time (seconds)
1,000 (observed)	**249072**	**0.062**
2,000 (observed)	**992896**	**0.242**
4,000 (observed)	**3999435**	**1.024**
8,000 (observed)	**16115122**	**4.073**
16,000 (observed)	**64321674**	**16.675**
32,000 (observed)	**257455598**	**63.333**
100,000 (estimated)	**2514214824**	**618.486**
500,000 (estimated)	**62855370600**	**15462.200**
1,000,000 (estimated)	**251421482400**	**61848.800**
10,000,000 (estimated)	**1005685929600**	**6184880.000**

Merge Sort

List Size	Comparisons	Time (seconds)
1,000 (observed)	**8703**	**0.003**
2,000 (observed)	**19409**	**0.008**
4,000 (observed)	**42774**	**0.016**
8,000 (observed)	**93591**	**0.035**
16,000 (observed)	**203267**	**0.075**
32,000 (observed)	**438603**	**0.151**
100,000 (estimated)	**15211681**	**0.523**
500,000 (estimated)	**86790929**	**2.984**
1,000,000 (estimated)	**182743769**	**6.283**
10,000,000 (estimated)	**2131991248**	**73.301**

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

At a general level, where all the values in a list are random, merge sort always better than insertion sort and selection sort. Because it is not likely that we have a best case for insertion sort, the time complexity of merge sort is nlog(n) and n^2 for insertion sort and selection sort. .Therefor, merge sort is better than insertion sort and selection sort at a general level.

Insertion sort is always better than selection sort because selection sort always have the same number of comparison for all the best, worst, 'average' cases. But, in insertion sort, it is likely to have half of the number of comparison compare to selection sort. Therefor, insertion sort is always better than selection sort at a general level.

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

Insertion sort will always better than merge sort and selection sort when it sorts a list that is already sorted or mostly sorted. Because, in a sorted or mostly sorted list, it is likely that insertion sort will do one comparison or a few comparisons when it puts the next unsorted element into the sorted list. So, we might get linear time complexity or slightly worst than linear when we used insertion sort.

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

When insertion sort puts the next value in a random list into the sorted list, in average, the number of comparison is half of the number of element is the sorted list; Because the list is generated randomly, insertion sort does not compare every single value to insert the next element in its correct order. So, in a random list, insertion sort has about half as many comparisons as selection sort.

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.)

Because every iteration of insertion sort is more expensive than selection sort. Insertion sort has to move a lot of memories for each iteration while selection sort only have to swap/move memory n - 1 times.