Lab 6

Selection Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)	499500	0.05202221870422363
2,000 (observed)	1999000	0.20905637741088867
4,000 (observed)	7998000	0.8422033786773682
8,000 (observed)	31996000	3.3417539596557617
16,000 (observed)	127992000	11.225546836853027
32,000 (observed)	511984000	35.77805972099304
100,000 (estimated)	2047872000	111.8064
500,000 (estimated)	8191488000	559.032
1,000,000 (estimated)	3.28 * 10^10	1118.064
10,000,000 (estimated)	1.3 * 10^11	11180.64

Insertion Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)	247986	0.04401206970214844
2,000 (observed)	1018717	0.1800389289855957
4,000 (observed)	3995264	0.6891758441925049
8,000 (observed)	16112194	2.7786262035369873
16,000 (observed)	64667449	10.432350635528564
32,000 (observed)	257507119	31.003979206085205
100,000 (estimated)	1030028476	96.88743502
500,000 (estimated)	4120113904	484.437
1,000,000 (estimated)	1.648 * 10^10	968.874
10,000,000 (estimated)	6.592 * 10^10	9688.74

1. Which sort do you think is better? Why?

Insertion sort is better since it scales better with the problem size. Additionally, while both are o(n^2) insertion sort is more optimized and cuts the amount of comparisons done greatly.

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

Insertion sort still beats selection sort. The amount of comparisons still stay the same within the selection while the insertion sort adjusts to the data in the list to reduce the amount of comparisons.

3. You probably found that insertion sort had about half as many comparisons as selection sort. Why? 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 is able to half the amount of comparisons of selection sort because selection sort still has to go through each element, but with insertion sort, by inserting the item into a sorted list, it is able to reduce the amount of unsorted items and therefore reducing the amount of comparisons done overall. The time is not halved since both are o(n) algorithms meaning they scale the same -- not affecting time..