Selection Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)	499500	0.1461009979248047
2,000 (observed)	1999000	0.551893949508667
4,000 (observed)	7998000	2.302361011505127
8,000 (observed)	31996000	8.941316843032837
16,000 (observed)	127992000	36.915199995040894
32,000 (observed)	511984000	143.6296398639679
100,000 (estimated)	4999950000	~512
500,000 (estimated)	124999750000	~2048
1,000,000 (estimated)	499999500000	~8192
10,000,000 (estimated)	49999995000000	~32768

Insertion Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)	241969	0.050013065338134766
2,000 (observed)	994282	0.20342397689819336
4,000 (observed)	4009852	0.8217380046844482
8,000 (observed)	16024529	3.2249701023101807
16,000 (observed)	64151892	12.34569501876831
32,000 (observed)	258233371	53.42255902290344
100,000 (estimated)	2499975000	~170
500,000 (estimated)	62499875000	~682
1,000,000 (estimated)	249999750000	~2730
10,000,000 (estimated)	24999997500000	~10922

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

Insertion sort is better because it makes less comparisons, operates quicker, and computes the same result.

2. Which sort is better when sorting a list that is already sorted (or mostly sorted)? Why?
   Insertion sort can have a Big-O of O(n) when a list is already sorted, whereas selection sort always has a Big-O of $O(n^2)$ no matter the input list.

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?
   Insertion sort does not have to iterate through the whole list for each insertion, or "pass", to insert a new value; it only looks the necessary values of the sorted list for insertion and rarely looks at the whole list, so the times for insertion sort are more variable to the relative position of unsorted items and therefore cannot be regarded as taking half the

time as selection sort. Each pass of a selection sort must iterate through the items that have not been sorted in order to find the minimum value to swap in the correct position. Additionally, if a shifting insertion is implemented, the computation time can be optimized in an insertion sort to further reduce operation time.