Selection Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)	12497500	0.14400482177734375
2,000 (observed)	1999000	0.589292049407959
4,000 (observed)	7998000	2.313969373703003
8,000 (observed)	31996000	9.043273687362671
16,000 (observed)	127992000	37.84829616546631
32,000 (observed)	511984000	162.12741804122925
100,000 (estimated)	4999950000	1704.3308799266815
500,000 (estimated)	61284900000	1902.4912895039551
1,000,000 (estimated)	901345920000	2001.2049488102401
10,000,000 (estimated)	100492550000	2005.5359429435602

Insertion Sort		
List Size	**Comparisons**	**Time (seconds)**
1,000 (observed)	247986	0.06004595756530762
2,000 (observed)	1018717	0.26194095611572266
4,000 (observed)	3995264	1.0184988975524902
8,000 (observed)	16112194	4.024745941162109
16,000 (observed)	64667449	16.17403292655945
32,000 (observed)	257507119	64.28958106040955
100,000 (estimated)	2505274077	1576.578488111496
500,000 (estimated)	30012429500	2000.239503820439
1,000,000 (estimated)	4002334192401	3024.240358693132
10,000,000 (estimated)	63845913454924	5058.3591435468104

1. **Which sort do you think is better? Why?**
   I believe insertion sort is better because it only uses one comparison when the (n+1)st element is greater than the nth element. Selection sort, on the other hand, must always check all remaining elements to find the absolute smallest element in the unsorted part of the list.

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

   Insertion sort is better than sorting a list that is already sorted or mostly sorted because it just compares A[k] and A[k-1] and does the implementation in one sweep. For a nearly sorted array, insertion sort will only need a handful of O(n) passes to complete.

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 will usually perform less comparisons than selection sort because in a selection sort, for each element that is to be added to the sorted part, you have to scan the entire unsorted part of the list to find the minimum remaining element. Whereas in insertion sort, it searches the sorted section to find out where the next element will go but the search ends once it finds the insertion point.
   However, the times for insertion sort are not half what they are for selection sort because insertion sort has an additional time penalty when sorting an array; insertion sort requires n/4 comparisons, and each of the n/4 comparisons requires the element to be moved up so there are also n/4 moves as well as comparisons. Selection sort only requires one swap.