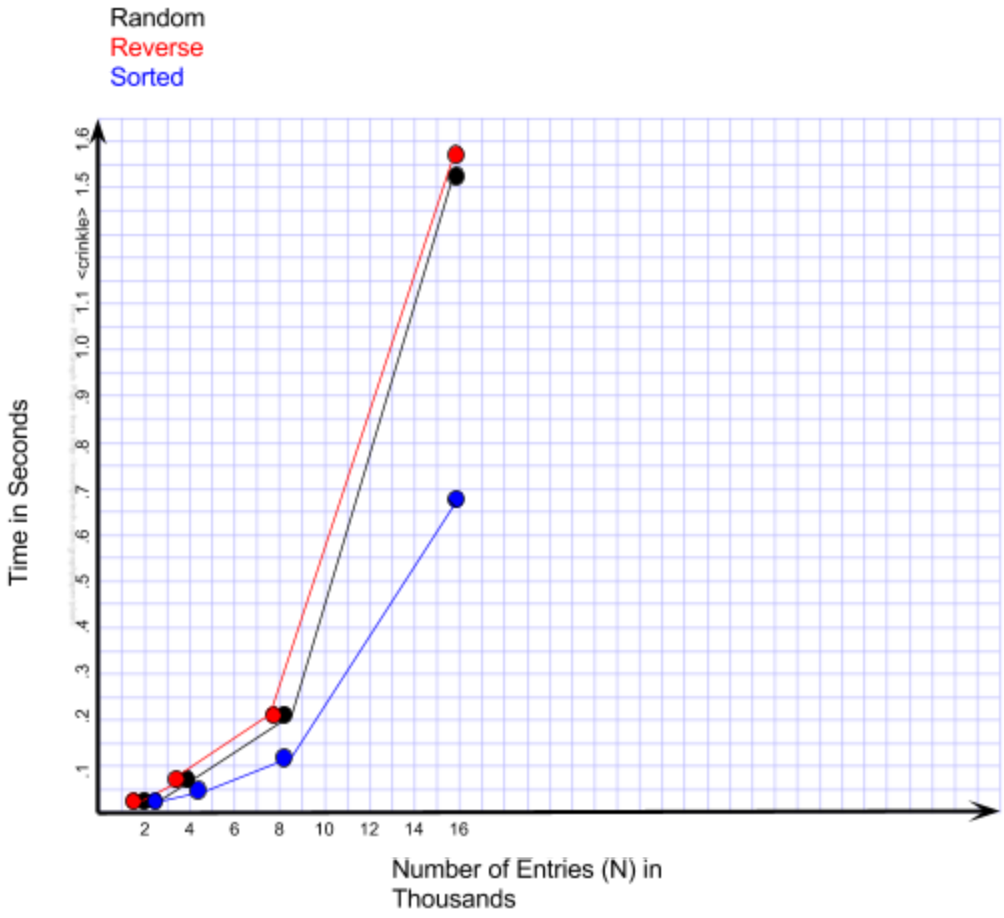

3) From your timing data analyze the rate of growth of SelectionSort using the doubling rule.

When we compare the runtimes of the SelectionSort after having doubled the input from the previous input we find that the run time also increases in a non-linear way. Looking at the times for both Random sort and Reverse sort, we can see that as we double the input, the run time increase by 2x, 4x, then 8x. This relationship is increasing by approximately n^2 this can be seen as the run time is being increased by growing multiples of two (2,4, and 8: powers of two). For the already sorted list, the growth is also quadratic, but likely with some coefficient attacked on that n^2, maybe it is 1/2n^2.

4) Analyze the SelectionSort code: use number of array accesses as your cost model, and describe the rate of growth in tilde (~) notation.

Array access on its own is constant time, that said, the array accesses are nested within a double for loop. Because of this, it will grow at ~n^2.

5) How does the order of the input data affect SelectionSort? Explain why, by referring to the algorithm or the code. (How many times **less** and **exch** called?)

The order of the input does not affect Selection Sort because it has to scan to the end of the remaining unsorted unsorted entries regardless of when it found the smallest entry. Once it finds that smallest entry, it moves it into its correct position within the sorted entries. The order does matter for an algorithm like insertion sort, because this algorithm will compare its smallest entry with every single sorted element until it finds the right position.