**Performance analysis of sorting**

A naïve analysis of the problem in the assignment would lead one to think that choosing an optimal O(nlgn) sorting algorithm would give us the best performance as we get larger numbers of records. However, in this case we’re adding each new record to the already sorted list. This would put us dangerously close to the worst case {O(n^2)} of a standard quicksort per pass (ie record) but alternatively puts us very close to the best case for a simple insertion sort {O(n)} pass. For my implementation I tried a quicksort implementation which ended up being very close to O(n^2) per pass as expected and then refactored to use a simple insertion sort. After using the insertion sort, I ended up with an O(n) per pass. In the end because there are n passes for each of the 6 sorts, the total time complexity ends up being around 6n(n+1)/2 which ends up being O(n^2) overall. This is still better than even the best case for a quicksort which would end up being O(n \* nlgn). Ultimately my times generally followed a linear increase per pass with some discrepancy due to real world cpu/memory constraints.

**Graphs for per pass insertion sort timing**

Y axis is the number microseconds per pass

X axis is the pass number