**CS 245 - Assignment 1**

**Bio Owens**

Throughout the development of this assignment, there were several ways to implement different parts of the program, though being conscious of what was and was not efficient was a big key in ensuring that this sorting algorithm would be faster than merge sort. Therefore, while implementing my entire program I made sure everything was efficient and this led to my hybrid algorithm being faster than merge sort. In my hybrid algorithm the in-place sorting algorithm I used was insertion sort. I chose insertion sort because it fulfills the requirement of being in-place, it is stable unlike quicksort (which is faster in some scenarios) and faster than bubble sort and selection sort. Though, my hybrid sort is only faster when a certain range of run\_sizes are used. The most optimal run size of base 2 I found was 128. My hybrid sort is faster with certain run sizes because it avoids unnecessarily sorting sequences, or runs in this case, that are already sorted. This means there are less values to compare overall. Continuing, merge sorting everything can be costly depending on the amount of data. With little variance in run times are very short amounts of data, such as an array of length 128, by using merge sort instead of say insertion sort, since merge sort is not in place it is creating extra variables and arrays that are adding overhead to its run time. Since insertion sort avoids this, it uses less memory and runs faster on smaller amounts of data. After all sequences are sorted, this is where merge sort shines. Merge sort comes in and merges all the sequences of sorted order, which sorts the entire array in the end.

After analyzing the run-time of both algorithms, merge sort seems as if it should be faster since it has a run time of nlog(n) and due to insertion sort being used in the hybrid algorithm and in the worst case being n2. Despite this being true after only looking at the non-constant dominating term of their runtimes, when analyzing the speeds of the algorithms themselves, a different conclusion is come to. Insertion sort is only used on sequences of digits less than the desired run\_size, meaning that as long as the run\_size configured isn’t significantly large the run time of that particular sort will be negligible in comparison to merge sort. Therefore, if an n2 run time is only happening there and it is negligible, the next dominant run time of the hybrid algorithm is less than nlog(n) as long as the run time is within a reasonable range. This leads to the hybrid algorithm being faster than merge sort with a good configured run\_size.

Based on my findings, any run\_size between 64 and 350 should result in the hybrid algorithm being faster than merge sort when the data is over 250,000 elements. The most fine tuned run\_size that was consistent I found was size 128.