**Lab 3: Timing Analysis and Report**

For this lab, a multithreaded C++ library that performs bubble sort, insertion sort, and merge sort on a vector of integers was converted into a JavaScript library using Web Assembly (WASM). Timing performance analysis was performed on the multithreaded C++ and JavaScript libraries, and single-threaded versions of the same C++ and the JavaScript libraries. An overview of the timing comparison between each library is shown below.

The top half of the charts represent performance of single-threaded versions of each algorithm in each respective library (WASM and native C++) and the bottom half represents that of multithreaded libraries. From this chart, the fact that the multithreaded algorithms in both libraries are much faster is immediately recognizable. The WASM performance in the multithreaded library is nearly comparable to that of native performance of the single-threaded counterpart. However, upon closer examination, it is seen that performance of merge sort is slower in the multithreaded versions than it is when single-threaded. The charts below take a closer look at the performance of merge sort in both single-threaded and multithreaded environments.

In both native C++ and WASM libraries, the single-threaded versions of the merge sort were noticeably faster than the multithreaded versions. The discrepancies in timing performance were much more pronounced in the WASM libraries. With no overlap in data at 95% confidence intervals, the results of merge sort performance for both native C++ and WASM in both multithreaded and single-threaded environments are statistically significant. It is also seen that that the confidence intervals for the multithreaded codebases are much larger. One explanation for this counterintuitive result of single-threaded algorithm outperforming its multithreaded counterpart is the fact that extra overhead introduced in creating and joining threads in the multithreaded approach plays a role in timing performance. With merge sort being a highly efficient and speedy sorting algorithm, the difference in the existence of these overheads may have influenced the timing performance in a much more significant way than the number of threads present.

Next, timing performance analysis and comparison are done on insertion sort and bubble sort between single-threaded and multithreaded WASM environments. The chart below shows the timing performance of bubble sort in both multithreaded and single-threaded WASM environments.

The multithreaded version of bubble sort performs significantly better than the single-threaded version. On average, the multithreaded version of the bubble sort is 7 times faster than its single-threaded counterpart. With data entries at 95% confidence intervals never coming close to overlapping, the results are seen to be statistically significant. This chart conforms to the widely accepted idea that multithreading is generally faster than single-threading since multithreading allows for simultaneous, parallel executions of algorithms. With bubble sort being significantly slower and less efficient than merge sort, the overhead introduced by multithreading mentioned earlier does not seem to affect or influence the timing performance.

The chart below plots the timing performance of multithreaded and single-threaded WASM based insertion sort.

Much like bubble sort, the multithreaded version of insertion sort outperforms its single-threaded counterpart by a great margin. With 95% confidence intervals never overlapping and staying far apart, the results of this timing performance data can be said to be statistically significant. On average, the multithreaded insertion sort is approximately 4 times faster than the single-threaded insertion sort. Much like bubble sort, insertion sort has the timing complexity of O(n2) and benefits significantly from having a multithreaded approach.