David Kim and JD Francis

For the Sorting Competition lab, we implemented 5 different iterations of sorts and improvements to improve the overall speed of our sorting algorithm. These iterations include quicksort, selection sort, bubble sort, threading, and length pre-fixed strings. In terms of types of sort algorithms, bubble was the slowest sort and quicksort was the fastest sort, leaving selection sort in the middle.

Because of bubble sort’s algorithm, its complexity is O(n2) for both average and worst case. Comparing and finding the largest number for n times (n being the number of elements) takes too much time and resources to be considered a good sort algorithm for large data sets. We got an average of \_\_\_\_\_\_\_\_ seconds when the program was run \_\_ times. Compared to the other sorts, it is a much slower and costly sorting method. Therefore, of the three sorts implemented, bubble sort was the worst one in terms of time.

Selection sort is another algorithm with a complexity of O(n2) for an average and worst case. It also must iterate through every element in the array every time it is called recursively. As a result, it is still a relatively inefficient method of sorting in terms of large data sets. Selection sort had an average time of \_\_\_\_\_\_ seconds when run \_\_\_ times. While it is still slow, it is much better than the speed for bubble sort. This is because selection sort’s swapping portion of the algorithm is only done once per iteration whereas bubble sort can swap multiple times in one iteration. Selection sort simply looks for the index of the longest word (or however and whatever is being sorted) and swaps according to that. On the other hand, bubble sort can potentially swap at every comparison of elements.

The fastest sort out of the ones that we implemented is the quicksort. With a O(n2) worst case and O(nlog2n) average case, quicksort is much faster and more efficient as a sorting algorithm than the other two. The utilization of splitting the algorithm (divide and conquer) into smaller and smaller parts and moving around a pivot give the log2n property that makes the algorithm so cost efficient. In terms of usefulness, quicksort is a good algorithm to use, even in bigger data sets. With a file size of \_\_\_\_ KB, we found an average of \_\_\_\_ seconds running it, ­\_\_\_ times, much faster than the previously mentioned algorithms. However, there is still room for improvement, especially with implementing the sort on our own. This is where the use of threading and length pre-fixed strings come into play.

Threading gives the ability to mimic usage of two paths to do different jobs. With threading, the computer seems like it is running two operations at the same time, halving the time necessary in certain lower complexity functions. For our implementation, we used threading within quicksort to thread the recursive calls, left and right. In doing so, we were able to get an average of \_\_\_\_ seconds (which includes length-prefixed strings) with a total run number of \_\_ times. Not necessarily “built” for large data sets, threading is still a powerful way to increase the full potential speed of the computer by utilizing its multiple cores (which most computers now have).

On top of this, we used length pre-fixed strings in order to take away most of the overhead that is associated with using the strcmp and strlen functions in which we have no control over. Length pre-fixed strings (LPF strings) are cstrings that have the first element as an integer that represents the total length of the following cstring (note that an integer, not a character of an integer, will be given and would give undefined behavior if printed with the whole string). Using LPF strings and functions, we were able to shave off a little time from the total with an average of \_\_\_\_ seconds (which includes threads) having been run \_\_\_ times. Even in terms of coding efficiency, the LPF strings are very efficient in that the new functionalities, such as strcmp and strlen except for LPF strings, are simple to write and do not take up much time, providing a simple solution to increase the speed, even if only by a little.

Using the three techniques/algorithms, quicksort, threading, and LPF strings, we came up with a version of our most efficient implementation of a sorting algorithm. The mixture of efficiency from a quicksort and the small additions that threading and LPF strings help to reach the efficiency that everyone, especially programmers in a competition to get a get-out-of-jail-free pass, strives towards.