**Group Members:** Colin Ackerley & Gregory Cook

**Framework:**

1. Map
2. Reduce
   1. The reduce framework creates a thread/fork for every index in the hashtable that contains the mapped data. Each thread then calls the reduce method. The reduce method then sorts each array of the hashtable, for both word count and sort. The reduce method will then count the number of occurrences of each word if using word count. After sorting and counting is completed for a thread, the data is added to a global array of input size that contains a reduce node struct. The reduce node struct contains a string for a word, an int for a number, and an int for count. Sort only uses the num variable, and word count uses count and word. The threads add to the global array in order, so the data will remain sorted. Once each thread has completed execution, the data in the global array will be added to the output file.

**Difficulties:**

1. Reduce
   1. Passing the counter of a for loop via a thread proved difficult, since doing so would not always pass the correct index value due to the way loops function. The solution was to create an array of ints that was the same size as number of loop iterations, and then pass the value from the array instead of the loop counter.
   2. Deciding on a way to keep all data sorted while still using threads was also a challenge, but the solution to have a thread for each index of the hashtable and then add that to a global array worked well.
   3. Preventing segfaults and ensuring that all data was sorted and added to the global array properly was also challenging, but was accomplished successfully by carefully examining and checking edge cases and a lot of debugging.