Sorting Competition Documentation

Jake Osselaer

Quincy Schurr

To start the sorting aspect of our sort, we decided to try a quick sort algorithm since it is known as one of the fastest sorting algorithms. We successfully implemented a basic quick sort for the task but it ran incredibly slow. Our code was running a 550 kb file in about 13 seconds. After re-evaluating the code we decided to implement a simpler version of one method. In the earliest version of our quicksort we switched elements of a char\*\* array by creating a temp array, but then decided to do the same thing by calling swap. We saw a decrease in time by doing this. Another thing we realized was slowing down our quick sort was the way we were calling strcmp and strlen. When we tested different sized files we saw that our time was not increasing linearly, but exponentially. For very large files, the strcmp and strlen functions were getting called a lot. This really slowed down our sorting, but we couldn’t see a way to get around calling strcmp and strlen where we did in our program. We then decided to try merge sort. We successfully implemented this sorting algorithm and it worked a lot faster than our quick sort algorithm. Our merge sort was still growing at an exponential rate though when we increased our test file size. We discovered that when we called the merge function we were creating a temp array the size of the number or words for each time merge was called and that was increasing the time it took to sort. When we tested a 12 mb file it took more than three minutes to run. To fix this we declared the temp array in the prepareData function so that the creation of the temp array would not be added to our sorting time. This worked really well because for big sets of data we only needed to create the temp array once, instead of every time we recursively called our merge function.