André Gras and Chris Perez

Project 3 Overview

The first sort we tried was Bubble Sort. We thought it would be easy implementation to just get a sort working and move from there. It performed moderately well with the small data sets we tried; however, when the data set grew larger Bubble became less efficient and took increasingly longer periods of time to sort the data sets. It was sorting at O(n2) complexity, and we needed something faster for the project. We knew Bubble was not where we wanted to end up, so we decided on attempting to optimize a QuickSort.

Our first iteration of QuickSort was simple, with the pivot being the first element of the array. We experienced faster sort speeds for the larger files, but the smaller were closer times, albeit still slightly faster than Bubble. When our data set was sorted already, QuickSort performed slowly, closer to O(n2) complexity. Having the pivot be the first or last element makes nearly sorted or reverse sorted arrays be around O(n2). Our average time to search through one Bible (out of 5 runs) was 2.2 seconds.

We decided to implement Merge Sort just to check its efficiency, but ultimately could not implement it without segmentation faults. From there, we decided to continue with our Quick Sort, but attempt to make it faster. Learning about the median of threes in class, we decided to optimize our pivot better.

In implementing the median of threes, we encountered many problems in our code. We could not figure out why there was a segmentation fault occurring in the code. Once we figured out the errors in our programming, we found the speed to be a lot greater. Our average time from 5 runs of the Bible gave us a sorting time of .716 seconds.

We also wanted to see if just choosing a random pivot would increase the speed more than doing the 3s median. The object of randomizing is to lower the chance of a worst case O(n2) resulting from a bad pivot. The 3s median solves that problem, but we still wanted to see if randomizing worked faster (we did 5 tries of averaging 5 runs to make sure). Our total average ended up being .928 seconds. We found that merely choosing one random variable ends up being too inconsistent and leaving us with slightly slower speeds.

We thought we could try to make it faster if we randomized the first pivot pick then did the median of threes. We found the times to be almost equal, sometimes slower, sometimes faster. We decided to keep the random picking, because it increased the likelihood of getting a better median closer to the actual median.

We are keeping the median of 3s as well over a median of 5s because the median of 5s would mean more strcmp and a lower speed. We believe the stability and consistency of this sort will keep our program strong and swift. We also decided to use length-prefixed c-strings to speed up the comparisons, although kept the vector, knowing access would be slower for ease of reading in data.