Clojure Project Summary

This was a difficult project. After spending nearly 20 hours on it in combined time and effort, this is the best that I could come up with. For starters, getting the REPL to run was nearly impossible, so I opted for an online run-time environment that took care of everything “under the hood.” It is called repl.io. From there, I had to figure out Clojure’s syntax, which took a lot of time to get used to. Simply learning how to implement a basic IO stream took several hours and search after search on Google.

Implementing the sort function took the most amount of time. At first, I tried to implement a pure merge sort algorithm, but that quickly turned out to be too complicated for the time that I had to work with. I ended up partitioning the data set, then doing built-in sorts on the partitions, and combining the results recursively. So, in the end, it is a merge sort, but not exactly a *pure* one. It was simply the best that I could do under the time constraints.

After I had that all figured out, I had to parallelize the merge sort, but this turned out to be much trickier than I would have thought. I originally thought that I could parallelize the sorts for each partition, but that failed to work. So, I used ‘pmap,’ which does give some parallelization. To my surprise, however, it actually increased the amount of time it took to complete the sorts. The only reason that I can think of is that the ‘pmap’ function somehow interfered with the built-in sort functions that I was calling, which ended up slowing them down. Since ‘pmap’ does the parallelization automatically, I could not choose how many threads I wanted to run at once. Regardless, my sorting algorithms are nothing compared to the built-in sort function, which is nearly a full order of magnitude faster on average.

Below are a graph comparing my two searches.

