SIT315 M2.T2C: Complex Threading With OpenMP

Codey Funston | S222250824

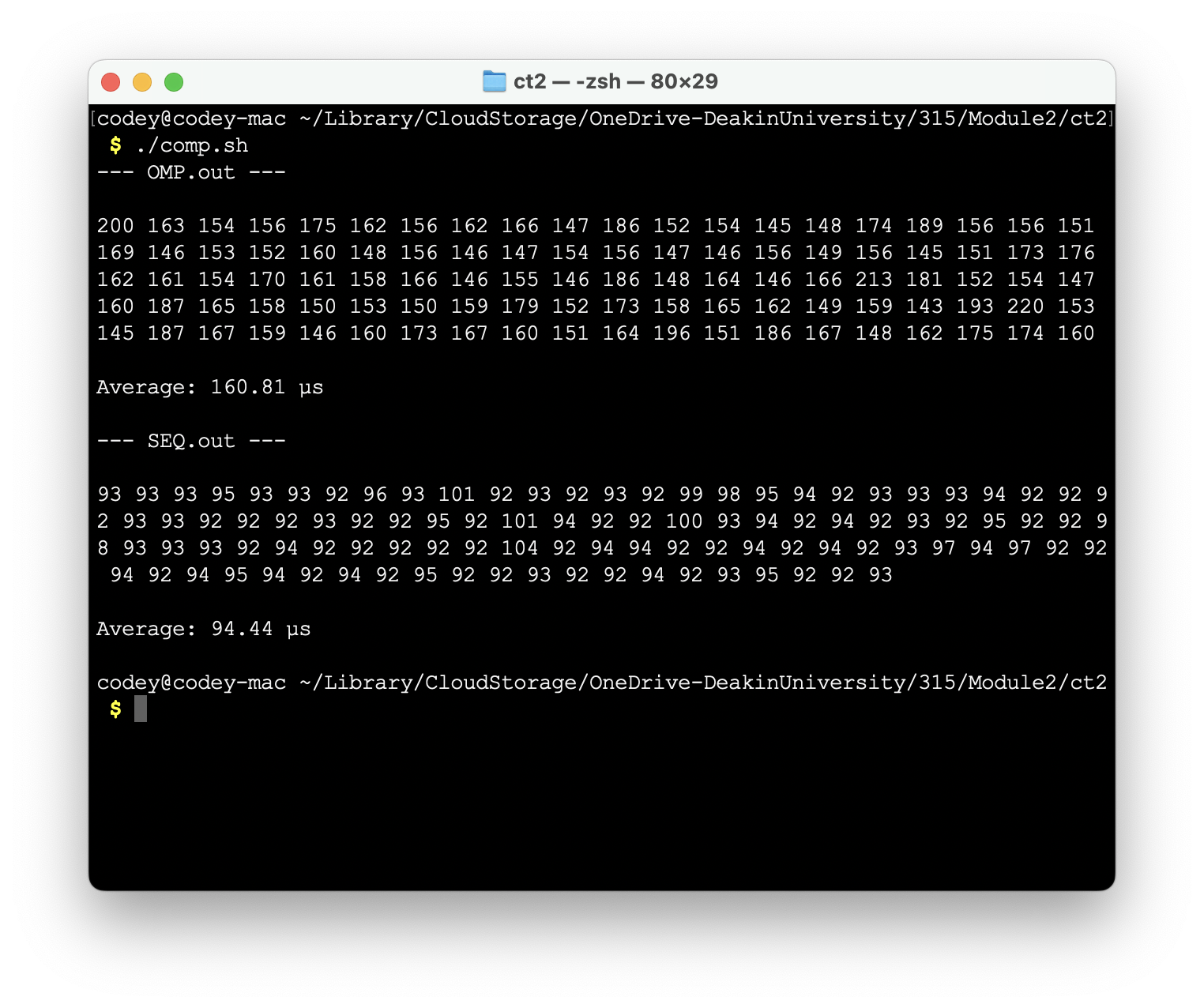
Decomposition and Performance Comparison

I decided to implement a quicksort algorithm that used recursion as that seemed to work better conceptually with parallelising the program. The function runs its body and then has two recursive calls, splitting the data in two based on a random pivot element. I used OpenMP to assign a thread to each call and to not use a barrier since both calls work on separate parts of the data that do not cross over.

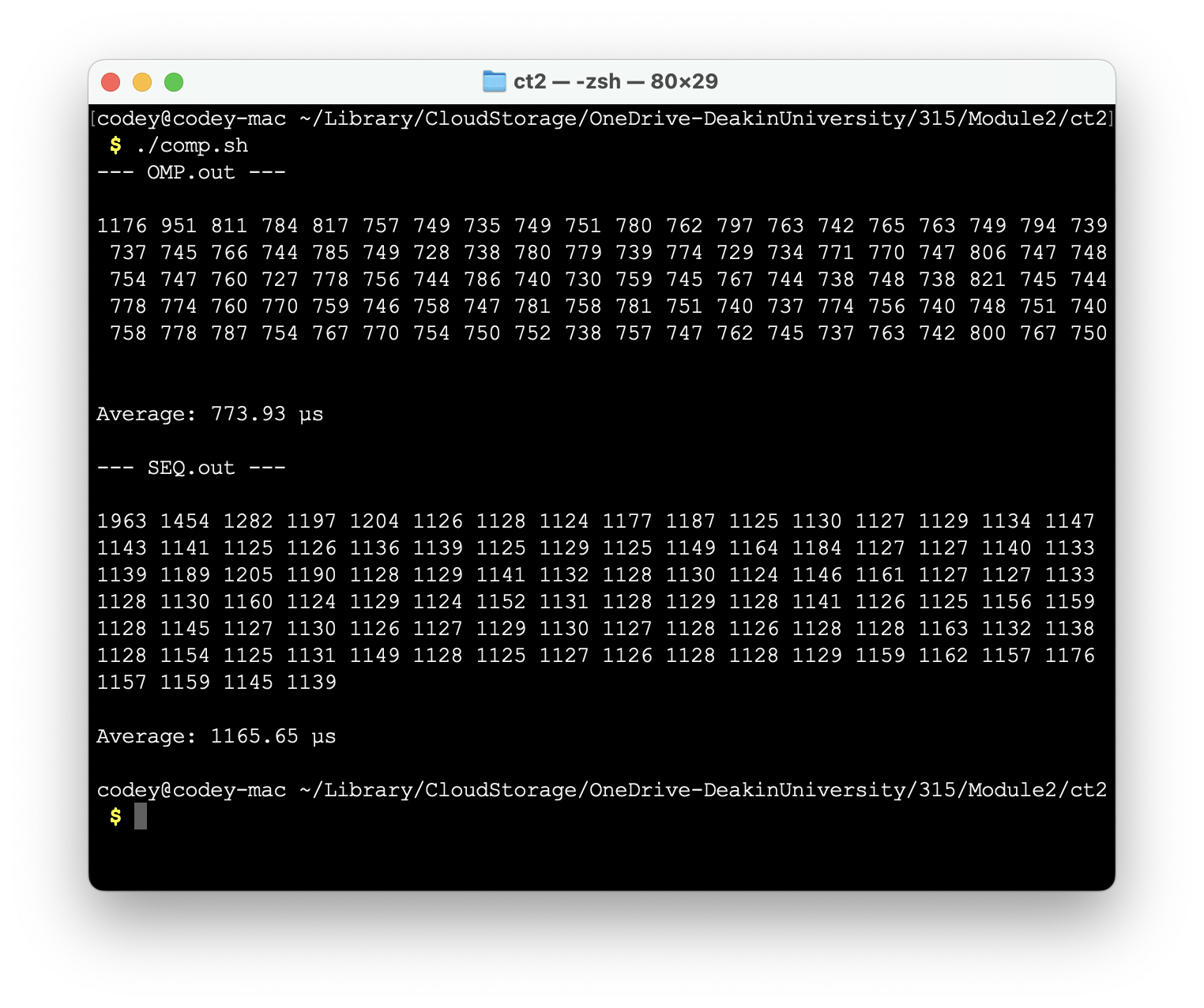
The runtime of the sequential program wasn’t that bad given that the time complexity for quicksort is . For very small data input the sequential version runs faster, for example with a vector of 2000 integers, over 100 tests for each program type the average for the sequential version was 96.51 microseconds, compared to the OMP version’s average of 166.02 microseconds. This is because the threading overhead is much too high. To help reduce the overhead I added a condition based on testing output to only assign a thread to a recursive call if the section of data being worked on consisted of more than 1000 integers.

Some Results and Corresponding Input Data Size

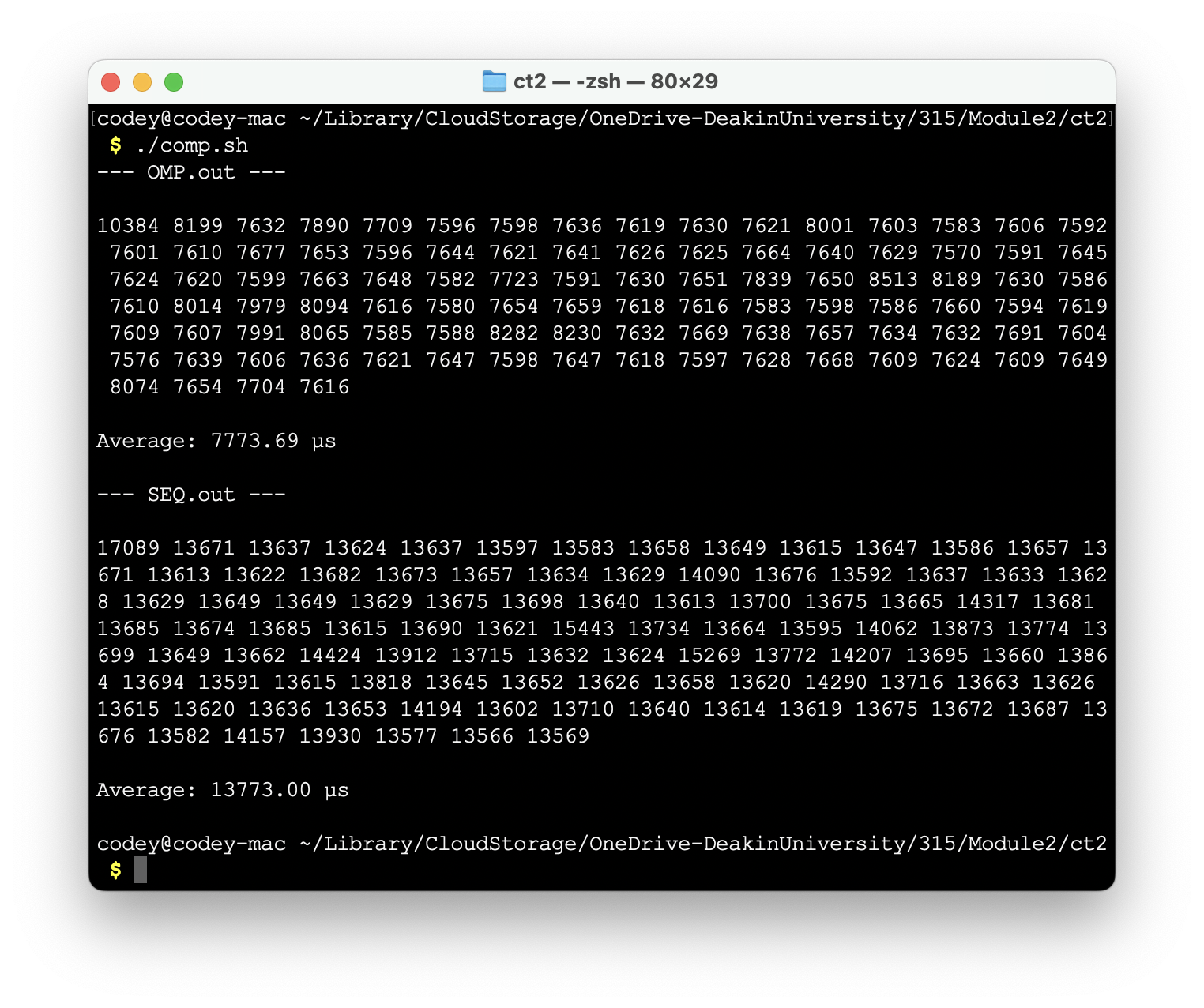
* 2,000



* 20,000 (OMP is slightly faster now)



* 200,000 (OMP is now almost twice as fast)



* 2000000 (no more improvement is seen now from OMP)

A screenshot of a computer

Description automatically generated

* 100 (the OMP version’s safeguard for data with less than 1000 integers means that for this incredibly low data size the program is run sequentially)

