When it came to designing the system for Project 2, the foundation seemed to be the input file containing 14,000 numbers. After validating command line arguments, the next step was to set up global arrays that the main thread and all descendants could access and mutate. The first array was to contain the 14,000 numbers, and the remaining 3 were arrays of length 14, meant to contain the values calculated by each worker thread for the main thread to access. After these foundational pieces were set up, reading the input file with the read system call seemed the simplest and fastest. I initially attempted to loop and read from the file 4 bytes (size of an integer) at a time until the file had been traversed. The resulting 4 byte string should then be converted into a long double data type for future operations. However reading 4 bytes at a time proved to be a problem since pieces of data on separate lines were being grouped together whenever the initial number was less than 3 digits (2 digits, 1 newline, next line's first character). Instead I initialize an empty character array and read from the file 1 byte at a time. On each read, if the character is not the newline character, it gets appended to the character array via reallocation of memory. If the character is the newline character, it is converted to an integer type via the atoi function, and inserted into the array. A counter num keeps track of the index of each final number so insertion is trivial. After 14,000 values have absolutely been read, an output file is created using the POSIX system call. Next the main thread declares and initializes an array of worker pthreads, supplying its runner function. An error came up when trying to give each pthread the counter value from the for loop as a function argument. Since each pthread was trying to use the same memory address from the variable i, an error resulted. Instead they should be allocated their own piece of memory that stores their function argument. A separate for loop was used to then wait for each pthread to terminate. The reason comes from the fact that having pthread_create and pthread_join in the same for loop leads to blocking, a thread would be created, do some operations, terminate, and these steps repeat 14 times. Rather we want the threads to execute concurrently, so they should be allowed to execute together. After each of the 14 worker threads were established, they were responsible for creating 3 threads of their own. To do this, I implemented much of the same method as the main thread does by creating an array of pthreads, initializing them with some runners, and separately waiting for the threads to join for the worker child thread to continue execution. After the worker child threads were verified to each create 3 threads of their own, the 3 grandchild threads were given separate runner functions to calculate the root of the square of sums, geometric average, and arithmetic average. When initially performing the root of the square of sums, I ran into a compilation issue and quickly realized that to link the math library, the option -lm needed to be appended to my command line statement. One of the main issues that I finally faced was returning values from the grandchildren pthreads up to the child worker threads. After some research on linux man pages, I found an address of memory can be returned up to the creator of the thread, by passing in the pointer to pthread_exit. To retrieve the value in the child worker thread, pthread_join allows an address of memory to be supplied as

a second argument. This is where the returning thread will store its return value. After some issues with type casting, I properly casted and dereference the value to update the global array in the child worker thread. The final issues to write to an output file were trivial, and simply required some string functions from the string library such as strlen, and sprintf to convert a numeric to a character array.