The steps are needed for the project program are, 1. read data from input file, 2. distribute data set for 12 thread to do 3 computations, and 3. Write all thread result and the maximum result of the 3 major computation into a file as output. Since the data number of the input file is fixed for this project, allocating an integer array with size 12000(amount of data in file) for the purpose of holding all reading data value. Also 3 integer arrays with size 12 for the purpose of holding each thread computation result. Those arrays variables should be in global scope to maximum the accessibility for the whole program. Step 1 simply using fget() from the I/O FILE class to read each line of data from input file. Then, using atoi() to convert reading line to int and store in our global InpArray. Step 3 using system call creat() to create output file and write() to append information into output file. But during the process of step 2, there are 2 problems encountered.

First, while testing the algorithm for geometric average, only five random numbers were introduced to test and the output result is correct. So on, head to actual run the algorithm with one thread’s data set, the result were getting as 0 or error message “Segmentation fault (core dumped)”. Why this happened was found by printing out every result of the computation. The algorithm had was multiplying all data first and then dividing 1/1000 (1000 data). While computing the product with more data, the result was getting over large which cause out of bound. So, come up with a solution approach as suggested by the professor. Instead of getting the product of all data set first, I divide each data by 1/1000 first and then multiply them all. Although another approach, which dividing the product by 1/1000 when the product getting related large, can save few calculate instruction time. But for code simplicity, I divide each data by 1/1000 first.

Second, during testing to pass arguments to pthread\_create() function with a fixed integer. The child pthread will get the exact value of the passing value, which should be work like that. And so, I used this single pthread as testing example for the program process. After that, when I apply a for loop to create 12 pthread instead one pthread for the program, I found out that the passing value for each child pthread is not correct and sometime repeated. But if I add a line of code to print out every passing value they received, it will work correctly. At first, I thought this situation because I did not terminate each pthread properly by pthread\_exit(). But pthread\_join() and pthread\_exit() is both do the same job to handle child thread terminate for the main program, and the situation does not change still after I introduced pthread\_exit() into my code. Then I assume it’s cause by the order of each pthread execute and finish not pair, so I use wait (1) to control the computation. But using wait() is taking off the advantage of thread parallel computation. Anyway, I went into look the definition of pthread\_create() at the end. The passing argument for pthread\_create() is passing the address of the parameters. Since the for loop is incremented every time after one pthread\_create() code, but the passing parameter “i” is just one memory address. Because of that, there’s a possibility for multiple pthread using same “i” value during computation. To solve this problem, I created another int array with size of 12 to hold the correct pair pthread number for each pthread.