110770716

Darren Kong

AMS326

Homework 1

**Problem 2.1**

The naive\_matrix\_multiply(a,b) takes in two matrices, a and b. It multiplies them using the naïve method with a triple nested for-loop. The time complexity is . In the deepest for-loop, there is only one multiplication and one addition, therefore, there are multiplications and additions.

The Strassen Matrix code is made out of many parts. Matrix\_add(a, b) takes additions to add two matrices, a and b. The value of n is always the length of a row in a square matrix. Matrix\_subtract(a, b) takes additions to add two matrices, a and b. matrix\_padding(a, newRow, newCol) appends 0’s to the matrix a so that is has a row length of newRow and column length of newCol. Strassen\_matrix\_split(a) take a matrix a and returns the four equal sized sub-matrices that make up matrix a. I split the actual recursion part of Strassen’s Algorithm into two parts. Strassen\_matrix\_multiply\_helper(a, b) takes two matrices and checks if they are square matrices with a power of two. If it isn’t, it would pad the matrices to the nearest greater power of two. Then it calls Strassen\_matrix\_multiply().

Strassen\_matrix\_multiply(a, b) is the meat of the code. It calls the relevant sub-routines to split the matrices into four equal sized parts. Then the magic of the code happens with m1-m7 and c1-c4. It recursively calls itself after adding and subtracting the relevant matrices. However, with each subsequent recursive call the size of the matrices begins decreasing by 1. Basically, the size of the matrix becomes . Once the recursion has gotten itself a matrix of size 2x2, it simply does a naïve multiplication. There are a total of 18 matrix additions and subtractions together in one call. But since it is a constant, we don’t care. The principal idea is that Strassen’s algorithm does only 7 multiplications of sub-blocks instead of 8 in the naïve method. Therefore and the complexity is . To get an accurate number, I had taken a global variable for additons and multiplications for each method.

The runtimes in the python code were unacceptable, in excess of ten minutes. I had rewritten the code in C to achieve better runtimes. The resultant code is very much similar. A matrix of size 4096x4096 takes about five minutes to compute using Strassen’s Algorithm in C rather than a full day in Python. The cause of this seems to be some latency in memory management or bandwidth.

**Problem 2.2**

The function infectionsperday(mu, sigma, initial, days, pool) takes in five parameters. Mu, sigma, and pool is used to create the pool of values from the normal distribution. Then, in a for-loop, a list of infections per day was produced using the initial parameter and the amount of days. The function returned this list. There were two lists created, one for the first 45 days and the other for the next 45 days based upon the different parameters given. I had then concatenated these two lists together and plotted it using matplotlib library in python. The graph below is from a run of the program. The graph is not going to be the same for all runs since it is based on randomly choosing from a pool of values from the normal distribution. The overall look of the graphs should be very similar though.

