Steven Dea
Algorithms for Bioinformatics Spring 2020

# Programming Assignment 1 Analysis

In my work on this project, I re-learned quite a bit in regards to file reading and parsing input using the Scanner class. I have not spent a lot of time in the past year and a half programming in Java, most of what I've worked on has been in either Python or C++ so it was definitely a learning experience trying to apply my knowledge to Java-specific syntax. I also spent quite a bit of time trying to understand how Strassen's algorithm actually works. I felt that the textbook did not explain it too well, and so I ended up watching quite a few Youtube videos in an attempt to understand it. I tried to follow the suggestions in the assignment by writing my logic and thought process down prior to actual programming and that ended up helping me a lot in the beginning phases of writing actual code. I spent a bit of time solving how to read the input file and do the naïve matrix multiplication method on paper, but I did not spend too much time working on my logic for my implementation of Strassen's Algorithm. I would definitely go back and spend a good amount of time working out my logic for Strassen's Algorithm on paper before going to code. I think that as I worked on the project, my understanding of the problem grew and as such, it would have been a good idea to go back to the drawing board once my knowledge of the problem changed.

My design for the naïve matrix multiplication method was to keep things efficient in both time and memory. As such, I thought about applying a similar logic to Strassen's Algorithm as the naïve matrix in using a recursive function to solve for matrix multiplication, but I decided against it because it did not seem to have any particular advantages over a simpler, more memory efficient iterative approach of using three nested for loops. My design for Strassen's Algorithm makes use of recursive calls to partition the input matrices down until they reach 2x2 matrices, which are then easy to apply Strassen's Algorithm to. One part of implementing Strassen's Algorithm for me was understanding that the formulas for calculating p1-p7 could be applied to matrices that were larger than just 2x2. The portion of the code that took me the longest to figure out was the recursive calls to solve each of the p1-p7 matrices by substituting the smaller problems of the sub-matrices into the formulas. I further added, subtraction, and combine methods to make it easy to solve for the p1-p7 matrices and the resulting combined matrix values. Comparing the amount of multiplications necessary for the naïve matrix multiplication versus my implementation of Strassen's Algorithm shows the correct difference of $O(n^3)$ vs $O(n^{\log_2 7})$. For a matrix of 8x8, my naïve method takes 512 multiplications where my implementation of Strassen's Algorithm takes 343 multiplications. One thing that is really important to note regarding the efficiencies of these two algorithms is that while Strassen's Algorithm takes significantly less time the larger the input matrices become, it also contains a much higher overhead in regards to space costs due to its nature as a recursive function.

## Applications for Bioinformatics

A possible application for Strassen's Algorithm in the field of Bioinformatics is that of matrix multiplication in regards to mutation matrices. Mutation matrices such as various PAM matrices work by multiplying themselves to measure evolutionary distance measured in PAMs. For example, a matrix of PAM1 is made up of amino acids that are on average, 1% different from the starting matrix and is considered in a distance of one PAM. As these matrices are multiplied together, we can figure out how far our input matrices are from each other by measuring them in PAM matrices. For very large inputs, this would be a very useful application of Strassen's Algorithm to cut down on computation time. A further application of Strassen's Algorithm in Bioinformatics could be for calculating Markov chains. From the first example of its application in Bioinformatics in calculating PAM matrices, we must also understand that amino acid mutations are not just completely random and must take into account the previous amino acids, the likelihood of an amino acid mutating to another particular amino acid, etc. This is where the Markov chains come into play. Markov chains can be calculated by multiplying matrices of probabilities and creating transition matrices where we can more accurately predict when an amino acid would mutating into another.