Jonathan Nguyen - jnguye36
Analysis

This lab covers matrix multiplication, and a comparison between the standard algorithm of multiplying out each member of a column against the corresponding row in the secondary matrix, versus Strassen's algorithm which uses a recursive divide and conquer strategy.

Generally the main data structures used were 2D arrays meant to act as matrices. Arrays are standard in most programming languages, and my unfamiliarity with Java led me to use it as a basic data structure. Alternatively using an ArrayList could also be done, but did not seem to provide any huge benefits for the purpose of this assignment. Likewise the nature of matrices lend itself to choosing arrays versus other potential implementations.

In order to implement Strassen's algorithms, I generated many new 2D arrays. Each recursive call would require new allocation of space, but splitting up all terms rather than reusing empty 2D arrays as available gives larger benefits to readability. Operations could have also been condensed and broken down into single statements, but readability is also lost there.

In terms of speed, Strassen's algorithm is faster or equal to the speed of the regular multiplication method. However, at very large N, Strassen's algorithm begins to greatly outperform, which can already be seen at n = 3, size 8 matrices. Normal multiplication requires 512 actions compared to Strassen's 400, roughly 25% in difference. However, space is greatly saved in the normal multiplication algorithm. Given the input space of 2 arrays, only a third 2D array is required to hold the output. On the first recursive call, Strassen's requires at least 17 other 2D arrays, but array size will eventually become smaller upon each recursive call. Even in the smaller case of n = 2, Strassen's requires more space allocated.

I have a better understanding of both Strassen's algorithm and how divide and conquer works now. Strassen's algorithm isn't very intuitive to me, but ultimately breaking it down into the simple terms for recursion made it not as relevant. For example, separating the algorithm out into a base case, recursive case, and a processing step to proceed with the recursive case made the implementation easier to handle.

Next time I'd like to plan a bit more and try to the understand the algorithm more before attempting implementation. I found myself adding in functions like the add and subtract function for the matrices because I didn't understand how Strassen's functioned. This also added some unneeded work where I had to rewrite some functions in order to better fit with the recursive model, to take only matrices as inputs. On top of that, planning ahead of time would have allowed me to better modularize my code.

In terms of bioinformatics, I can see matrices being used for scoring purposes or for setting up alignment scores. It might also be useful for generating motifs and patterns in DNA sequences where you could search for similar patterns.