605.420 Foundations of Algorithms

Anam Ahsan

Project 1 Analysis

Due Date: September 27th, 2022

Dated Turned In: September 28th, 2022

Description and justification of your data structures, implementation choices, and design decisions

The main data structure used throughout is an array as it is very malleable in terms of space, which is useful in the Strassen algorithm due to the many partitions it requires. Furthermore, arrays allow for random access which as we attempt to add, subtract, and recombine values, is a useful feature.

In order to effectively implement Strassen algorithm without losing track of numbers, many functions were designed to be standalone methods that were called upon rather than coded into one method. The Strassen algorithm, called the 'mainStrass' method, which calls for partition/recombination of arrays ('partition' method), addition ('add' method), and subtraction ('subtract' method). Each of these functions is a method within the same class. Only multiplication is within the mainStrass method. Other methods include the method for the Naïve algorithm, called the 'ordinary' method, the' fillMatrix' method which takes the input and puts it into arrays, and 'printMatrix' method which prints the input and output arrays. The 'main' method simply opens and reads the file and creates the destination file for the output.

Efficiency with respect to both time and space

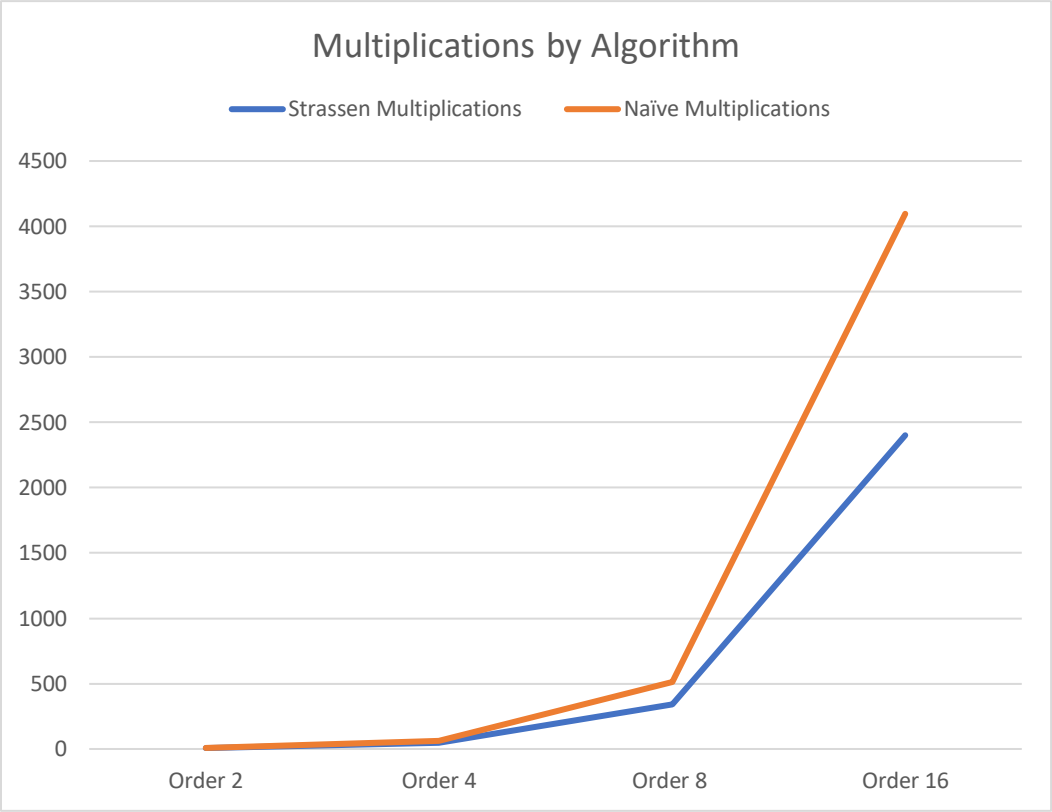
The partitioning required to compute the Strassen algorithm, may seem like it requires more space leading to a higher space complexity, but that is not necessarily the case. The matrices created during the partition are recombined leading to a more dynamic space complexity. In this case the space complexity will be n = size of array, which for this program was n^2 (n x n). This would be subject to change if we used matrices that were not power of two. Likewise, for the naïve method which does all the multiplication within its index location the space complexity is $\Theta(n^2)$.

In terms of efficiency with respect to both time and space. For an array, the average cost of time complexity to search, insert, and delete are $\Theta(n)$ and the cost to access is $\Theta(1)$. The division of the input matrices takes $\Theta(1)$ time, the creation of 10 matrices from the sums and difference takes $\Theta(n^2)$ time. To calculate the 7 matrix products using the divided matrices it also takes $\Theta(n^2)$ time. Together this takes $7T(n/2) + \Theta(n^2)$ when $n > 1$.

In contrast the naive method simply has 3 nested for-loops which calculate the new matrix and store them in fixed indices. The time complexity for this method is a $O(n^3)$.

A table and a graph summarizing your observations with respect to asymptotic efficiency.

	Order 2	**Order 4**	**Order 8**	**Order 16**
Strassen Multiplications	**7**	**49**	**343**	**2401**
Naïve Multiplications	**8**	**64**	**512**	**4096**



A comparison of the theoretical efficiency to the observed efficiency

The theoretical and observed efficiency are comparable, but only once the order becomes larger. When the order is less than eight the efficiency is nearly negligible, but as we get to the higher orders, especially 16, the difference in number of multiplications is significant. The naïve algorithm requires nearly double, going on triple, the multiplications that are required by the Strassen algorithm. Therefore, at least for the sake of time, when the size of the matrix is great, the Strassen algorithm may be the more efficient choice.

What you learned / What you might do differently next time

During this project I learned the scope of recursions and what reducing just one recursive multiplication can do for processing time. Initially, when doing the reading, it seemed difficult to grasp how just one less recursive multiplication would be worthwhile but seeing it in action was very enlightening. It has broadened the scope of what one can do to make their code efficient, and how unique algorithms can help us solve complex and larger problems.

One thing I would do differently is that I would use different classes so that my main class is not over-crowded with methods. For this project, I would only leave the parsing of the input data and the printing method in the main page. The main code for the Strassen Algorithm would be its own class, the naive algorithm would have its own class, and finally the partition, addition, and subtraction, and methods would be grouped together in a separate class.

Applicability to Bioinformatics

This is applicable to bioinformatics because with the ability to sequence DNA has become more accessible, leading to data sets that are becoming exponentially larger and more complex. To efficiently process such large amount of data we need computations like the Strassen's Algorithm which can handle larger inputs in shorted amount of time. For example, if we're looking at a sequence alignment matrix of codons or nucleotides, there are a number of calculations where the Strassen algorithm would be helpful.