# COMS4032A - Applications of Algorithms

## Assignment 1

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Nihal Ranchod - 2427378

August 6, 2024

## Table of Contents

All C++ scripts created for this assignment can be found at this GitHub Repository

The C++ code for each of the algorithms can be found in the smm_algorithms.cpp file.

## 1 Graphs of Running Times & Empirical Analysis

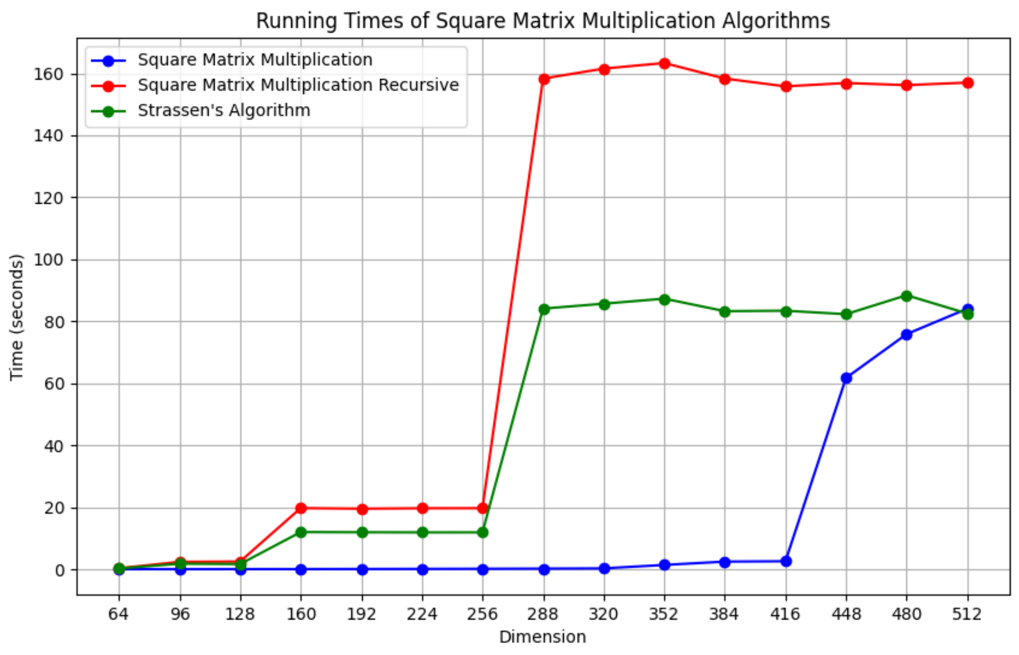

Figure 1: Plot showing the running times of the three different square matrix multiplication algorithms for dimensions `(64x64)` to `(512x512)`.

### 1.1 Square Matrix Multiplication

The square matrix multiplication algorithm has a time complexity of $O(n^3)$. This is shown in the running times shown in Figure 1 as well as CSV file of running times. The running time of this algorithm increases as the matrix dimension increases:

- For dimension 64, the run time is 0.0016426 seconds.
- For dimension 512, the run time is 84.03659 seconds.

The cubic growth is consistent with the $O(n^3)$ complexity.

### 1.2 Square Matrix Multiplication Recursive

The recursive approach has a time complexity of $O(n^3)$. However, the implementation details can lead to significant overhead, especially for larger matrices. Evident in the running times shown in Figure 1:

- For dimension 64, the run time is 0.287641 seconds.
- For dimension 512, the run time is 156.989 seconds.

The recursive approach shows a much higher running time compared to the square matrix multiply method, due to the overhead of recursive calls and additional memory usage.

## 1.3 Strassen's Algorithm

Strassen's algorithm has a time complexity of approximately $O(n^{2.81})$. Achieved by reducing the number of multiplications required as shown by the run times in Figure 1:

- For dimension 64, the run time is 0.222157 seconds.
- For dimension 512, the run time is 82.4961 seconds.

Strassen's algorithm is faster than the recursive method, it is still slower than the square matrix method for smaller matrices due to the overhead of additional operations. However, for larger matrices, the reduced asymptotic complexity becomes more beneficial.

## 1.4 Summary

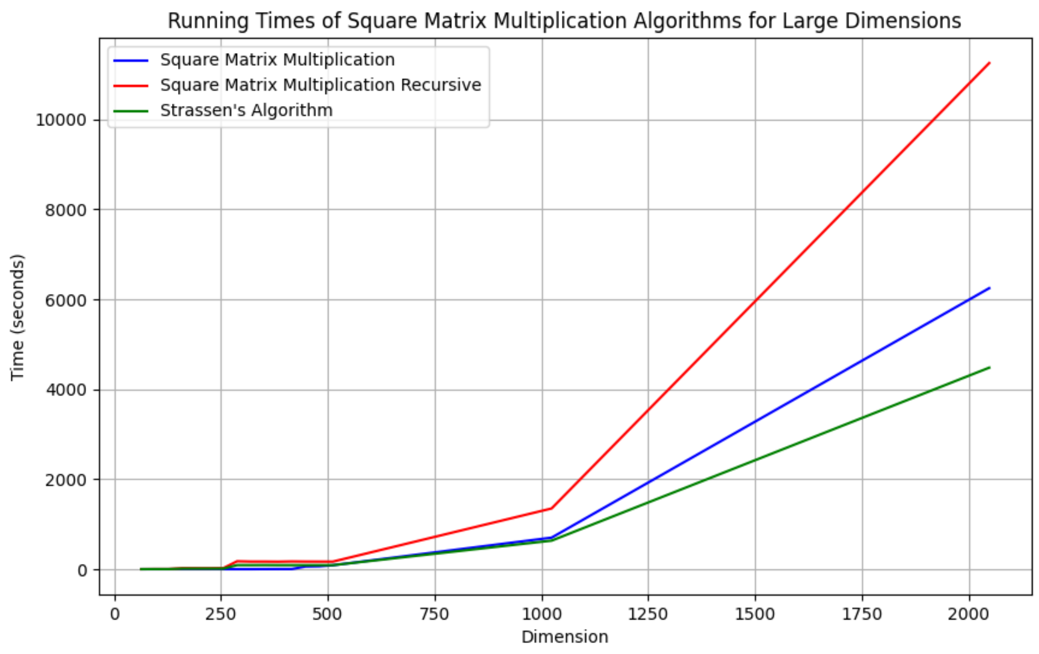

Figure 2: Plot showing the running times of the three different square matrix multiplication algorithms for large dimensions `(2048x2048)`.

- **Square Matrix Multiply**: $O(n^3)$ - Fastest for small matrices because it has no overhead from recursive calls or additional operations, but becomes inefficient for larger matrices. Its implementation allows efficient execution for smaller dimensions.
- **Square Matrix Multiply Recursive**: $O(n^3)$ - Generally slower due to overhead, not practical for large matrices. The recursive calls and additional memory usage introduce significant overhead, making it less efficient.

- **Strassen's Algorithm:** $O(n^{2.81})$ - More efficient for larger matrices, but has overhead that makes it slower for smaller matrices. The overhead from additional operations and padding for non-power-of-two dimensions affects its performance for smaller matrices. Strassen's algorithm offers significant performance improvements due to its lower asymptotic complexity, despite the initial overhead.

## 2 Methodology for Obtaining Graphs

### 2.1 Range of Dimensions

The experiments were conducted on square matrices with dimensions ranging from `(64x64)` to `(512x512)`, Figure 1, and up to `(2048x2048)`, Figure 2. This range was chosen to observe the performance of each algorithm across different scales and to clearly indicate the asymptotic growth of each algorithm as the dimensions increase.

### 2.2 Number of Matrices

For each dimension, three randomly generated matrices were used. The repetition helped in averaging out any anomalies and provided a more accurate measure of the running times.

### 2.3 Types of Matrices

The matrices used in the experiment were randomly generated with integer values. This randomness ensured that the results were not biased by any specific matrix structure.

### 2.4 Padding for Non-Power-of-Two Dimensions

The Recursive approach and Strassen's Algorithm require the matrices to be of dimensions that are powers of two. To accommodate matrices that are not exact powers of two, padding was applied:

- If the original matrix dimension (n) is not a power of two, the matrix is padded to the next power of two dimension.
- Padding involves adding rows and columns with zero values to the original matrix to reach the required dimension.

### 2.5 Timing Measurements

The running times of the algorithms were measured using the `chrono::high_resolution_clock` library in C++. The time taken to multiply the square matrices using each algorithm was recorded, including the padding operation in the case of the Recursive approach and Strassen's algorithm if it was needed. Each timing measurement was repeated three times for each dimension, and the average time was calculated to ensure accuracy.

## 2.6 Correctness Check

To ensure the correctness of the matrix multiplication algorithms, a correctness check was performed using the following steps:

1. **Matrix Generation:** For each dimension, two random matrices (A) and (B) were generated.

2. **Square Matrix Multiplication:** The result of the square matrix multiplication (C1) was computed.

3. **Square Matrix Multiplication Recursive:** The matrices (A) and (B) were padded if necessary, and the result (C2) was computed using the recursive algorithm. The result was then unpadded if padding was applied.

4. **Strassen's Algorithm:** Similarly, the matrices (A) and (B) were padded if necessary, and the result (C3) was computed using Strassen's algorithm. The result was then unpadded if padding was applied.

5. **Comparison:** The results (C1), (C2), and (C3) were printed and visually inspected to ensure they matched.

The correctness check ensures that the implementations of the Recursive and Strassen's algorithms produce the same results as the Square Matrix Multiplication algorithm, validating their correctness.

## 2.7 Important Considerations

- **Padding Overhead:** The padding process introduces some overhead, especially for smaller matrices. This overhead is considered in the analysis.

- **Consistency:** The same set of randomly generated matrices is used for all three algorithms to ensure a fair comparison.

- **Environment:** All experiments were conducted on the same machine to avoid discrepancies due to hardware differences.