**WEEK 1 PROGRAMMING ASSIGNMENT**

**Task 1:**

TASK DESCRIPTION

This task is asking me to estimate the asymptotic complexity of D = ABC using the naïve multiplication function used in lecture (Gongora, J. 2025). The asymptotic complexity of a system can be determined to measure how the runtime scales with the input size, which is very important especially for larger datasets.

I will do this by separating the resultant matrix, D, into two separate matrix multiplications, the first being AB = X, and then this will be multiplied with C to find D.

METHOD USED

I know that for each operation, the columns of the first matrix and the rows of the second matrix for each multiplication must be equal for this approach to be valid, so firstly I will have to ensure that this is true. In this case, as A is a (M x N) matrix and B is a (N x N) matrix, therefore this requirement is met. The resultant matrix therefore consists of (M x N) elements after N multiplications being performed. This part of the multiplication results in an asymptotic complexity of O(MN^2), showing that if the dataset increases with M, the runtime will increase linearly, and if it increases with N, the runtime will increase as a polynomial function.

I can then proceed to repeat this with the second matrix multiplication operation to find the final matrix D, by multiplying X (M x N) and C (N x P). This operation passes the check as the number of columns in X matches the number of rows in C. This should result in a matrix D that scales with data M and P, i.e. it has M x P elements, and each element is a result of N multiplications. The asymptotic complexity for this equation is therefore O(MNP).

RESULTS

This then results in a total asymptotic complexity of O(MN^2 + MNP). This asymptotic complexity shows that for values of M and P, the runtime increases linearly if these data values are changed, but the runtime changes as a polynomial factor with the increase of input value N.

DISCUSSION

One way in which this operation could be varied is to instead multiply BC first, which will result in a different matrix X, and then find the result matrix D, which may result in a reduced upper bound (O notation), and therefore a quicker runtime. This method would conclude with an asymptotic complexity of O(PN^2 +MNP) instead of the previous O(MN^2 +MNP), showing a slighter dependency in input values P rather than M. However, if the data inputs were similar/equal (i.e. M ~ N ~ P), this complexity would evolve into O(N^3), as the O(MN^2) and the O(MNP) dominate if all dimensions scale together. Consequently, the runtime increases cubically with the matrix size iteration. Although this operation is limited in the orders in which it can be performed, either (AB)C or A(BC), using this method of finding complexity would introduce an optimised algorithm that can perform this task faster.

**Task 2:**

TASK DESCRIPTION

The aim of this task is to measure how execution time scales with the size of input data, and to compare it with theoretical complexity:

1. Generate a random matrix of size N x M
2. Use the naïve multiplication method to multiply 2 N x N matrices
3. Calculate the runtime of different multiplications as a result of input data size
4. Plot the runtime as a function the input data to visualise the complexity of the program

This is very important as most programs rely on much larger datasets and perform a large number of operations at a time, so knowing the complexity/runtime of a system will determine the efficiency of the program.

METHOD USED

Part 1:

Numpy was used to generate a random array of integers 1-100 (line 25) used as a matrix (W3Schools, n.d.), and user inputs were stored as rows and columns (lines 10-11), allowing for testing of different matrix sizes. Input validation was included to ensure data was of the correct integer type, and between certain values to make testing more accessible (0-1000, lines 16-23). The function then returns a random matrix, making it easier to implement for later use and saving time and memory

Part 2:

N was set as an arbitrary number in order to test the input (line 35), and using the previously made function to generate a random matrix, I call the function to create matrices A and B (lines 41-45), and then printed to verify generation. Following this, I initialise an array that can act as my matrix for matrix C and fill it with zeroes to act as placeholders for when each element is changed during the naïve multiplication operation (line 49). A for loop is then called, and following the same logic as discussed in the lecture (Gongora, J. 2025), the loop goes through the number of rows in matrix A, the columns in matrix B, and performs multiplications based on the shared dimension, N, changing each element in C as a result of each multiplication operation (GeeksforGeeks, n.d.). The result is then printed to visualise the matrix and confirm correct operation.

Part 3:

The multiplication logic from the previous function was modularised into a reusable function (lines 70-83This function can now take two input matrices and return a resultant matrix, as well as performing some more checks to ensure this operation is possible: now checks for dimensional compatibility (a requirement to be able to perform matrix multiplication), and then if dimensions are not the same, display an error highlighting the incompatible dimensions (lines 75-76). This then allows me to input any matrices to execute a matrix multiplication, highlighting its reusability and efficiency when it comes to time and memory saving.

The task in then continued by setting a starting value for N (line 86), then initialising an empty array to store each iteration of the matrix size N (line 88), along with each calculated runtime. A while loop iterates from 1-99, with each iteration creating 2 new random matrices, and then after these two matrices are created, a timer is started (lines 91-95). It is important in this process to start the timer after the generation of each matrix, as generating these matrices can add to the total time taken for the multiplication if the timer is started before these matrices are generated. Once the timer begins, the naïve multiplication is performed (line 97), then immediately followed by ending the timer (line 99), and calculating the runtime using (end time – start time). This is then stored into the variable “runtime” (line 101), which allows it to then easily be appended to the results array, along with the iteration number N (line 104). The matrix size is then appended by 1 (line 106), and the while loop repeats the process.

Part 4:

I have used matplotlib to visualise the results array to display the runtime and how it scales with the input size (Scaler, n.d.). The x axis is set as the first column in the results array, N, and the y axis as the second column in the results array, the calculated runtime (lines 115-117). This is then plotted on a graph with a set title and labels and then show once executed (lines 122-124).

RESULTS

A graph with a line

AI-generated content may be incorrect.A non-linear trend is followed in the results, describing the polynomial relationship as predicted by the asymptotic complexity of the naïve multiplication:

DISCUSSION

The runtime shows a consistent outcome with the asymptotic complexity of D = ABC, showing that runtime increases rapidly with increasing N. This plot confirms the polynomial scaling of the naïve multiplication approach

The best way of testing and improving this algorithm would be to vary the matrix size N and increase its limit to a much larger value test how the input size affects the runtime of the system, which allow a more accurate comparison to operations performed on larger datasets, which is more common when handling data.

**Task 3:**

TASK DESCRIPTION

The aim of this task is very similar to the previous one, the implementations being almost identical, however instead of it being just one matrix multiplication, this one is now performing two. I have opted to implement this operation as the combination of 2 naïve matrix multiplications to if this has a significant effect on the runtime of the system compared to just 1 operation. I will test it firstly doing (AB)C, and then A(BC), and see which one has a faster runtime.

METHOD USED

Due to having made functions in the previous task in order to perform these operations, such as generating a random matrix and performing naïve multiplication, I can reuse these in this task as the operation is essentially the same. Due to this, the only observable difference is the creation of 3 matrices in the while loop, A, B and C, and then performing an intermediary operation to find X, and then D, which are both included in the runtime calculation. The runtime is then calculated and plotted against the iterated size of the matrix using matplotlib

RESULTS

A graph with a line

AI-generated content may be incorrect.

Figure 1.1: plotted graph of runtime against size of matrix for (AB)C

A graph with a line drawn on it

AI-generated content may be incorrect.

Figure 1.2: plotted graph of runtime against size of matrix for A(BC)

DISCUSSION

The graph for (AB)C (figure 1.1) as plotted by matplotlib originally behaves as intended, when N is small the time taken to perform the operations is also small, effectively instantaneous, and then follows a similar non-linear relationship as N iterates in size. The same can be said for the graph plotted for A(BC) (figure 1.2), showing the same polynomial relationship, with the runtime being almost instantaneous at lower matrix sizes, and then increasing rapidly at a non-linear rate as N increases. Both results are clearly consistent with the predicted asymptotic complexity of O(MN^2 + MNP)

The best way to improve this program in future implementations would be to test it with larger datasets, as well as taking an average of multiple runtime measurements to reduce noise. This would be able to more accurately describe the scaling of runtime compared to the input data size, allowing for a better representation of the complexity of the system.

REFERENCES:

Gongora, J. (2025) *Data, Algorithms and Numerical Optimisation – Week 1 Lecture Slides*. Available on Learn (accessed 3 October 2025).

W3Schools (n.d.) *NumPy Random*. Available at: <https://www.w3schools.com/python/numpy/numpy_random.asp> (Accessed: 8 October 2025).

GeeksforGeeks (n.d.) *Multiplication of two matrices in one line using NumPy in Python*. Available at: <https://www.geeksforgeeks.org/numpy/multiplication-two-matrices-single-line-using-numpy-python> (Accessed: 8 October 2025).

Scaler (n.d.) *Matplotlib 2D plot*. Available at: <https://www.scaler.com/topics/matplotlib/matplotlib-2d-plot> (Accessed: 8 October 2025).