

Name: adham khater

Code: 1170529

Table of Content

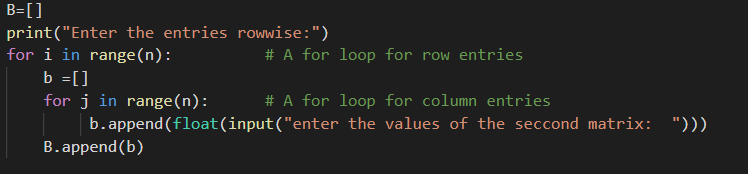
Naïve Algorithm -----------------------------------------1

Divide and conquer --------------------------------------2

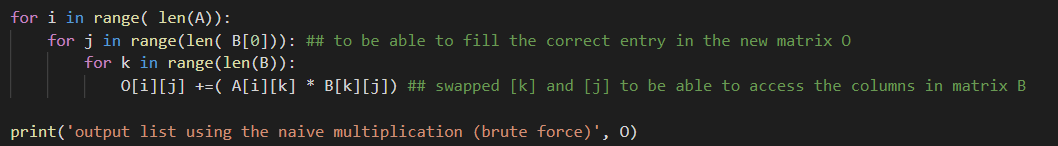
Strassen’s Alg --------------------------------------------4

Naïve Algorithm

It is a very basic way of joining matrices by multiplying them using 3 forloops and the output matrix would have the same size of the 2 input matrices since they all have size NxN. I’m taking the matrices as an input from the user by the following code:



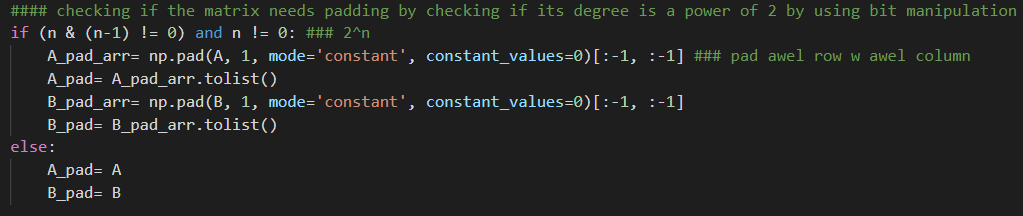
Then I multiply both matrices by the following code:



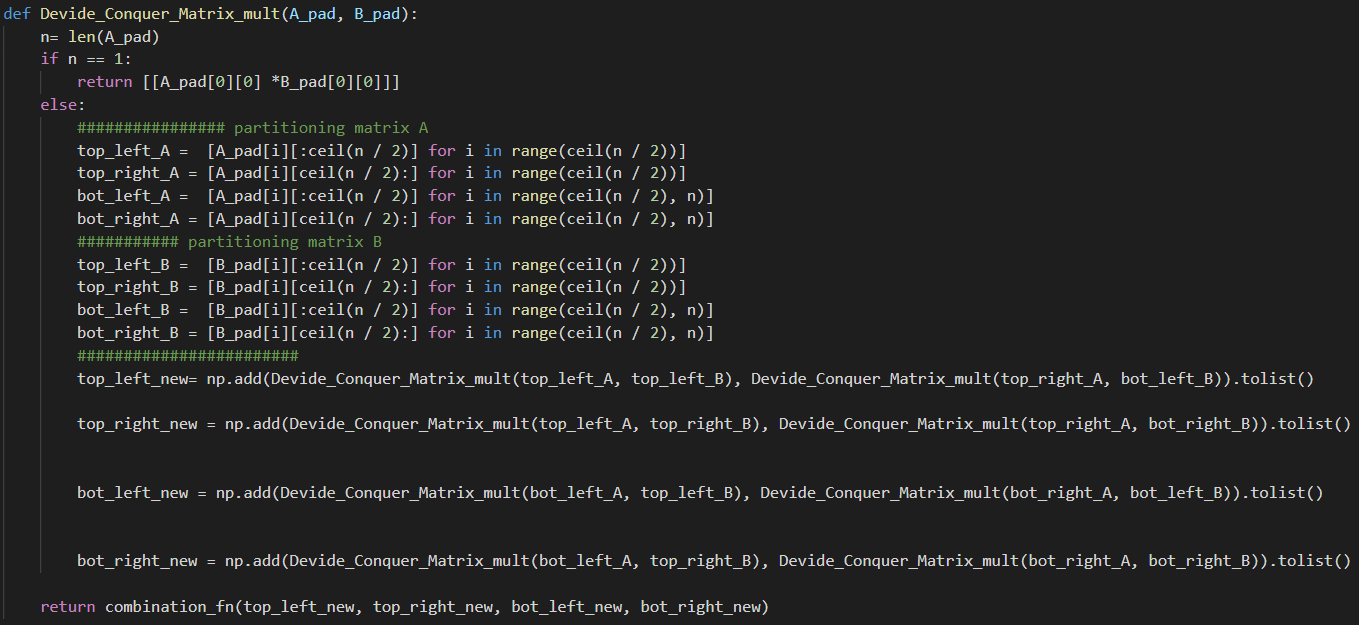
The 2 input matrices has time complexity of N^2, while the third of the matrix multiplication has a time complexity if N^3, so the Big O notation is O^3 since we have 3 nested for loops.

Divide and Conquer

As it was done previously, I’m taking the size of both matrices and their elements from the user as an input then I’m checking if the size of both matrices is a power of 2 if not, padding would be done – I made it on the first row and column. This check up is done by the following code using the bit manipulation. A code snippet is available below:

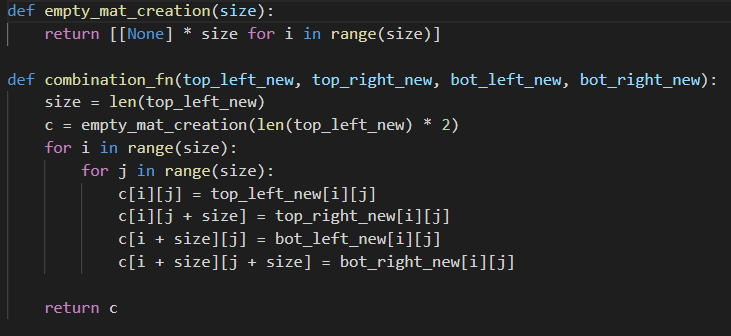


Next the matrices would enter the recursive function where they would be split each into four matrices of size N/2 and would continue to split until the base condition is met which is n, the size of both matrices, equals 1. Code snippet is available below:

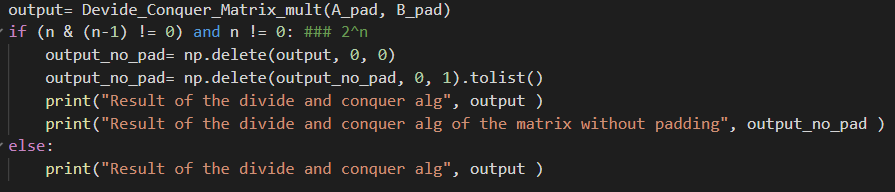


As it is shown my return would be another function, combination\_fn.

This function combines the separate sectors of the multiplied matrices and forms the output matrix as shown in the code below:



Eventually, if the padding was made, I would remove it then display both output matrices, padded and unpadded. If not, I would simply display the output matrix, as shown in the code below:



The time complexity of this code is: 2N^2 for the 2 input matrices.

Partitioning has a time complexity of N. while the recursion call has a time complexity which is 8T(N/2)+O(N^2) which equals O(N^3) using the master theorem. The combination matrix has a time complexity of N^2, so the Big(O) notation equals N^3

Recursion tree:

N

N/2 N/2

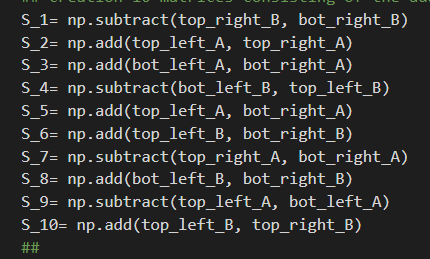
N/4 N/4 N/4 N/4

1 1 1 1

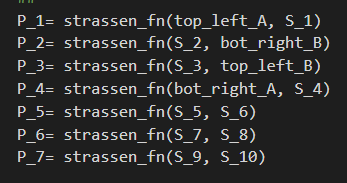
Each time the recursion is called it has 2 matrices with size N/2 and the main call would have 8 childerns. So, 2^log(n)\*cn^2= N^3

Strassen’s Algorithm

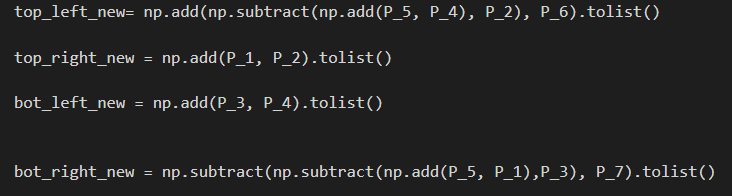
The Strassen uses the same divide and conquer approach. The only difference is that it uses only 7 recursive calls instead of 8. It divides the 2 input matrices into 4 main sectors. It then creates 10 matrices that consist of the addition/ subtraction of these portioned matrices as shown in the code below:



Using these matrices the recursion is called 7 times multiplying these matrices with the previously made partitions as shown in the code below:



Then the new matrix partitions are made by adding and subtracting the product outputs from the previous step :



Then using the same functions in the divide and conquer the matrix would be unpadded, if it was padded, put in an output matrix and then displayed for the user.

The time complexity of the code is:

2N^2 for the input matrices : N^2 +N^2

The partitioning is N

The creation of the new matrix is N

And the recursive calls are 7T(N/2)O(N^2), which also equals N^3

And the output matrix filling which is N^2.

Therefore, the Big(O) notation is N^3

Recursion tree:

N

N/2 N/2

N/4 N/4 N/4 N/4

1 1 1 1

Each time the recursion is called it has 2 matrices with size N/2 and the main call would have 7 children instead of 8.