CSPB 2270 Final Project - Matrix Multiplication Algorithms - Andrew Schiavetta

I did my final project on matrices and matrix multiplication algorithms. I used nested arrays or 2-dimensional arrays to represent a matrix with n number of rows and columns.

I implemented two matrix multiplication algorithms: a basic iterative approach or dot product multiplication and Strassen's matrix multiplication algorithm.

Strassen's approach uses the divide and conquer approach, breaking up a matrix into smaller quadrants. Breaking a larger matrix into a smaller 2x2 matrices and multiplying the smaller matrices requires only 7 multiplication steps instead of 8, but requires more addition and subtraction steps.

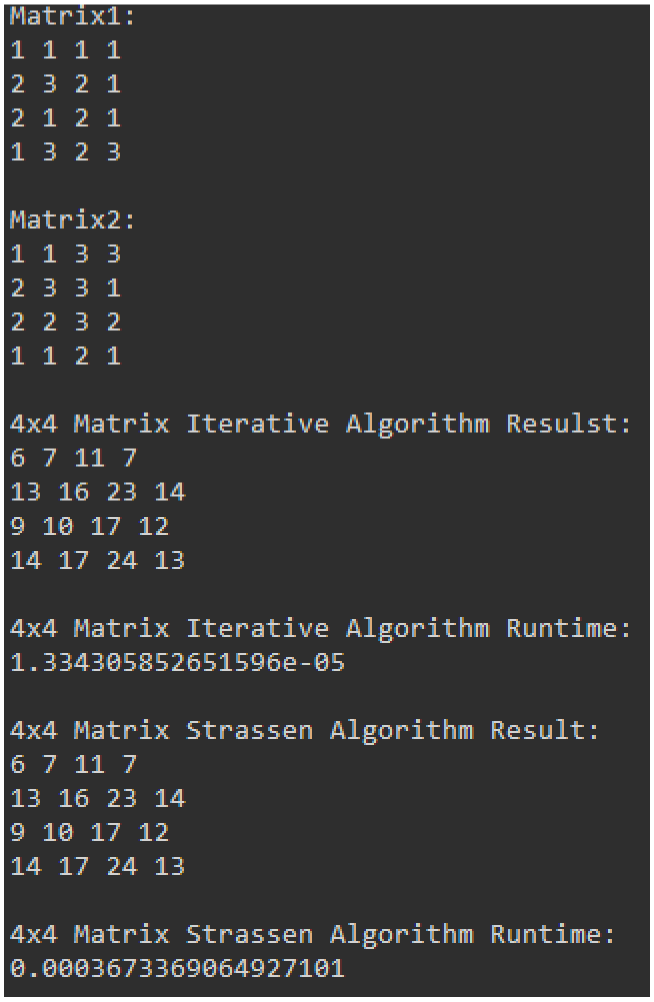
The program will generate two matrices of a given size using the create_matrix function. It will then perform the multiplication of them using both algorithms and print the results with the print_matrix function. I’ve used the “timeit” module to time the two algorithms.

In my initial research I had read that Strassen's algorithm is faster that the iterative approach and since creating his algorithm in 1969, there have been several improvements by both Winograd and then Karstad and Schwartz who reduced the number of additions and subtractions by replacing them with basis transformations which are less expensive computationally. Implementations of Strassen's algorithm can be parallelized to improve performance further. AI has even been used to find faster algorithms for matrix multiplication.

After implementing the two algorithms and testing them on various sized matrices I found that was not the case for my implementation and that the iterative approach was faster in all cases up to n = 512 for an nxn matrix. (I didn't test beyond that as Strassen's algorithm was taking exponentially longer (n^2.8 according to the literature.)

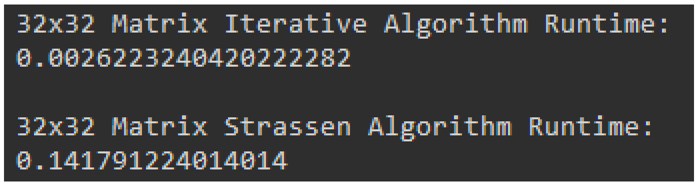
I ran the program in the python terminal by typing: python3 ./CSPB2270_Final_Project.py

My results from running the program are below:

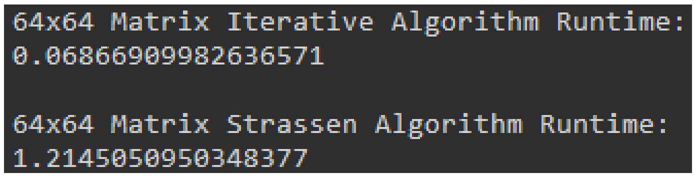


```
Matrix1:
1 1 1 1
2 3 2 1
2 1 2 1
1 3 2 3

Matrix2:
1 1 3 3
2 3 3 1
2 2 3 2
1 1 2 1

4x4 Matrix Iterative Algorithm Resulst:
6 7 11 7
13 16 23 14
9 10 17 12
14 17 24 13

4x4 Matrix Iterative Algorithm Runtime:
1.334305852651596e-05

4x4 Matrix Strassen Algorithm Result:
6 7 11 7
13 16 23 14
9 10 17 12
14 17 24 13

4x4 Matrix Strassen Algorithm Runtime:
0.0003673369064927101
```

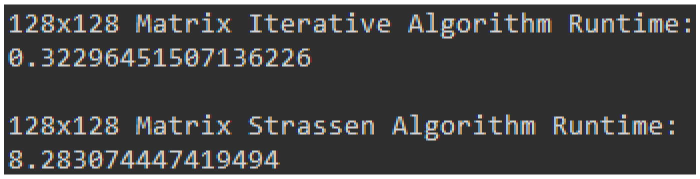
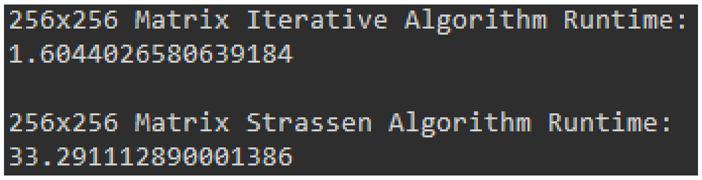
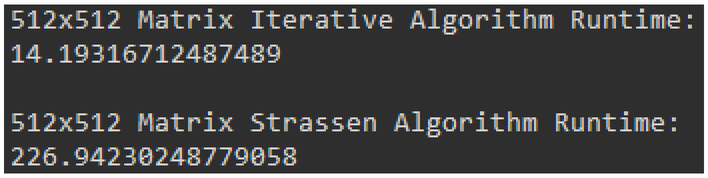
My results from running the program are below:



```
32x32 Matrix Iterative Algorithm Runtime:
0.0026223240420222282

32x32 Matrix Strassen Algorithm Runtime:
0.141791224014014
```



```
64x64 Matrix Iterative Algorithm Runtime:
0.06866909982636571

64x64 Matrix Strassen Algorithm Runtime:
1.2145050950348377
```


```
128x128 Matrix Iterative Algorithm Runtime:
0.32296451507136226

128x128 Matrix Strassen Algorithm Runtime:
8.283074447419494
```

```
256x256 Matrix Iterative Algorithm Runtime:
1.6044026580639184

256x256 Matrix Strassen Algorithm Runtime:
33.291112890001386
```

```
512x512 Matrix Iterative Algorithm Runtime:
14.19316712487489

512x512 Matrix Strassen Algorithm Runtime:
226.94230248779058
```

I've looked into why this might be the case and have found a couple of possibilities. One is that I haven't tested big enough matrices to see the difference yet, but I've also seen that Strassen should begin to be faster around n = 500 to n = 1,000. Doing some quick math, the chart below shows how many times longer it takes to complete the next calculation. It's not clear if the iterative approach was growing at a faster rate.

Increase in matrix size	Iterative Algorithm	Strassen's Algorithm
32 vs 64	34.0x	8.6x
64 vs 128	5.3x	6.8x
128 vs 256	5.3x	4.0x
256 vs 512	8.8x	6.8x

Alternatively it could have to do with how the iterative algorithm is accessing matrices in memory. The iterative approach could already be cache optimized and the Strassen algorithm might not be. It could be that the Strassen algorithm is not memory optimized and each recursive call is creating a copy of the new array. I would think that the recursive nature of the Strassen algorithm would give it an advantage since all variables would be accessible on the stack.

I also wonder if using C++ instead of Python would have made a difference. If I had used pointers in the Strassen implementation perhaps it would have run faster. I chose Python since

I’m new to programming and it was more beginner friendly in the first intro course, but after this past semester I really appreciate the use of static typing in C++.

I’ve definitely gained an appreciation for the importance of algorithms and data structures and how they can significantly improve computation times when the scale of data massively increases.

Sources:
https://en.wikipedia.org/wiki/Strassen_algorithm
https://www.geeksforgeeks.org/python-matrix-product/?ref=lbp
https://www.geeksforgeeks.org/strassens-matrix-multiplication/
https://github.com/hardianlawi/algorithm-implementation/blob/master/src/multiplication/strassenAlgorithm.py