# Assignment 3: Matrix Multiplication

## Learning Outcomes

* Learn a divide and conquer approach to a difficult problem.

## Motivation

Matrix Multiplication is a difficult problem and one that is classicly solved in time. Over the years, efforts have been made to get this time lower.

To multipliy two matrices, order matters. The width of the first matrix must have a width that is equal to the height of the second matrix. The size of the resulting matrix will be the width of the second matrix by the height of the first matrix. In the resulting matrix at any location (i,j), the value of that location is found by taking the dot product of the i-th row in the first matrix and the j-th column of the second matrix. This classical approach to matrix multiplication takes time and was believed to be unbeatable.

In 1969, the Strassen method was developed that reduces the number of multiplications necessary to compute the product of two matrices. One requirement in this method is that both matrices must be the same dimensions and those dimensions must be -by-. This means that the two matrices must be square and of size 1 by 1, 4 by 4, 8 by 8, 16 by 16, etc. in size.

Given the rules set forth in the first paragraph, that means a -by- matrix times a -by- matrix will equal a -by- matrix.

## Inputs

This program will accept two matrices, each entered one at a time:

* The first input is the height, an integer, which should be a value.
* The second input is the width, an integer, which should be a value.
* The next values will be read into the matrix in row-column order. These should be of type double. The term "row-column order" means that values are read into the matrix much like words of English on a page: We read from left to right, then top to bottom.

## Outputs

Your program should output the first matrix, the second matrix, and then the product of the two matrices.

## Options

* You can roll your own solution. I found this to be a fun exercise, but given that most of this has nothing to do with the assignment objectives, you are not required to do this.
* Or you can use my prewritten library that does everything except solve the problem. If you use my library, you are not allowed to change it. (If you find a bug, which is always possible, please let me know quickly.)

If you get started on this assignment late, use the library. If you get started early, try to roll your own solution. Once I got all of the bugs worked out in my library, this took about three hours to solve.

The current prewritten code will do this entire assignment using the traditional matrix multiplication method. You will need to rewrite one line of code in driver.cpp to use the fast version which you wrote instead of the traditional version.

## How to multiply matrices using the Strassen method.

Imagine that you have these two starting matrices. We'll call the first one F and the second on S. The product of the two matrices is X. F, S, and the result matrix X should be the same dimensions.

4 4  
8 1 9 6  
9 4 6 3  
5 2 3 3  
7 4 4 6  
  
4 4  
5 8 7 1  
1 4 7 1  
5 9 7 4  
3 3 2 4

In this example, because these are all 4-by-4 matrices. "half" is 2.

## Base case

All recursive algorithms need a base case. In this circumstance, if , call the traditional matrix multiplication and return whatever the result is. Here are the first four lines of my strassen algorithm.

void strassen(int n, const matrix& F, const matrix& S, matrix& X) {  
 if (n == 2) {  
 X = F.timesTraditional(S);  
 return;  
 }

### Step 1.

Divide each matrix into four quadrants, each with one-fourth of the squares in the space. For the first matrix, I named these A, B, C, and D. For the second matrix, I named these AA, BB, CC, DD. You may use whatever naming scheme you like. This will require that you create eight matrices and copy all of the information from each quadrant into the representative matrix.

A B  
C D  
  
AA BB  
CC DD

For example, A will be this:

8 1  
9 4

For example, BB will be this:

7 1  
7 1

There is no easy way to do this except for a bunch of for-loops.

### Step 2.

Strassen's algorithm requires that you declare seven additional matrices, each one-fourth the size of your original matrix. (Thus they will be the same size as one of your quadrants from Step 1).

M1 = (A + D)(AA+DD)  
M2 = (C + D)AA  
M3 = A(BB-DD)  
M4 = D(CC-AA)  
M5 = (A+B)DD  
M6 = (C-A)(AA+BB)  
M7 = (B-D)(CC+DD)

You will notice that each of these matrices involves multiplying two matrices. This is done recursively and the key to the speedup of the algorithm. You'll also notice that you must add and subtract matrices. The code for this called plus and minus is provided for you in the library so that you can focus on the problem at hand.

For an example of a recursive call, here's how I compute M1:

strassen(half, A.plus(D), AA.plus(DD), M1);

In this case, half is whatever the width of the one of the matrices divided by 2.

### Step 3.

Compile the results of each of your M matrices. I named these X1, X2, X3, X4. This quadrant matrices will be of size "half by half".

X1 = M1 + M4 - M5 + M7  
X2 = M3 + M5  
X3 = M2 + M4  
X4 = M1 - M2 + M3 + M6

## Step 4

Build your results matrix X:

X = [ [ X1 X2 ]  
 [ X3 X4 ] ]

There is no easy way to create this X matrix except for a bunch of for-loops. It will be of size by .

## Run the program.

Here's one where I ran the program with these values.

[ [ 8 1 9 6 ]  
 [ 9 4 6 3 ]  
 [ 5 2 3 3 ]  
 [ 7 4 4 6 ] ] 4x4  
  
 [ [ 5 8 7 1 ]  
 [ 1 4 7 1 ]  
 [ 5 9 7 4 ]  
 [ 3 3 2 4 ] ] 4x4  
  
 [ [ 104 167 138 69 ]  
 [ 88 151 139 49 ]  
 [ 51 84 76 31 ]  
 [ 77 126 117 51 ] ] 4x4

Here's an 8 by 8 case.

[ [ 3 3 5 9 3 1 2 2 ]  
 [ 8 6 7 7 4 9 7 9 ]  
 [ 4 9 5 4 6 4 9 2 ]  
 [ 8 4 1 1 7 2 8 2 ]  
 [ 4 3 1 5 3 6 3 8 ]  
 [ 3 8 2 3 1 5 1 9 ]  
 [ 5 9 8 3 5 4 2 3 ]  
 [ 3 3 3 8 1 1 4 8 ] ] 8x8  
  
 [ [ 7 8 8 4 5 2 2 3 ]  
 [ 9 2 4 1 2 1 9 8 ]  
 [ 5 1 7 1 1 8 8 6 ]  
 [ 8 6 4 5 7 9 3 2 ]  
 [ 3 4 3 1 4 6 3 2 ]  
 [ 2 6 5 2 8 7 1 1 ]  
 [ 3 4 5 9 8 5 4 7 ]  
 [ 8 8 7 3 3 9 1 1 ] ] 8x8  
  
 [ [ 178 131 145 94 131 183 120 104 ]  
 [ 324 295 320 192 279 344 205 203 ]  
 [ 235 179 216 151 205 220 201 203 ]  
 [ 170 167 176 131 170 151 120 138 ]  
 [ 194 193 181 111 170 211 93 93 ]  
 [ 215 170 178 84 133 184 124 114 ]  
 [ 233 160 210 92 149 205 194 172 ]  
 [ 208 171 173 121 148 210 109 106 ] ] 8x8

## Note

Your solution must work for all of the provided test cases. In theory, it should work for any two matrices that are by . I generated the test cases from random.org.

This assignment requires that you build several matrices. There are two ways to do that:

* Use the provided set method. If you know the (i,j) coordinate of where you want to place a value, use it.
* Create a vector and push values onto that vector assuming that they will be understood in row-column order, then pass that vector to the matrix constructor.

I used a combination of both of these when creating the assignment.

## Turn it in

Upload a zip file containing the files that use used or wrote yourself to the drop box on D2L:

* Make sure your name, CSCI 4270, and the Programming Assignment Number appear in comments in all of your files.
* Note: NO CREDIT will be given to programming assignments that do not compile.
* Make sure you have compiled and tested your program before handing it in.

8 8

3 3 5 9 3 1 2 2

8 6 7 7 4 9 7 9

4 9 5 4 6 4 9 2

8 4 1 1 7 2 8 2

4 3 1 5 3 6 3 8

3 8 2 3 1 5 1 9

5 9 8 3 5 4 2 3

3 3 3 8 1 1 4 8

8 8

7 8 8 4 5 2 2 3

9 2 4 1 2 1 9 8

5 1 7 1 1 8 8 6

8 6 4 5 7 9 3 2

3 4 3 1 4 6 3 2

2 6 5 2 8 7 1 1

3 4 5 9 8 5 4 7

8 8 7 3 3 9 1 1