This project had a lot of variables. These results are based on my computers capabilities and my choice of code implementation. For this project we had to simply multiply two nxn matrices using three different ways; classical, divide-and-conquer and Strassen’s. The value n was calculated using the formula n = 2^x, where the user chooses the value for x. The minimum value for x was 1, meaning n = 2, and the maximum value was 12, meaning n = 4,096. While testing the limits, I entered 13 and I got a “OutOfMemoryError” error. It seemed like multiplying matrices of over 8,000 was too much for my computer.

I decided to fill matrices A and B randomly with 0’s and 1’s. I wanted the memory to hold actual values instead of just 0’s. I thought this would be a little more accurate since matrices should have non zero values or there would be no need for a computer to calculate the outcome. From my code, you can see that I started timing the classical way after matrices A, B and C where created and matrices A and B where filled. I stopped and printed the results right after I sent matrices A and B to a multiply method and return the matrix C.

For Divide-and-Conquer, I reset matrix C and created the eight new divided matrices that will be used for this method. I am not 100% sure if this really matters because the computer will just override the old value but to not take any chances, I reset matrix C to zeros. I started the timer and filled the eight sub matrices from matrices A and B. I believe this should be counted in the time taking to complete because these eight sub matrices are not given and it is divide- and –conquers decision to tackle the problem this way. After filling in the appropriate eight sub matrices, I simply used my multiply and add methods to calculate the four sub matrices for matrix C. After all of this I stopped and printed the time.

As for Strassen’s, I believed it would be more accurate to delete all values of the eight sub matrices from matrices A and B. Strassen’s method also uses these eight sub matrices and I start my timer before I start filling them in. after these eight matrices are filled in, I make seven new matrices using my multiply, add, and subtract methods just like Strassen’s method demands. After all of this, I used these new seven matrices to calculate the four sub matrices of matrix C and stop and print the timer.

To obtain the most accurate results I ran this program ten times for each value of n. After many hours of program running, I have written the average in the table seen before my graphs. The classical way is O(n^3) because of the three nested for loops that are needed. Divide-and-Conquer method is O(n^3) like the classical usually takes longer than the classical way because there is a few additional steps like dividing the matrices A and B and addition step. Strassen’s method is O(n^(Lg7)) = O(n^2.81) which makes it better than the other two methods but not until you have large matrices do you see improvement in time comparison.

So now to talk about my results. As you can see from my table, when n is in the range of 2 and 32, the classical multiplication is faster than the other two methods. When n is 64 or 128, the divide-and-conquer multiplication is faster than the other two methods. But when n is in the range of 256 and 4,096, the Strassen’s multiplication is the fastest. I took the liberty of making three graphs so detail can be seen and a final graph showing all values. My conclusion is that is you have matrices of size 32x32 and below, the classical way is the what you want. But when you get to larger values of n, Strassen’s method will be better for time purposes.