

Overview:

Strassen’s divide and conquer matrix multiplication algorithm for n by n matrices is asymptotically faster than the conventional $O(n^3)$ algorithm. This means that for sufficiently large values of n , Strassen’s algorithm will run faster than the conventional algorithm. For small values of n , however, the conventional algorithm is faster.

Since Strassen’s algorithm is a recursive algorithm, at some point in the recursion, once the matrices are small enough, we may want to switch from recursively calling Strassen’s algorithm and just do a conventional matrix multiplication. That is, the proper way to do Strassen’s algorithm is to not recurse to a “base case” of a 1 by 1 matrix, but to switch earlier and use conventional matrix multiplication. Let us define the *cross-over point* between the two algorithms to be the value of n for which we want to stop using Strassen’s algorithm and switch to conventional matrix multiplication. The goal of this assignment is to implement the conventional algorithm and Strassen’s algorithm and to determine their cross-over point, both analytically and experimentally. One important factor our simple analysis will not take into account is memory management, which may significantly affect the speed of your implementation.

Tasks:

1. Assume that the cost of any single arithmetic operation (adding, subtracting, multiplying, or dividing two real numbers) is 1, and that all other operations are free. Consider the following variant of Strassen’s algorithm: to multiply two n by n matrices, start using Strassen’s algorithm, but stop the recursion at some size n_0 , and use the conventional algorithm below that point. You have to find a suitable value for n_0 – the cross-over point. Analytically determine the value of n_0 that optimizes the running time of this algorithm in this model. (That is, solve the appropriate equations.) This gives a crude estimate for the cross-over point between Strassen’s algorithm and the standard matrix multiplication algorithm.
2. Implement your variant of Strassen’s algorithm and the standard matrix multiplication algorithm to find the cross-over point experimentally. Experimentally optimize for n_0 and compare the experimental results with your estimate from above. Make both implementations as efficient as possible. The actual cross-over point, which you would like to make as small as possible, will depend on how efficiently you implement Strassen’s algorithm. Your implementation should work for any matrices, not just those whose dimensions are a power of 2.

To test your algorithm, you might try matrices where each entry is randomly selected to be 0 or 1; similarly, you might try matrices where each entry is randomly selected to be 0, 1 or 2, or instead 0, 1, or -1 . You might also try matrices where each entry is a randomly selected real number in the range $[0, 1]$. You need not try all of these, but do test your algorithm adequately.

Code setup:

So that we may test your code ourselves as necessary, please make sure your code runs as follows:

```
$ python strassen.py 0 dimension inputfile
```

The flag 0 is meant to provide you some flexibility; you may use other values for your own testing, debugging, or extensions. The dimension, which we refer to henceforth as d , is the dimension of the matrix you are multiplying, so that 32 means you are multiplying two 32 by 32 matrices together. The inputfile is

an ASCII file with $2d^2$ integer numbers, one per line, representing two matrices A and B ; you are to find the product $AB = C$. The first integer number is matrix entry $a_{1,1}$, followed by $a_{1,2}, a_{1,3}, \dots, a_{1,d}$, where $a_{i,j}$ denotes the entry of A in the i th row and j th column; next comes $a_{2,1}, a_{2,2}$, and so on, for the first d^2 numbers. The next d^2 numbers are similar for matrix B .

Your program should print to standard output a list of the values of the *diagonal entries* $c_{1,1}, c_{2,2}, \dots, c_{d,d}$, one per line, including a trailing newline. We reserve the right to check the output by a script, so add no clutter. (You should not output the whole matrix, although of course all entries should be computed.)

What to hand in:

As before, you may work in pairs, or by yourself. Refer to the Collaboration and Academic Honesty Policies on the course website for rules regarding resources and discussion other than your partner. You should not use any code outside of the Python Standard Library.

Hand in a project report (on paper) describing your analytical and experimental work (for example, carefully describe optimizations you made in your implementations). Be sure to discuss the results you obtain, and try to give explanations for what you observe. How low was your cross-over point? What difficulties arose? What types of matrices did you multiply, and does this choice matter?

Your grade will be based primarily on the correctness of your program, the crossover point you find, your interpretation of the data, and your discussion of the experiment.

Hints:

It is hard to make the conventional algorithm inefficient; however, you may get better caching performance by looping through the variables in the right order (really, try it!). For Strassen's algorithm:

- Avoid copying large blocks of data unnecessarily. This requires some thinking.
- Your implementation of Strassen's algorithm should work even when n is odd! This requires some additional work, and thinking. (One option is to pad with 0's; how can this be done most effectively?) However, you may want to first get it to work when n is a power of 2 – this will get you most of the credit – and then refine it to work for more general values of n .