Strassen's Algorithm (Optional Extension)

Matrix multiplication in the naive implementation requires $O(n^3)$ floating point operations on a matrix with n^2 entries. In fact it seems unlikely one could do better. Each of the n^2 entries in the product requires a dot product of two length n vectors. Yet in 1969, Volken Strassen shocked numerical analysists with his algorithm requiring $O(n^{\log_2 7}) \approx O(n^{2.73})$ floating point operations. The search for a lower bound on the complexity of matrix multiplication continues today, with recent results hovering around $O(n^{2.3})$.

Goal

You will implement Strassen's algorithm using Python and numpy. You will then perform tests to verify the achieved asymptotic running time of Strassen is actually lower than the running time of your previously implemented (non-numpy) matrix multiplication.

Details

Read up on Strassen online (wikipedia has a good article, for example) and implement his algorithm using numpy. **Your algorithm can assume** $n = 2^k$. Making Strassen work otherwise requires really messy memory management best left to C-type languages. Test correctness of your code on several random matrices by comparing your output to numpy ($A@B$). To simplify things, you should work with **integer** matrices only (floating-point introduces rounding errors that we don't want to deal with.)

Once you know it works, do some regression analysis on your original algorithm and Strassen and compare their running times. The easiest way to do this is in Python is with `timeit.timeit`. Here's an example

```
import timeit

i = 4
A = np.random.randint(-10,10,(2**i, 2**i))
timeit.timeit('A@A', number=25, globals = globals())/25
```

The `timeit` function evaluates the first string argument. If the string references values defined elsewhere you need to pass in the current state of the python interpreter which is what the `globals` argument is for. Note we divide by 25 to get an average time per operation. (Running times are usually very centrally distributed so you need not worry about outliers – the mean is a perfectly good measure of center.)

Warning – these algorithms run pretty quickly on $n = 2^6$ or $n = 2^7$ but after that become *very* slow. You will need to proceed judiciously but also gather enough data for a valid conclusion. (In my testing I let my Lenovo laptop run for about 30 minutes).

N.B.: Strassen is recursive and has a base case. Please use the base case $n = 4$ and return `A@B`. This will ensure we have similar results.

Results

You should graph the averaged running times of Strassen and Naive multiplication on the same graph. Then perform an appropriate regression analysis to determine the respective orders of growth to prove Strassen is asymptotically faster. Support your conclusions with calculations and graphs. (See the Gaussian elimination lab for regression operations).

Finally, just for humility's sake, print out a nicely formatted table comparing the above two algorithms with numpy's built in multiply operation. (Don't feel too bad, it's using a compiled C library).