4.2 Strassen's algorithm for matrix multiplication

4.2-1

> Use Strassen's algorithm to compute the matrix product
>
> $$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}.$$
>
> Show your work.

The first matrices are

$$\begin{array}{ll} S_1 = 6 & S_6 = 8 \\ S_2 = 4 & S_7 = -2 \\ S_3 = 12 & S_8 = 6 \\ S_4 = -2 & S_9 = -6 \\ S_5 = 6 & S_{10} = 14. \end{array}$$

The products are

$$\begin{aligned} P_1 & = 1 \cdot 6 = 6 \\ P_2 & = 4 \cdot 2 = 8 \\ P_3 & = 6 \cdot 12 = 72 \\ P_4 & = -2 \cdot 5 = -10 \\ P_5 & = 6 \cdot 8 = 48 \\ P_6 & = -2 \cdot 6 = -12 \\ P_7 & = -6 \cdot 14 = -84. \end{aligned}$$

The four matrices are

$$\begin{aligned} C_{11} & = 48 + (-10) - 8 + (-12) = 18 \\ C_{12} & = 6 + 8 = 14 \\ C_{21} & = 72 + (-10) = 62 \\ C_{22} & = 48 + 6 - 72 - (-84) = 66. \end{aligned}$$

The result is

$$\begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}.$$

4.2-2

> Write pseudocode for Strassen's algorithm.

```
STRASSEN(A, B)
    n = A.rows
    if n == 1
        return a[1, 1] * b[1, 1]
    let C be a new n × n matrix
    A[1, 1] = A[1..n / 2][1..n / 2]
    A[1, 2] = A[1..n / 2][n / 2 + 1..n]
    A[2, 1] = A[n / 2 + 1..n][1..n / 2]
    A[2, 2] = A[n / 2 + 1..n][n / 2 + 1..n]
    B[1, 1] = B[1..n / 2][1..n / 2]
    B[1, 2] = B[1..n / 2][n / 2 + 1..n]
    B[2, 1] = B[n / 2 + 1..n][1..n / 2]
    B[2, 2] = B[n / 2 + 1..n][n / 2 + 1..n]
    S[1] = B[1, 2] - B[2, 2]
    S[2] = A[1, 1] + A[1, 2]
    S[3] = A[2, 1] + A[2, 2]
    S[4] = B[2, 1] - B[1, 1]
    S[5] = A[1, 1] + A[2, 2]
    S[6] = B[1, 1] + B[2, 2]
    S[7] = A[1, 2] - A[2, 2]
    S[8] = B[2, 1] + B[2, 2]
    S[9] = A[1, 1] - A[2, 1]
    S[10] = B[1, 1] + B[1, 2]
    P[1] = STRASSEN(A[1, 1], S[1])
    P[2] = STRASSEN(S[2], B[2, 2])
    P[3] = STRASSEN(S[3], B[1, 1])
    P[4] = STRASSEN(A[2, 2], S[4])
    P[5] = STRASSEN(S[5], S[6])
    P[6] = STRASSEN(S[7], S[8])
    P[7] = STRASSEN(S[9], S[10])
    C[1..n / 2][1..n / 2] = P[5] + P[4] - P[2] + P[6]
    C[1..n / 2][n / 2 + 1..n] = P[1] + P[2]
    C[n / 2 + 1..n][1..n / 2] = P[3] + P[4]
    C[n / 2 + 1..n][n / 2 + 1..n] = P[5] + P[1] - P[3] - P[7]
    return C
```

4.2-3

> How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{\lg 7})$.

We can just extend it to an $n \times n$ matrix and pad it with zeroes. It's obviously $\Theta(n^{\lg 7})$.

4.2-4

> What is the largest k such that if you can multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices is

> time $o(n^{\lg 7})$? What would the running time of this algorithm be?

Assume $n = 3^m$ for some m. Then, using block matrix multiplication, we obtain the recursive running time $T(n) = kT(n/3) + O(1)$.

By master theorem, we can find the largest k to satisfy $\log_3 k < \lg 7$ is $k = 21$.

4.2-5

> V. Pan has discovered a way of multiplying 68×68 matrices using 132464 multiplications, a way of multiplying 70×70 matrices using 143640 multiplications, and a way of multiplying 72×72 matrices using 155424 multiplications. Which method yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm? How does it compare to Strassen's algorithm?

Using what we know from the last exercise, we need to pick the smallest of the following

$$\begin{aligned}\log_{68} 132464 &\approx 2.795128 \\ \log_{70} 143640 &\approx 2.795122 \\ \log_{72} 155424 &\approx 2.795147.\end{aligned}$$

The fastest one asymptotically is 70×70 using 143640.

4.2-6

> How quickly can you multiply a $kn \times n$ matrix by an $n \times kn$ matrix, using Strassen's algorithm as a subroutine? Answer the same question with the order of the input matrices reversed.

- $(kn \times n)(n \times kn)$ produces a $kn \times kn$ matrix. This produces k^2 multiplications of $n \times n$ matrices.
- $(n \times kn)(kn \times n)$ produces an $n \times n$ matrix. This produces k multiplications and $k - 1$ additions.

4.2-7

> Show how to multiply the complex numbers $a + bi$ and $c + di$ using only three multiplications of real numbers. The algorithm should take a, b, c and d as input and produce the real component $ac - bd$ and the imaginary component $ad + bc$ separately.

The three matrices are

$$
\begin{aligned}
A &= (a+b)(c+d) = ac + ad + bc + bd \\
B &= ac \\
C &= bd.
\end{aligned}
$$

The result is

$$(B - C) + (A - B - C)i.$$