Strassen's Matrix Multiplication

Divide and Conquer with a Twist

Prof. Antonio Khalil Moretti

SCIS 313: Data Structures and Algorithm Analysis

Spelman College

Today's Roadmap

Three questions, in order

1. How does normal matrix multiplication work, and why is it $O(n^3)$?
2. Can divide & conquer do better? (Spoiler: not obviously.)
3. What did Strassen do differently — and does it actually help?

Tools we'll use

4. The D&C template: divide, conquer, combine
5. Recurrence relations
6. Master Theorem (Case 3 both times)

Key moral: one fewer recursive call per level changes the exponent.

Normal Matrix Multiplication

The formula

Given $n \times n$ matrices A and B, their product $C = A \cdot B$ is defined by:

$$C[i][j] \;=\; \sum_{k=1}^{n} A[i][k] \cdot B[k][j]$$

For 2×2 matrices, written out explicitly:

$$C_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$C_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$C_{21} = a_{21}b_{11} + a_{22}b_{21}$$
$$C_{22} = a_{21}b_{12} + a_{22}b_{22}$$

Count the multiplications: $1, 2, 3, 4, 5, 6, 7, 8$ — exactly **8**. This number will matter a lot.

Normal Matrix Multiplication

```
// C = A * B, all n x n
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) {
        C[i][j] = 0;
        for (k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];  // <-- innermost
    }
```

Why $O(n^3)$?

- The innermost statement executes once per (i, j, k) triple.
- Each index runs from 0 to $n-1$: that's $n \times n \times n = n^3$ executions.

$$T(n) = \Theta(n^3)$$

This is our baseline. Can divide & conquer beat it?

Applying Divide & Conquer

The key idea: treat sub-matrices as elements

Assume n is a power of 2. (If not, pad with zeros.)

Partition each matrix into four $(n/2) \times (n/2)$ blocks:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Now multiply *as if the blocks are scalars* — the same 2×2 formulas apply:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Each product (e.g. $A_{11}B_{11}$) is a recursive call on an $(n/2) \times (n/2)$ problem.

Applying Divide & Conquer

Base case and small problems

When is a problem "small"?

When $n \leq 2$. At that point we stop recursing and use the four explicit formulas directly — constant time.

Ask yourself before we count calls:

Looking at the four block-product equations above, how many recursive multiplications does one level of the D&C require?

Applying Divide & Conquer

Base case and small problems

When is a problem "small"?

When $n \leq 2$. At that point we stop recursing and use the four explicit formulas directly — constant time.

Ask yourself before we count calls:

Looking at the four block-product equations above, how many recursive multiplications does one level of the D&C require?

Answer

8. Each of the four C blocks needs two sub-products: C_{11} needs $A_{11}B_{11}$ and $A_{12}B_{21}$, and so on. That's $4 \times 2 = 8$ recursive calls.

D&C Recurrence Relation

Break down the cost:

- **Divide:** Split indices to find block boundaries — $O(1)$.
- **Conquer:** 8 recursive multiplications on $(n/2) \times (n/2)$ matrices — $8\,T(n/2)$.
- **Combine:** Add the block results. Each matrix addition touches n^2 entries: $O(n^2)$.

Recurrence:

$$T(1) = c_0 \qquad T(n) = 8\,T(n/2) + O(n^2) \quad \text{for } n > 2$$

Apply Master Theorem with $a = 8$, $b = 2$, $d = 2$:

$$\log_b a = \log_2 8 = 3 \;>\; d = 2 \quad \Rightarrow \quad \textbf{Case 3}$$

$$T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$

Same as the simple method but now with $O(n^2)$ extra stack space. Worse in practice.

The Key Bottleneck

Why multiplications dominate

What drives the exponent?

In $T(n) = 8\,T(n/2) + O(n^2)$, the 8 recursive calls dominate because $\log_2 8 = 3 > 2$. The combine step ($O(n^2)$) is essentially free by comparison.

Strassen's observation (1969):

The core idea

Additions and subtractions on $n \times n$ matrices cost $O(n^2)$. **Multiplications cost** $T(n/2)$ — exponentially more expensive. Therefore: trade multiplications for additions, even if the formulas look messier.

The question: Can we compute all four C_{ij} blocks using **fewer than 8** matrix multiplications?

The Key Bottleneck

Why multiplications dominate

What drives the exponent?

In $T(n) = 8\,T(n/2) + O(n^2)$, the 8 recursive calls dominate because $\log_2 8 = 3 > 2$. The combine step ($O(n^2)$) is essentially free by comparison.

Strassen's observation (1969):

The core idea

Additions and subtractions on $n \times n$ matrices cost $O(n^2)$. **Multiplications cost** $T(n/2)$ — exponentially more expensive. Therefore: trade multiplications for additions, even if the formulas look messier.

The question: Can we compute all four C_{ij} blocks using **fewer than 8** matrix multiplications?

Strassen's 7 Sub-Products

The algebraic trick

Define seven matrix products P_1 through P_7:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \qquad P_5 = (A_{11} + A_{12})\, B_{22}$$
$$P_2 = (A_{21} + A_{22})\, B_{11} \qquad P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$P_3 = A_{11}\,(B_{12} - B_{22}) \qquad P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$P_4 = A_{22}\,(B_{21} - B_{11})$$

Each P_i is **exactly one** recursive multiplication. That's 7 total.

Note: The additions and subtractions inside (e.g. $A_{11} + A_{22}$) each cost $O(n^2)$ — acceptable.

Recovering the Result from P_1–P_7

Only additions and subtractions from here

The four output blocks are:

$$C_{11} = P_1 + P_4 - P_5 + P_7$$
$$C_{12} = P_3 + P_5$$
$$C_{21} = P_2 + P_4$$
$$C_{22} = P_1 + P_3 - P_2 + P_6$$

Verify C_{12} as a sanity check:

$$P_3 + P_5 = A_{11}(B_{12} - B_{22}) + (A_{11} + A_{12})B_{22} = A_{11}B_{12} - A_{11}B_{22} + A_{11}B_{22} + A_{12}B_{22} = A_{11}B_{12} + A_{12}B_{22} \checkmark$$

Key point: No new multiplications anywhere in this step — only $O(n^2)$ additions. The combine step is still $O(n^2)$.

Strassen's Recurrence Relation

Now the cost breakdown changes:

- **Divide:** $O(1)$
- **Conquer:** **7** recursive multiplications on $(n/2) \times (n/2)$ matrices — $7\,T(n/2)$
- **Combine:** additions/subtractions to recover C_{ij} — $O(n^2)$

Recurrence:

$$T(1) = c_0 \qquad T(n) = 7\,T(n/2) + O(n^2) \quad \text{for } n > 2$$

Apply Master Theorem with $a = 7$, $b = 2$, $d = 2$:

$$\log_b a = \log_2 7 \approx 2.807 \;>\; d = 2 \quad \Rightarrow \quad \textbf{Case 3}$$

$$\boxed{T(n) = \Theta\left(n^{\log_2 7}\right) \approx \Theta(n^{2.807})}$$

The Payoff — Comparing All Three Methods

Method	Recursive calls	Recurrence	Complexity
Naive loops	—	—	$\Theta(n^3)$
D&C, 8 calls	8	$8\,T(n/2) + O(n^2)$	$\Theta(n^3)$
Strassen	7	$7\,T(n/2) + O(n^2)$	$\Theta(n^{2.807})$

Key lessons

- D&C with 8 calls gives $\Theta(n^3)$ — same as naive, but with extra stack overhead.
- Dropping to 7 calls changes $\log_2 8 = 3$ to $\log_2 7 \approx 2.807$: a strictly better exponent.
- The trick: more additions, fewer multiplications. Additions are $O(n^2)$; they're irrelevant at the top of Case 3.
- **One fewer recursive call per level beats the entire combine step.**

The Broader Moral

This same idea appears across algorithms

The pattern

When a D&C algorithm falls under Master Theorem Case 3, the *number of recursive calls* a controls the exponent $\log_b a$. Reducing a by even 1 can meaningfully improve the bound.

You have seen this before:

- **Karatsuba multiplication:** $4T(n/2) \to 3T(n/2)$ cuts $O(n^2)$ to $O(n^{1.585})$.
- **Strassen:** $8T(n/2) \to 7T(n/2)$ cuts $O(n^3)$ to $O(n^{2.807})$.

Open question (active research): Can matrix multiplication be done in $O(n^2)$ or close to it? The current record is approximately $O(n^{2.371})$. Strassen (1969) was the first to break $O(n^3)$ — that was a genuinely surprising result.