# Lecture 4: Integer and Matrix Multiplication

More complicated examples of divide-and-conquer

# Integer Arithmetic

Add. Given two $n$-bit integers $a$ and $b$, compute $a + b$.

- $\Theta(n)$ time

Multiply. Given two $n$-bit integers $a$ and $b$, compute $a \cdot b$.

- Primary school method: $\Theta(n^2)$ time.
  - A.k.a. "long multiplication"

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| + | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | × | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | |
| | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | |
| | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | |
| | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | | |
| | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

# Divide-and-Conquer Multiplication: First Attempt

Observation:

```
   0100 1011 × 1101 1010
 = (0100 0000 + 1011) × (1101 0000 + 1010)
 = (0100 × 1101) << 8 +
   (0100 × 1010) << 4 +
   (1011 × 1101) << 4 +
   1011 × 1010
```

In general:

- Let $a = a_1 \ll n/2 + a_0$, and $b = b_1 \ll n/2 + b_0$, where $a_1, a_0, b_1, b_0$ are all $(n/2)$-bit integers.
- We have $ab = a_1 b_1 \ll n + a_1 b_0 \ll n/2 + a_0 b_1 \ll n/2 + a_0 b_0$

# The first divide-and-conquer algorithm for integer multiplication

Suppose the bits are stored in arrays $A[1..n]$ and $B[1..n]$, $A[1]$ and $B[1]$ are the least significant bits

<u>**Multiply**($A, B$)**:**</u>
$n \leftarrow$ **size of** $A$
**if** $n = 1$ **then return** $A[1] \cdot B[1]$
$mid \leftarrow \lfloor n/2 \rfloor$
$U \leftarrow$ **Multiply**($A[mid+1..n], B[mid+1..n]$)
$V \leftarrow$ **Multiply**($A[mid+1..n], B[1..mid]$)
$W \leftarrow$ **Multiply**($A[1..mid], B[mid+1..n]$)
$Z \leftarrow$ **Multiply**($A[1..mid], B[1..mid]$)
$M[1..2n] \leftarrow 0$
$M[1..n] \leftarrow Z$
$M[mid+1..] \leftarrow M[mid+1..] \oplus V \oplus W$
$M[2mid+1..] \leftarrow M[2mid+1..] \oplus U$
**return** $M$

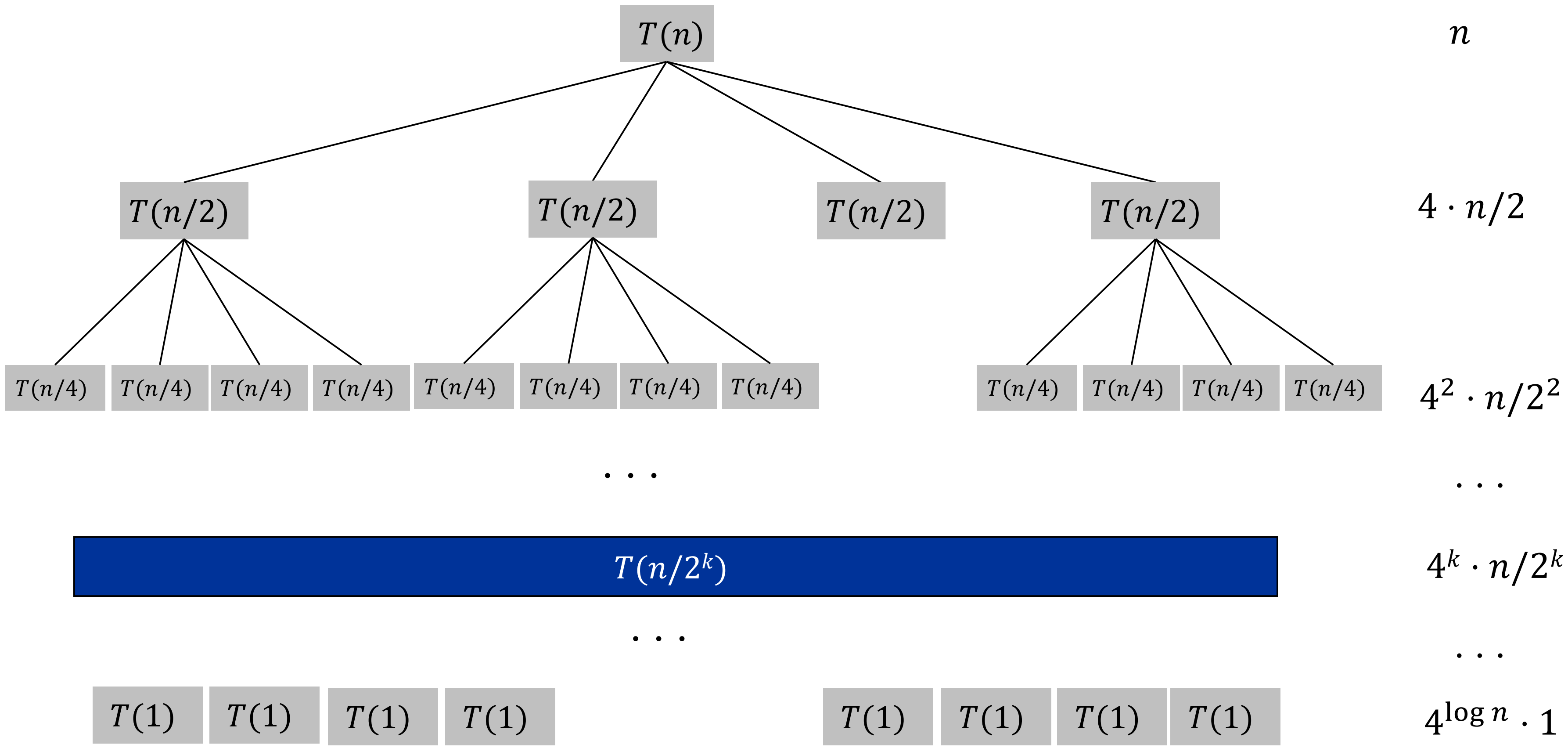
$\oplus$: denotes the integer addition algorithm

# Analysis

## Recurrence:

$$T(n) = 4T(n/2) + n$$

## Solve the recurrence:



# Analysis (continued)

$$n + \left(\frac{4}{2}\right) n + \left(\frac{4}{2}\right)^2 n + \cdots + 4^{\log n} = \Theta\left(4^{\log n}\right) = \Theta\left(n^{\log 4}\right) = \Theta(n^2)$$

The divide-and-conquer algorithm is as bad as the primary school method!

- Essentially, the algorithm still multiplies every bit of $A$ with every bit of $B$.
- Compared with merge sort, the key difference is that one problem generates 4 subproblems of size $n/2$.

# Karatsuba Multiplication

Recall:

- Let $a = a_1 \ll n/2 + a_0$, and $b = b_1 \ll n/2 + b_0$, where $a_1, a_0, b_1, b_0$ are all $(n/2)$-bit integers.
- We have

$$\begin{aligned} ab &= a_1 b_1 \ll n + a_1 b_0 \ll n/2 + a_0 b_1 \ll n/2 + a_0 b_0 \\ &= a_1 b_1 \ll n + (a_1 b_0 + a_0 b_1) \ll n/2 + a_0 b_0 \end{aligned}$$

The trick:

$$a_1 b_0 + a_0 b_1 = (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0$$

This only requires 3 subproblems of size $n/2$!

# Karatsuba's multiplication algorithm

```
Multiply(A, B):
n ← size of A
if n = 1 then return A[1] · B[1]
mid ← ⌊n/2⌋
U ← Multiply(A[mid + 1..n], B[mid + 1..n])
Z ← Multiply(A[1..mid], B[1..mid])
A′ ← A[mid + 1..n] ⊕ A[1..mid]
B′ ← B[mid + 1..n] ⊕ B[1..mid]
Y ← Multiply(A′, B′)
M[1..2n] ← 0
M[1..] ← M[1..n] ⊕ Z
M[mid + 1..] ← M[mid + 1..] ⊕ Y ⊖ U ⊖ Z
M[2mid + 1..] ← M[2mid + 1..] ⊕ U
return M
```

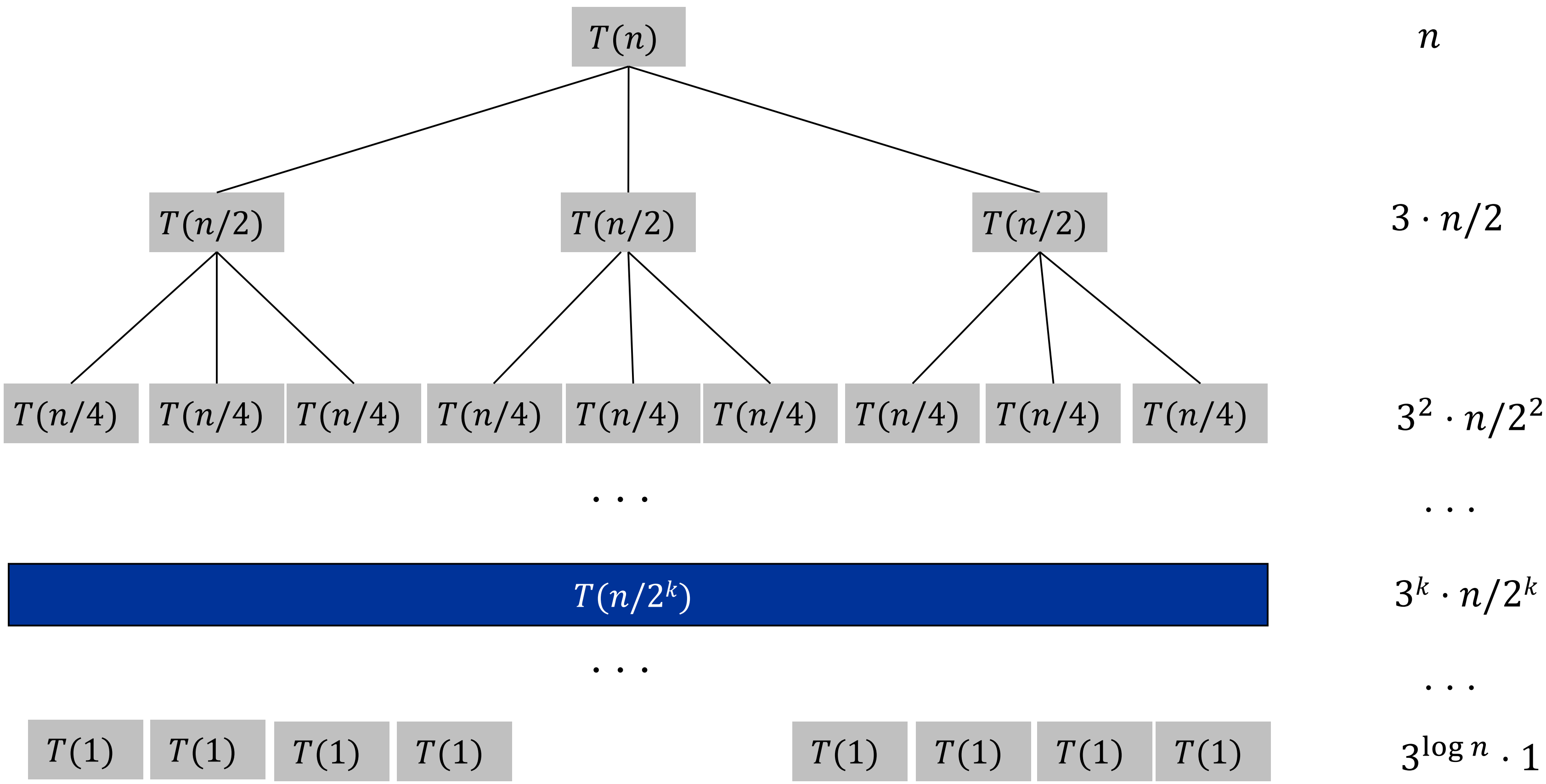
$\oplus$ $\ominus$ : denotes the integer addition/subtraction algorithm

# Analysis

**Recurrence:**

$$T(n) = 3T(n/2) + n$$

**Solve the recurrence:**



# Analysis (continued)

$$n + \left(\frac{3}{2}\right)n + \left(\frac{3}{2}\right)^2 n + \cdots + 3^{\log n} = \Theta\left(3^{\log n}\right) = \Theta\left(n^{\log 3}\right) = \Theta(n^{1.585})$$

Progressive improvements:

- Dividing each integer into $3$ parts, and solve $5$ subproblems
  - $T(n) = 5T(n/3) + n$, $T(n) = \Theta\left(n^{\log_3 5}\right) = \Theta(n^{1.465})$
- Dividing each integer into $4$ parts, and solve $7$ subproblems
  - $T(n) = 7T(n/4) + n$, $T(n) = \Theta\left(n^{\log_4 7}\right) = \Theta(n^{1.404})$
- …
- An $\Theta(n \log n \log\log n)$ algorithm (based on FFT)
- An $\Theta(n \log n \log\log\log n)$ algorithm
- The fastest algorithm runs in time $O(n \log n\, 2^{\log * n})$
- The conjecture is that the problem can be solved in $\Theta(n \log n)$ time. It is still an open problem.

# Integer Multiplication in Practice

## Work on the word level

- Example (using 16-bit words):
  - Decimal: `1316103040073424382`
  - Hexadecimal: `1243 BCBD EF63 5DFE`
  - Stored using an array of 4 words

## In practice:

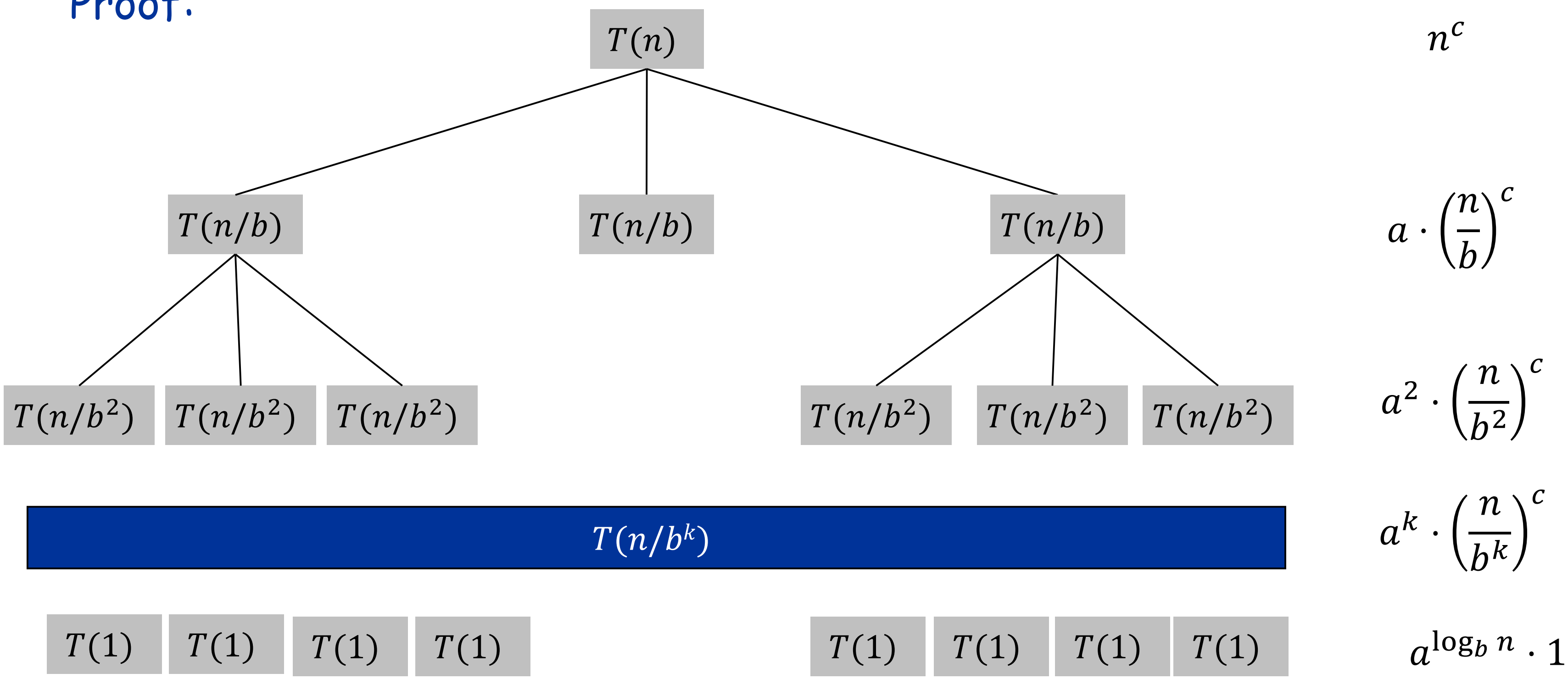
- Long multiplication: Best for < 20 words
- Karatsuba's algorithm: Best for 20 ~ 2000 words
- FFT based algorithm: Best for > 2000 words

# The Master Theorem

**Theorem:** Let $a \geq 1, b > 1, c \geq 0$ be constants. The recurrence $T(n) = aT(n/b) + n^c$ have the following solutions.

- Case 1: $c < \log_b a$: $T(n) = \Theta\left(n^{\log_b a}\right)$.
- Case 2: $c = \log_b a$: $T(n) = \Theta(n^c \log n)$.
- Case 3: $c > \log_b a$: $T(n) = \Theta(n^c)$.

**Proof:**



# Proof of the Master Theorem (continued)

$$n^c + n^c \left(\frac{a}{b^c}\right)^1 + n^c \left(\frac{a}{b^c}\right)^2 + \cdots + a^{\log_b n}$$

If $a > b^c$, i.e., $c < \log_b a$, this is an increasing geometric series, so it is asymptotically bounded by the last term $\Theta\left(a^{\log_b n}\right) = \Theta\left(n^{\log_b a}\right)$.

If $a = b^c$, i.e., $c = \log_b a$, all terms are equal, so the sum is $n^c \cdot \log_b n = \Theta(n^c \log n)$.

If $a < b^c$, i.e., $c > \log_b a$, this is a decreasing geometric series, so it is asymptotically bounded by the first term $\Theta(n^c)$.

# Matrix Multiplication

Matrix multiplication. Given two $n$-by-$n$ matrices $A$ and $B$, compute $C = AB$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Brute force. $\Theta(n^3)$ time.

Fundamental question. Can we improve upon brute force?

# Matrix Multiplication: First Attempt

Divide-and-conquer.

- Divide: partition $A$ and $B$ into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply 8 $\frac{1}{2}n$-by-$\frac{1}{2}n$ submatrices recursively.
- Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

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

$$\mathrm{T}(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{n^2}_{\text{add, form submatrices}} \quad \Rightarrow \mathrm{T}(n) = O(n^3)$$

# Strassen's Matrix Multiplication Algorithm

Key idea. multiply 2-by-2 block matrices with only 7 multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

- 7 multiplications of $\frac{1}{2}n$-by-$\frac{1}{2}n$ submatrices.
- $\Theta(n^2)$ additions and subtractions.
- $T(n) = 7T(n/2) + n^2$, $T(n) = \Theta\left(n^{\log_2 7}\right) = \Theta(n^{2.807})$

In practice: Used to multiply large matrices (e.g., $n > 100$)

# Fast Matrix Multiplication in Theory

Q. Multiply two 2-by-2 matrices with only 7 multiplications?
A. Yes! $\Theta(n^{2.807})$ [Strassen, 1969]

Q. Multiply two 2-by-2 matrices with only 6 multiplications?
A. Impossible.

Q. Two 3-by-3 matrices with only 21 multiplications?
A. Also impossible.

Q. Two 70-by-70 matrices with only 143,640 multiplications?
A. Yes! $\Theta(n^{2.795})$

The competition goes on...

- $\Theta(n^{2.376})$ [Coppersmith-Winograd, 1990.]
- $\Theta(n^{2.374})$ [Stothers, 2010.]
- $\Theta(n^{2.3728642})$ [Williams, 2011.]
- $\Theta(n^{2.3728639})$ [Le Gall, 2014.]
- Conjecture: close to $\Theta(n^2)$