CS 161: Design and Analysis of Algorithms

Divide & Conquer I: Multiplication, Master Method

- Integer multiplication
- Recurrence relations
- Master method
- Matrix Multiplication

Divide & Conquer

- Break problem up into smaller subproblems
- Recursively solve sub problems
- Stitch solutions together, obtaining solution for original problem

Example: Integer Multiplication

- Algorithm 1:
  - Input: two n-digit integers x and y in base b
  - Let k = $\lceil n/2 \rceil$
  - Write x as $b^k x_1 + x_0$, y as $b^k y_1 + y_0$
  - Recursively compute x_1y_1, x_1y_2 , x_2y_1 , x_2y_2
  - Compute $xy = b^{2k} x_1y_1 + b^k (x_1y_2 + x_2y_1) + x_2y_2$

Example: Integer Multiplication

- Running time?
  - Say the running time on n-digit integers is T(n)
  - Computing xy consists of:
    - Four recursive calls of size $\lceil n/2 \rceil$: 4 T($\lceil n/2 \rceil$)
    - Multiplications by $b^{\lceil n/2 \rceil}$ and $b^{2 \lceil n/2 \rceil}$: O(n)
    - Additions of 4 O(n)-digit integers: O(n)
  - Running time: $T(n) = 4\,T(\lceil n/2 \rceil) + O(n)$

Recurrence Relations

- A **recurrence relation** expresses a function T(n) in terms of T with smaller inputs
- To specify function exactly, need to also give base case
- For asymptotic analysis, base cases are not usually necessary

Example Recurrence 1

- $T(n) = T(n-1) + 7$
- Base case? Say $T(0) = 2$
- $T(n) = 7 + T(n-1) = 7 + 7 + T(n-2) = \ldots = 7k + T(n-k)$
- Set $k = n$, $T(n) = 7n + T(0) = 7n+2$
- No matter what $T(0)$ is, $T(n) = O(n)$
- If replace 7 by any constant, $T(n) = O(n)$
- If multiply $T(n-1)$ by 2, $T(n) \neq O(n)$

Example Recurrence 2

- $T(n) = T(n-1) + O(n)$
- There are constants c and n_0 such that $T(n) \leq T(n-1) + c\, n$ for all $n \geq n_0$
- Base case: $T(n_0) = d$ for some d
- $T(n) \leq T(n-1) + c\, n \leq T(n-2) + c\, n + c\,(n-1) = \ldots$

  $\leq d + c\,(n + (n-1) + (n-2) + \ldots + n_0)$

  $\leq d + c\, n^2 = O(n^2)$

Example Recurrence 3

- $T(n) = T(\log n) + O(1)$
- There are constants c and n_0 such that $T(n) \leq T(\log n) + c$ for all $n \geq n_0$
- Base case: $T(n_0) = d$ for some d
- $T(n) \leq T(\log n) + c \leq T(\log \log n) + 2c = \ldots$

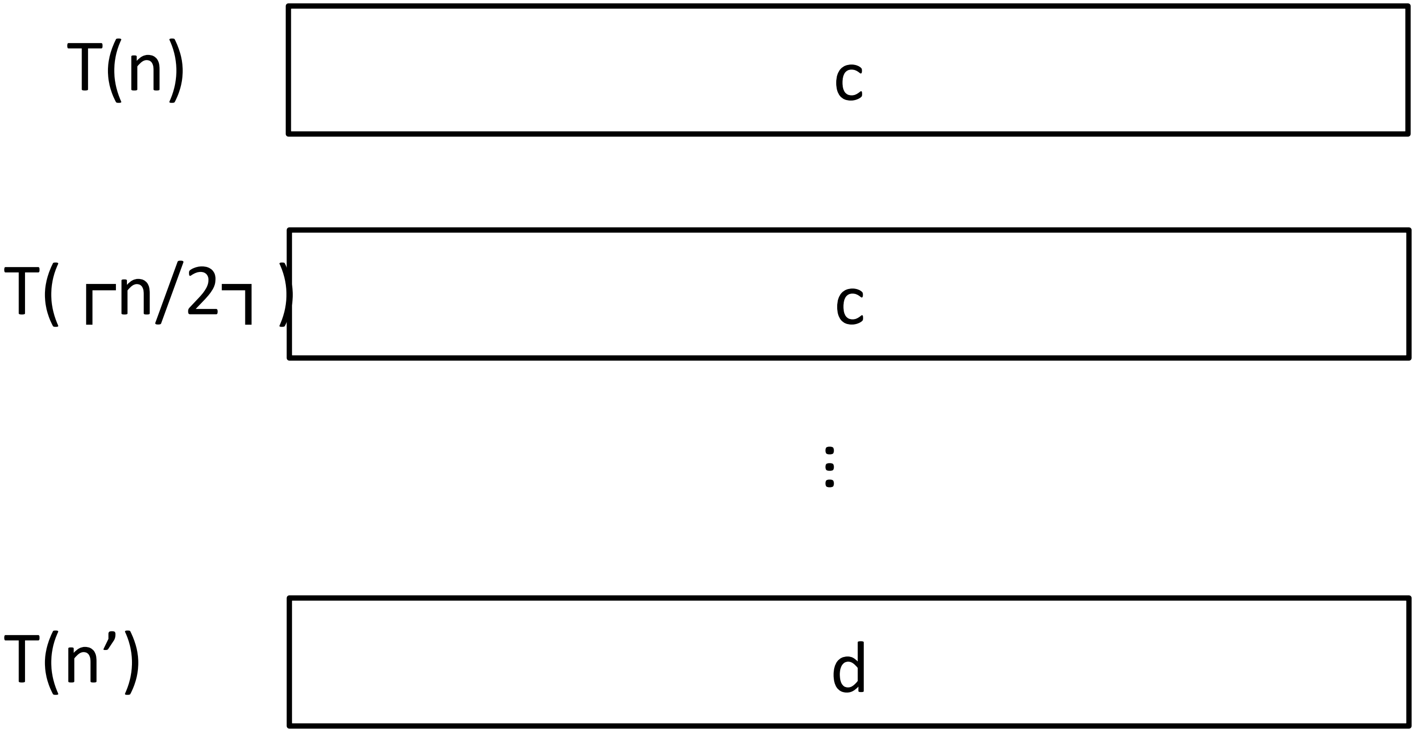
  $\leq d + c \log^* n = O(\log^* n)$

Example Recurrence 4

- $T(n) = T(\lceil n/2 \rceil) + c$
- There are constants c and n_0 such that $T(n) \leq T(\lceil n/2 \rceil) + c$ for all $n \geq n_0$
- Say $T(n) \leq d$ for all $n < n_0$

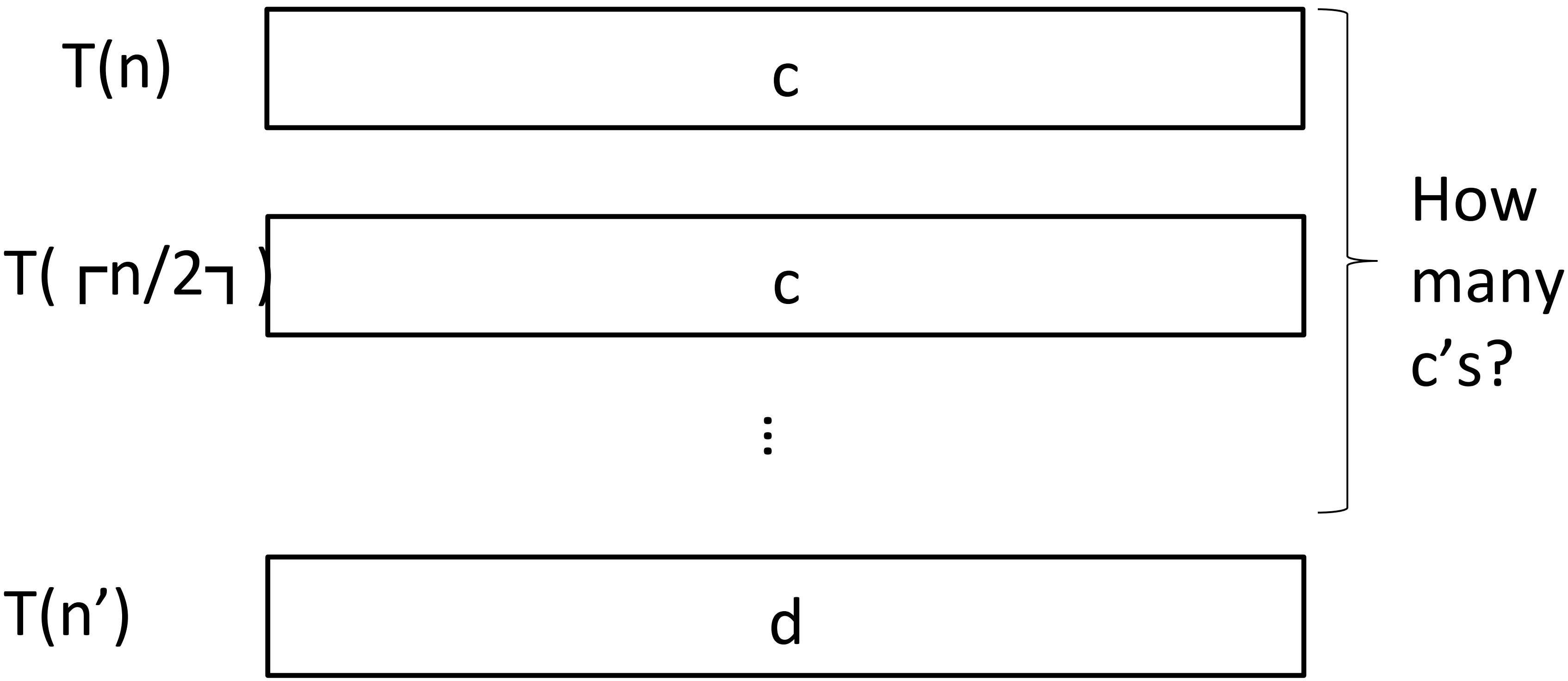
Example Recurrence 4

- $T(n) \leq T(\lceil n/2 \rceil) + c$



Example Recurrence 4

- $T(n) \leq T(\lceil n/2 \rceil) + c$



Example Recurrence 4

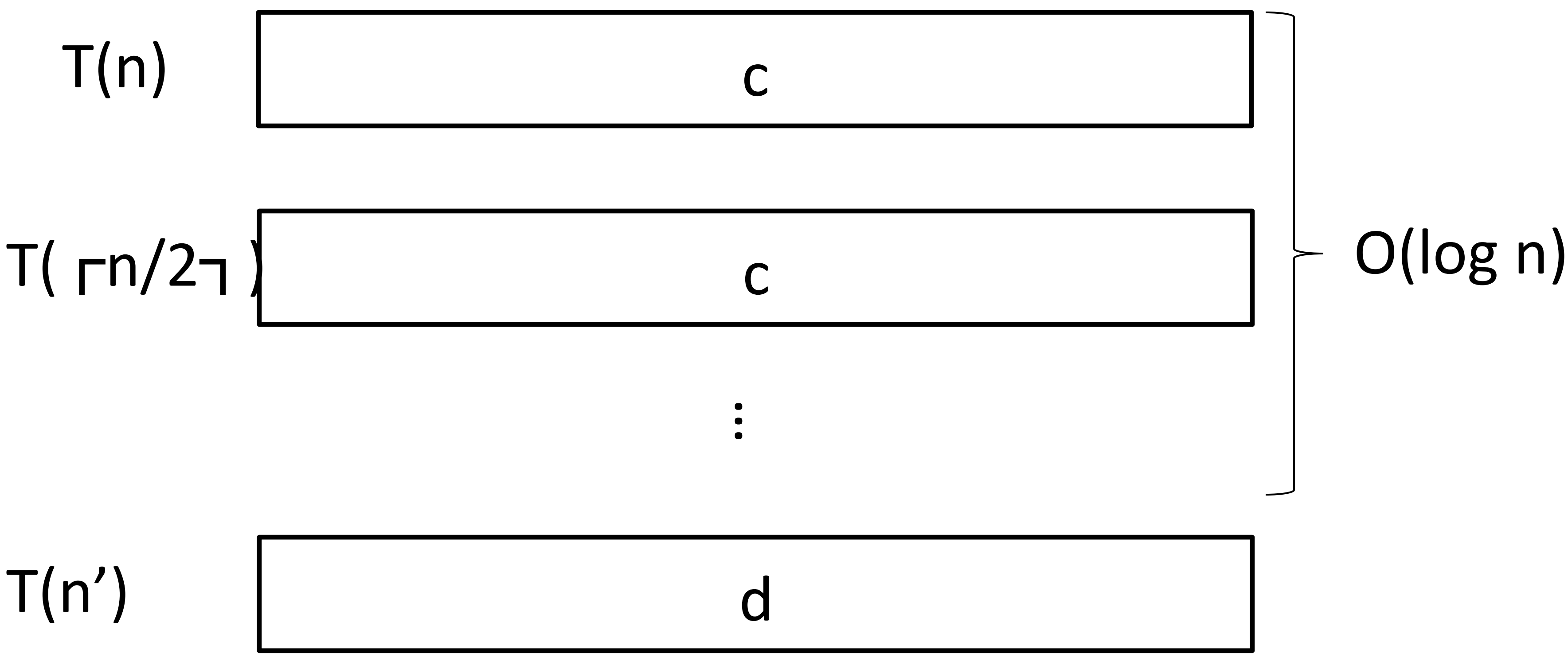
- Theorem: Consider transforming $n \rightarrow n/k + i$. There is a constant b such that the number of transformations needed to get $n < b$ is $O(\log n)$
  - Proof: Let $m = n - ik/(k-1)$
  - If $n' = n/k + i$, then $m' = m/k$
  - Let $b = ik/(k-1) + 1$. Number of times to get $n < b$ is number of times to get $m < 1$
    - $= \log_k m = \log_k (n-ik/(k-1)) = O(\log n)$

Example Recurrence 4

- $T(n) \leq T(\lceil n/2 \rceil) + c \leq T(n/2 + \frac{1}{2}) + c$
- There is a b such that after O(log n) iterations, $n < b$
- $T(n) \leq T(b) + c\, O(\log n) = O(\log n)$

Example Recurrence 4

- $T(n) \leq T(\lceil n/2 \rceil) + c$



Master Method

- Suppose $T(n) = a\,T(n/b + O(1)) + O(n^d)$
- Then:
  - If $a < b^d$, $T(n) = O(n^d)$
  - If $a > b^d$, $T(n) = O(n^{\log_b a})$
  - If $a = b^d$, $T(n) = O(n^d \log n)$

Master Method

- Proof:
  - $T(n) = a\,T(n/b + O(1)) + O(n^d)$
  - $T(n) \leq a\,T(n/b + c) + O(n^d)$ for large enough n
  - Let $m = n – c\,b/(b-1)$
  - $T'(m) = T(n – c\,b/(b-1))$
  - Recurrence: $T'(m) \leq a\,T'(m/b) + O(n^d)$
  - $T'(m) \leq a\,T'(m/b) + fn^d$ for large enough n
  - Suffices to prove theorem for $T'(m)$

Master Method

T’(m) | $f\, m^d$

a T’(n/b) | $f\,(m/b)^d$ | $f\,(m/b)^d$ | $f\,(m/b)^d$

⋮

Master Method

- $T'(m) \leq a\, T'(m/b) + f\, n^d$
- Claim: At level k, amount is at most

$$f \times \left(\frac{a}{b^d}\right)^k \times m^d$$

Master Method

- Total amount:

$$T'(m) \le \sum_{k=0}^{\max k} f \times \left(\frac{a}{b^d} \right)^k \times m^d$$

- What is maximum k? $\log_b m$

Geometric Sums

- Need to analyze sums of the form

$$\sum_{k=0}^{k_{\max}} \alpha^k$$

- Cases:
  - $\alpha = 1$: $1+k_{max}$
  - $\alpha \neq 1$: $$\frac{1-\alpha^{k_{\max}+1}}{1-\alpha}$$

Geometric Sums

$$\frac{1-\alpha^{k_{\max}+1}}{1-\alpha}$$

- If α < 1, $< \frac{1}{1-\alpha} = O(1)$
- If α > 1, $O\left(\alpha^{k_{\max}}\right)$

Master Method

$$T'(m) \leq fm^d \sum_{k=0}^{\log_b m} \left(\frac{a}{b^d}\right)^k$$

- If a < b^d, $fm^d O(1) = O(m^d)$
- If a > b^d, $fm^d O\left(\left(a/b^d\right)^{\log_b m}\right) = O\left(m^d m^{\log_b (a/b^d)}\right) = O\left(m^{\log_b a}\right)$
- If a = b^d, $fm^d O(\log m) = O(m^d \log m)$

Master Method

- Suppose $T(n) = a\,T(n/b + O(1)) + O(n^d)$
- Then:
  - If $a < b^d$, $T(n) = O(n^d)$
  - If $a > b^d$, $T(n) = O(n^{\log_b a})$
  - If $a = b^d$, $T(n) = O(n^d \log n)$

Ω Version

- Suppose $T(n) = a\,T(n/b - \Omega(1)) + \Omega(n^d)$
- Then:
  - If $a < b^d$, $T(n) = \Omega(n^d)$
  - If $a > b^d$, $T(n) = \Omega(n^{\log_b a})$
  - If $a = b^d$, $T(n) = \Omega(n^d \log n)$

Master Method Examples

- $T(n) = T(n/2+6) + O(1)$
  - $a = 1$, $b = 2$, $d = 0$
  - $a = b^d$, so $O(n^d \log n) = O(\log n)$

Master Method Examples

- $T(n) = T(\lceil n/2 \rceil + 1) + O(n)$
  - $a = 1$, $b = 2$, $d = 1$
  - $a < b^d$, so $O(n^d) = O(n)$

Master Method Examples

- $T(n) = 2T(n/2+3\sin(n)) + O(1)$
  - $a = 2$, $b = 2$, $d = 0$
  - $a > b^d$, so $O(n^{\log_b a}) = O(n)$

Master Method Examples

- $T(n) = 2T(\lceil n/2 \rceil + 1/n) + O(n)$
  - $a = 2$, $b = 2$, $d = 1$
  - $a = b^d$, so $O(n^d \log n) = O(n \log n)$

Integer Multiplication

- Algorithm 1: $T(n) = 4\,T(\lceil n/2 \rceil) + O(n)$
- $a = 4$, $b = 2$, $d = 1$
- $a > b^d$, so $T(n) = O(n^{\log_b a}) = O(n^2)$

Integer Multiplication

- Algorithm 2:
  - Input: two n-digit integers x and y in base b
  - Let $k = \lceil n/2 \rceil$
  - Write x as $b^k x_1 + x_0$, y as $b^k y_1 + y_0$
  - Recursively compute x_1y_1, x_2y_2, $(x_1 + x_0)(y_1 + y_0)$
  - Compute $x_1y_2 + x_2y_1 = (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_2y_2$
  - Compute $xy = b^{2k} x_1y_1 + b^k (x_1y_2 + x_2y_1) + x_2y_2$

Integer Multiplication

- Running Time?
  - Let $T(n)$ be the time to multiply n bit integers
  - Computing xy consists of:
    - Some additions and subtractions: $O(n)$
    - 3 calls of size at most $\lceil n/2 \rceil +1$: $3T(\lceil n/2 \rceil +1)$
  - Recurrence: $T(n) = 3T(\lceil n/2 \rceil +1) + O(n)$
- $a = 3$, $b = 2$, $d = 1$
- $a > b^d$, so $T(n) = O(n^{\log_b a}) = O(n^{1.59})$

Integer Multiplication

- Can we do better?
  - Sequence of algorithms achieving $O(n^{\log_k(k+1)})$
  - Best algorithms achieve approximately $O(n \log n)$

Matrix Multiplication

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,p} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,p} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,p} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,p} \end{pmatrix}$$

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

Matrix Facts

- Any m by n matrix can by multiplied by and n by p matrix. The result is an m by p matrix
- Associative: A (BC) = (AB) C
- Not Commutative: AB ≠ BA in general
- Addition is component-wise: If A and B are m be n matrices, then we can add A+B by adding individual components

Block Multiplication

- The $a_{i,j}$ and $b_{i,j}$ can be any values that can be added and multiplied.
  - Can be matricies!
- To compute matrix product AB, can group off elements of A and B into submatrices, and compute product using these smaller matrices

Block Multiplication Example

$$\begin{pmatrix} 7 & 5 & 9 & 8 \\ 2 & 8 & 4 & 6 \\ 4 & 8 & 1 & 2 \\ 9 & 3 & 4 & 1 \end{pmatrix} \times \begin{pmatrix} 6 & 2 & 6 & 1 \\ 8 & 4 & 7 & 2 \\ 4 & 3 & 2 & 9 \\ 1 & 4 & 9 & 8 \end{pmatrix}$$

Block Multiplication Example

$$\begin{pmatrix} \begin{bmatrix} 7 & 5 \\ 2 & 8 \end{bmatrix} & \begin{bmatrix} 9 & 8 \\ 4 & 6 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 9 & 3 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 4 & 1 \end{bmatrix} \end{pmatrix} \times \begin{pmatrix} \begin{bmatrix} 6 & 2 \\ 8 & 4 \end{bmatrix} & \begin{bmatrix} 6 & 1 \\ 7 & 2 \end{bmatrix} \\ \begin{bmatrix} 4 & 3 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 2 & 9 \\ 9 & 8 \end{bmatrix} \end{pmatrix}$$

Block Multiplication Example

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Block Multiplication Example

$$\begin{pmatrix} \begin{bmatrix} 7 & 5 \\ 2 & 8 \end{bmatrix} & \begin{bmatrix} 9 & 8 \\ 4 & 6 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 9 & 3 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 4 & 1 \end{bmatrix} \end{pmatrix} \times \begin{pmatrix} \begin{bmatrix} 6 & 2 \\ 8 & 4 \end{bmatrix} & \begin{bmatrix} 6 & 1 \\ 7 & 2 \end{bmatrix} \\ \begin{bmatrix} 4 & 3 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 2 & 9 \\ 9 & 8 \end{bmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} \begin{bmatrix} 7 & 5 \\ 2 & 8 \end{bmatrix} \times \begin{bmatrix} 6 & 2 \\ 8 & 4 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ 4 & 6 \end{bmatrix} \times \begin{bmatrix} 4 & 3 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 7 & 5 \\ 2 & 8 \end{bmatrix} \times \begin{bmatrix} 6 & 1 \\ 7 & 2 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ 4 & 6 \end{bmatrix} \times \begin{bmatrix} 2 & 9 \\ 9 & 8 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 9 & 3 \end{bmatrix} \times \begin{bmatrix} 6 & 2 \\ 8 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 4 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 3 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 4 & 8 \\ 9 & 3 \end{bmatrix} \times \begin{bmatrix} 6 & 1 \\ 7 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 4 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 9 \\ 9 & 8 \end{bmatrix} \end{pmatrix}$$

Matrix Multiplication

- Recursive Algorithm:
  - Write product as $\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix}$
  - Recursively compute AE, AF, BG, GH, CE, CF, DG, DG
  - Return $\begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$

Matrix Multiplication

- Work outside of recursion: $O(n^2)$ ($d=2$)
- Size of recursive calls: $O(n/2)$ ($b=2$)
- Number of recursive calls: 8 ($a=8$)
- $a > b^d$, so $O(n^{\log 8})=O(n^3)$
  - no better than naïve algorithm

Strassen's Algorithm

- Need AE+BG, AF+BH, CE+DG, CF+DH
  - Can we compute using fewer than 8 multiplications?

$$P_1 = A(F - H) \qquad P_5 = (A + D)(E + H)$$
$$P_2 = (A + B)H \qquad P_6 = (B - D)(G + H)$$
$$P_3 = (C + D)E \qquad P_7 = (A - C)(E + F)$$
$$P_4 = D(G - E)$$

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

Strassen's Algorithm

- 7 recursive calls of size n/2 ($a = 7$, $b = 2$)
- $O(n^2)$ time outside of calls ($d = 2$)
- $a > b^d$, so $O(n^{\log_2 7}) \approx O(n^{2.8074})$
- Better than naïve solution!

Matrix Multiplication

- What is the smallest ω such that matrix multiplication can be done in time essentially $O(n^{\omega})$?
  - Strassen (1969): $\omega < 2.808$
  - Coppersmith, Wenograd (1990): $\omega < 2.376$
  - Stothers (2010): $\omega < 2.3736$
  - Williams (2011): $\omega < 2.3727$
- Widely believed $\omega = 2$