CSE 417 Autumn 2025

Lecture 6: Divide and Conquer

Nathan Brunelle

Homeworks

HW 1 due today at 11:59pm.

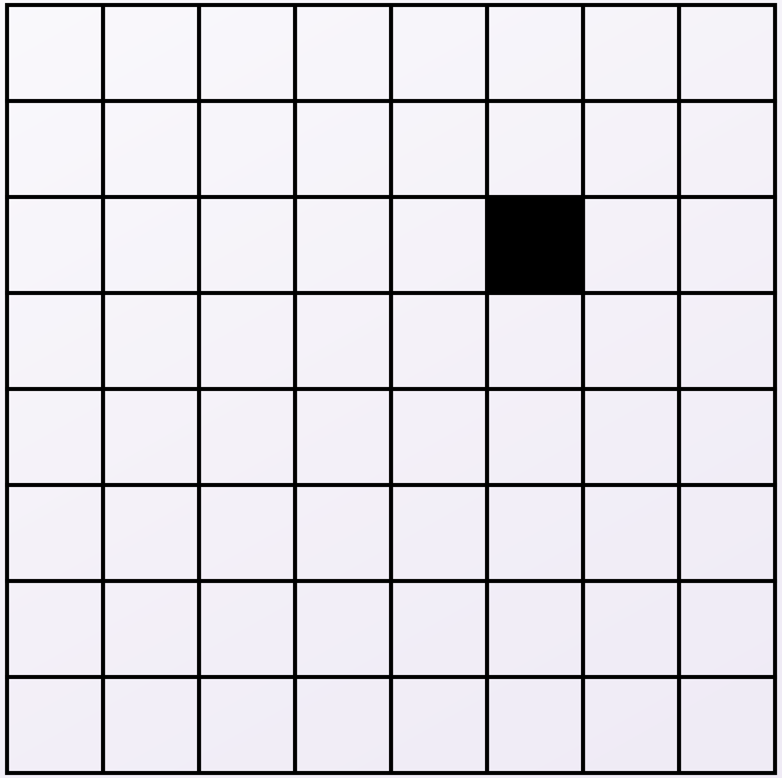
HW 2 out today at 11:30am.

Motivating Example

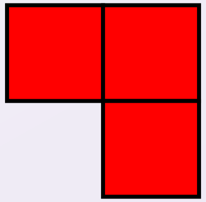
Trominos Tiling

Given an 8x8 grid with 1 cell missing, can we exactly cover it with "trominoes"?

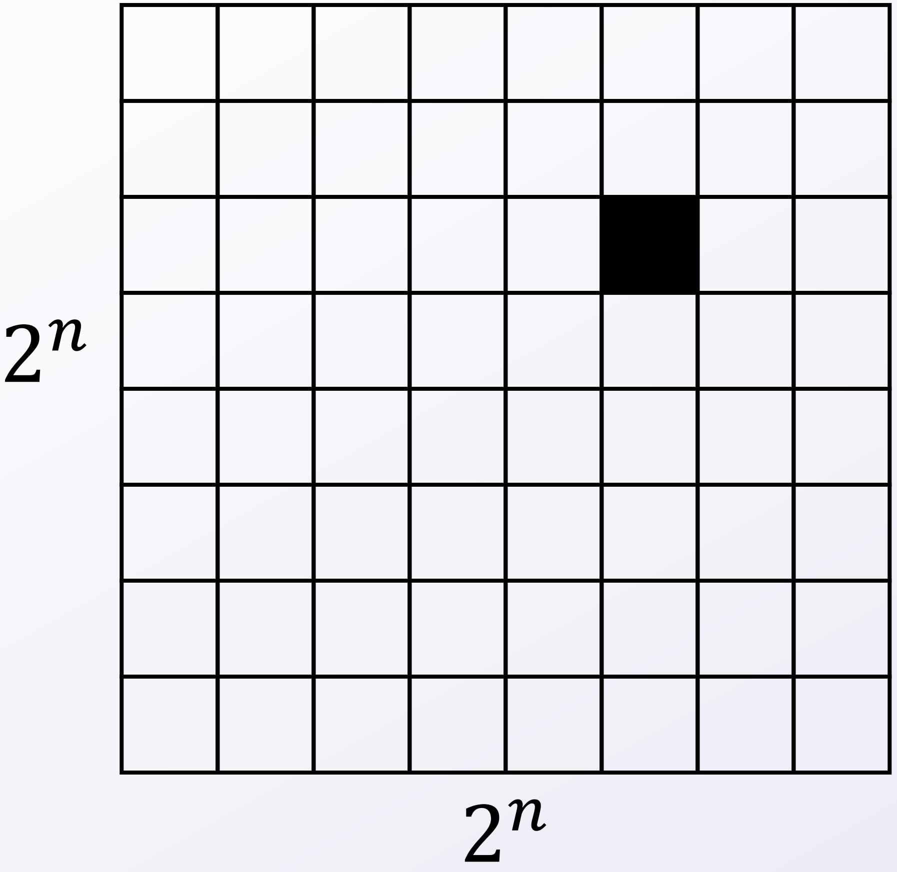
Can you cover this?



With these?

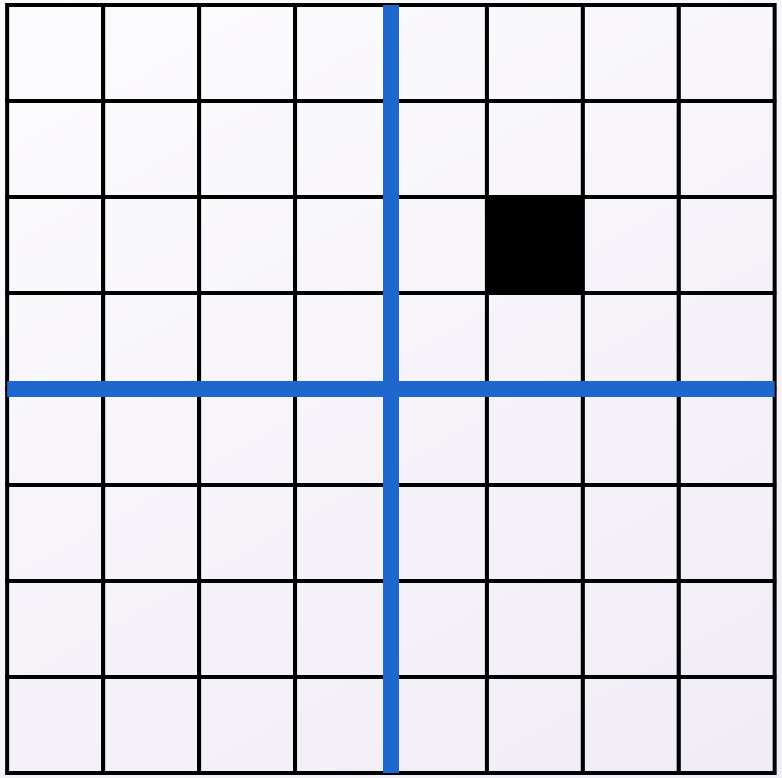


Trominoes Puzzle - Generalizing



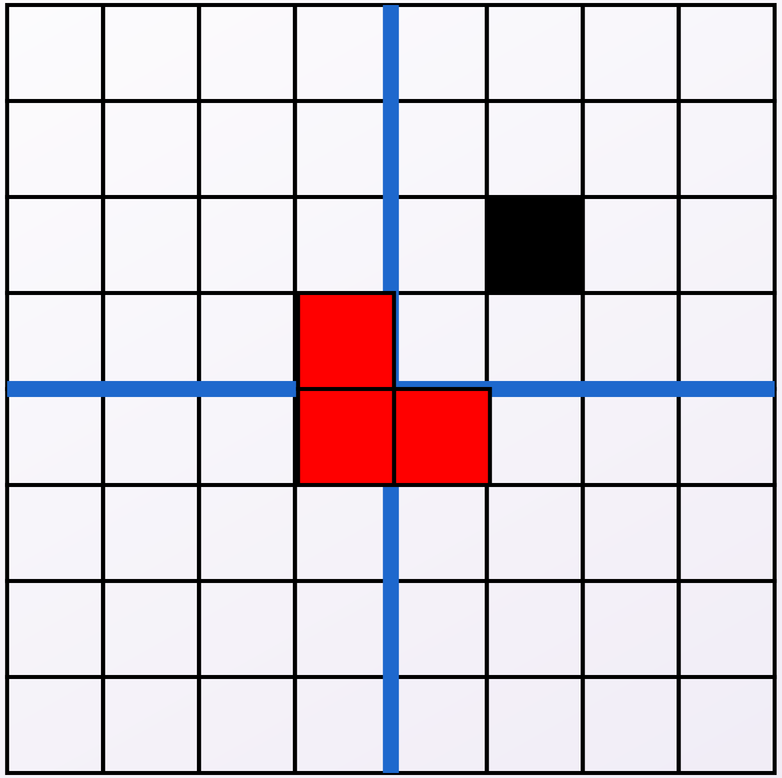
What about larger boards?

Trominoes Puzzle Solution – Step 1



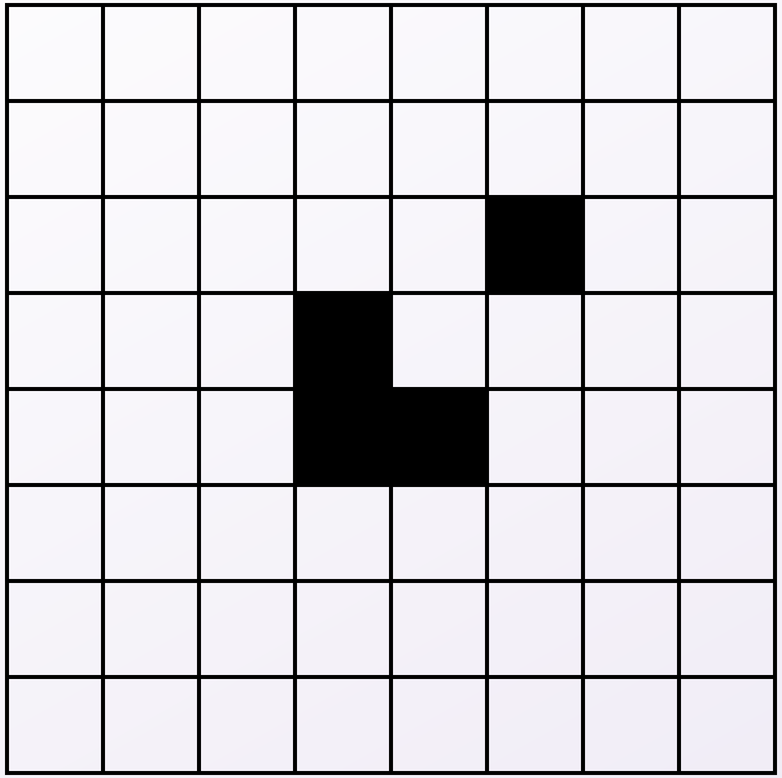
Divide the board into quadrants

Trominoes Puzzle Solution – Step 2



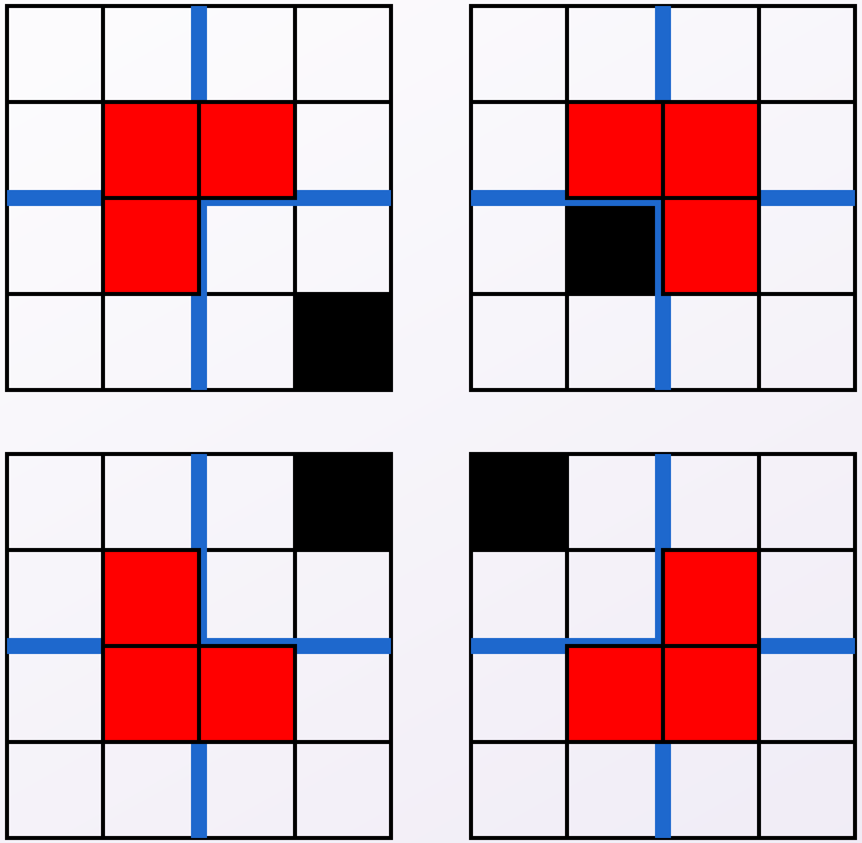
Place a tromino to occupy the three quadrants without the missing piece

Trominoes Puzzle Solution – Step 3



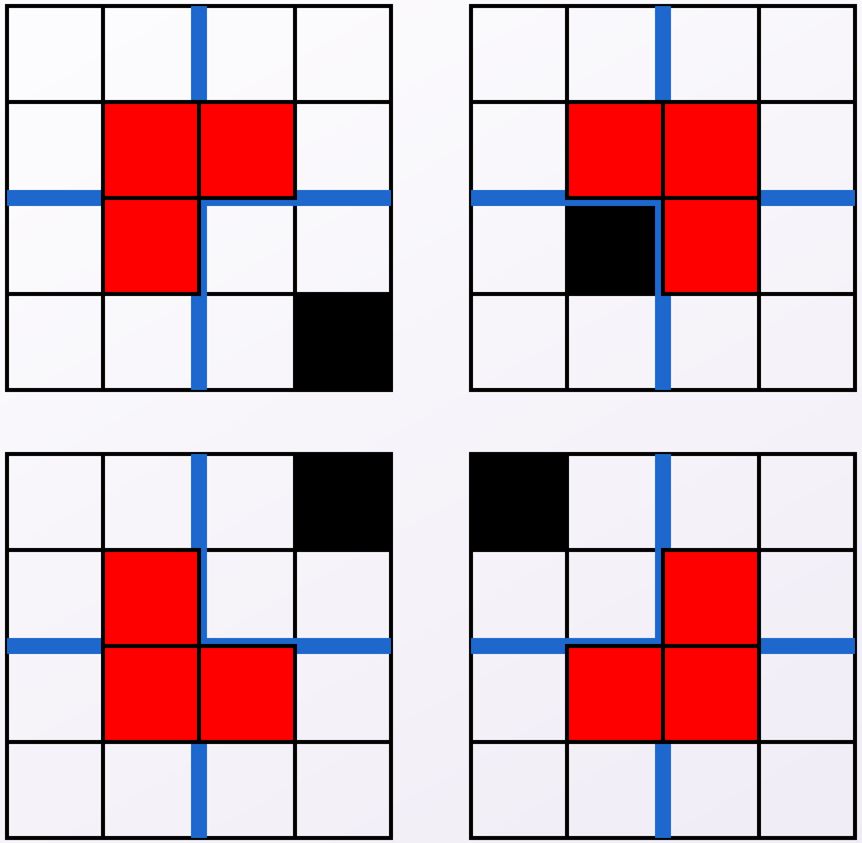
Each quadrant is now a smaller subproblem

Trominoes Puzzle Solution – Step 4

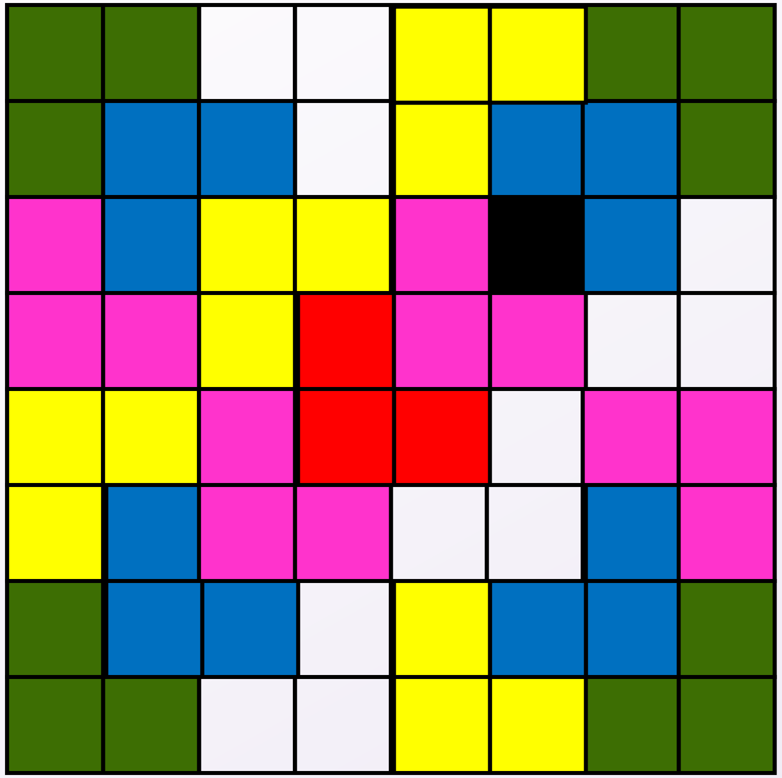


Solve **Recursively**

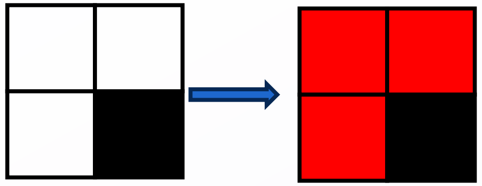
Divide and Conquer



Trominoes Puzzle Solution

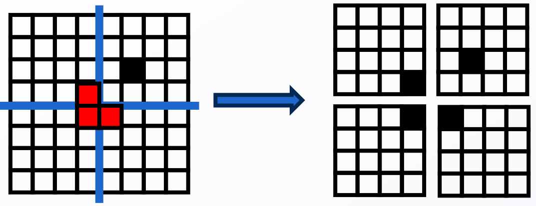


Divide and Conquer (Trominoes)



Base Case:

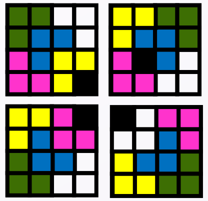
For a 2×2 board, the empty cells will be exactly a tromino



Divide:

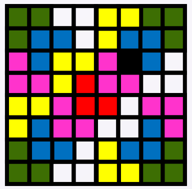
Break of the board into quadrants of size $2^{n-1} \times 2^{n-1}$ each

Put a tromino at the intersection such that all quadrants have one occupied cell



Conquer:

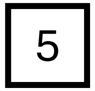
Cover each quadrant



Combine:

Reconnect quadrants

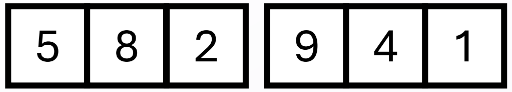
Divide and Conquer (Merge Sort)



Base Case:

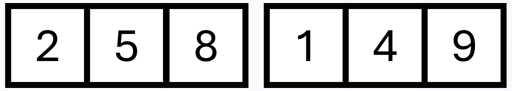
If the list is of length 1 or 0, it’s already sorted, so just return it

(Alternative: when length is ≤ 15, use insertion sort)



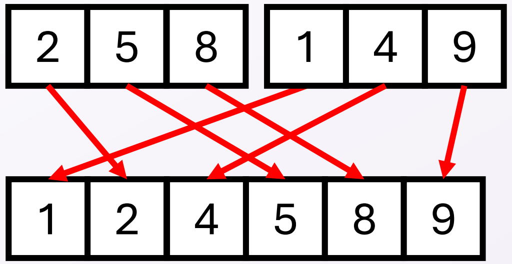
Divide:

Split the list into two “sublists” of (roughly) equal length



Conquer:

Sort both lists recursively



Combine:

Merge sorted sublists into one sorted list

Divide and Conquer (Running Time)

$\boxed{T(c) = k}$

Base Case:

When the problem size is small ($\leq c$), solve non-recursively

$a =$ *number of subproblems*

$\frac{n}{b} =$ *size of each subproblem*

$f_d(n) =$ time to divide

Divide:

When problem size is large, identify 1 or more smaller versions of exactly the same problem

$\boxed{a \cdot T\left(\frac{n}{b}\right)}$

Conquer:

Recursively solve each smaller subproblem

$f_c(n) =$time to combine

Combine:

Use the subproblems' solutions to solve to the original

Overall: $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ where $f(n) = f_d(n) + f_c(n)$

Merge Sort Running Time

$\boxed{T(c) = k}$

$a = \textit{number of subproblems} = 2$

$\frac{n}{b} = \textit{size of each subproblem} = \frac{n}{2}$

$f_d(n) = \text{time to divide} = O(1)$

$\boxed{2T\left(\frac{n}{2}\right)}$

$f_c(n) = \text{time to combine} = O(n)$

Base Case:

If the list is of length 1 or 0, it's already sorted, so just return it

(Alternative: when length is ≤ 15, use insertion sort)

Divide:

Split the list into two "sublists" of (roughly) equal length

Conquer:

Sort both lists recursively

Combine:

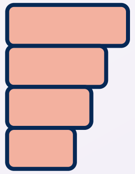
Merge sorted sublists into one sorted list

Overall: $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)}$ **where** $\boldsymbol{f(n) = f_d(n) + f_c(n)}$

Master Theorem

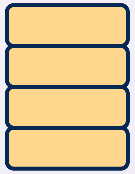
Master Theorem: Suppose that $T(n) = a \cdot T(n/b) + O(n^k)$ for $n > b$.

If $a < b^k$ then $T(n)$ is $O(n^k)$



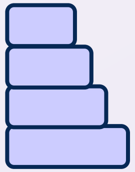
- Cost is dominated by work at top level of recursion

If $a = b^k$ then $T(n)$ is $O(n^k \log n)$



- Total cost is the same for all $\log_b n$ levels of recursion

If $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$



- Note that $\log_b a > k$ in this case
- Cost is dominated by total work at lowest level of recursion

Binary search: $a = 1$, $b = 2$, $k = 0$ so $a = b^k$: Solution: $O\left(n^0 \log n\right) = O(\log n)$

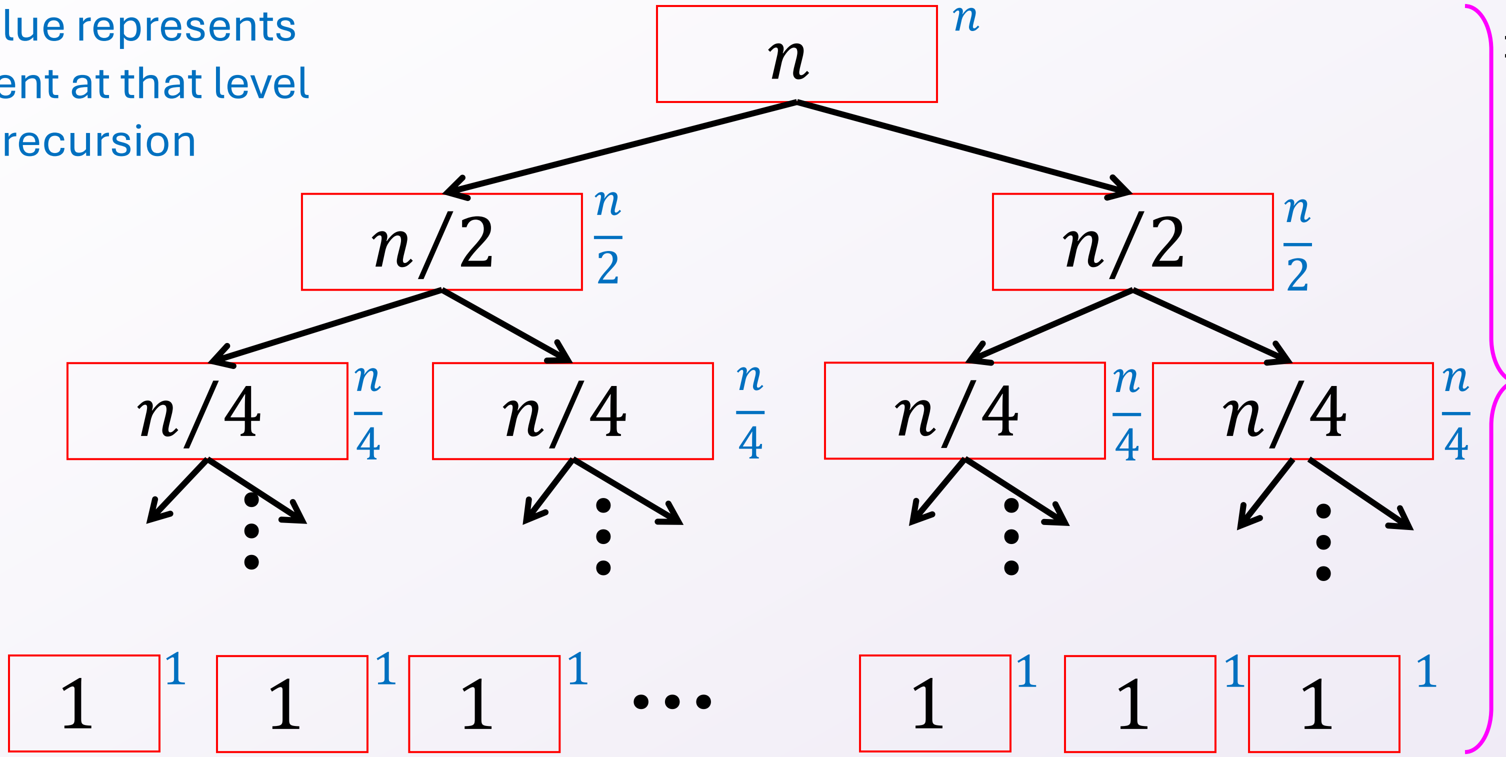
Mergesort: $a = 2$, $b = 2$, $k = 1$ so $a = b^k$: Solution: $O(n^1 \log n) = O(n \log n)$

Visualizing the Merge Sort Recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$\Rightarrow$ n comparisons / level

$\log_2 n$ levels of recursion

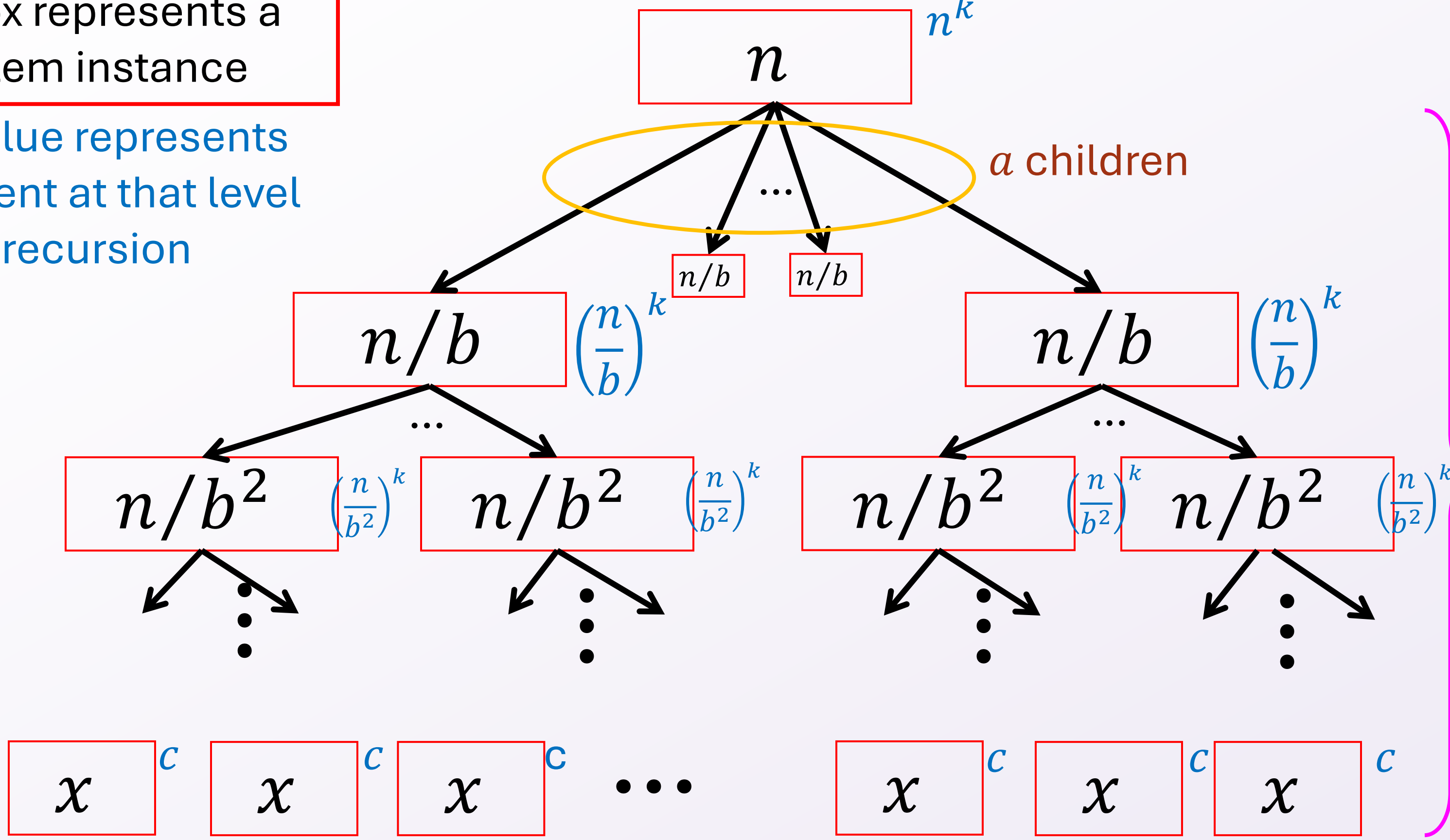
$T(n) = \Theta(n \log n)$

Visualizing Generic Recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + n^k$$

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$\Rightarrow a^i \left(\frac{n}{b^i}\right)^k$ work for each level i

$\approx \log_b n$ levels of recursion

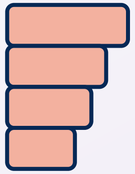
$\Rightarrow a^{\log_b n} = n^{\log_b a}$ work for the last level

$T(n) = ???$ Depends on how quickly the number of boxes increases vs. how quickly the running time of each decreases

In comes the Master Theorem!

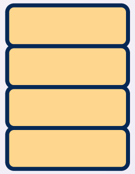
Master Theorem: Suppose that $T(n) = a \cdot T(n/b) + O(n^k)$ for $n > b$.

If $a < b^k$ then $T(n)$ is $O(n^k)$



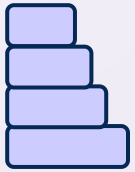
- Cost is dominated by work at top level of recursion

If $a = b^k$ then $T(n)$ is $O(n^k \log n)$



- Total cost is the same for all $\log_b n$ levels of recursion

If $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$



- Note that $\log_b a > k$ in this case
- Cost is dominated by total work at lowest level of recursion

Binary search: $a = 1$, $b = 2$, $k = 0$ so $a = b^k$: Solution: $O\left(n^0 \log n\right) = O(\log n)$

Mergesort: $a = 2$, $b = 2$, $k = 1$ so $a = b^k$: Solution: $O(n^1 \log n) = O(n \log n)$

Integer Multiplication

```
      695273
    × 123412
    --------
     1390546
     695273
   2781092
  2085819
 1390546
 695273
-----------
 85805031476
```

Decimal

```
      110110
    × 101110
    --------
      000000
     110110
    110110
   110110
  000000
 110110
-----------
 100110110100
```

Binary

Elementary school algorithm

$O(n^2)$ time for n-bit integers

Divide and Conquer method

$$\boxed{x_1}\boxed{x_2} = 2^{\frac{n}{2}}\boxed{x_1} + \boxed{x_2}$$

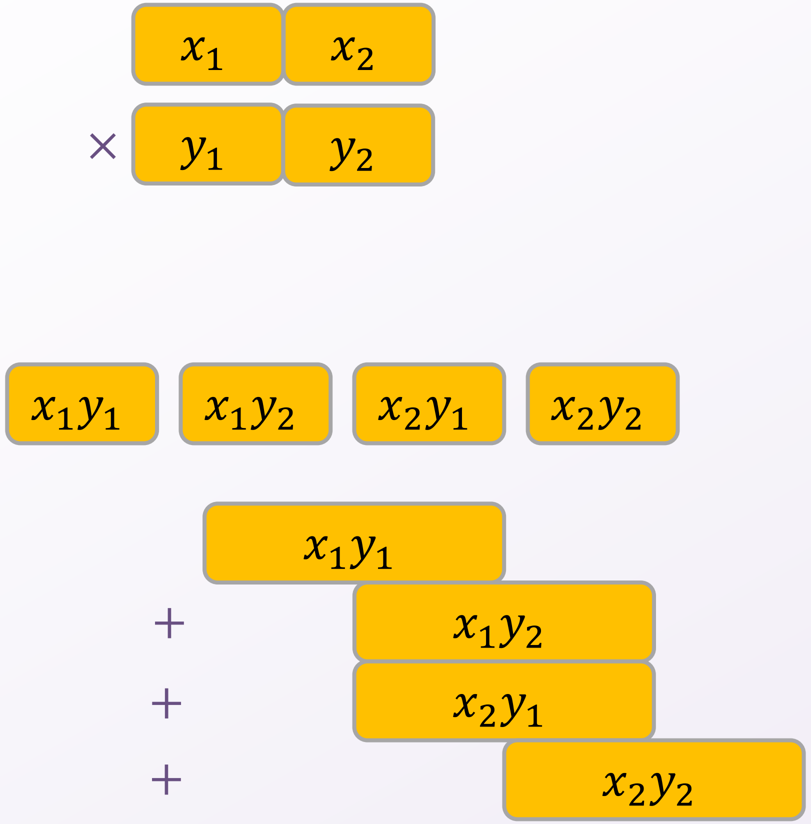
$$\times\ \boxed{y_1}\boxed{y_2} = 2^{\frac{n}{2}}\boxed{y_1} + \boxed{y_2}$$

$$2^n\ (\boxed{x_1} \times \boxed{y_1}) +$$

$$2^{\frac{n}{2}}\ (\boxed{x_1} \times \boxed{y_2} + \boxed{x_2} \times \boxed{y_1}) +$$

$$(\boxed{x_2} \times \boxed{y_2})$$

Divide and Conquer (Integer Multiplication)



Base Case:

If there is only 1 place value, just multiply them

Divide:

Break the operands into 4 values:

- x_1 is the most significant $\frac{n}{2}$ digits of x
- x_2 is the least significant $\frac{n}{2}$ digits of x
- y_1 is the most significant $\frac{n}{2}$ digits of y
- y_2 is the most significant $\frac{n}{2}$ digits of y

Conquer:

Compute each of x_1y_1, x_1y_2, x_2y_1, and x_2y_2

Combine:

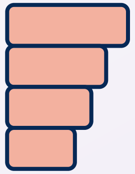
Return $2^n(x_1y_1) + 2^{\frac{n}{2}}(x_1y_2 + x_2y_1) + (x_2y_2)$

Integer Multiplication Recurrence Solution

Master Theorem: Suppose that $\boldsymbol{T(n) = a \cdot T(n/b) + O(n^k)}$ for $\boldsymbol{n > b}$.

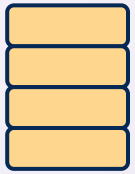
If $\boldsymbol{a < b^k}$ then $\boldsymbol{T(n)}$ is $\boldsymbol{O(n^k)}$

- Cost is dominated by work at top level of recursion



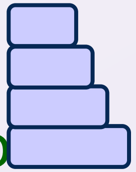
If $\boldsymbol{a = b^k}$ then $\boldsymbol{T(n)}$ is $\boldsymbol{O(n^k \log n)}$

- Total cost is the same for all $\boldsymbol{\log_b n}$ levels of recursion



If $\boldsymbol{a > b^k}$ then $\boldsymbol{T(n)}$ is $\boldsymbol{O(n^{\log_b a})}$

- Note that $\boldsymbol{\log_b a > k}$ in this case
- Cost is dominated by total work at lowest level of recursio



$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$\boldsymbol{a = 4}$, $\boldsymbol{b = 2}$, $\boldsymbol{k = 1}$, so $\boldsymbol{a > b^k}$: Solution: $\boldsymbol{O(n^{\log_b a}) = O(n^2)}$

Karatsuba Method

Can't avoid these

$$2^n(\boxed{x_1y_1}) + 2^{\frac{n}{2}}(\boxed{x_1y_2 + x_2y_1}) + \boxed{x_2y_2}$$

Can we do this with one multiplication?

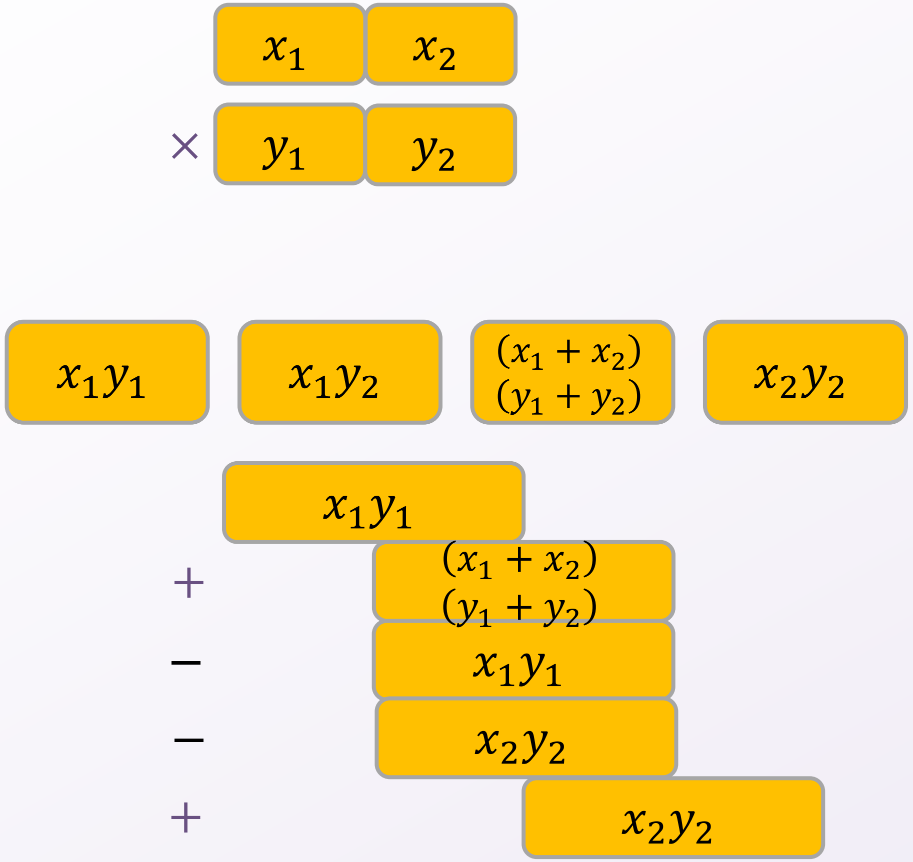
$$(x_1 + x_2)(y_1 + y_2) =$$
$$x_1y_1 + x_1y_2 + x_2y_1 + x_2y_2$$

$$\boxed{x_1y_2 + x_2y_1} = \boxed{(x_1 + x_2)(y_1 + y_2) - x_1y_1 - x_2y_2}$$

Two multiplications

One multiplication

Divide and Conquer (Karatsuba Method)



Base Case:

If there is only 1 place value, just multiply them

Divide:

Break the operands into 4 values:

- x_1 is the most significant $\frac{n}{2}$ digits of x
- x_2 is the least significant $\frac{n}{2}$ digits of x
- y_1 is the most significant $\frac{n}{2}$ digits of y
- y_2 is the most significant $\frac{n}{2}$ digits of y

Conquer:

Compute each of x_1y_1, $(x_1 + x_2)(y_1 + y_2)$, and x_2y_2

Combine:

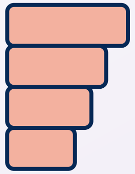
Return

$$2^n(x_1y_1) + 2^{\frac{n}{2}}\big((x_1 + x_2)(y_1 + y_2) - x_1y_1 - x_2y_2\big) + (x_2y_2)$$

Karatsuba Method Recurrence Solution

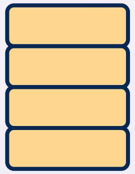
Master Theorem: Suppose that $T(n) = a \cdot T(n/b) + O(n^k)$ for $n > b$.

If $a < b^k$ then $T(n)$ is $O(n^k)$



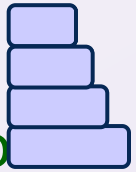
- Cost is dominated by work at top level of recursion

If $a = b^k$ then $T(n)$ is $O(n^k \log n)$



- Total cost is the same for all $\log_b n$ levels of recursion

If $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$



- Note that $\log_b a > k$ in this case
- Cost is dominated by total work at lowest level of recursio

$$T(n) = 3T\left(\frac{n}{2}\right) + n$$

$a = 3, b = 2, k = 1$, so $a > b^k$: Solution: $O(n^{\log_b a}) = O\left(n^{\log_2 3}\right) = O(n^{1.585})$

Matrix Multiplication

$$
\begin{array}{cc} & n \\ n & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \end{array} \times \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}
$$

$$
= \begin{bmatrix} 1 \cdot 2 + 2 \cdot 8 + 3 \cdot 16 & 1 \cdot 4 + 2 \cdot 10 + 3 \cdot 16 & 1 \cdot 6 + 2 \cdot 12 + 3 \cdot 18 \\ \vdots & \vdots & \vdots \end{bmatrix}
$$

$$
= \begin{bmatrix} 60 & 72 & 84 \\ 132 & 162 & 192 \\ 204 & 252 & 300 \end{bmatrix}
$$

Run time? $O(n^3)$

Multiplying Matrices

for $i \leftarrow 1$ to n

    for $j \leftarrow 1$ to n

        $C[i,j] \leftarrow 0$

        for $k \leftarrow 1$ to n

            $C[i,j] \leftarrow C[i,j] + A[i,k] \cdot B[k,j]$

        endfor

    endfor

endfor

Can we improve this with divide and conquer?

We can see subproblems!

$$A_{11} \qquad\qquad\qquad\qquad\qquad B_{11}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$A \times B =$$

$$A_{11} \times B_{11} \qquad\qquad\qquad\qquad A_{11} \times B_{11}$$

$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & \cdot & \cdot \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & \cdot & \cdot \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & \cdot & \cdot \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & \cdot & \cdot \end{bmatrix}$$

Matrix Multiplication D&C

Multiply $n \times n$ matrices (A and B)

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

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

Divide and Conquer Matrix Multiplication

A_{11}	A_{12}		B_{11}	B_{12}
A_{21}	A_{22}		B_{21}	B_{22}

P_1	P_2	P_3	P_4
P_5	P_6	P_7	P_8

$P_1 + P_2$	$P_3 + P_4$
$P_5 + P_6$	$P_7 + P_8$

Base Case:

For a 1×1 matrices, return the product in a 1×1 matrix

Divide:

Use each quadrant of the input $n \times n$ matrices as it's own $\frac{n}{2} \times \frac{n}{2}$ matrix

Conquer:

Compute each of:

$$\begin{aligned} P_1 &= A_{11} \times B_{11} & P_5 &= A_{21} \times B_{11} \\ P_2 &= A_{12} \times B_{21} & P_6 &= A_{22} \times B_{21} \\ P_3 &= A_{11} \times B_{12} & P_7 &= A_{21} \times B_{12} \\ P_4 &= A_{12} \times B_{22} & P_8 &= A_{22} \times B_{22} \end{aligned}$$

Combine:

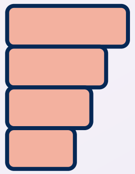
Compute the value of each quadrant by summing $P_1 \ldots P_8$ as shown

Matrix Multiplication Recurrence Solution

Master Theorem: Suppose that $T(n) = a \cdot T(n/b) + O(n^k)$ for $n > b$.

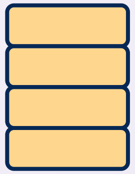
If $a < b^k$ then $T(n)$ is $O(n^k)$

- Cost is dominated by work at top level of recursion



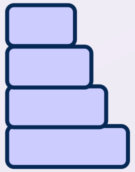
If $a = b^k$ then $T(n)$ is $O(n^k \log n)$

- Total cost is the same for all $\log_b n$ levels of recursion



If $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$

- Note that $\log_b a > k$ in this case
- Cost is dominated by total work at lowest level of recursion



$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

$a = 8$, $b = 2$, $k = 2$, so $a > b^k$: Solution: $O(n^{\log_b a}) = O\left(n^{\log_2 8}\right) = O(n^3)$

How to Improve?

Multiply $n \times n$ matrices (A and B)

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

$$A \times B = \boxed{\begin{bmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{bmatrix}}$$

Idea: Use an idea like Karatsuba! Can we derive these products using addition/subtraction?

Strassen's Algorithm

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

Calculate:

$$Q_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$
$$Q_2 = (A_{21} + A_{22}) \times B_{11}$$
$$Q_3 = A_{11} \times (B_{12} - B_{22})$$
$$Q_4 = A_{22} \times (B_{21} - B_{11})$$
$$Q_5 = (A_{11} + A_{12}) \times B_{22}$$
$$Q_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$
$$Q_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

Find $A \times B$:

$$\begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix} =$$

$$\begin{bmatrix} Q_1 + Q_4 - Q_5 + Q_7 & Q_3 + Q_5 \\ Q_2 + Q_4 & Q_1 - Q_2 + Q_3 + Q_6 \end{bmatrix}$$

Divide and Conquer – Strassen's Algorithm

A_{11}	A_{12}		B_{11}	B_{12}
A_{21}	A_{22}		B_{21}	B_{22}

Q_1	Q_2	Q_3	Q_4
Q_5	Q_6	Q_7	

$Q_1 + Q_4 - Q_5 + Q_7$	$Q_3 + Q_5$
$Q_2 + Q_4$	$Q_1 - Q_2 + Q_3 + Q_6$

Base Case:

For a 32×32 matrices, use the textbook algorithm

Divide:

Use each quadrant of the input $n \times n$ matrices as it's own $\frac{n}{2} \times \frac{n}{2}$matrix

Conquer:

Compute each of:

$$
\begin{aligned}
Q_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
Q_2 &= (A_{21} + A_{22}) \times B_{11} \\
Q_3 &= A_{11} \times (B_{12} - B_{22}) \\
Q_4 &= A_{22} \times (B_{21} - B_{11}) \\
Q_5 &= (A_{11} + A_{12}) \times B_{22} \\
Q_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}) \\
Q_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22})
\end{aligned}
$$

Combine:

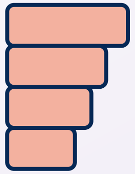
Compute the value of each quadrant by summing $Q_1 \ldots Q_8$ as shown

Strassen's Recurrence Solution

Master Theorem: Suppose that $\boldsymbol{T(n) = a \cdot T(n/b) + O(n^k)}$ for $\boldsymbol{n > b}$.

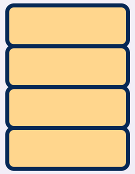
If $\boldsymbol{a < b^k}$ then $\boldsymbol{T(n)}$ is $\boldsymbol{O(n^k)}$

- Cost is dominated by work at top level of recursion



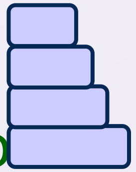
If $\boldsymbol{a = b^k}$ then $\boldsymbol{T(n)}$ is $\boldsymbol{O(n^k \log n)}$

- Total cost is the same for all $\log_b n$ levels of recursion



If $\boldsymbol{a > b^k}$ then $\boldsymbol{T(n)}$ is $\boldsymbol{O(n^{\log_b a})}$

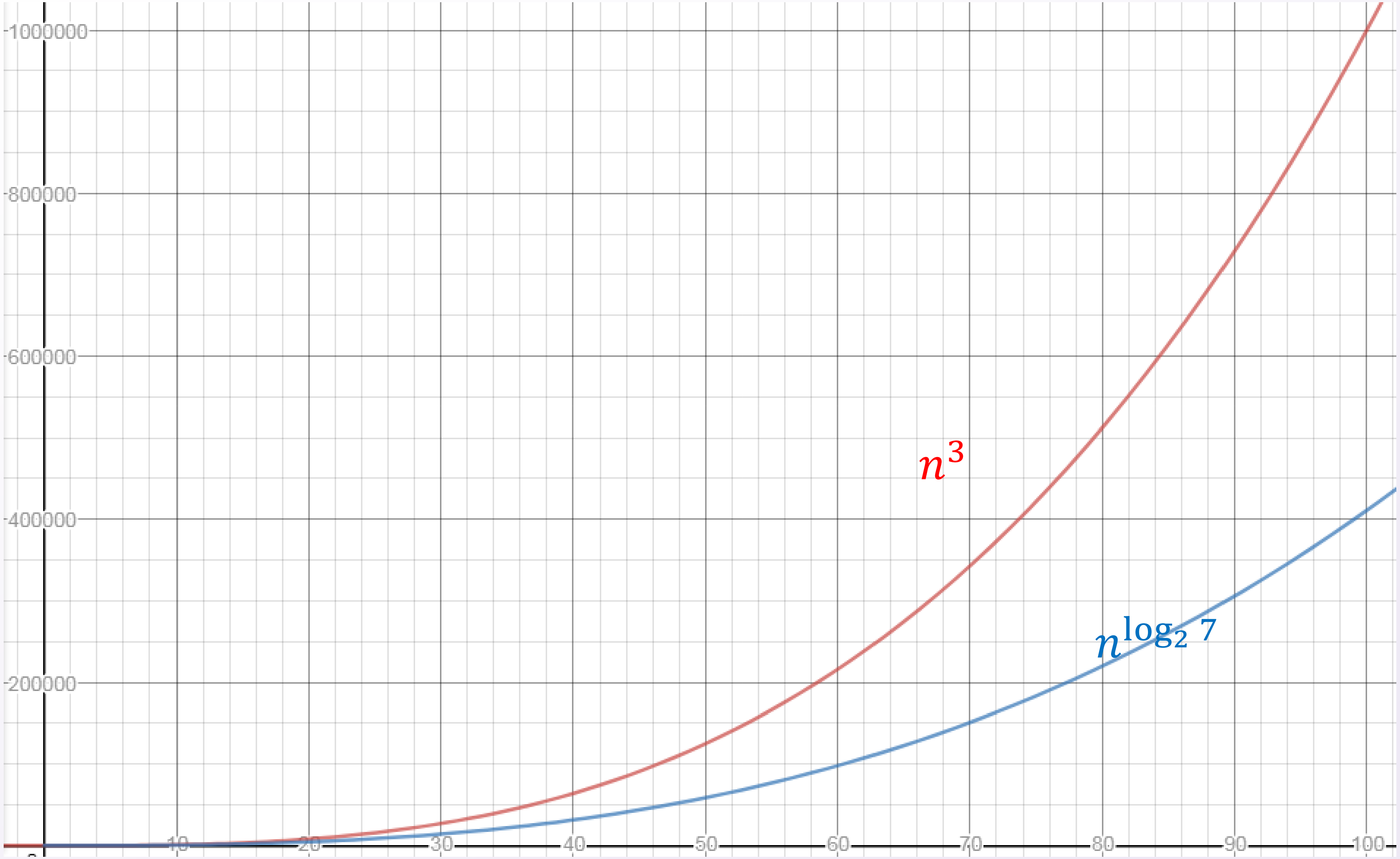
- Note that $\log_b a > k$ in this case
- Cost is dominated by total work at lowest level of recursio



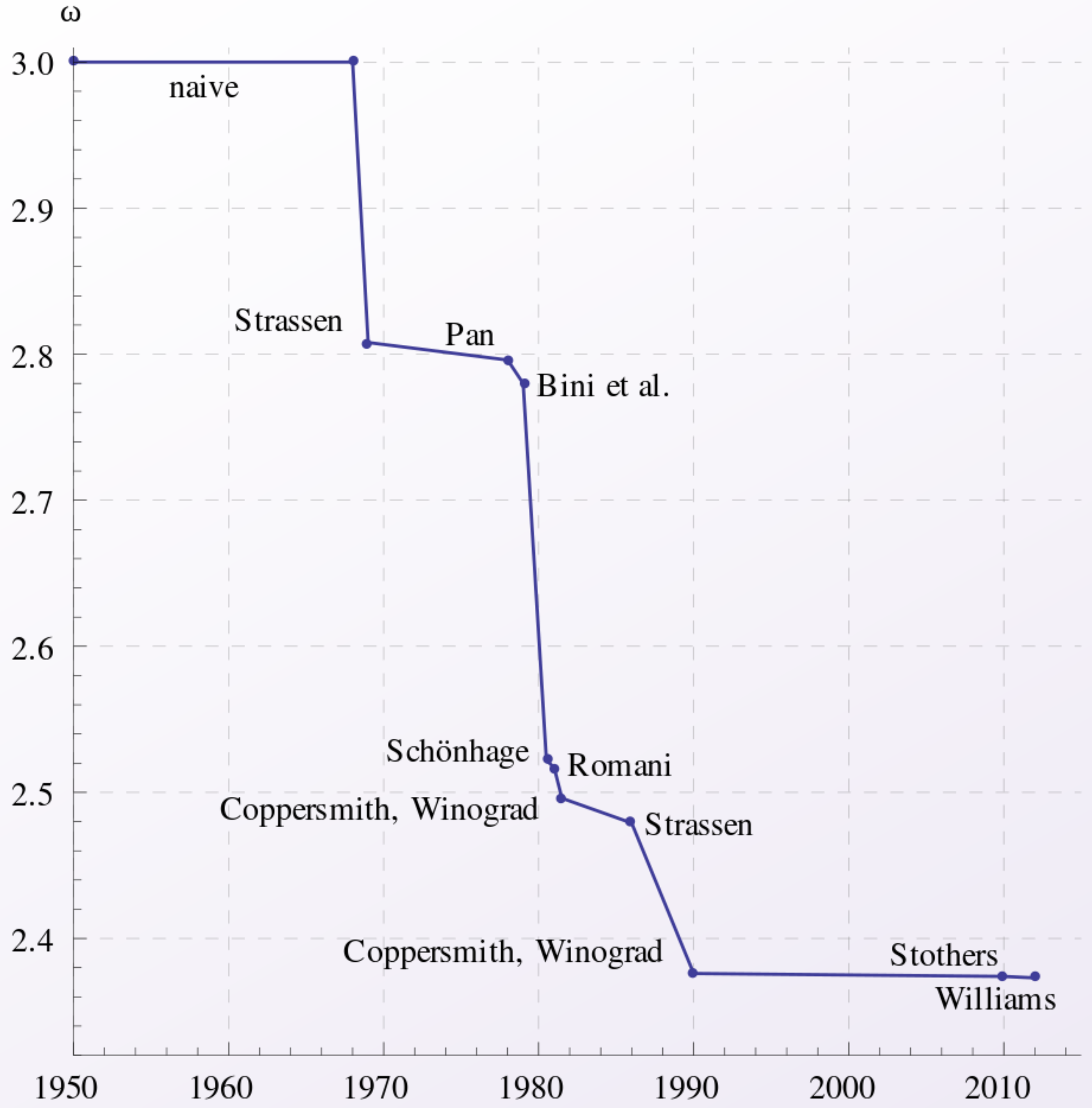
$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

$\boldsymbol{a = 7}$, $\boldsymbol{b = 2}$, $\boldsymbol{k = 2}$, so $\boldsymbol{a > b^k}$: Solution: $O(n^{\log_b a}) = O\left(n^{\log_2 7}\right) = O(n^{2.807})$

Strassen's Algorithm vs. Naïve Running Time



Is this the fastest?



Every few years someone comes up with an asymptotically faster algorithm

Current best is $O(n^{2.3728596})$, but it requires input sizes in the millions to actually be faster

We know there is no algorithm with running time $o(n^2)$

The best possible running time is unknown!
(and weirdly, may not exist!)