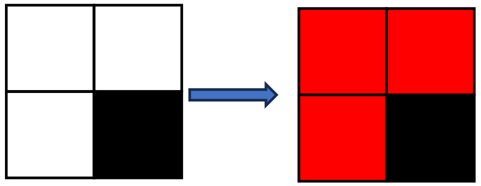
CSE 421 Winter 2025
Lecture 10: Divide and Conquer 2

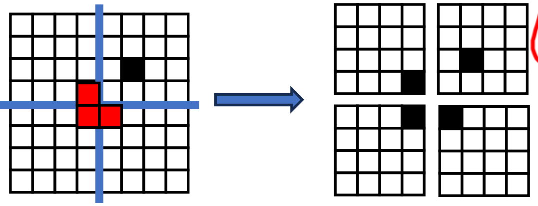
Nathan Brunelle

http://www.cs.uw.edu/421

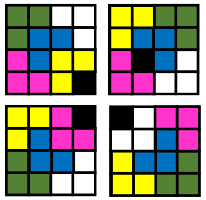
Divide and Conquer (Trominoes)



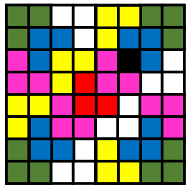
- **Base Case**:
  - For a 2×2 board, the empty cells will be exactly a tromino



- **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



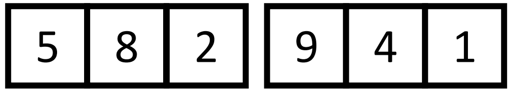
- **Conquer**:
  - Cover each quadrant



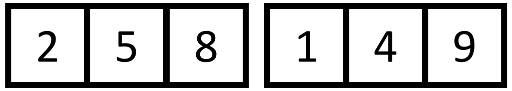
- **Combine**:
  - Reconnect quadrants

Divide and Conquer (Merge Sort)

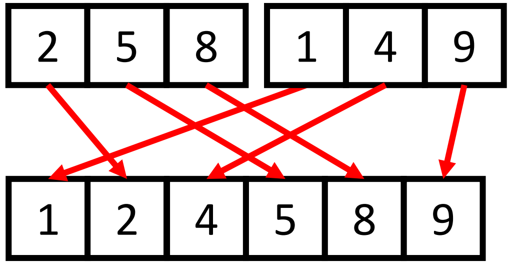
- **Base Case**:
  - If the list is of length 1 or 0, it's already sorted, so just return it
  - (Alternative: when length is ≤ 15, use insertion sort)



- **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)

$T(c) = k$

a = number of subproblems

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

$f_d(n)$ = time to divide

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

$f_c(n)$ = time to combine

- **Base Case**:
  - When the problem size is small ($\leq c$), solve non-recursively
- **Divide**:
  - When problem size is large, identify 1 or more smaller versions of exactly the same problem
- **Conquer**:
  - Recursively solve each smaller subproblem
- **Combine**:
  - Use the subproblems' solutions to solve to the original

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

Divide and Conquer (Running Time)

$T(c) = k$

$a = number\ of\ subproblems$

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

$f_d(n) =$ time to divide

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

$f_c(n) =$time to combine

- **Base Case**:
  - When the problem size is small ($\leq c$), solve non-recursively
- **Divide**:
  - When problem size is large, identify 1 or more smaller versions of exactly the same problem
- **Conquer**:
  - Recursively solve each smaller subproblem
- **Combine**:
  - Use the subproblems' solutions to solve to the original

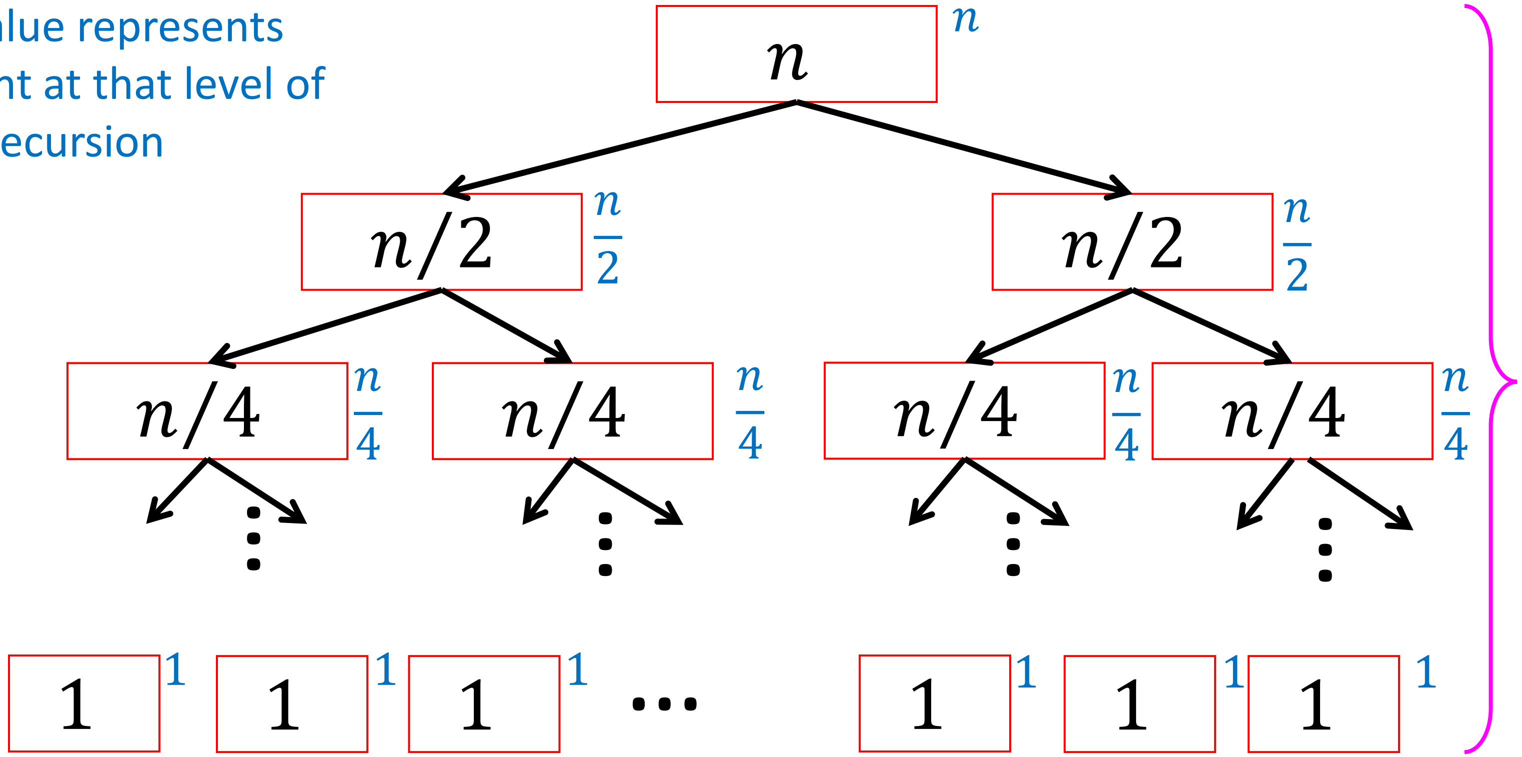
Overall: $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k)}$ **where** $\boldsymbol{f_d(n) + f_c(n) \in \Theta(n^k)}$

Tree Method (Merge Sort)

Red box represents a problem instance

Blue value represents time spent at that level of recursion

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



$\Rightarrow n$ comparisons / level

$\log_2 n$ levels of recursion

$$T(n) = \sum_{i=0}^{\log_2 n} n = \Theta(n \log n)$$

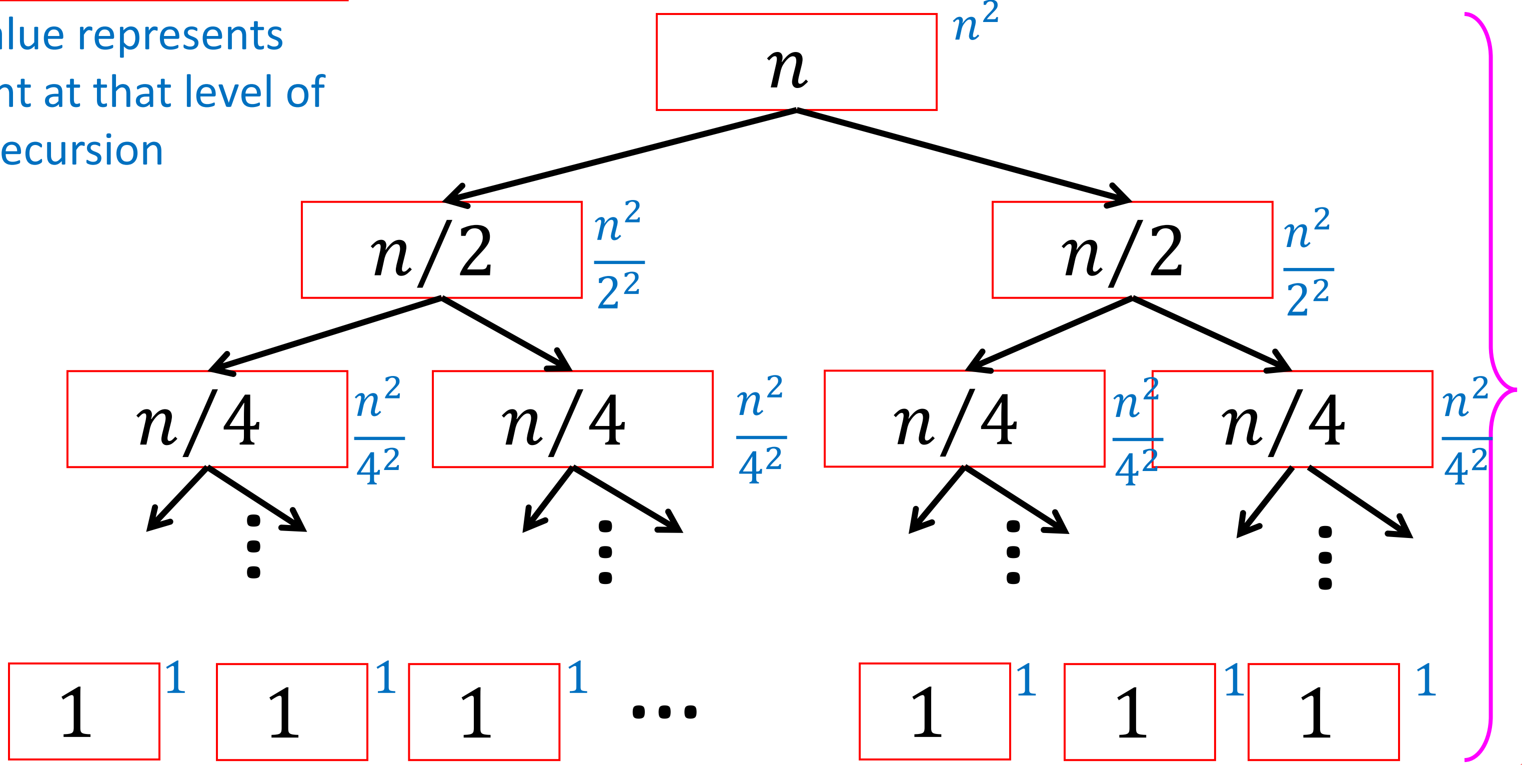
$n^2 \cdot 2^i \left(\frac{1}{2^i}\right)^2$

Tree Method (Slow CPP from last time)

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

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$\Rightarrow 2^i \frac{n^2}{2^{2i}} = \frac{n^2}{2^i}$ work for level i

$\log_2 n$ levels of recursion

$$T(n) = \sum_{i=0}^{\log_2 n} \frac{n^2}{2^i} = \Theta(n^2)$$

Tree Method (More Subproblems)

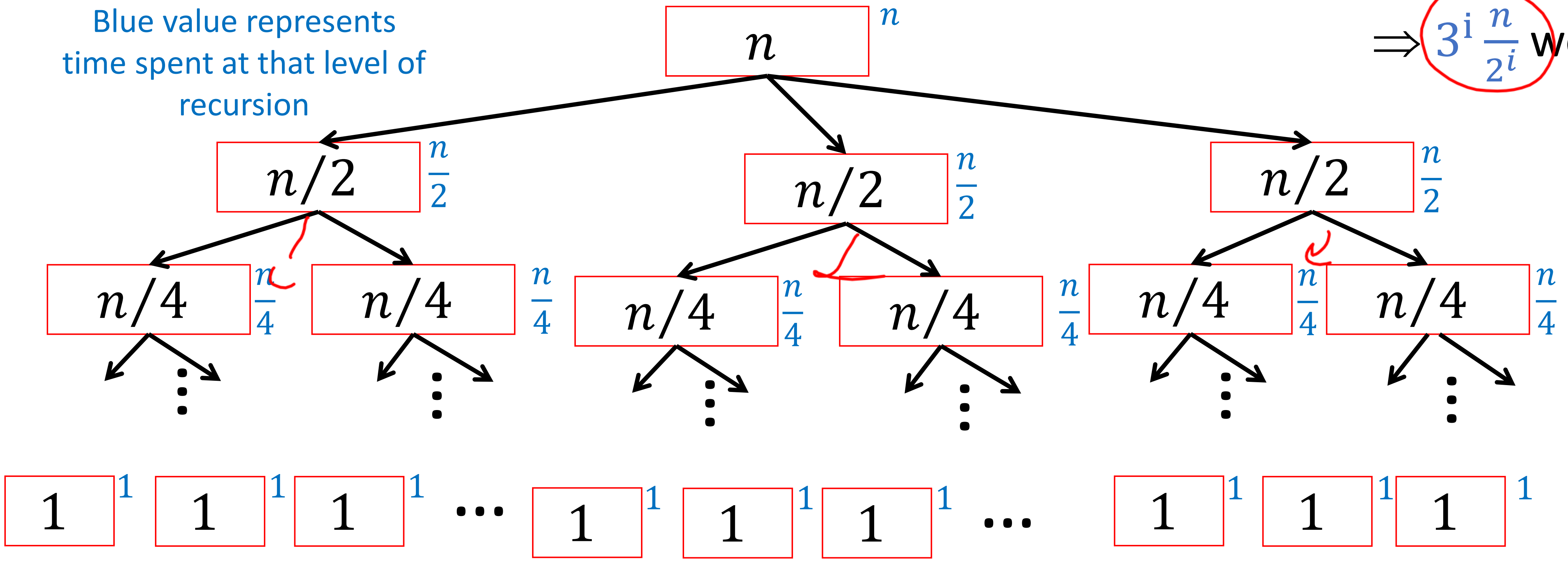
Red box represents a problem instance

Blue value represents time spent at that level of recursion

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

$a = 3$

$b = 2$

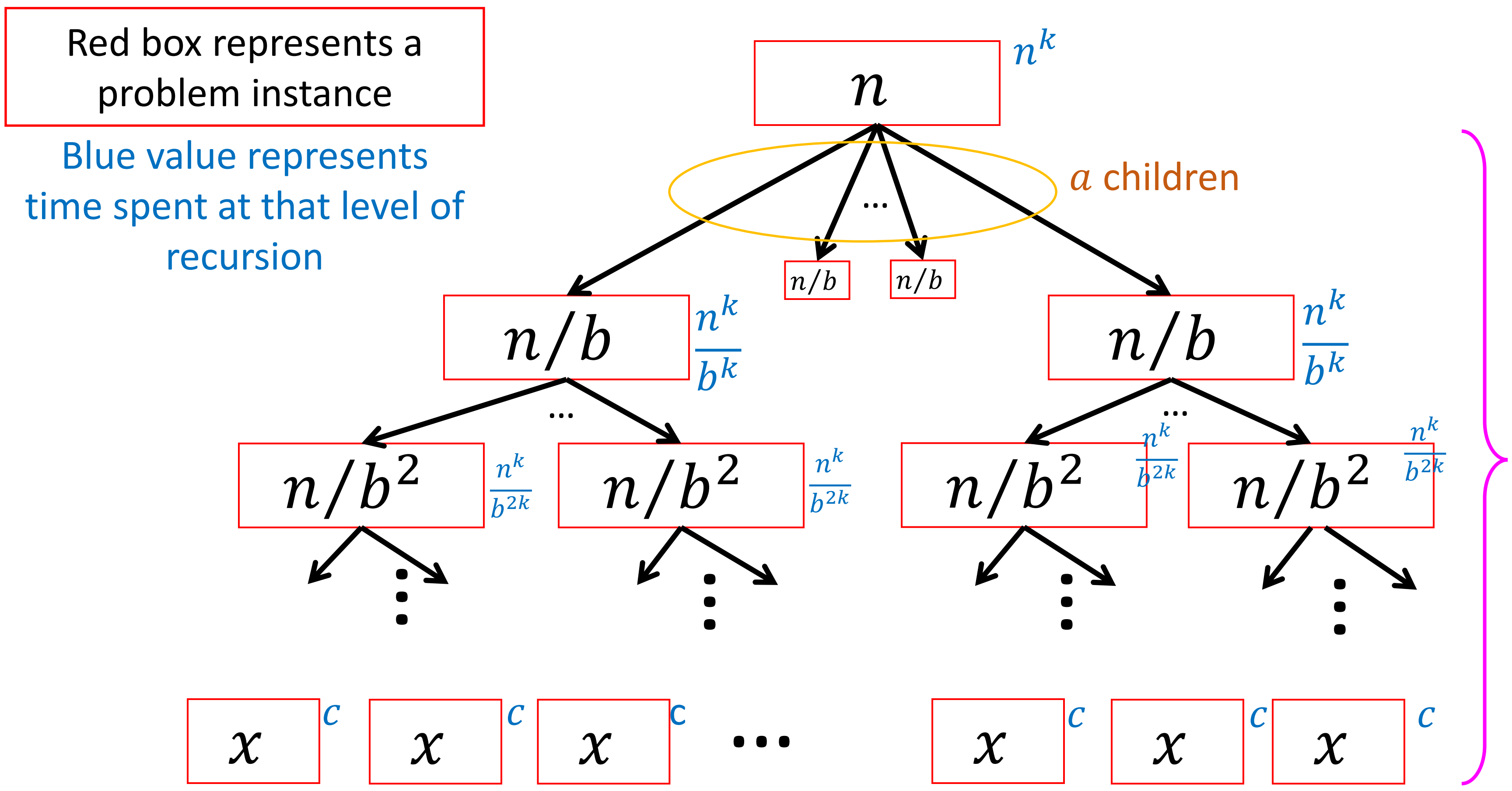


$\Rightarrow 3^i \frac{n}{2^i}$ work for level i

$$T(n) = \sum_{i=0}^{\log_2 n} n\left(\frac{3}{2}\right)^i = \Theta\left(n^{\log_2 3}\right) \approx \Theta\left(n^{1.585}\right)$$

Tree Method

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



$\Rightarrow a^i \frac{n^k}{b^{ik}}$ work for level i

$\approx \log_b n$ levels of recursion

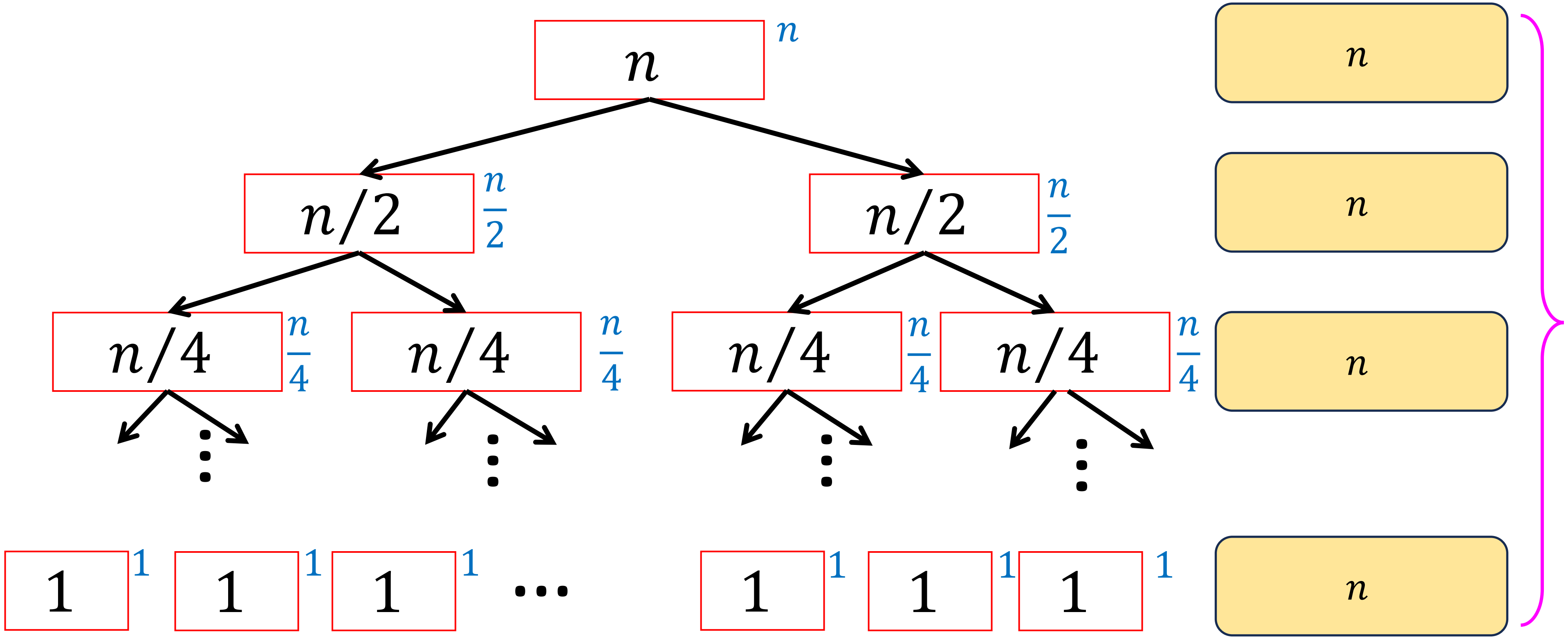
$$T(n) = \sum_{i=0}^{\log_b n} n^k \left(\frac{a}{b^k}\right)^i$$

Work Stays Constant

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

$$\frac{a}{b^k} = \frac{2}{2^1} = 1$$

$$T(n) = \sum_{i=0}^{\log_2 n} n(1)^i = \Theta(n \log n)$$



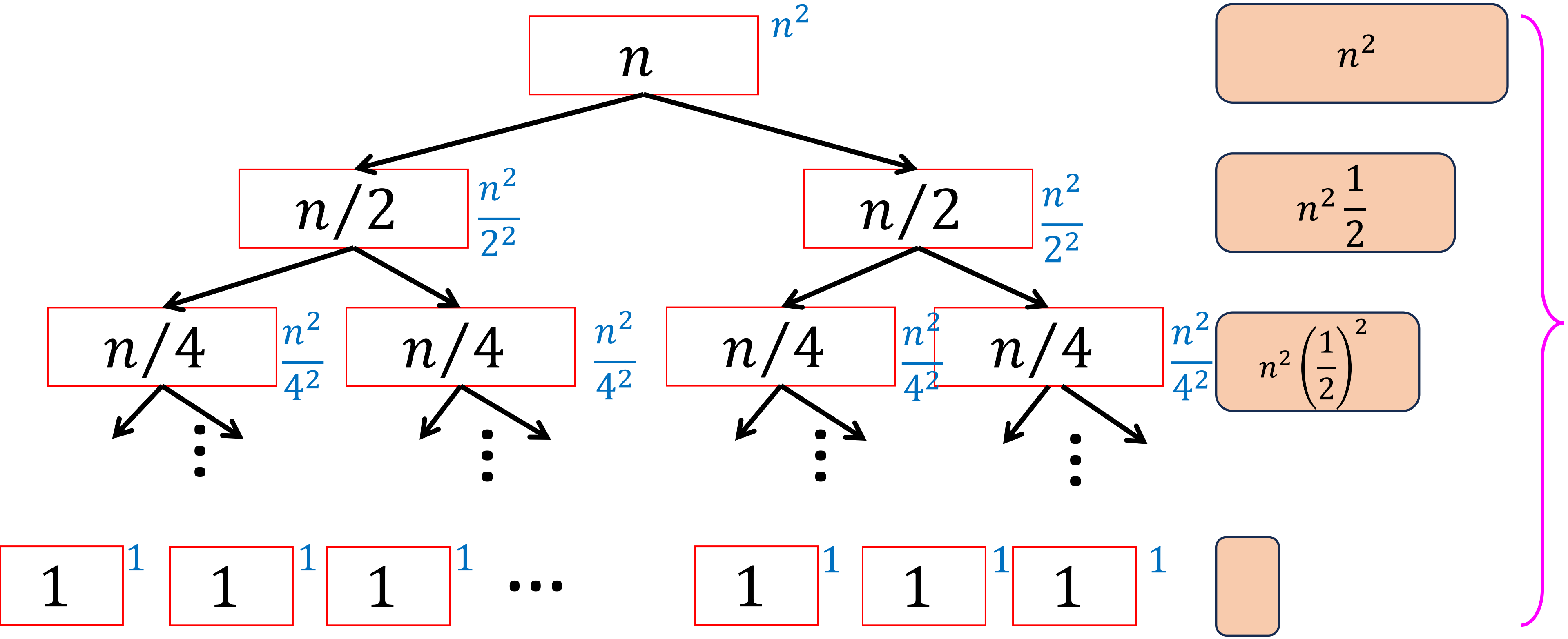
Total work is the work for any level, times the height

Work Decreases

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

$$\frac{a}{b^k} = \frac{2}{2^2} = \frac{1}{2}$$

$$T(n) = \sum_{i=0}^{\log_2 n} n^2 \left(\frac{1}{2}\right)^i = \Theta(n^2)$$



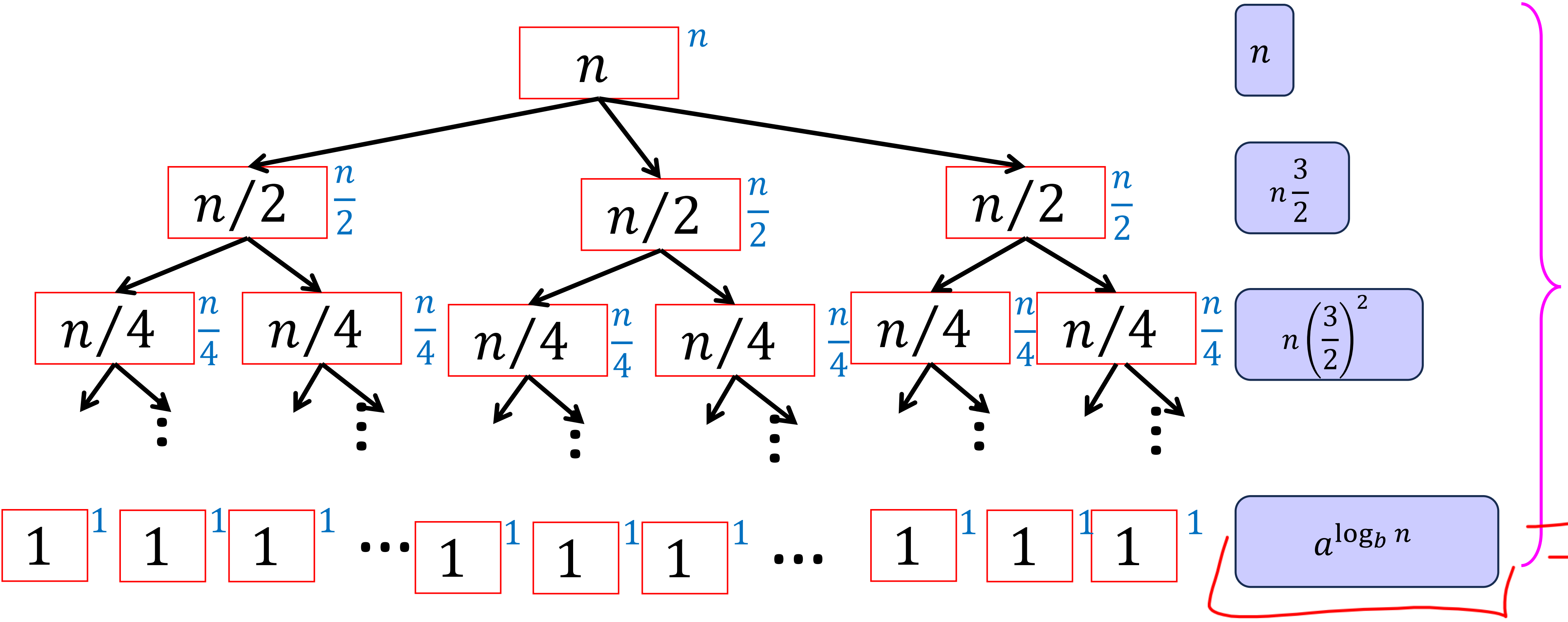
Total work is asymptotically dominated by the root

Work Increases

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

$$\frac{a}{b^k} = \frac{2}{2^2} = \frac{1}{2}$$

$$T(n) = \sum_{i=0}^{\log_2 n} n^2 \left(\frac{1}{2}\right)^i = \Theta(n^2)$$



Total work is asymptotically dominated by the leaves

Summary

When solving a recurrence of the form

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

The tree method will produce the series

$$T(n) = \sum_{i=0}^{\log_b n} n^k \left(\frac{a}{b^k}\right)^i$$

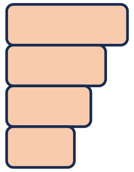
An asymptotic bound on $T(n)$ then only depends on the value of $\frac{a}{b^k}$

$\frac{a}{b^k} = 1$	$\frac{a}{b^k} < 1$	$\frac{a}{b^k} > 1$
n^k	$n^k \left(\frac{a}{b}\right)^0$	n^k
n^k	$n^k \left(\frac{a}{b}\right)^1$	$n^k \left(\frac{a}{b}\right)^1$
n^k	$n^k \left(\frac{a}{b}\right)^2$	$n^k \left(\frac{a}{b}\right)^2$
n^k	1	$a^{\log_b n} = n^{\log_b a}$
$\Theta(n^k \log n)$	$\Theta(n^k)$	$\Theta(n^{\log_b a})$

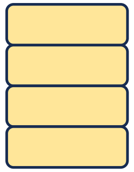
Solving Divide and Conquer Recurrences

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

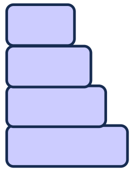
- If $a < b^k$ then $T(n)$ is $O(n^k)$
  - Cost is dominated by work at top level of recursion



- 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



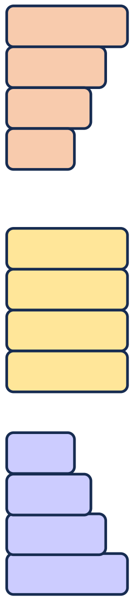
Binary search: $a = 1$, $b = 2$, $k = 0$ so $a = b^k$: Solution: $O(n^0 \log n) = O(\log n)$

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

Beware! It doesn't always apply!

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



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

$a = 4, b = 2, k = ???$

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

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

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

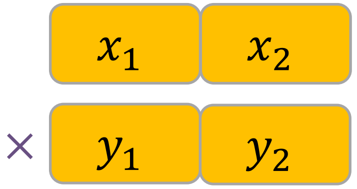
$$2^n \left(x_1 \times y_1 \right) +$$

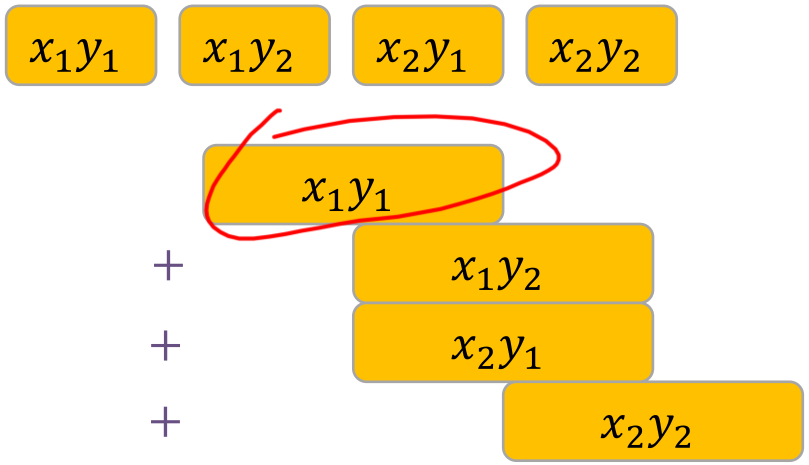
$$2^{\frac{n}{2}} \left(x_1 \times y_2 + x_2 \times y_1 \right) +$$

$$\left(x_2 \times y_2 \right)$$

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

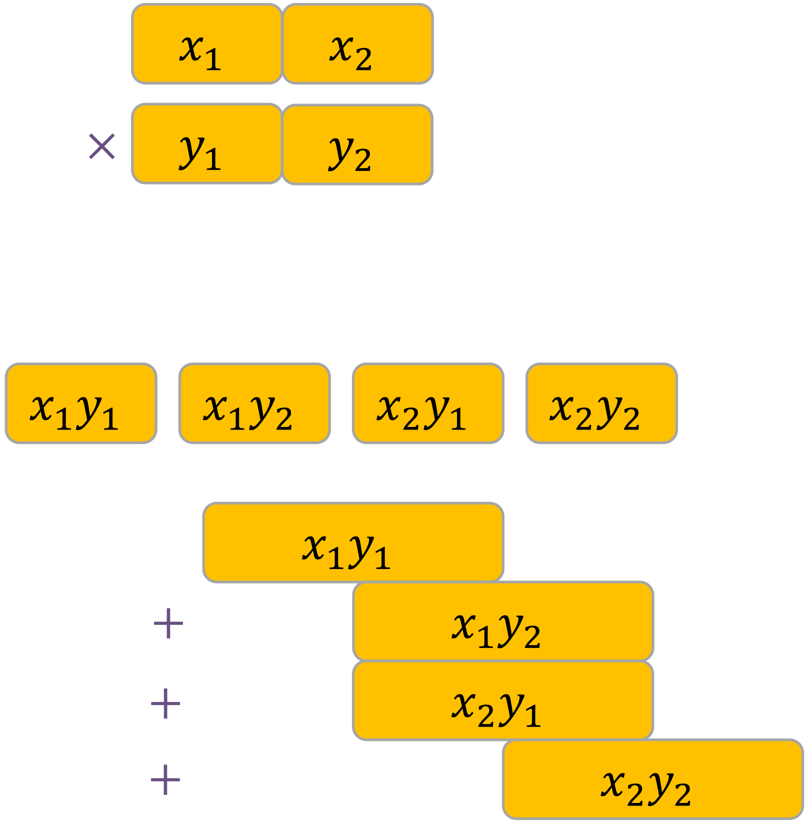
Divide and Conquer (Integer Multiplication)





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

Divide and Conquer (Integer Multiplication)

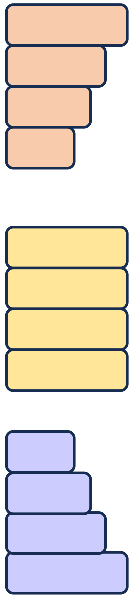


- **Base Case**:
  - If there is only 1 place value, just multiply them
- **Divide**:
  - Break the operands into 4 values:
    - x_1 is the most significant $\frac{n}{2}$ digits of x
    - x_2 is the least significant $\frac{n}{2}$ digits of x
    - y_1 is the most significant $\frac{n}{2}$ digits of y
    - y_2 is the most significant $\frac{n}{2}$ digits of y
- **Conquer**:
  - Compute each of $x_1 y_1$, $x_1 y_2$, $x_2 y_1$, and $x_2 y_2$
- **Combine**:
  - Return $2^n (x_1 y_1) + 2^{\frac{n}{2}} (x_1 y_2 + x_2 y_1) + (x_2 y_2)$

Integer 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



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

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

Karatsuba Method

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

Can't avoid these

Can we do this with one multiplication?

$$(x_1 + x_2)(y_1 + y_2) =$$

$$x_1y_1 + x_1y_2 + x_2y_1 + x_2y_2$$

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

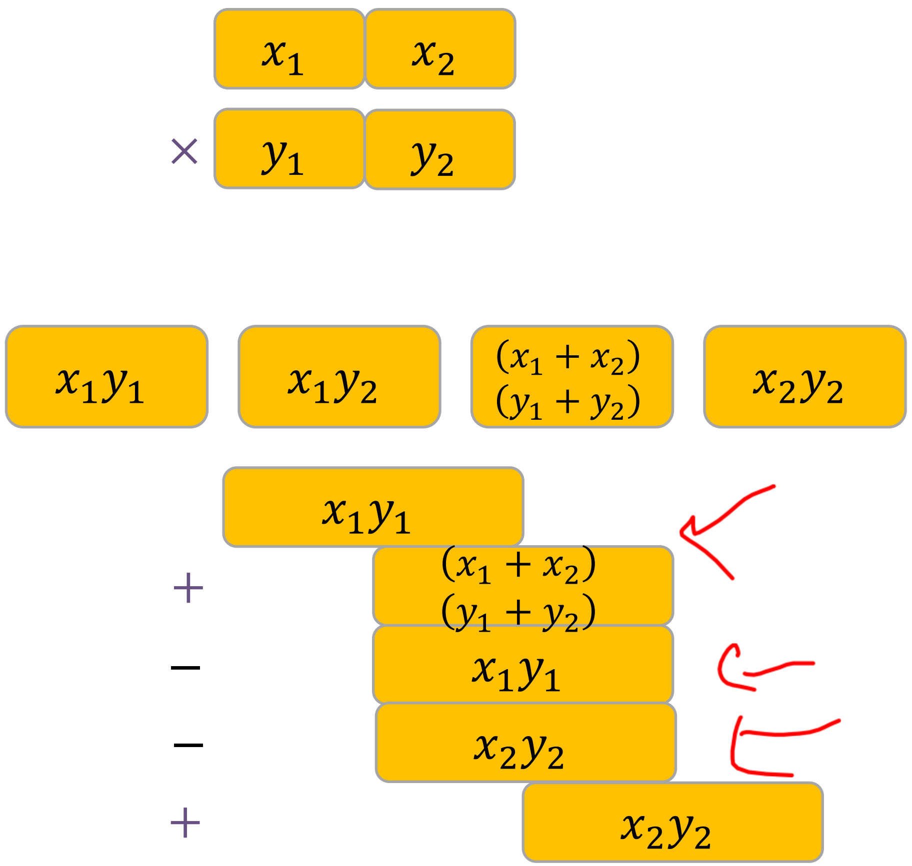
Two multiplications

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

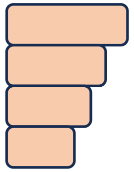
$$2^n(x_1y_1) + 2^{\frac{n}{2}}\big((x_1 + x_2)(y_1 + y_2) - x_1y_1 - x_2y_2\big) + (x_2y_2)$$



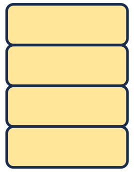
Karatsuba Method Recurrence Solution

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

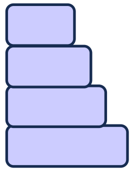
- If $a < b^k$ then $T(n)$ is $O(n^k)$
  - Cost is dominated by work at top level of recursion



- 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 recursion



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

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

Matrix Multiplication

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

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

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

Run time? $O(n^3)$

Multiplying Matrices

for $\boldsymbol{i} \leftarrow \mathbf{1}$ to $\boldsymbol{n}$

  for $\boldsymbol{j} \leftarrow \mathbf{1}$ to $\boldsymbol{n}$

    $\boldsymbol{C}[\boldsymbol{i}, \boldsymbol{j}] \leftarrow \mathbf{0}$

    for $\boldsymbol{k} \leftarrow \mathbf{1}$ to $\boldsymbol{n}$

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

    endfor

  endfor

endfor

Can we improve this with divide and conquer?

We can see subproblems!

A_{11}

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

B_{11}

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

$A \times B =$

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

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

Matrix Multiplication D&C

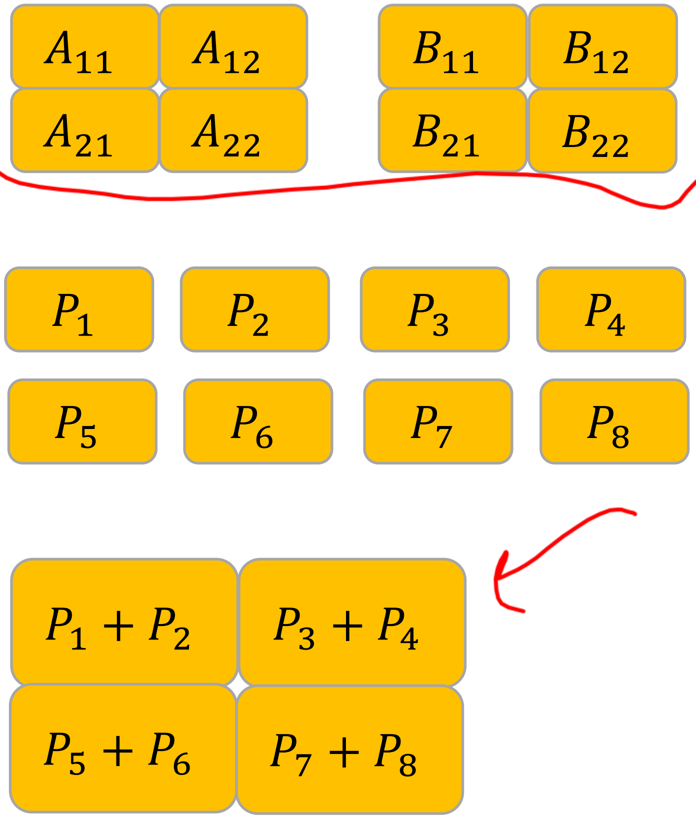
Multiply $n \times n$ matrices (A and B)

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

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

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

Divide and Conquer Matrix Multiplication



- **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:

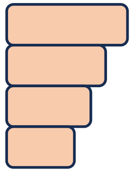
$$P_1 = A_{11} \times B_{11} \qquad P_5 = A_{21} \times B_{11}$$
$$P_2 = A_{12} \times B_{21} \qquad P_6 = A_{22} \times B_{21}$$
$$P_3 = A_{11} \times B_{12} \qquad P_7 = A_{21} \times B_{12}$$
$$P_4 = A_{12} \times B_{22} \qquad P_8 = A_{22} \times B_{22}$$

- **Combine**:
  - Compute the value of each quadrant by summing $P_1 \ldots P_8$ as shown

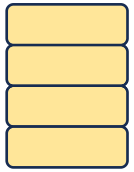
Karatsuba Method Recurrence Solution

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

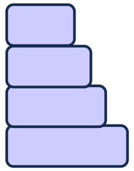
- If $\boldsymbol{a < b^k}$ then $\boldsymbol{T(n)}$ is $\boldsymbol{O(n^k)}$
  - Cost is dominated by work at top level of recursion



- 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 recursion



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

$\boldsymbol{a = 8}$, $\boldsymbol{b = 2}$, $\boldsymbol{k = 2}$, so $\boldsymbol{a > b^k}$: 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)

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

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

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:

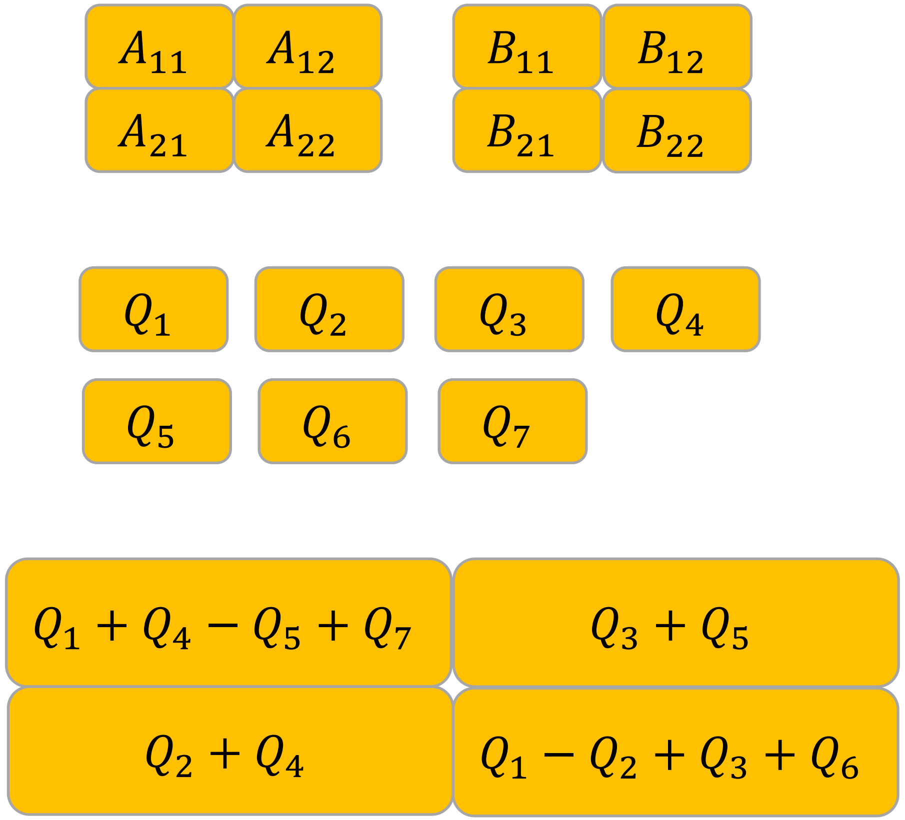
$$Q_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$
$$Q_2 = (A_{21} + A_{22}) \times B_{11}$$
$$Q_3 = A_{11} \times (B_{12} - B_{22})$$
$$Q_4 = A_{22} \times (B_{21} - B_{11})$$
$$Q_5 = (A_{11} + A_{12}) \times B_{22}$$
$$Q_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$
$$Q_7 = (A_{12} - A_{22}) \times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

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 Matrix Multiplication



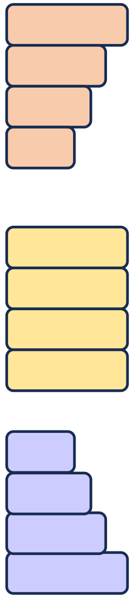
- **Base Case**:
  - For a 32×32 matrices, use the textbook algorithm
- **Divide**:
  - Use each quadrant of the input $n \times n$ matrices as it's own $\frac{n}{2} \times \frac{n}{2}$matrix
- **Conquer**:
  - Compute each of:

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

- **Combine**:
  - Compute the value of each quadrant by summing $Q_1 \ldots Q_8$ as shown

Karatsuba Method Recurrence Solution

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

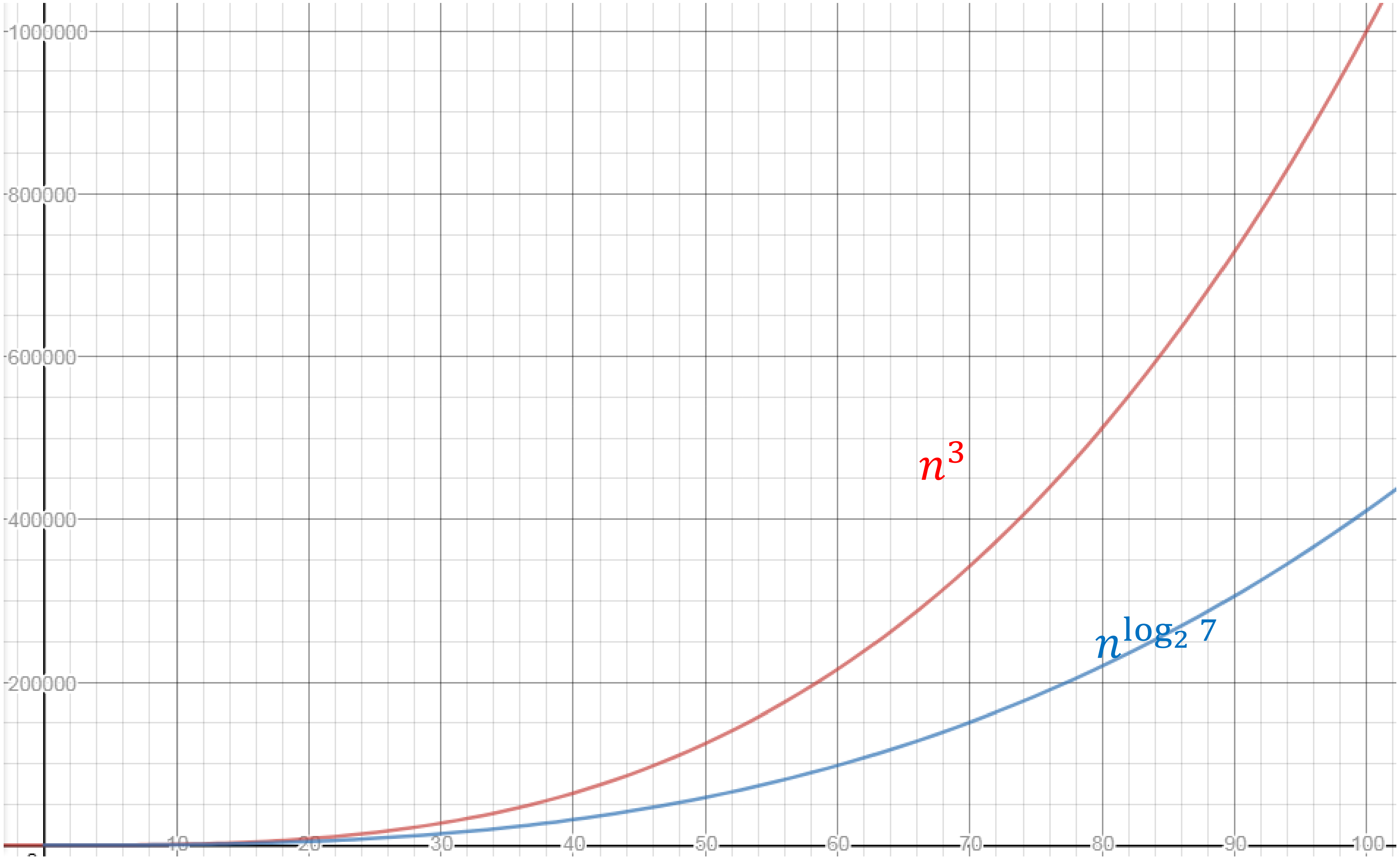


- If $a < b^k$ then $T(n)$ is $O(n^k)$
  - Cost is dominated by work at top level of recursion
- If $a = b^k$ then $T(n)$ is $O(n^k \log n)$
  - Total cost is the same for all $\log_b n$ levels of recursion
- If $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$
  - Note that $\log_b a > k$ in this case
  - Cost is dominated by total work at lowest level of recursion

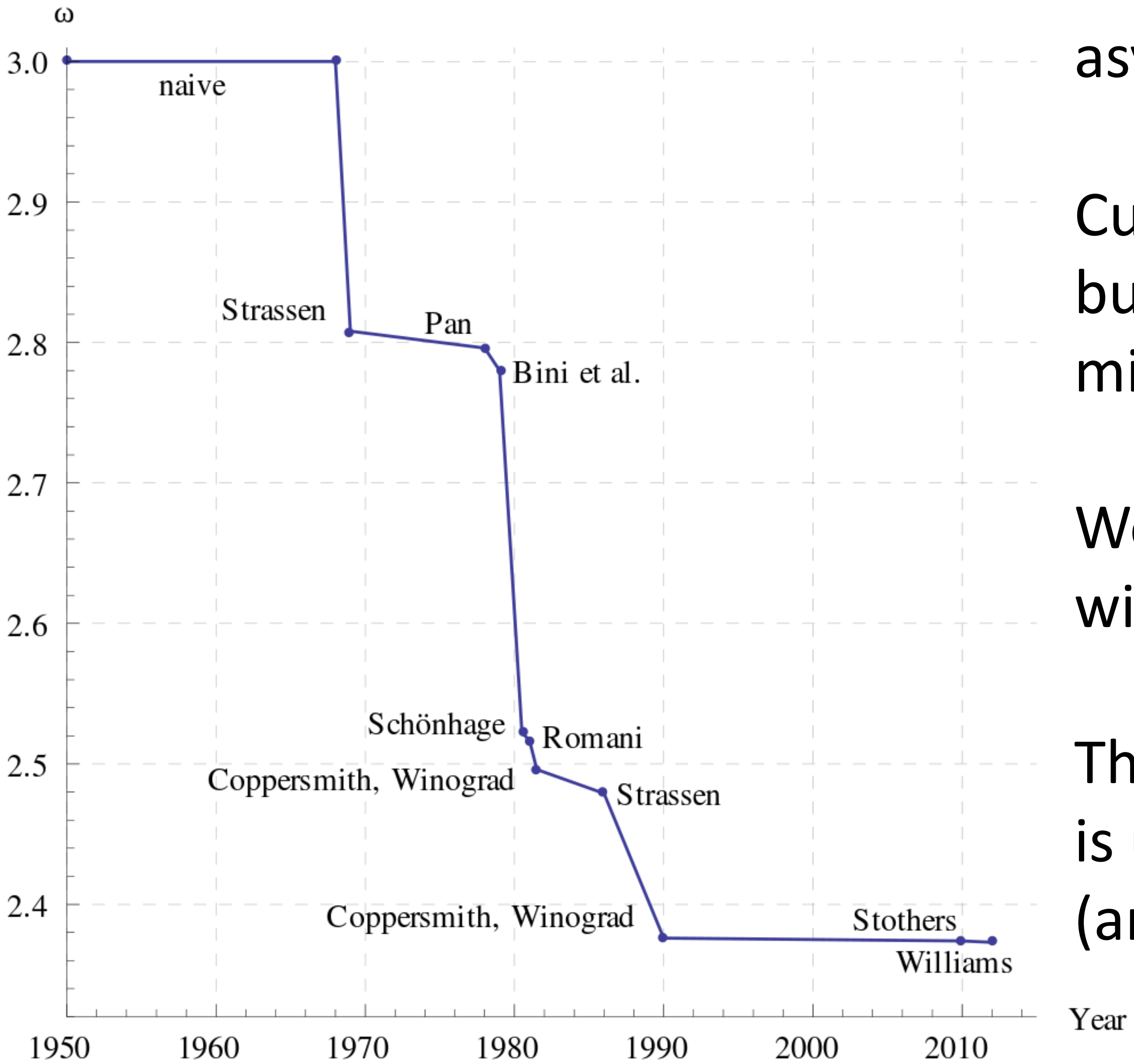
$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

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

Strassen's Algorithm



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!)