# Divide-and-Conquer Algorithms

## Part Three

# Announcements

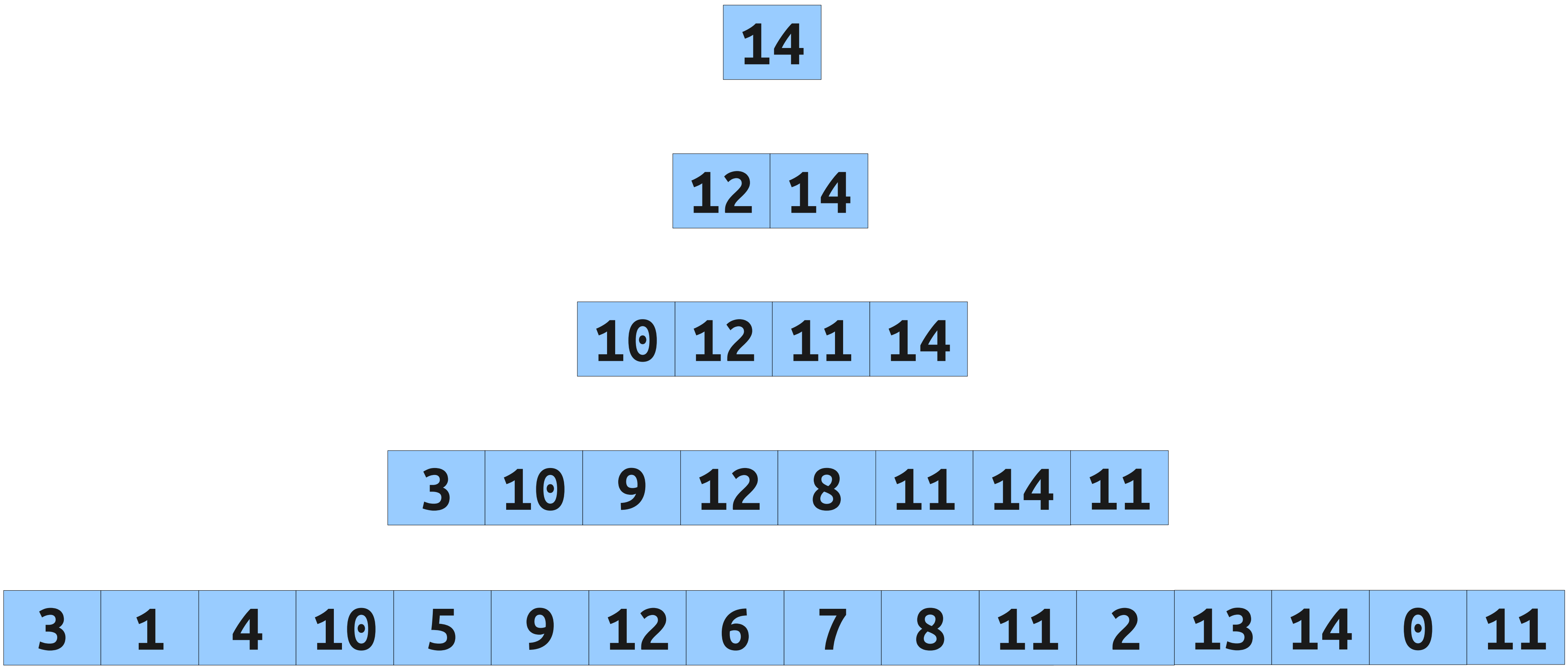
- Problem Set One graded; will be returned at the end of lecture.
  - If you submitted by email, let us know if you don't hear back by 5PM today.
  - If you submitted through the SCPD office, we'll return your problem set through the SCPD office.
- Handout: “Mathematical Terms and Identities.”
  - Covers useful mathematical definitions, terms, and identities that we'll be using over the rest of the quarter.
  - Let us know if there's anything you'd like us to add for future quarters!

# Outline for Today

- **The Master Theorem**
  - A powerful tool for solving recurrences.
- **Applications of the Master Theorem**
  - Rapidly solving a variety of recurrence relations!

# One More Recurrence Relation

# Finding the Maximum Value




$$T(1) \leq c$$
$$T(n) \leq T(n / 2) + cn$$

$$\begin{aligned} & cn + cn / 2 + \ldots + c \\ = \; & cn\,(1 + \tfrac{1}{2} + \ldots + {}^{1}/_{n}) \\ \leq \; & cn\,(1 + \tfrac{1}{2} + \tfrac{1}{4} + \ldots) \\ = \; & 2cn = \mathbf{O}(\boldsymbol{n}) \end{aligned}$$

# Three Recurrences

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = T(\lceil n / 2 \rceil) + T(\lfloor n / 2 \rfloor) + \Theta(n)$$

Solves to $O(n \log n)$

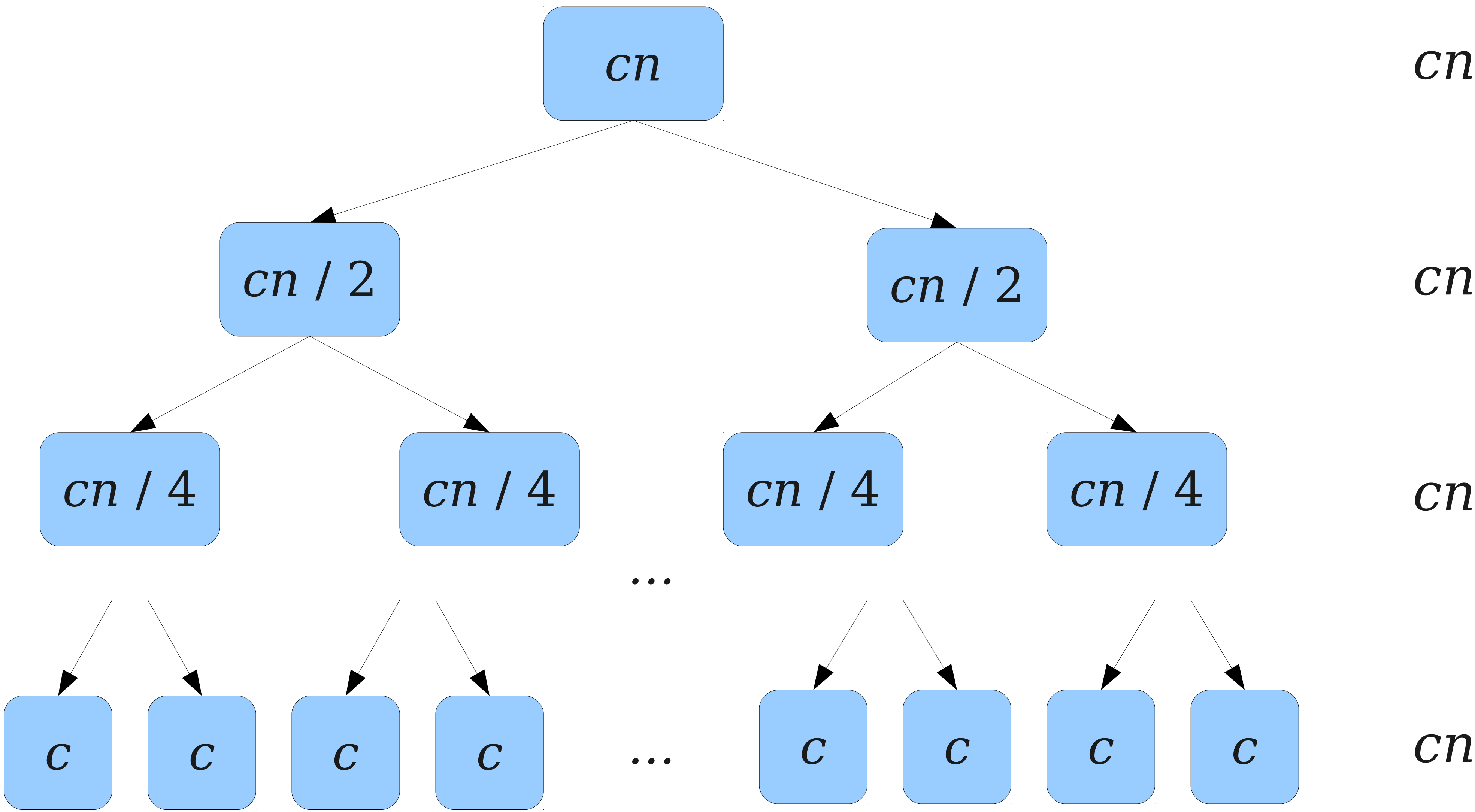
$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = T(\lceil n / 2 \rceil) + T(\lfloor n / 2 \rfloor) + \Theta(1)$$

Solves to $O(n)$

$$T(1) = \Theta(1)$$
$$T(n) = T(\lceil n / 2 \rceil) + \Theta(n)$$

Solves to $O(n)$

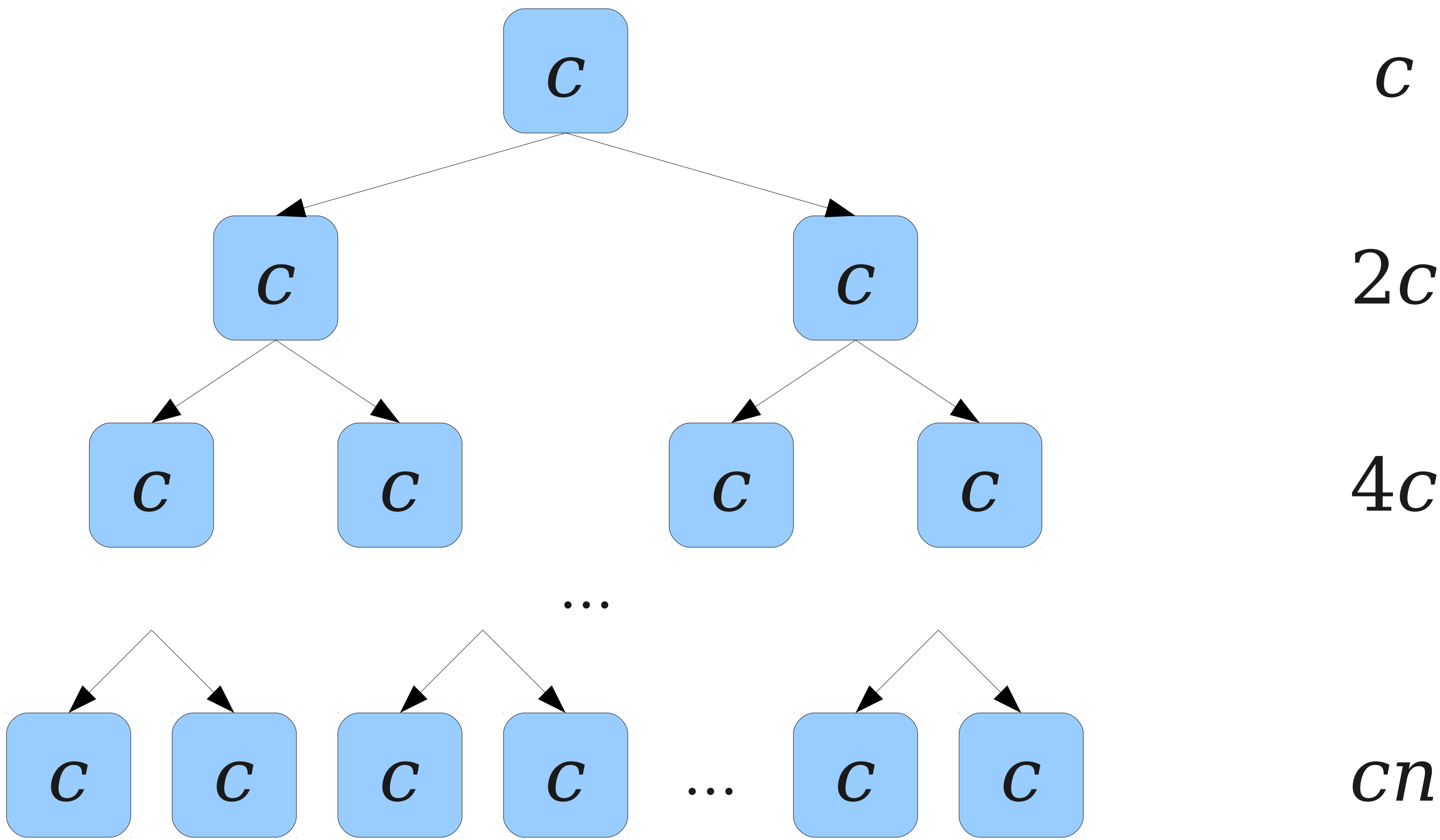
$$T(1) \leq c$$
$$T(n) \leq 2T(n / 2) + cn$$




$$O(n \log n)$$

$T(1) \leq c$

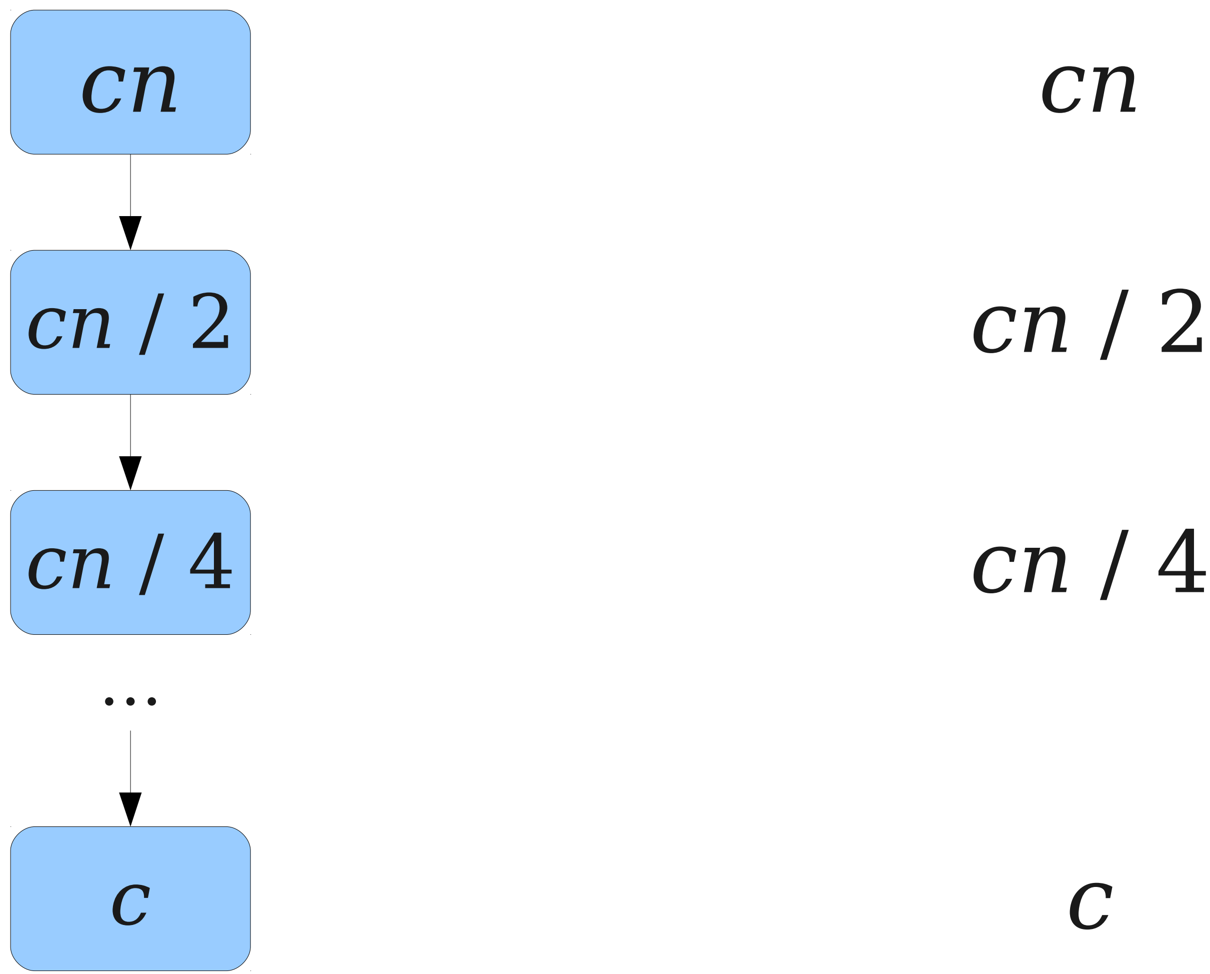
$T(n) \leq 2T(n / 2) + c$




$O(n)$

$T(1) \leq c$

$T(n) \leq T(n / 2) + cn$




$O(n)$

# Categorizing Recurrences

- The recurrences we have seen so far can be categorized into three groups:
  - **Topheavy recurrences**, where the majority of the runtime is dominating by the initial call.
    - Runtime is dominated by initial call.
  - **Balanced recurrences**, where each level in the tree does the same amount of work.
    - Runtime is determined by number of layers times the work per layer.
  - **Bottomheavy recurrences**, where the majority of the runtime is accounted for in the leaves.
    - Runtime is dominated by the work per leaf times the number of leaves.

# The Master Theorem

- The **Master Theorem** (given on the next slide) is a theorem for asymptotically bounding recurrences of the type we've seen so far.
- Intuitively, categorizes recurrences into one of the three groups just mentioned, then determines the runtime based on that category.

# The Master Theorem

***Theorem:*** Let $T(n)$ be defined as follows:

$$T(1) \leq \Theta(1)$$
$$T(n) \leq aT(\lceil n / b \rceil) + O(n^d)$$

Then

$$T(n) = \begin{cases} O\left(n^d\right) & \text{if } \log_b a < d \\ O\left(n^d \log n\right) & \text{if } \log_b a = d \\ O\left(n^{\log_b a}\right) & \text{if } \log_b a > d \end{cases}$$

# Solving Existing Recurrences

- Consider the mergesort recurrence

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) \leq 2T(\lceil n / 2 \rceil) + \Theta(n)$$

- What are $a$, $b$, and $d$? $\boldsymbol{a = 2, b = 2, d = 1}$.
- What is $\log_b a$? **1**
- By the Master Theorem, $T(n) = \mathbf{O(n \log n)}$.

# Solving Existing Recurrences

- Consider the weakly unimodal maximum recurrence:

$$T(1) \leq c$$
$$T(n) \leq 2T(\lceil n / 2 \rceil) + c$$

- What are $a$, $b$, $d$? $\boldsymbol{a = 2, b = 2, d = 0}$
- What is $\log_b a$? **1**
- By the Master Theorem, $T(n) = \boldsymbol{O(n)}$

# Solving Existing Recurrences

- Consider the recurrence for the code to find the maximum value in an array:

$$T(1) \leq c$$
$$T(n) \leq T(\lceil n / 2 \rceil) + cn$$

- What are $a$, $b$, $d$? $\boldsymbol{a = 1, b = 2, d = 1}$
- What is $\log_b a$? **0**
- By the Master Theorem, $T(n) = \mathbf{O(\boldsymbol{n})}$

# Proving the Master Theorem

- We can prove the Master Theorem by writing out a generic proof using a recursion tree.
  - Draw out the tree.
  - Determine the work per level.
  - Sum across all levels.
- The three cases of the Master Theorem correspond to whether the recurrence is topheavy, balanced, or bottomheavy.

# Simplifying the Recurrence

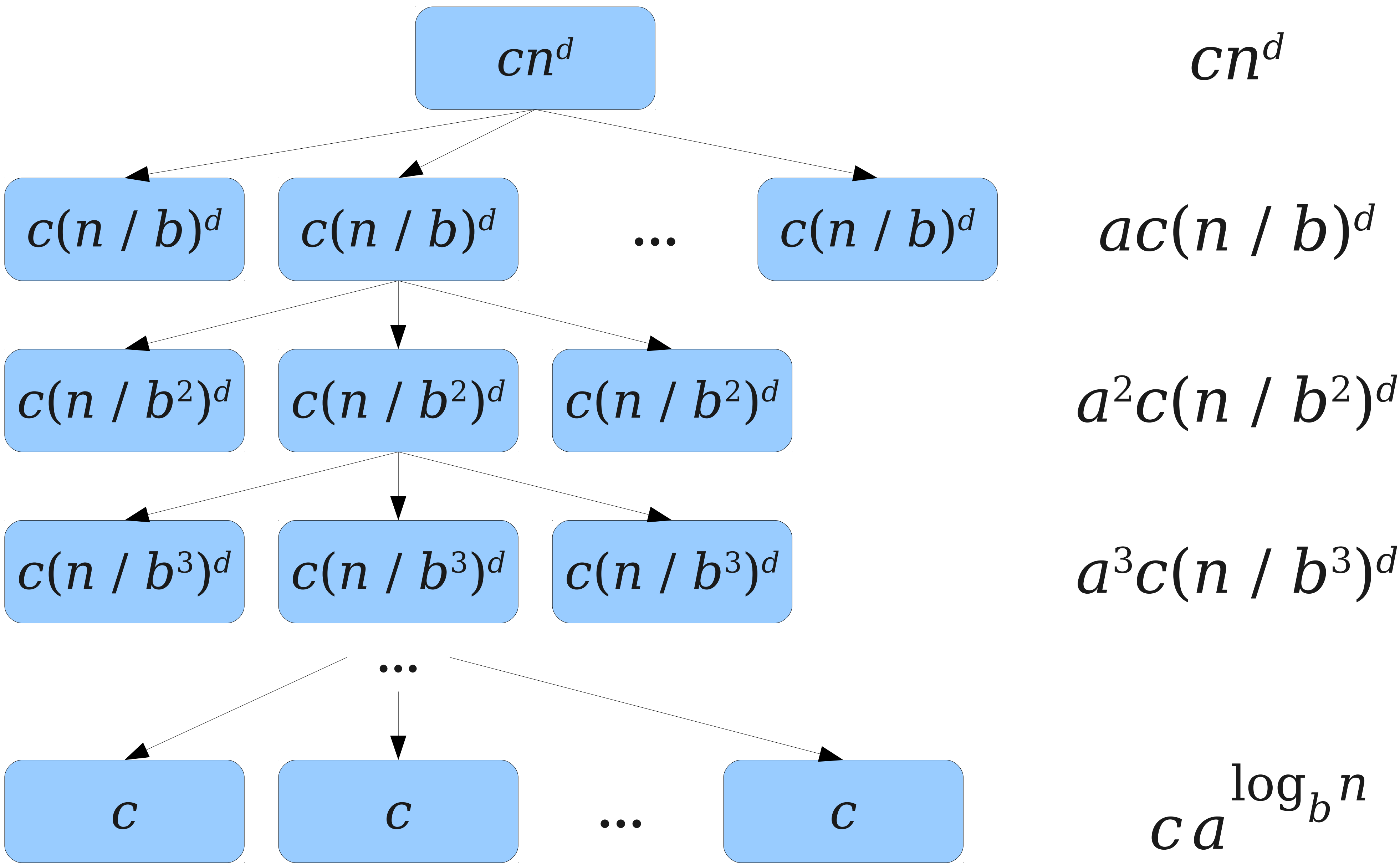
- The recurrence given by the Master Theorem is shown here:

$$T(1) \leq \Theta(1)$$
$$T(n) \leq a T(\lceil n / b \rceil) + O(n^d)$$

- We will apply our standard simplifications to this recurrence:
  - Assume inputs are powers of *b*.
  - Replace Θ and O with constant multiples.

$$T(1) \leq c$$
$$T(n) \leq a T(n / b) + c n^d$$

$$T(1) \le c$$
$$T(n) \le aT(n / b) + cn^d$$



# Hairy Scary Math

- At internal level $k$ of the tree, the work done is

$$a^k c(n / b^k)^d$$

- Rearranging:

$$\begin{aligned} a^k c (n / b^k)^d &= cn^d a^k / b^{dk} \\ &= cn^d (a / b^d)^k \end{aligned}$$

- Therefore:

$$\begin{aligned} T(n) &\le c a^{\log_b n} + \sum_{k=0}^{\log_b n - 1} c n^d \left(\frac{a}{b^d}\right)^k \\ &= c a^{\log_b n} + c n^d \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^k \end{aligned}$$

# Icky Tricky Math

- Let's see if we can simplify

$$\mathrm{T}(n) \leq c\,a^{\log_b n} + \sum_{k=0}^{c} n^d \log_b n - 1 \left(\frac{a}{b^d}\right)^k$$

- Let's look at the first term. Note that

$$\begin{aligned} a^{\log_b n} &= \left(b^{\log_b a}\right)^{\log_b n} \\ &= b^{(\log_b a)(\log_b n)} \\ &= \left(b^{\log_b n}\right)^{\log_b a} \\ &= n^{\log_b a} \end{aligned}$$

so $\quad \mathrm{T}(n) \leq c\,n^{\log_b a} + c\,n^d \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^k$

# Frightening Enlightening Math

- All that's left to do now is to simplify

$$T(n) \leq cn^{\log_b a} + cn^d \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^k$$

- **Case 1:** What if $a / b^d = 1$? Then $\log_b a = d$, so

$$\begin{aligned} T(n) &\leq cn^d + cn^d \sum_{k=0}^{\log_b n - 1} 1 \\ &= cn^d + cn^d \log_b n \\ &= O(n^d \log n) \end{aligned}$$

# Frightening Enlightening Math

- All that's left to do now is to simplify

$$T(n) \leq cn^{\log_b a} + cn^d \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^k$$

- **Case 2:** What if $a / b^d < 1$? Then $\log_b a < d$, so

$$\begin{aligned} T(n) &< cn^d + cn^d \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^k \\ &< cn^d + cn^d \sum_{k=0}^{\infty} \left(\frac{a}{b^d}\right)^k \\ &< cn^d \left(1 + \frac{1}{1 - a/b^d}\right) \\ &= O(n^d) \end{aligned}$$

**Case 3:** What if $a / b^d > 1$? Then $\log_b a > d$, so

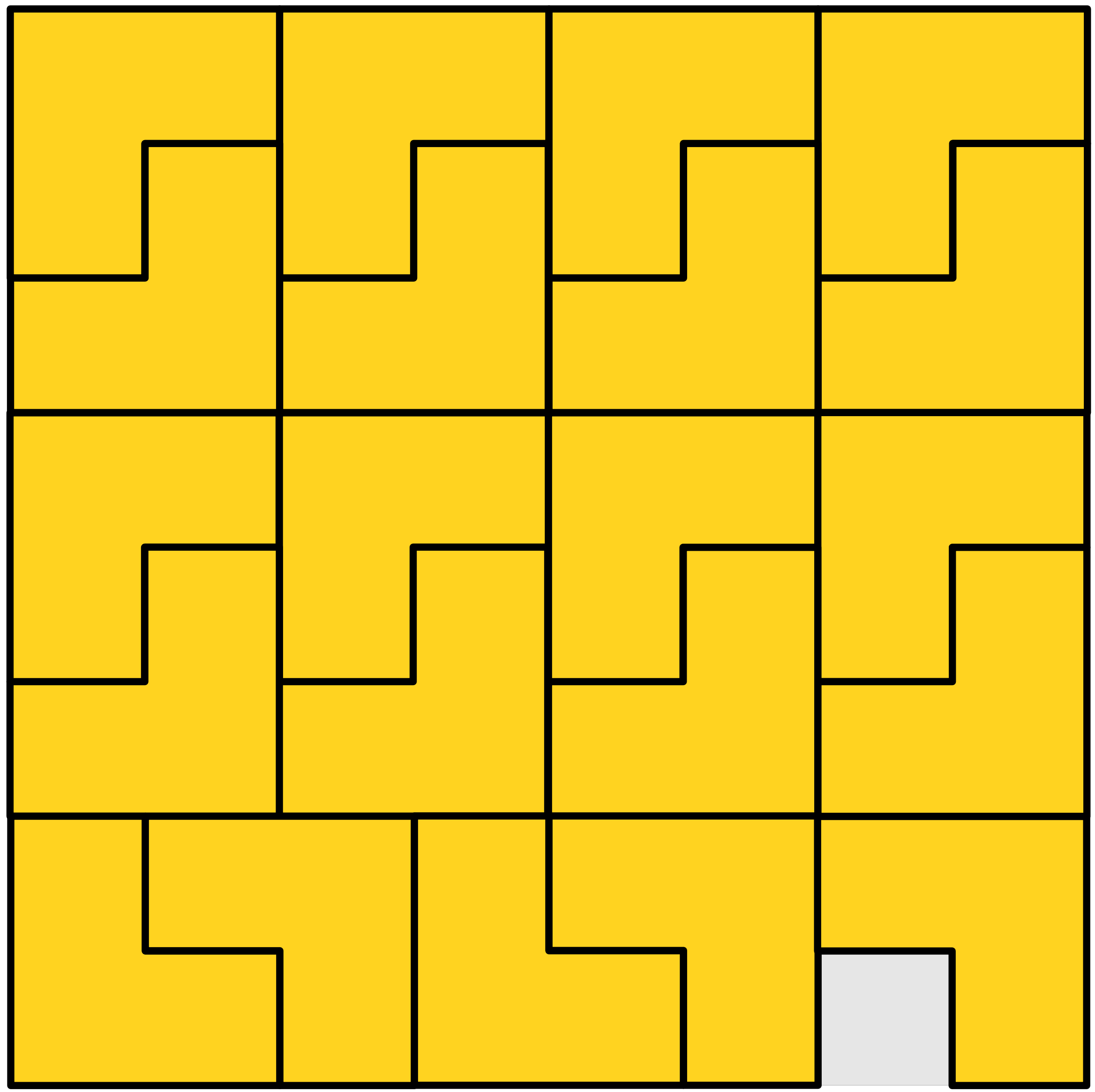
$$
\begin{aligned}
T(n) &\leq cn^{\log_b a} + cn^d \sum_{k=0}^{\log_b n - 1} \left(\frac{a}{b^d}\right)^k \\
&= cn^{\log_b a} + cn^d \frac{(a/b^d)^{\log_b n} - 1}{(a/b^d) - 1} \\
&< cn^{\log_b a} + cn^d (a/b^d)^{\log_b n} \frac{1}{(a/b^d) - 1} \\
&= cn^{\log_b a} + cn^d (a/b^d)^{\log_b n} \Theta(1) \\
&= cn^{\log_b a} + cn^d (a^{\log_b n} / b^{d \log_b n}) \Theta(1) \\
&= cn^{\log_b a} + cn^d (n^{\log_b a} / n^d) \Theta(1) \\
&= cn^{\log_b a} + cn^{\log_b a} \Theta(1) \\
&= O(n^{\log_b a})
\end{aligned}
$$

# Why the Master Theorem Matters

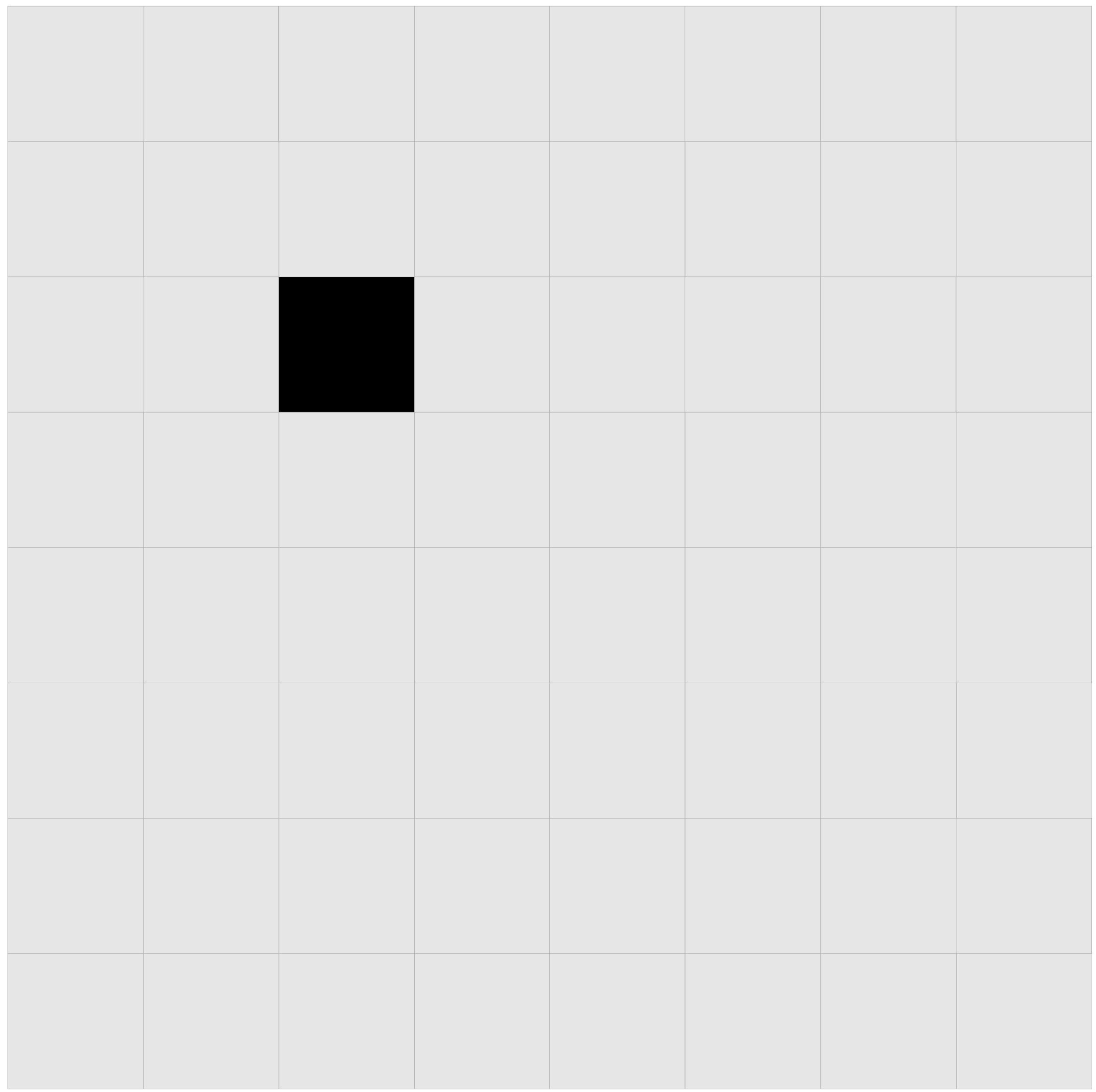
- The proof of the Master Theorem can be thought of as a single proof that works for all recurrences of the form handled by the theorem.
- From this point forward, we can just call back to the Master Theorem when applicable.
- Not all recurrences can be solved by the Master Theorem; more on that next time.

# Applications of the Master Theorem: A Sampler of Algorithms

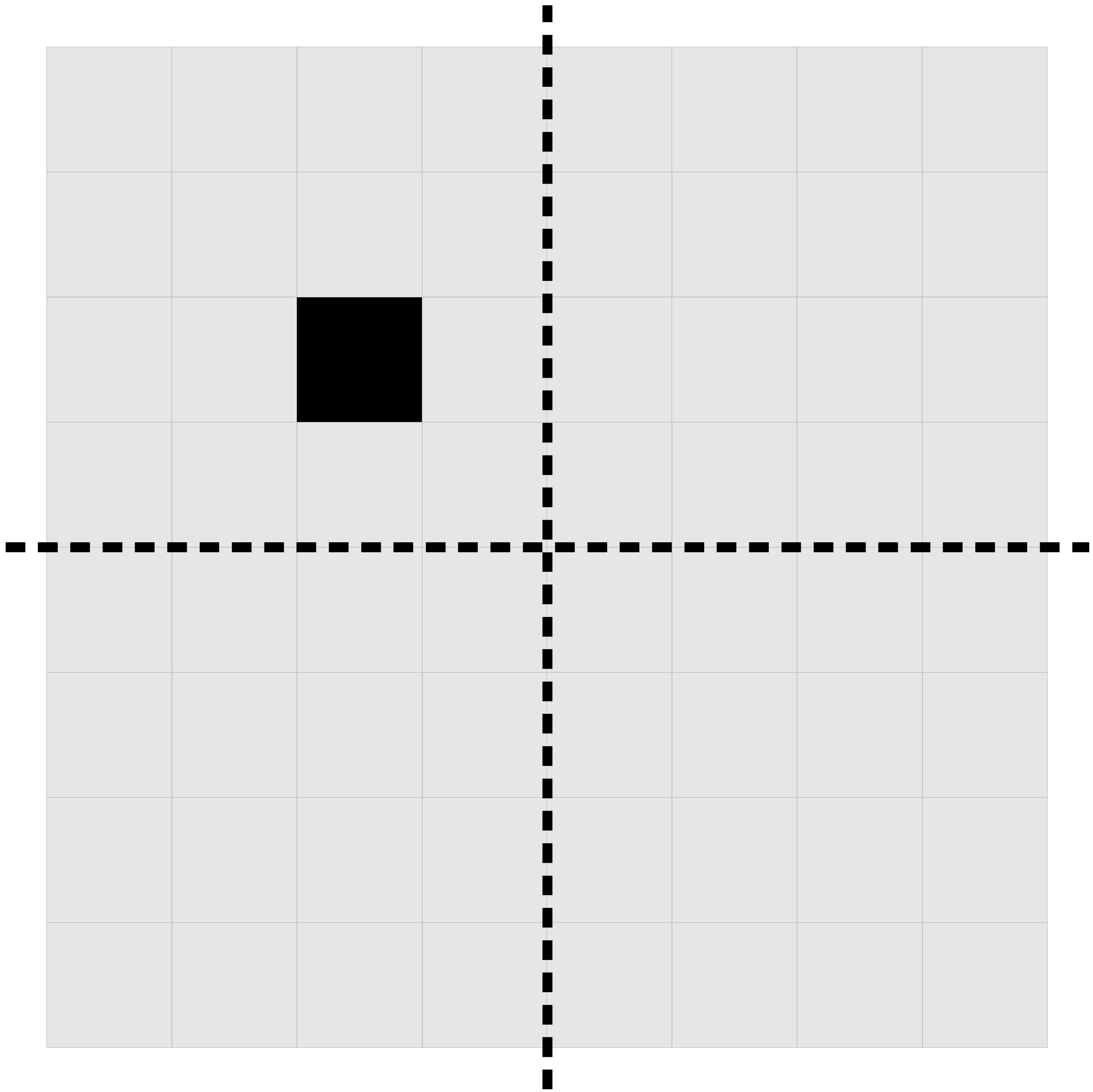
# Tiling with Triominoes



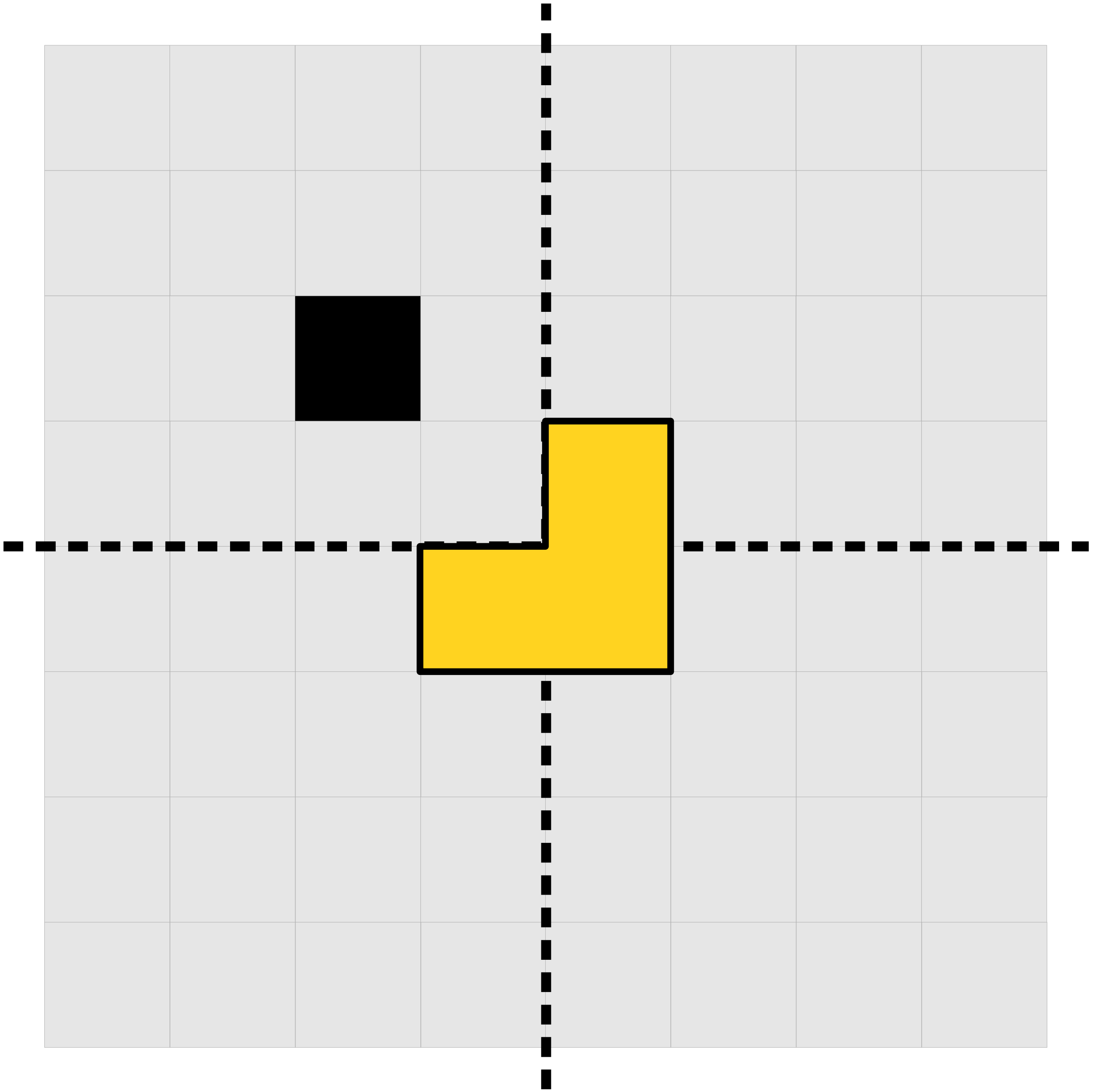
# Tiling with Triominoes



# Tiling with Triominoes



# Tiling with Triominoes



# Tiling with Triominoes

- To tile a $2^k \times 2^k$ board missing a single square, do the following:
  - If the board has size $1 \times 1$, is has no uncovered squares (because one square is missing) and we're done.
  - Otherwise, place a triomino in the center to cover up one square from each quadrant that isn't missing a square, then recursively fill in the four smaller squares.

$$T(1) = \Theta(1)$$
$$T(n) = 4T(n / 2) + \Theta(1)$$

# Solving the Recurrence

- We have the recurrence

$$T(1) = \Theta(1)$$
$$T(n) = 4T(n / 2) + \Theta(1)$$

- What are *a*, *b*, and *d*?
- What is $\log_b a$?
- What runtime do we get from the Master Theorem?
- Does that make sense?

# Searching a Grid, Take Two

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 10 | 12 | 13 | 21 | 32 | 34 | 43 | 51 |
| 16 | 21 | 23 | 26 | 40 | 54 | 65 | 67 |
| 21 | 23 | 31 | 33 | 54 | 58 | 74 | 77 |
| 32 | 46 | 59 | 65 | 74 | 88 | 99 | 103 |
| 53 | 75 | 96 | 115 | 124 | 131 | 132 | 136 |
| 85 | 86 | 98 | 145 | 146 | 151 | 173 | 187 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 10 | 12 | 13 | 21 | 32 | 34 | 43 | 51 |
| 16 | 21 | 23 | 26 | 40 | 54 | 65 | 67 |
| 21 | 23 | 31 | 33 | 54 | 58 | 74 | 77 |
| 32 | 46 | 59 | 65 | 74 | 88 | 99 | 103 |
| 53 | 75 | 96 | 115 | 124 | 131 | 132 | 136 |
| 85 | 86 | 98 | 145 | 146 | 151 | 173 | 187 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 10 | 12 | 13 | 21 | 32 | 34 | 43 | 51 |
| 16 | 21 | 23 | 26 | 40 | 54 | 65 | 67 |
| 21 | 23 | 31 | 33 | 54 | 58 | 74 | 77 |
| 32 | 46 | 59 | 65 | 74 | 88 | 99 | 103 |
| 53 | 75 | 96 | 115 | 124 | 131 | 132 | 136 |
| 85 | 86 | 98 | 145 | 146 | 151 | 173 | 187 |

$T(0) = \Theta(1)$

$T(1) = \Theta(1)$

$T(Z) \leq 3T(\lceil Z / 4 \rceil) + \Theta(1)$

# Recursive Sorted Searching

- We now have the recurrence

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(Z) \leq 3T(\lceil Z / 4 \rceil) + \Theta(1)$$

- What are *a*, *b*, and *d*?
- What does this recurrence solve to?
- Since $T(Z) = O(Z^{\log_4 3})$, the runtime is **$O((mn)^{\log_4 3})$** $\approx O((mn)^{0.79})$

# One More Example:
# **Integer Multiplication**

# Some Efficiency Claims

- Claim: The following can be done in $\Theta(1)$ time:
  - Multiplying two one-digit numbers.
  - Adding two one-digit numbers.
- Suppose that $A$ and $B$ have $n$ digits each. Then these operations have the following costs:
  - Computing $A + B$: $\Theta(n)$
  - Computing $A - B$: $\Theta(n)$
  - Computing $A \cdot 10^k$: $O(n + k)$
  - Computing $A \bmod 10^k$: $O(n + k)$

# Algorism Efficiency

- Recall: **Algorism** refers to place-value arithmetic.
- What is the cost of computing $A \cdot B$, where $A$ and $B$ are $n$-digit numbers?
  - Does $\Theta(n)$ rounds of the following:
    - Multiply each digit in $A$ by a digit in $B$: $\Theta(n)$ time, including time to carry across columns.
    - Shift the resulting number $O(n)$ places: $O(n)$ time.
  - $\Theta(n)$ additions of $O(n)$-digit numbers: time $\Theta(n^2)$.
  - Overall runtime: $\boldsymbol{\Theta(n^2)}$.

# A Quick History Lesson

# Multiplying with Divide-and-Conquer

- Suppose that you want to multiply together two numbers $X$ and $Y$, both of which are $n$ digits long.
- Write

$$X = a \cdot 10^{\lfloor n / 2 \rfloor} + b$$

$$Y = c \cdot 10^{\lfloor n / 2 \rfloor} + d$$

where $b, d < 10^{\lfloor n / 2 \rfloor}$

- If $X = 13579$ and $Y = 24680$, what are $a$, $b$, $c$ and $d$?

# Multiplying with Divide-and-Conquer

- If $X = a \cdot 10^{\lfloor n / 2 \rfloor} + b$ and $Y = c \cdot 10^{\lfloor n / 2 \rfloor} + d$, then

$$\begin{aligned} X \cdot Y &= (a \cdot 10^{\lfloor n / 2 \rfloor} + b) \cdot (c \cdot 10^{\lfloor n / 2 \rfloor} + d) \\ &= ac \cdot 10^{2\lfloor n / 2 \rfloor} + ad \cdot 10^{\lfloor n / 2 \rfloor} + bc \cdot 10^{\lfloor n / 2 \rfloor} + bd \\ &= ac \cdot 10^{2\lfloor n / 2 \rfloor} + (ad + bc) \cdot 10^{\lfloor n / 2 \rfloor} + bd \end{aligned}$$

- What is the cost of directly evaluating this expression?
  - Does 4 multiplications on numbers with $\lceil n / 2 \rceil$ digits.
  - Does three additions of numbers with $O(n)$ digits.
  - Does two multiplications by powers of ten, each of which takes $O(n)$ time.

$$T(1) = \Theta(1)$$
$$T(n) = 4T(\lceil n / 2 \rceil) + O(n)$$

# Solving the Recurrence

- We now have the recurrence

$$T(1) = \Theta(1)$$
$$T(n) = 4T(\lceil n / 2 \rceil) + O(n)$$

- What does the Master Theorem say?
- Runtime is $O(n^2)$. But that's no better than before…

# Karatsuba's Observation

- Karatsuba arrived at this expression:

  $$X \cdot Y = \boldsymbol{ac} \cdot 10^{2\lfloor n/2 \rfloor} + (\boldsymbol{ad + bc}) \cdot 10^{\lfloor n/2 \rfloor} + \boldsymbol{bd}$$

- Karatsuba's key question: Is it possible to compute ***ac***, ***ad + bc***, and ***bd*** without making four multiplications?

# Karatsuba's Observation

- Consider these three products:

$$E = ac$$

$$F = bd$$

$$G = (a + b)(c + d) = ac + ad + bc + bd$$

- We can compute these values with two additions and three multiplications.
- Note that

$$ac = E$$

$$bd = F$$

$$ad + bc = G - E - F$$

# Karatsuba's Algorithm

- Write $\boldsymbol{X = a \cdot 10^{\lfloor n/2 \rfloor} + b}$ and $\boldsymbol{Y = c \cdot 10^{\lfloor n/2 \rfloor} + d}$
- Recursively compute

$$\boldsymbol{E = ac \qquad F = bd \qquad G = (a + b)(c + d)}$$

- Then

$$\boldsymbol{X \cdot Y = E \cdot 10^{2\lfloor n/2 \rfloor} + (G - E - F) \cdot 10^{\lfloor n/2 \rfloor} + F}$$

- Does two multiplications by powers of ten ($O(n)$ each), four additions ($O(n)$ each), two subtractions ($O(n)$ each), and three recursive multiplies on numbers with at most $\lceil n / 2 \rceil$ digits.

$T(1) = \Theta(1)$
$T(n) = 3T(\lceil n / 2 \rceil) + O(n)$

# Karatsuba's Algorithm

- We now have the recurrence

$$T(1) = \Theta(1)$$
$$T(n) = 3T(\lceil n / 2 \rceil) + O(n)$$

- What does the Master Theorem tell us?
- Runtime is $O(n^{\log_2 3}) \approx$ **$O(n^{1.585})$**
- This is asymptotically better than the normal algorithm!
- **Standard algorism is not the optimal algorism algorithm!**

# After Karatsuba

- Several other algorithms for multiplying numbers have arisen since Karatsuba's algorithm.
- **Toom-Cook** uses a similar set of techniques to multiply $n$-digit numbers in time $O(n^{\log_3 5})$.
- **Schönhage–Strassen** uses a completely different approach (based on the fast Fourier transform) to achieve $O(n \log n \log \log n)$ runtime.
- Recently (2008), **Fürer's algorithm** achieved runtime $n \log n \, 2^{O(\log^* n)}$, where $\log^* n$ is an *extremely* slowly-growing function.
- **Finding an optimal multiplication algorithm is still an open problem!**

# Next Time

- The Selection Problem
- The Median of Medians Algorithm
- The Substitution Method