Lecture 8: Solving Recurrences

CSE 373: Data Structures and Algorithms

Warm Up!

Please fill out the Poll at- pollev.com/21sp373

What's the theta bound for the runtime function for this piece of code?

```
public void method1(int n) {
    if (n <= 100) {
        System.out.println(":3");
    } else {
        System.out.println(":D");
        for (int i = 0; i<16; i++) {
            method1(n / 4);
        }
    }
}
```

$$T(n) = \begin{cases} \text{constant work} & \text{if } n \leq 100 \\ 16T\left(\frac{n}{4}\right) + \text{constant work} & \text{otherwise} \end{cases}$$

a = 16, b = 4, c = 0

$\log_4 16 = 2$

$\log_4 16 > 0$

$T(n) \in \Theta\left(n^{\log_b a}\right)$

$\Theta\left(n^{\log_4 16}\right) = \boldsymbol{\Theta\left(n^2\right)}$

Master Theorem

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

Where $f(n)$ is $\Theta(n^c)$

If $\log_b a < c$ then $T(n) \in \Theta(n^c)$

If $\log_b a = c$ then $T(n) \in \Theta(n^c \log n)$

If $\log_b a > c$ then $T(n) \in \Theta\left(n^{\log_b a}\right)$

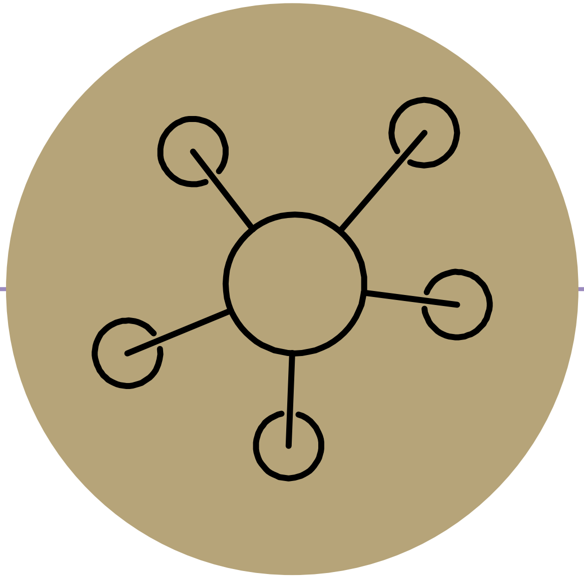
Announcements

Exercise 1 – Algorithm Analysis – Due Friday April 16th

Project 1 – Deques – Due Wednesday April 14th

Project 2 Goes out this Friday, due Wednesday April 28th

Midterm goes out Friday April 30th



Questions

Modeling Recursive Code

Meet the Recurrence

A recurrence relation is an equation that defines a sequence based on a rule that gives the next term as a function of the previous term(s)

It's a lot like recursive code:

- At least one base case and at least one recursive case
- Each case should include the values for n to which it corresponds
- The recursive case should reduce the input size in a way that eventually triggers the base case
- The cases of your recurrence usually correspond exactly to the cases of the code

$$T(n) = \begin{cases} 5 & \text{if } n < 3 \\ 2T\left(\frac{n}{2}\right) + 10 & \text{otherwise} \end{cases}$$

Recursive Patterns

Modeling and analyzing recursive code is all about finding patterns in how the input changes between calls and how much work is done within each call

Let's explore some of the more common recursive patterns

Pattern #1: Halving the Input

Pattern #2: Constant size input and doing work

Pattern #3: Doubling the Input

Review Why Include Non-Recursive Work?

```
public int recurse(int n) {
  if (n < 3) {
    return 80;
  }
  for (int i = 0; i < n; i++) {
    System.out.println(i);
  }
  int val1 = recurse(n / 3);
  int val2 = recurse(n / 3);
  int val3 = recurse(n / 3);

  return val1 + val2 + val3;
}
```

Base Case

Recursive Case

+n

+3

Think of it this way:

"work that happens if we enter base case"

$$T(n) = \begin{cases} 2 & \text{if } n < 3 \\ 3T\left(\frac{n}{3}\right) + n & \text{otherwise} \end{cases}$$

"work that happens if we enter recursive case"

Non-recursive parts of recursive cases are sometimes where the bulk of the work takes place!

Recurrence to Big-Θ

$$T(n) = \begin{cases} 2 & \text{if } n < 3 \\ 2T\left(\frac{n}{3}\right) + n & \text{otherwise} \end{cases}$$

It's still really hard to tell what the big-O is just by looking at it.

But fancy mathematicians have a formula for us to use!

Master Theorem

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

Where $f(n)$ is $\Theta(n^c)$

If $\log_b a < c$ then $T(n) \in \Theta(n^c)$

If $\log_b a = c$ then $T(n) \in \Theta(n^c \log n)$

If $\log_b a > c$ then $T(n) \in \Theta\left(n^{\log_b a}\right)$

a=2 b=3 and c=1

$y = \log_b x$ *is equal to* $b^y = x$

$\log_3 2 = x \Rightarrow 3^x = 2 \Rightarrow x \cong 0.63$

$\log_3 2 < 1$

We're in case 1

$T(n) \in \Theta(n)$

Understanding Master Theorem

Master Theorem

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

Where $f(n)$ is $\Theta(n^c)$

If $\log_b a < c$ then $T(n) \in \Theta(n^c)$

If $\log_b a = c$ then $T(n) \in \Theta(n^c \log n)$

If $\log_b a > c$ then $T(n) \in \Theta(n^{\log_b a})$

- A measures how many recursive calls are triggered by each method instance
- B measures the rate of change for input
- C measures the dominating term of the non recursive work within the recursive method
- D measures the work done in the base case

The $\log_b a < c$ case

- Recursive case does a lot of non recursive work in comparison to how quickly it divides the input size
- Most work happens in beginning of call stack
- Non recursive work in recursive case dominates growth, n^c term

The $\log_b a = c$ case

- Recursive case evenly splits work between non recursive work and passing along inputs to subsequent recursive calls
- Work is distributed across call stack

The $\log_b a > c$ case

- Recursive case breaks inputs apart quickly and doesn't do much non recursive work
- Most work happens near bottom of call stack

Recursive Patterns

Pattern #1: Halving the Input

Binary Search $\Theta(\log n)$

Pattern #2: Constant size input and doing work

Merge Sort

Pattern #3: Doubling the Input

Merge Sort

Divide

0	1	2	3	4	5	6	7	8	9
8	2	91	22	57	1	10	6	7	4

0	1	2	3	4
8	2	91	22	57

5	6	7	8	9
1	10	6	7	4

Conquer

0
8

0
8

Combine

0	1	2	3	4
2	8	22	57	91

5	6	7	8	9
1	4	6	7	10

0	1	2	3	4	5	6	7	8	9
1	2	4	6	7	8	10	22	57	91

Merge Sort

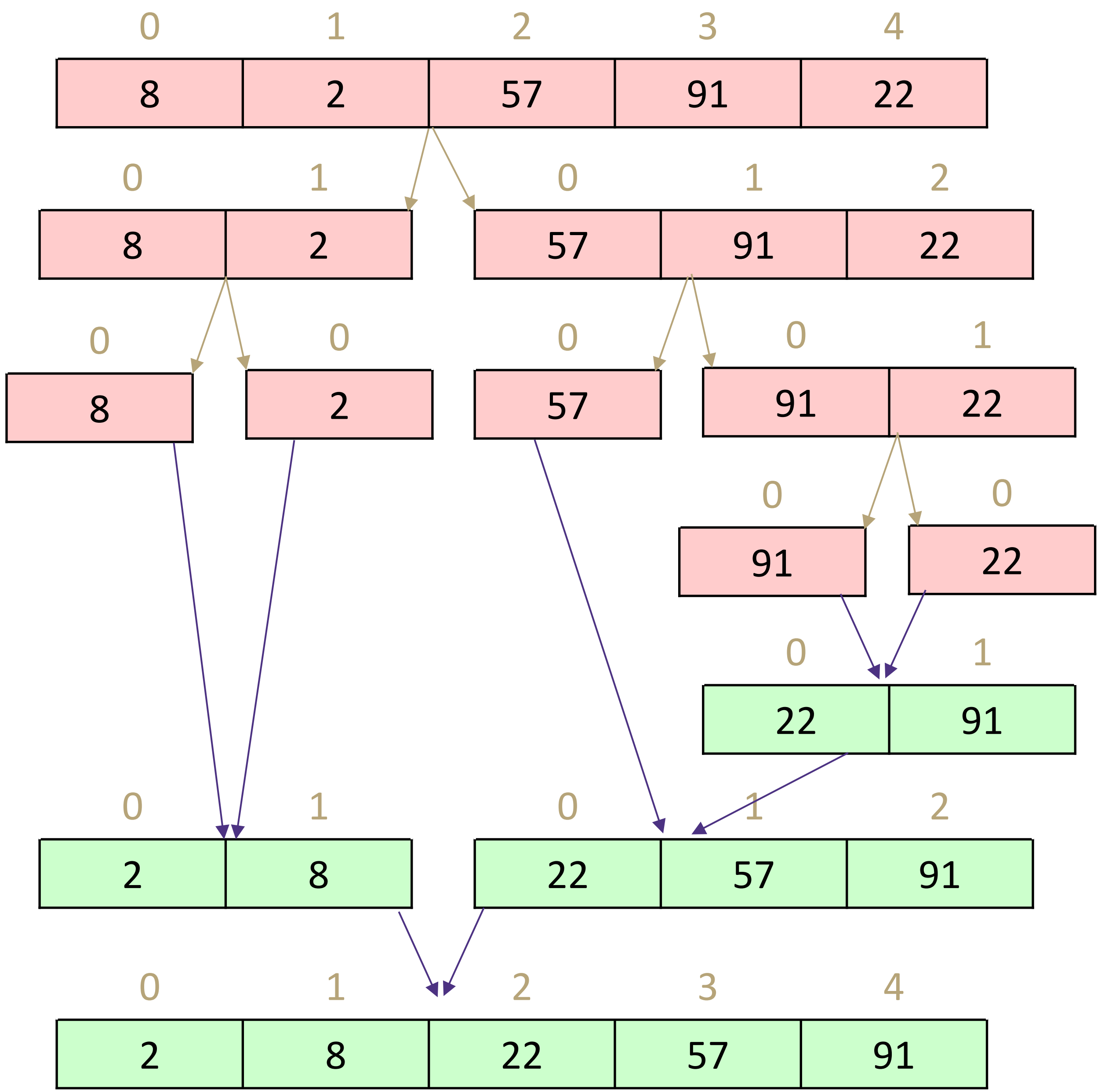
```
mergeSort(input) {
   if (input.length == 1)
      return
   else
      smallerHalf = mergeSort(new [0, ..., mid])
      largerHalf = mergeSort(new [mid + 1, ...])
      return merge(smallerHalf, largerHalf)
}
```

$$T(n) = \begin{cases} 1 \text{ if } n <= 1 \\ 2T(n/2) + n \text{ otherwise} \end{cases}$$

Pattern #2 – Constant size input and doing work

Take 1 min to respond to activity

www.pollev.conm/cse373activity
Take a guess! What is the Big-O of worst case merge sort?



Merge Sort Recurrence to Big-Θ

$$T(n) = \begin{cases} 1 \text{ if } n <= 1 \\ 2T(n/2) + n \text{ otherwise} \end{cases}$$

<u>Master Theorem</u>

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

Where $f(n)$ is $\Theta(n^c)$

If $\log_b a < c$ then $T(n) \in \Theta(n^c)$

If $\log_b a = c$ then $T(n) \in \Theta(n^c \log n)$

If $\log_b a > c$ then $T(n) \in \Theta\left(n^{\log_b a}\right)$

a=2 b=2 and c=1

$y = \log_b x$ *is equal to* $b^y = x$

$\log_2 2 = x \Rightarrow 2^x = 2 \Rightarrow x = 1$

$\log_2 2 = 1$

We're in case 2

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

Recursive Patterns

Pattern #1: Halving the Input

Binary Search Θ(logn)

Pattern #2: Constant size input and doing work

Merge Sort Θ(nlogn)

Pattern #3: Doubling the Input

Calculating Fibonacci

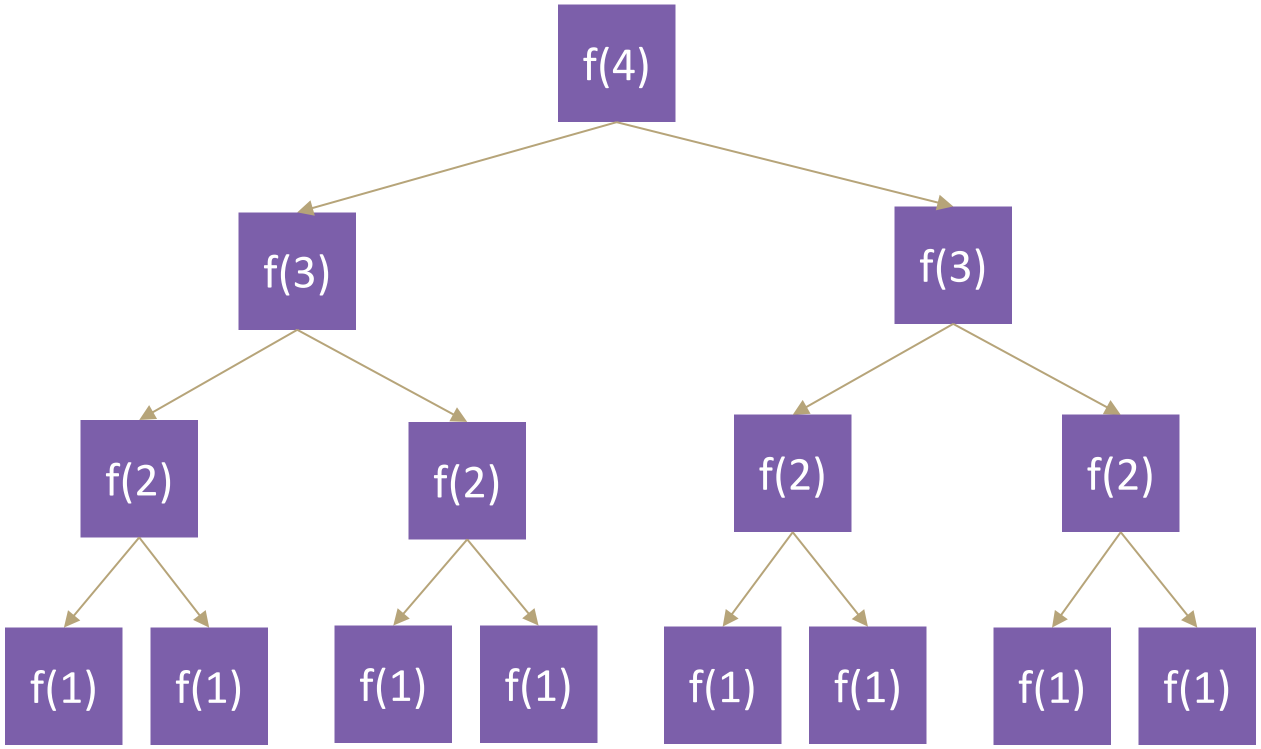
Calculating Fibonacci

```
public int fib(int n) {
    if (n <= 1) {
        return 1;
    }
    return fib(n-1) + fib(n-1);
}
```

- Each call creates 2 more calls
- Each new call has a copy of the input, almost
- Almost doubling the input at each call

Pattern #3 – Doubling the Input

Almost



Calculating Fibonacci Recurrence to Big-Θ

```
public int f(int n) {
    if (n <= 1) {
        return 1;
    }
    return f(n-1) + f(n-1);
}
```

d

2T(n-1) + c

$$T(n) = \begin{cases} d \text{ when } n \le 1 \\ 2T(n-1) + c \text{ otherwise} \end{cases}$$

Can we use master theorem?

<u>Master Theorem</u>

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

Uh oh, our model doesn't match that format…

- Can we intuit a pattern?
- T(1) = d
- T(2) = 2T(2-1) + c = 2(d) + c
- T(3) = 2T(3-1) + c = 2(2(d) + c) + c = 4d + 3c
- T(4) = 2T(4-1) + c = 2(4d + 3c) + c = 8d + 7c
- T(5) = 2T(5-1) + c = 2(8d + 7c) + c = 16d +25c
- Looks like something's happening but it's tough
  - Maybe geometry can help!

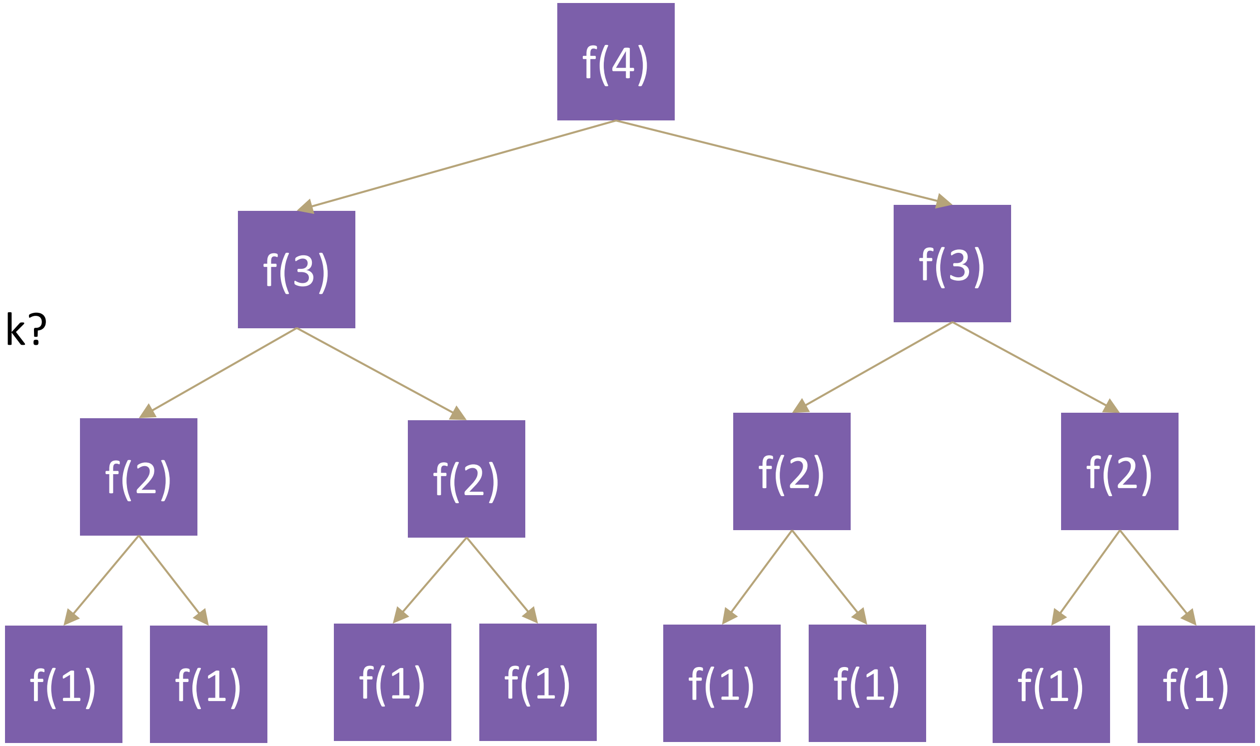
Calculating Fibonacci Recurrence to Big-Θ

How many layers in the function call tree?

How many layers will it take to transform "n" to the base case of "1" by subtracting 1

For our example, 4 -> Height = n

$$T(n) = \begin{cases} d \text{ when } n \leq 1 \\ 2T(n-1) + c \text{ otherwise} \end{cases}$$



How many function calls per layer?

Layer	Function calls
1	1
2	2
3	4
4	8

How many function calls on layer k?

2^{k-1}

How many function calls TOTAL for a tree of k layers?

$1 + 2 + 3 + 4 + \ldots + 2^{k-1}$

Calculating Fibonacci Recurrence to Big-Θ

Patterns found:

How many layers in the function call tree? n

How many function calls on layer k? 2^{k-1}

How many function calls TOTAL for a tree of k layers?

$1 + 2 + 4 + 8 + \ldots + 2^{k-1}$

Total runtime = (total function calls) x (runtime of each function call)

Total runtime = $(1 + 2 + 4 + 8 + \ldots + 2^{k-1})$ x (constant work)

$$1 + 2 + 4 + 8 + \ldots + 2^{k-1} = \sum_{i=1}^{k-1} 2^i = \frac{2^k - 1}{2 - 1} = 2^k - 1$$

Summation Identity
Finite Geometric Series

$$\sum_{i=1}^{k-1} x^i = \frac{x^k - 1}{x - 1}$$

$$\boldsymbol{T(n) = 2^n - 1 \in \Theta(2^n)}$$

Recursive Patterns

Pattern #1: Halving the Input

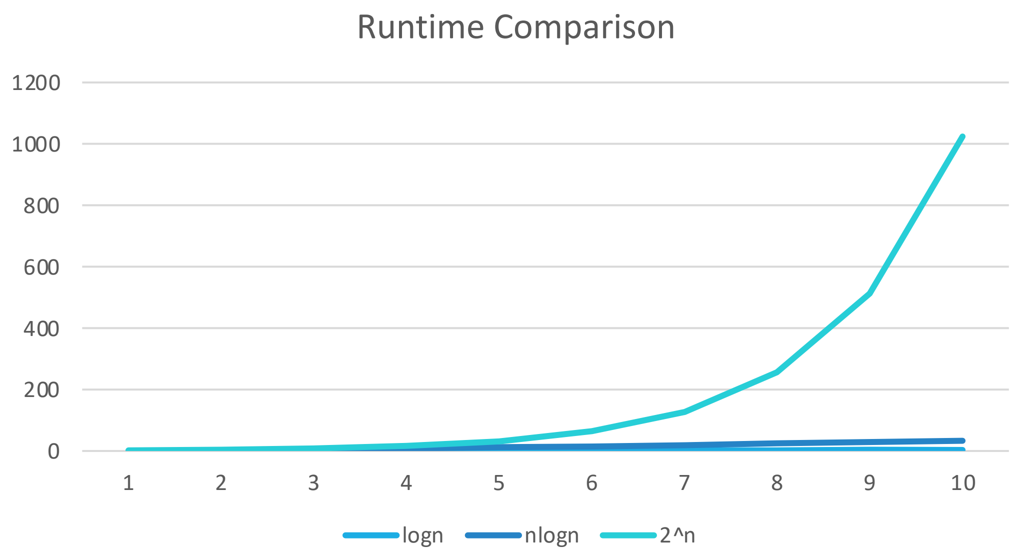
Binary Search Θ(logn)

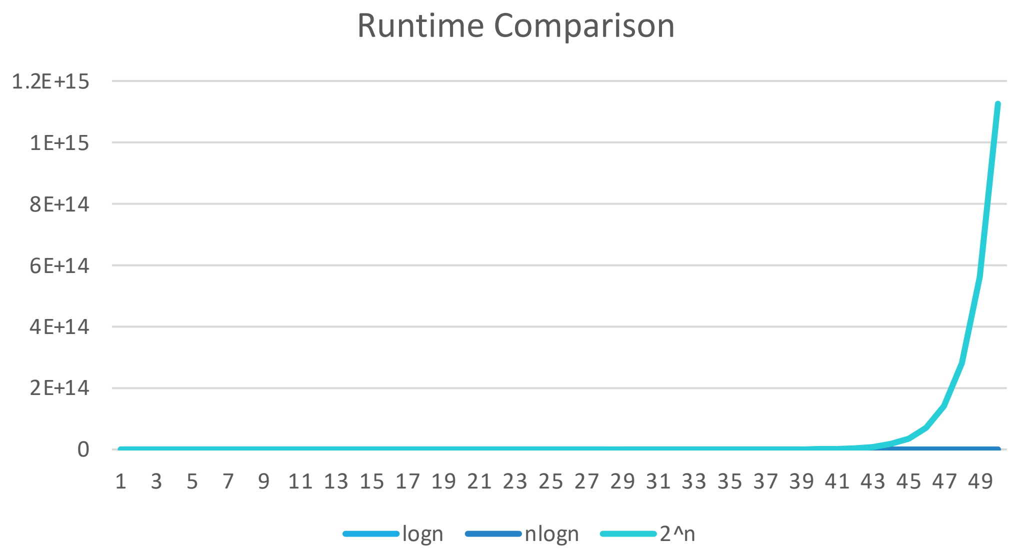
Pattern #2: Constant size input and doing work

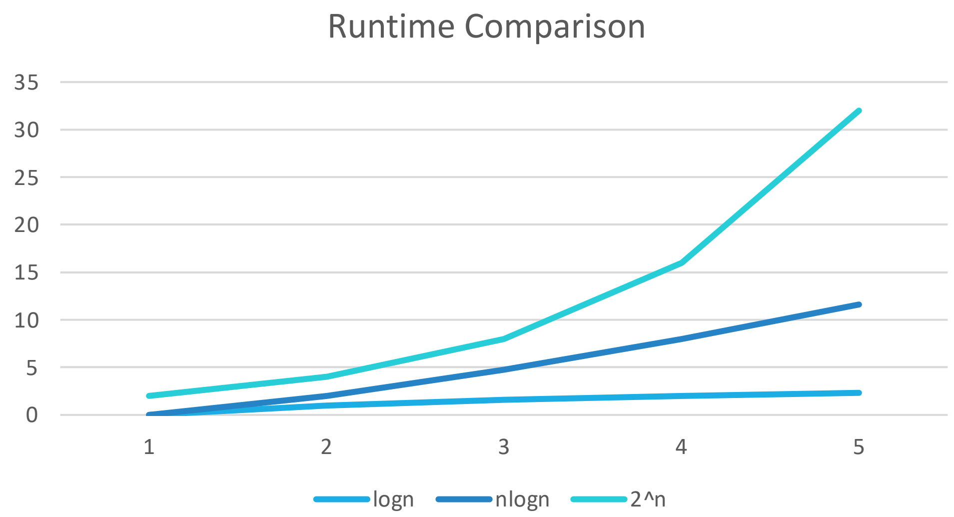
Merge Sort Θ(nlogn)

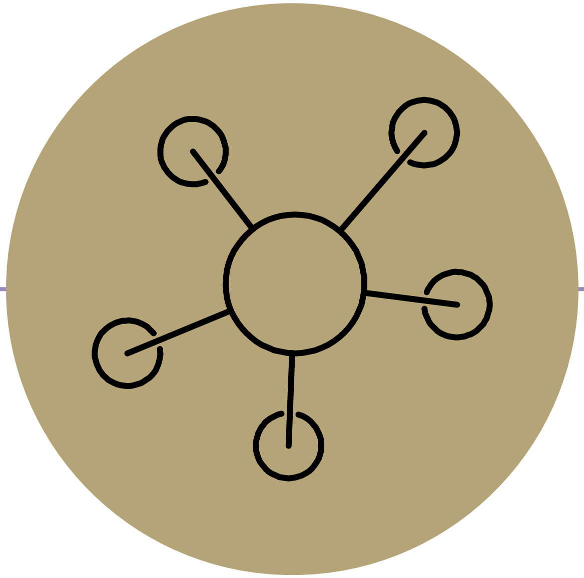
Pattern #3: Doubling the Input

Calculating Fibonacci Θ(2^n)



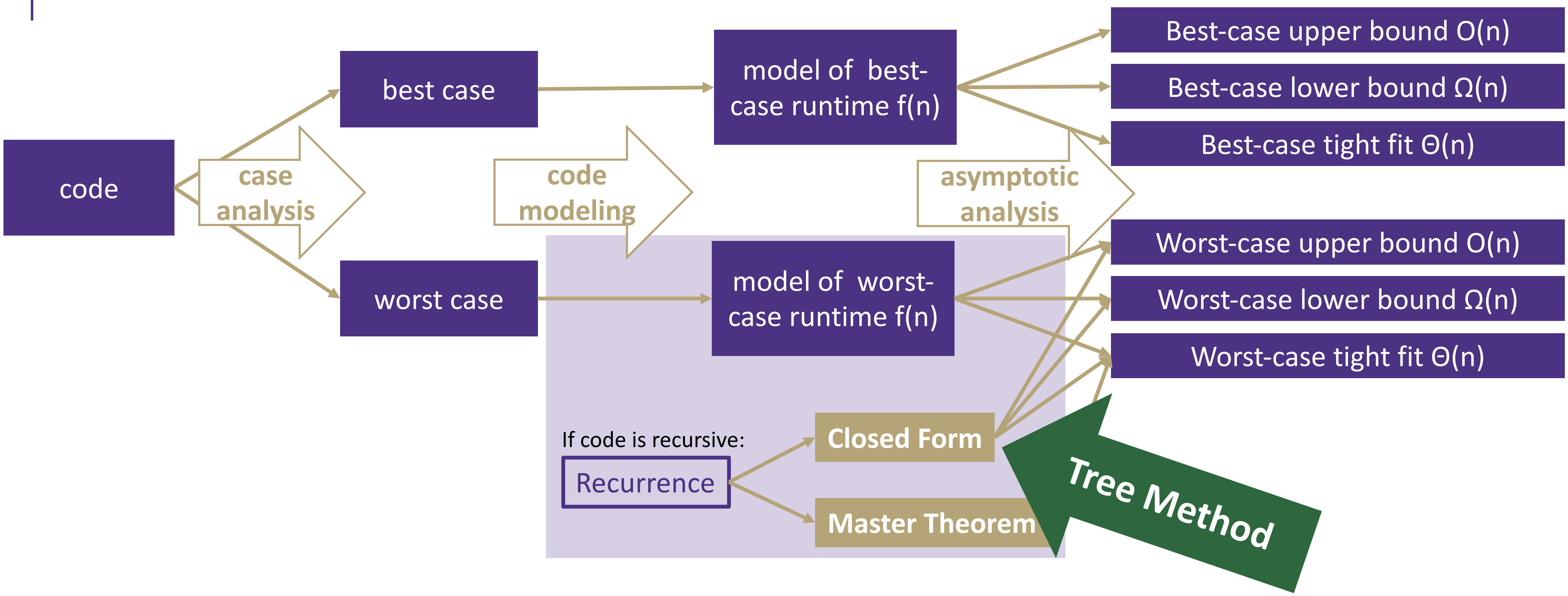






Questions?

Code Analysis Process



Recurrence to Big Θ Techniques

A recurrence is a mathematical function that includes itself in its definition

This makes it very difficult to find the dominating term that will dictate the asymptotic growth

Solving the recurrence or “finding the closed form” is the process of eliminating the recursive definition. So far, we’ve seen three methods to do so:

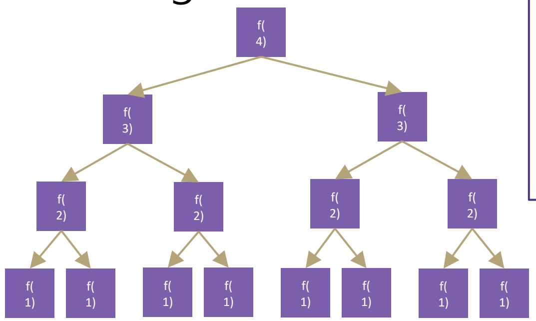
$$T(n) = \begin{cases} d \text{ when } n \leq 1 \\ 2T(n-1) + c \text{ otherwise} \end{cases}$$

1. Apply Master Theorem
   - Pro: Plug and chug convenience
   - Con: only works for recurrences of a certain format
2. Unrolling
   - Pro: Least complicated setup
   - Con: requires intuitive pattern matching
3. Tree Method
   - Pro: Plug and chug
   - Con: Complex setup

Master Theorem

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

T(1) = d
T(2) = 2T(2-1) + c = 2(d) + c
T(3) = 2T(3-1) + c = 2(2(d) + c) + c = 4d + 3c
T(4) = 2T(4-1) + c = 2(4d + 3c) + c = 8d + 7c
T(5) = 2T(5-1) + c = 2(8d + 7c) + c = 16d +25c



Tree Method

Draw out call stack, what is the input to each call? How much work is done by each call?

<u>How much work is done at each layer?</u>

64 for this example -> n work at each layer

Work is variable per layer, but across the entire layer work is constant - always n

<u>How many layers are in our function call tree?</u>

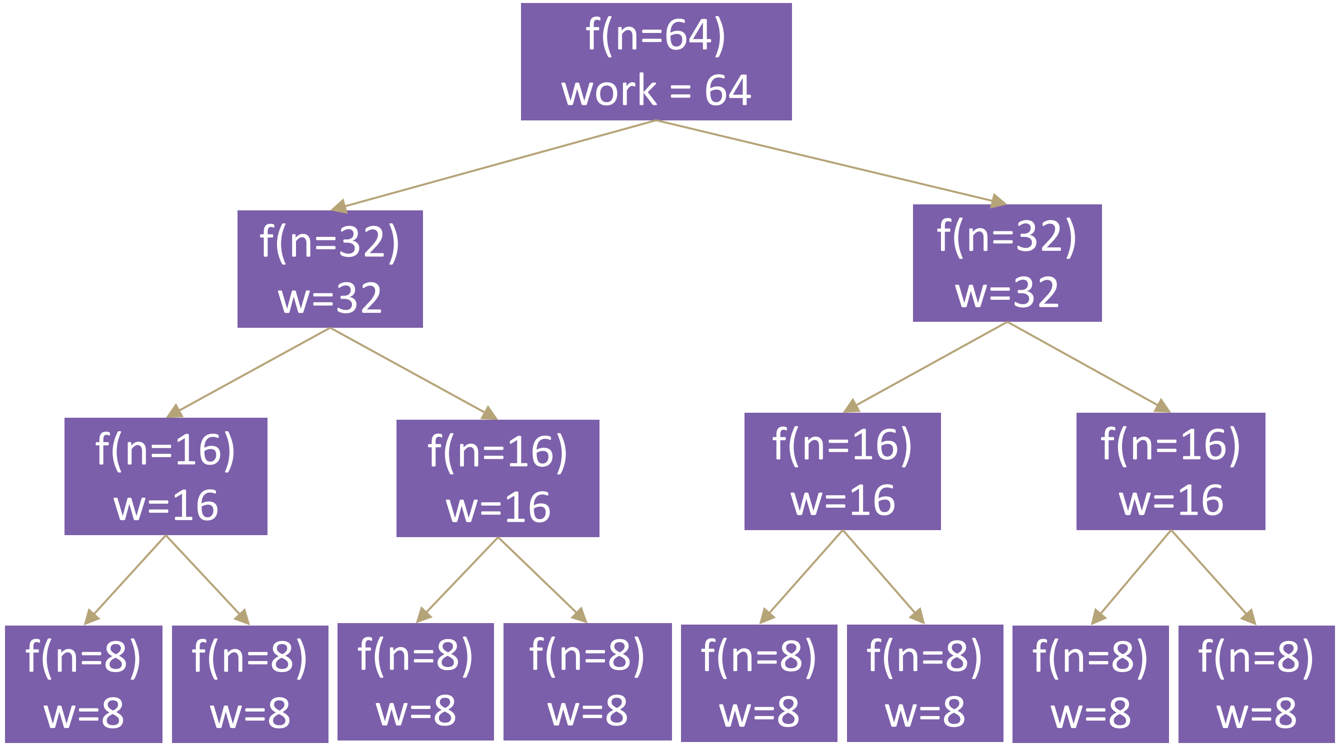
Hint: how many levels of recursive calls does it take *binary search* to get to the base case?

Height = $\log_2 n$

It takes $\log_2 n$ divisions by 2 for n to be reduced to the base case 1

$\log_2 64 = 6$ -> 6 levels of this tree

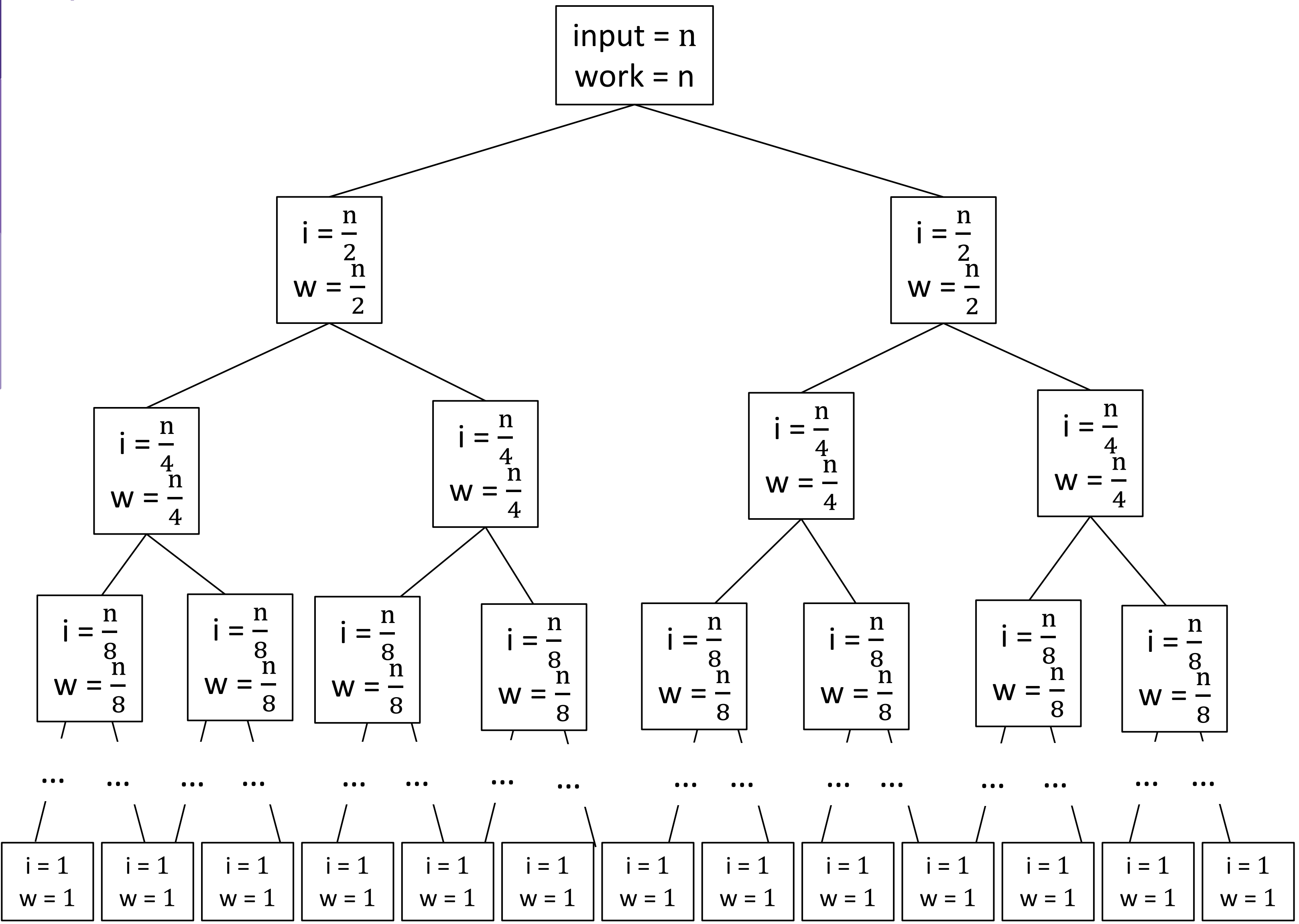
Merge Sort $T(n) = \begin{cases} 1 \text{ if } n <= 1 \\ 2T(n/2) + n \text{ otherwise} \end{cases}$



… and so on…

Tree Method

$$T(n) = \begin{cases} 1 \text{ when } n \le 1 \\ 2T\left(\frac{n}{2}\right) + n \text{ otherwise} \end{cases}$$



Recursive level	How many nodes at each level?	How much work done by each node?	How much work across each level?
0	1	n	n
1	2	$\frac{n}{2}$	n
2	4	$\frac{n}{4}$	n
3	8	$\frac{n}{8}$	n
logn	n	1	n

Tree Method Practice

$$T(n) = \begin{cases} 1 \text{ when } n \le 1 \\ 2T\left(\frac{n}{2}\right) + n \text{ otherwise} \end{cases}$$

1. What is the size of the input on level i? $\frac{n}{2^i}$

2. What is the work done by each node on the i^{th} recursive level? $\left(\frac{n}{2^i}\right)$

3. What is the number of nodes at level i? 2^i

4. What is the total work done at the i^{th} recursive level?

   $$numNodes * workPerNode = 2^i\left(\frac{n}{2^i}\right) = n$$

5. What value of i does the last level occur?

   $$\frac{n}{2^i} = 1 \rightarrow n = 2^i \rightarrow i = \log_2 n$$

6. What is the total work across the base case level?

   $$numNodes * workPerNode = 2^{\log_2 n}(1) = n$$

Level (i)	Number of Nodes	Work per Node	Work per Level
0	1	n	n
1	2	$\frac{n}{2}$	n
2	4	$\frac{n}{4}$	n
3	8	$\frac{n}{8}$	n
$\log_2 n$	n	1	

Combining it all together…

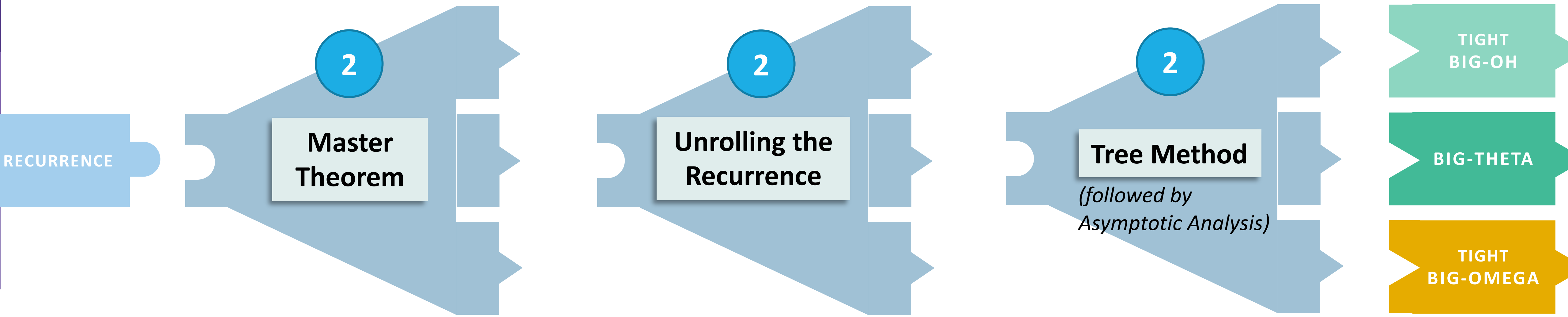
$$T(n) = \sum_{i=0}^{\log_2 n - 1} n + n = nlog_2 n + n = \Theta(\text{nlogn})$$

power of a log

$$x^{\log_b y} = y^{\log_b x}$$

Summation of a constant $\sum_{i=0}^{n-1} c = cn$

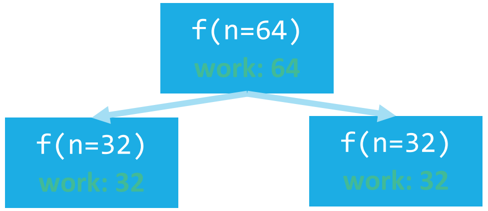
Recurrence to Big-Theta: Our Toolbox



MASTER THEOREM

$$T(n) = \begin{cases} d & \text{if } n \text{ is at most some constant} \\ aT\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

T(1) = d
T(2) = 2T(2-1) + c = 2(d) + c
T(3) = 2T(3-1) + c = 2(2(d) + c) + c = 4d + 3c



PROS: Convenient to plug ‘n’ chug
CONS: Only works for certain format of recurrences

PROS: Least complicated setup
CONS: Requires intuitive pattern matching, no formal technique

PROS: Convenient to plug ‘n’ chug
CONS: Complicated to set up for a given recurrence

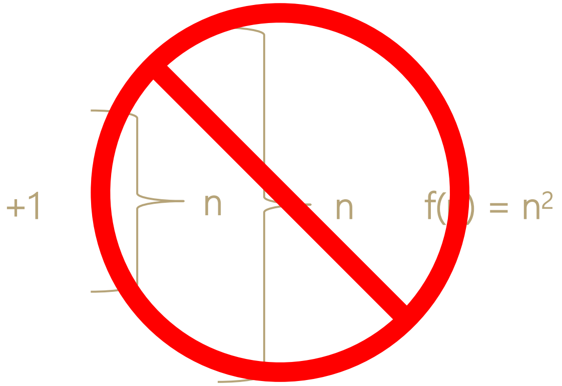
Questions

Summations

Modeling Complex Loops

Write a mathematical model of the following code

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!");
    }
}
```



Keep an eye on loop bounds!

Modeling Complex Loops

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.print(“Hello! ”);
    }
    Sysem.out.println();
}
```

$$\underbrace{\underbrace{+1}_{0 + 1 + 2 + 3 + \ldots + i-1}}_{n}$$

T(n) = $\underbrace{(0 + 1 + 2 + 3 + \ldots + i-1)}_{\text{How do we model this part?}}$

Summations!

$1 + 2 + 3 + 4 + \ldots + n = \sum_{i=1}^{n} i$

Definition: Summation

$\sum_{i=a}^{b} f(i)$ = f(a) + f(a + 1) + f(a + 2) + … + f(b-2) + f(b-1) + f(b)

T(n) = $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} 1$

What is the Big O?

Simplifying Summations

Find closed form using summation identities [(given on exams)]

```
for (int i = 0; i < n; i++) {
   for (int j = 0; j < i; j++) {
      System.out.println("Hello!");
   }
}
```

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} 1$$

closed form

simplified tight big O

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} 1 \quad = \sum_{i=0}^{n-1} 1 \cdot i \quad = 1\sum_{i=0}^{n-1} i \quad = \frac{n(n-1)}{2} \quad = \frac{1}{2}n^2 - \frac{1}{2}n \quad = \boldsymbol{O(n^2)}$$

Summation of a constant

$$\sum_{i=0}^{n-1} c = cn$$

Factoring out a constant

$$\sum_{i=a}^{b} cf(i) = c\sum_{i=a}^{b} f(i)$$

Gauss's Identity

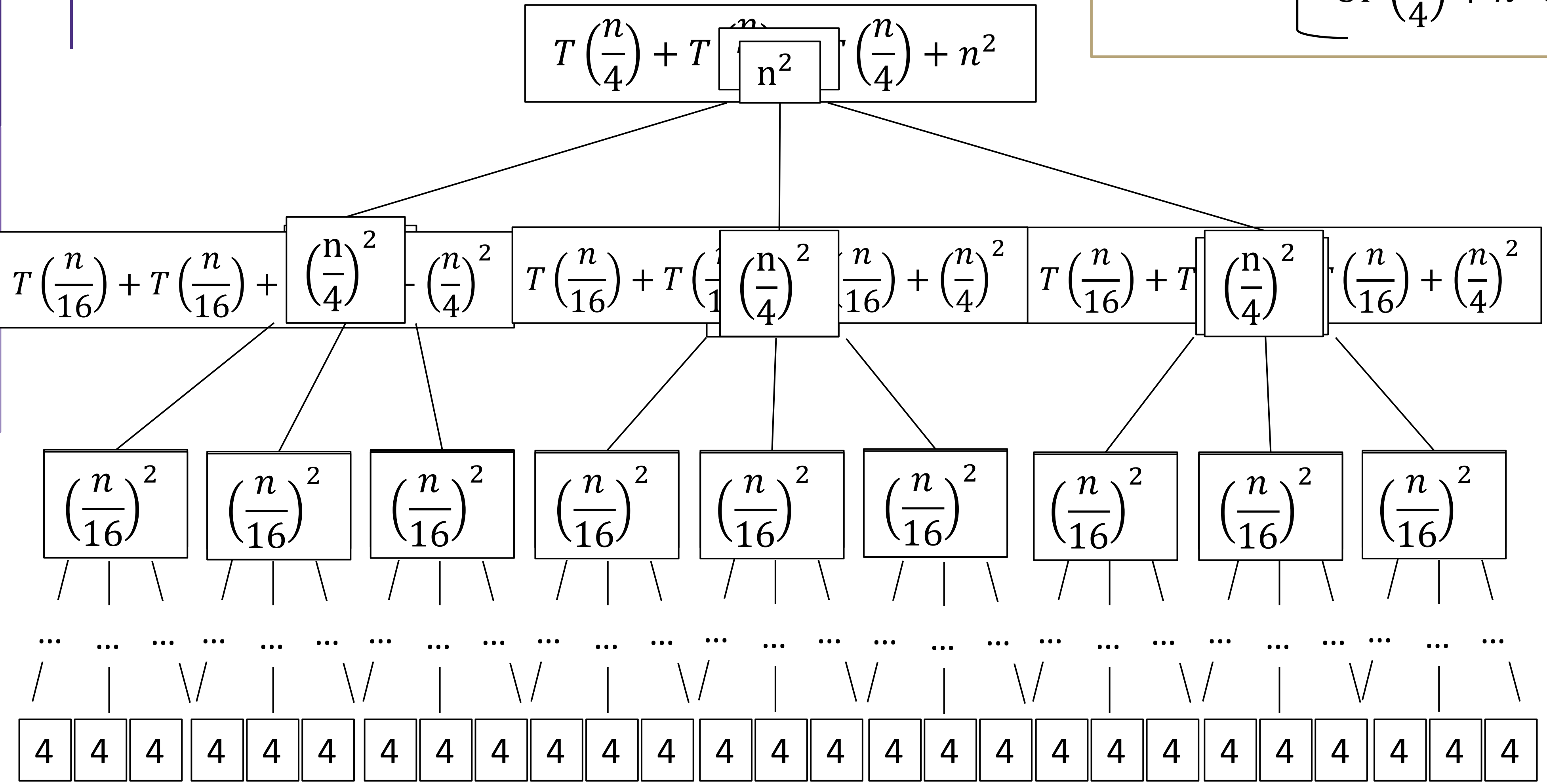
$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

Questions

Appendix

Tree Method Practice

$$T(n) = \begin{cases} 4 \; when \; n \le 1 \\ 3T\left(\frac{n}{4}\right) + n^2 \; otherwise \end{cases}$$



Answer the following questions:

1. What is the size of the input on level i?
2. What is the work done by each node on the i^{th} recursive level
3. What is the number of nodes at level i?
4. What is the total work done at the i^th recursive level?
5. What value of i does the last level occur?
6. What is the total work across the base case level?

Tree Method Practice

5 Minutes

$$T(n) = \begin{cases} 4 \text{ when } n \le 1 \\ 3T\left(\frac{n}{4}\right) + n^2 \text{ otherwise} \end{cases}$$

1. What is the size of the input on level i? $\frac{n}{4^i}$

2. What is the work done by each node on the i^{th} recursive level? $\left(\frac{n}{4^i}\right)^2$

3. What is the number of nodes at level i? 3^i

4. What is the total work done at the i^{th} recursive level?
$$3^i\left[\left(\frac{n}{4^i}\right)\right]^2 = \left(\frac{3}{16}\right)^i n^2$$

5. What value of i does the last level occur?
$$\frac{n}{4^i} = 1 \rightarrow n = 4^i \rightarrow i = \log_4 n$$

6. What is the total work across the base case level? $3^{\log_4 n} \cdot 4$

power of a log
$x^{\log_b y} = y^{\log_b x}$

$4 \cdot n^{\log_4 3}$

Level (i)	Number of Nodes	Work per Node	Work per Level
0	1	n^2	n^2
1	3	$\left(\frac{n}{4}\right)^2$	$\frac{3}{4^2}n^2$
2	9	$\left(\frac{n}{4^2}\right)^2$	$\frac{3^2}{4^4}n^2$
base	$3^{\log_4 n}$	4	$4 * 3^{\log_4 n}$

Combining it all together…

$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i n^2 + 4n^{\log_4 3}$$

Tree Method Practice

$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i n^2 + 4n^{\log_4 3}$$

factoring out a constant

$$\sum_{i=a}^{b} cf(i) = c\sum_{i=a}^{b} f(i)$$

$$T(n) = n^2 \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i + 4n^{\log_4 3}$$

Identities are on the webpage.
You don't need to memorize them.

finite geometric series

$$\sum_{i=0}^{n-1} x^i = \frac{x^n - 1}{x - 1}$$

Closed form:

$$T(n) = n^2 \left(\frac{\left(\frac{3}{16}\right)^{\log_4 n} - 1}{\frac{3}{16} - 1}\right) + 4n^{\log_4 3}$$

So what's the big-Θ...

$$T(n) = n^2 \left(-\frac{16}{13}\right)\left(\frac{3}{16}\right)^{\log_4 n} + \left(\frac{16}{13}\right) n^2 + 4n^{\log_4 3}$$

$$T(n) = n^2 \left(-\frac{16}{13}\right)(n)^{\log_4 \frac{3}{16}} + \left(\frac{16}{13}\right) n^2 + 4n^{\log_4 3}$$

$$T(n) \in \Theta(n^2)$$

More Tree Method

$$T(n) = \begin{cases} 6T\left(\frac{n}{2}\right) + 2n & \text{if } n > 8 \\ 3 & \text{otherwise} \end{cases}$$

Tree Method Practice

$$T(n) = \begin{cases} 6T\left(\frac{n}{2}\right) + 2n \text{ if } n > 8 \\ 3 \qquad\qquad \text{otherwise} \end{cases}$$

Answer the following questions:

1. What is the size of the input on level i?
2. What is the work done by each node on the i^{th} recursive level
3. What is the number of nodes at level i?
4. What is the total work done at the i^th recursive level?
5. What value of i does the last level occur?
6. What is the total work across the base case level?

Tree Method Practice

$$T(n) = \begin{cases} 6T\left(\frac{n}{2}\right) + 2n \text{ if } n > 2 \\ 3 \quad\quad\quad \text{otherwise} \end{cases}$$

5 Minutes

1. What is the size of the input on level i? $\frac{n}{2^i}$

2. What is the work done by each node on the i^{th} recursive level? $2\frac{n}{2^i}$

3. What is the number of nodes at level i? 6^i

4. What is the total work done at the i^{th} recursive level?
$$6^i\left[2\frac{n}{2^i}\right] = 2 \cdot 3^i \cdot n$$

5. What value of i does the last level occur?
$$\frac{n}{2^i} = 2 \rightarrow n = 2^{i+1} \rightarrow i = \log_2(n) - 1$$

6. What is the total work across the base case level?
$$6^{\log_2(n) - 1} \cdot 3$$

power of a log

$$x^{\log_b y} = y^{\log_b x}$$

$$\frac{3 \cdot 6^{\log_2 n}}{6} = \frac{1}{2} \cdot n^{\log_2 6} = \frac{1}{2} \cdot n^{\log_2 6}$$

Level (i)	Number of Nodes	Work per Node	Work per Level
0	1	$2n$	$2n$
1	2	$\frac{2n}{8}$	$\frac{n}{2}$
2	4	$2\left(\frac{n}{8^2}\right)$	$\frac{n}{8}$
base	$2^{\log_8 n - 1}$	3	$\frac{3}{2}n^{1/3}$

Combining it all together…

$$T(n) = \sum_{i=0}^{\log_2(n) - 2} 2 \cdot 3^i n + \frac{1}{2} n^{\log_2 6}$$

$$T(n) = \sum_{i=0}^{\log_2(n)-2} 2 \cdot 3^i n + \frac{1}{2} n^{\log_2 6}$$

$$= 2n \sum_{i=0}^{\log_2(n)-2} 3^i + \frac{1}{2} n^{\log_2 6}$$

$$= 2n \frac{3^{\log_2(n)-1}}{3-1} + \frac{1}{2} n^{\log_2 6}$$

$$= n \cdot \frac{n^{\log_2(3)}}{3} + \frac{1}{2} n^{\log_2 6}$$

$$= \frac{n^{\log_2(3)+1}}{3} + \frac{1}{2} n^{\log_2 6}$$

$$= \frac{n^{\log_2(6)}}{3} + \frac{1}{2} n^{\log_2 6} = \frac{5}{6} n^{\log_2 6}$$

finite geometric series

$$\sum_{i=0}^{n-1} x^i = \frac{x^n - 1}{x - 1}$$

power of a log

$$x^{\log_b y} = y^{\log_b x}$$

$$1 = \log_2 2$$

$$\log_a b + \log_a c = \log_a(bc)$$

Summation Practice

```
public static void primesUpToN(int n) {
   System.out.print("1 2 "); +1
   for (int i = 3; i <= n; i++) {
      for (int j = 2; j < i; j++){
         if (j != i && j % i == 0) {
            System.out.print(i + " ");+4
            break;
         }                              +1
      }
   System.out.println();+1
}
```

$$\sum_{j=2}^{i-1} 5 \qquad \sum_{i=3}^{n}\sum_{j=2}^{i-1} 5$$

$$T(n) = 1 + \sum_{i=3}^{n}\sum_{j=2}^{i-1} 5 = 1 + \sum_{i=0}^{n-3}\sum_{j=0}^{i-3} 5 = 1 + \sum_{i=0}^{n-3} 5(i-2) = 1 + 5\left(\sum_{i=0}^{n-3} i - \sum_{i=0}^{n-3} 2\right) - = 1 + 5\left(\frac{(n-2)(n-3)}{2} - (n-2)(2)\right)$$

Adjusting summation bounds

Summation of a constant

Factoring out a constant

Gauss's identity