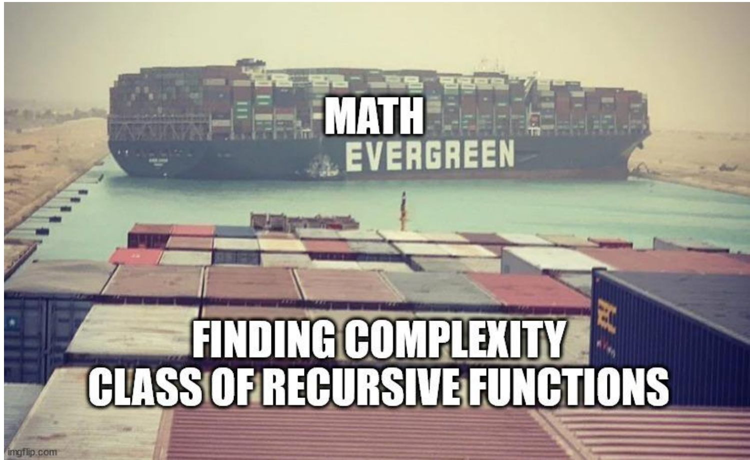
CSE 373 SP21 Section 3

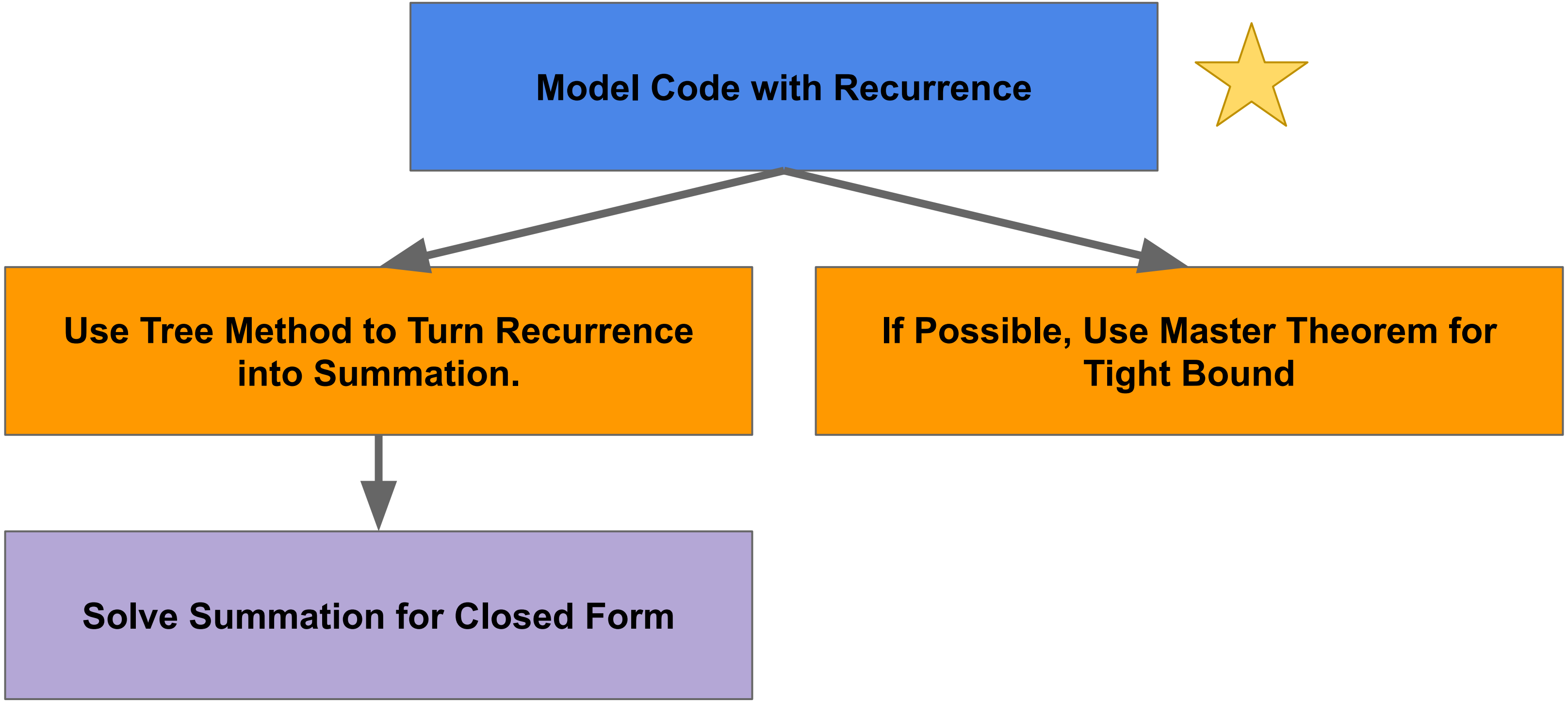
Recurrence Party 🎉



MicroTeach: Recurrences

Big Idea

Three* steps in solving for the runtime of a recursive function.



Modeling Code with Recurrences?

What is a Recurrence?

An expression that lets us express the runtime of a function **recursively.**

$$T(n) = \begin{cases} C_1 & \text{when} \quad n = 0, 1 \\ C_2 + T(n-1) & \text{otherwise} \end{cases}$$

What is a Recurrence?

An expression that lets us express the runtime a function **recursively.**

Base Case

$$T(n) = \begin{cases} \boxed{C_1 \quad \text{when} \quad n = 0, 1} \\ \boxed{C_2 + T(n-1) \quad \text{otherwise}} \end{cases}$$

Recursive Case

What is a Recurrence?

An expression that lets us express the runtime a function **recursively.**

Base Case

$$T(n) = \begin{cases} C_1 & \text{when} \quad n = 0, 1 \\ C_2 + T(n-1) & \text{otherwise} \end{cases}$$

Recursive Case

Recursive Call

Do Problems 1A + 3A Together!

Hint 1: This is helpful for 1A (see "cheat-sheet" at end of PDF):

Gauss's identity

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

Hint 2: You should use the answer from 1A to get 3A more quickly!

1A: Finding Bounds

1A: Finding Bounds

```
int result = 0;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < i; j++) {
        result++;
    }
}
```

Outer loop runs N times

1A: Finding Bounds

```
int result = 0;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < i; j++) {
        result++;
    }
}
```

The inner loop depends on the outer loop

1A: Finding Bounds

```
int result = 0;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < i; j++) {
        result++;
    }
}
```

Inside the inner loop, we have a constant time operation

1A: Finding Bounds

Outer loop runs 0 ~ n - 1 times which defines i
Inner loop runs 0 ~ i - 1 times
As inner loop depends on the outer loop, we can express as the summation below.

```
int result = 0;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < i; j++) {
        result++;
    }
}
```

$$\sum_{i=0}^{N-1}\sum_{j=0}^{i-1} 1$$

Remember: Gauss's identity

$$\boxed{\sum_{i=0}^{N-1}\sum_{j=0}^{i-1} 1}$$

$$\sum_{i=0}^{N-1} i = 0 + 1 + 2 + \ldots + N - 1 = \frac{N(N-1)}{2}$$

1A: Finding Bounds

POSSIBLE RUNTIME:

$T(N) = N(N-1)/2 = \frac{1}{2} N^2 - \frac{1}{2} N$

WORST CASE RUNTIME:

$\Theta(N^2)$

$N^2/2 + N/2 \Rightarrow N^2$

Best/worst cases agree!

3A: Code To Recurrence

3A: Code To Recurrence

```
public static int f(int N) {
    if (N <= 1) {
        return 0;
    }
    int result = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }
    return 5 * f(N / 2) + 3 * result + 2 * f(N / 2) + f(N / 2) + f(N / 2);
}
```

3A: Code To Recurrence

```
public static int f(int N) {
    if (N <= 1) {
        return 0;
    }
    int result = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }
    return 5 * f(N / 2) + 3 * result + 2 * f(N / 2) + f(N / 2) + f(N / 2);
}
```

Base case when the input N <= 1

3A: Code To Recurrence

```
public static int f(int N) {
    if (N <= 1) {
        return 0;
    }
    int result = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }
    return 5 * f(N / 2) + 3 * result + 2 * f(N / 2) + f(N / 2) + f(N / 2);
}
```

We saw the runtime for this loop earlier!

3A: Code To Recurrence

```
public static int f(int N) {
    if (N <= 1) {
        return 0;
    }
    int result = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }
    return 5 * f(N / 2) + 3 * result + 2 * f(N / 2) + f(N / 2) + f(N / 2);
}
```

It's N(N-1)/2!

(see Finding Bounds *a*)

3A: Code To Recurrence

```
public static int f(int N) {
    if (N <= 1) {
        return 0;
    }
    int result = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }
    return 5 * f(N / 2) + 3 * result + 2 * f(N / 2) + f(N / 2) + f(N / 2);
}
```

Call f(N/2)
Constant addition
Call f(N/2)
Call f(N/2)
Call f(N/2)

3A: Code To Recurrence

Calling 4 functions and the input is divided by half each level

$$T(N) = \begin{cases} 1 & \text{if } N \leq 1 \\ 4T(\frac{N}{2}) + \frac{N(N-1)}{2} & \text{otherwise} \end{cases}$$

Work happening in both outer and inner loops (see 1A)

3A: Code To Recurrence

Calling 4 functions and the input is divided by half each level

$$T(N) = \begin{cases} 1 & \text{if } N \leq 1 \\ 4T(\frac{N}{2}) + \frac{N(N-1)}{2} & \text{otherwise} \end{cases}$$

Work happening in both outer and inner loops (see 1A)

Can also simplify **non-recursive term** (and only this term) using Big-O rules.

So you could also have put $\mathbf{N^2}$ here.

3A: Code To Recurrence

Calling 4 functions and the input is divided by half each level

$$T(N) = \begin{cases} 1 & \text{if } N \leq 1 \\ 4T(\frac{N}{2}) + \frac{N(N-1)}{2} & \text{otherwise} \end{cases}$$

Work happening in both outer and inner loops (see 1A)

Can also simplify **non-recursive term** (and only this term) using Big-O rules.

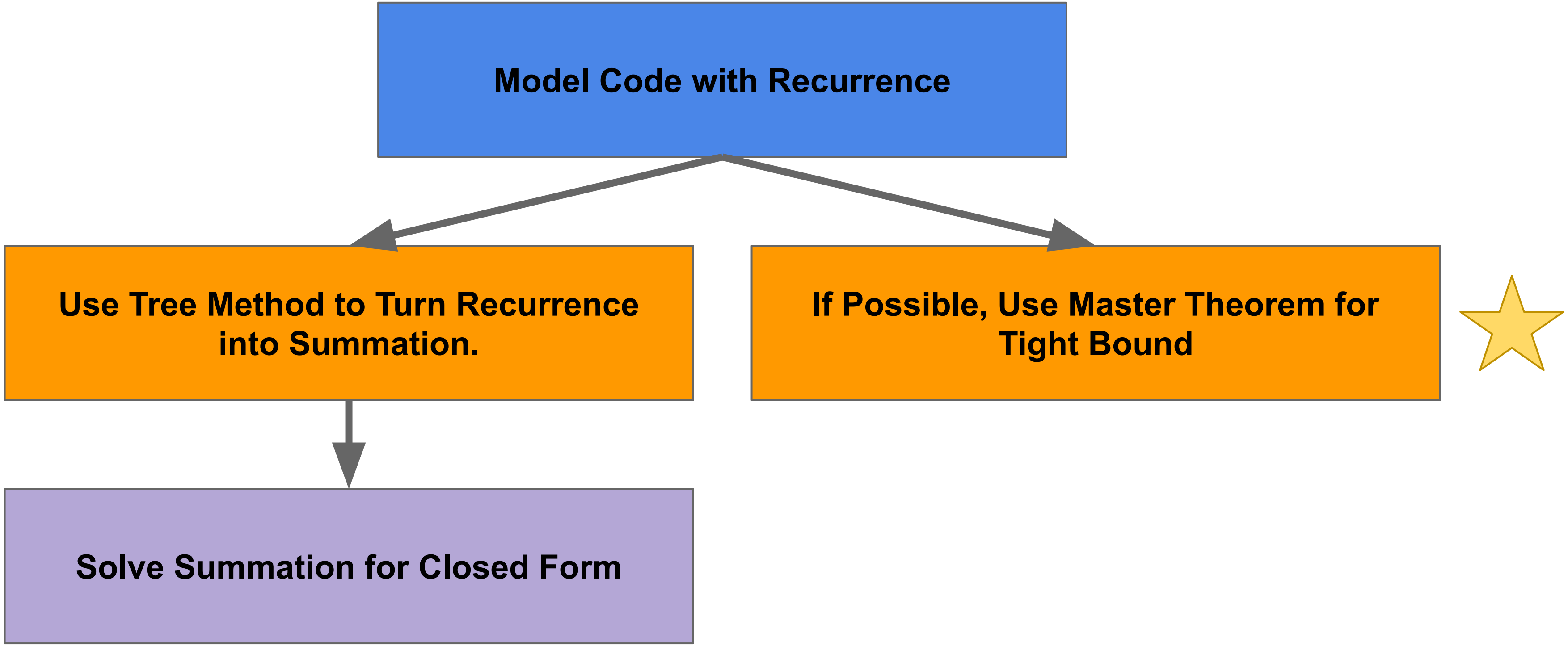
So you could also have put $\mathbf{N^2}$ here.

In fact, you **should** simplify before doing Tree Method!

MicroTeach: Master Theorem

Where are we?

Three* steps in solving for the runtime of a recursive function.



Master Theorem: A "Cheat Code" (Kinda)

For recurrences in this form, where a, b, c, e are constants:

$$T(n) = \begin{cases} d & \text{if } n \leq \text{some constant} \\ aT(n/b) + e \cdot n^c & \text{otherwise} \end{cases}$$

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

NOTE: **If** you get **lucky** and your recurrence matches the Master Theorem form, then you may use this formula to **jump** right to the final closed form.

(Of course, a lot of the time it **won't** match. That's why tree method is still important.)

Use Master Theorem to find closed form

Original recurrence: $$T(N) = \begin{cases} 1 & \text{if } N \leq 1 \\ 4T(\frac{N}{2}) + \frac{N(N-1)}{2} & \text{otherwise} \end{cases}$$

Recall: could also have put **N^2** here.

Step 1: Does T(n) match Master Theorem form?

For recurrences in this form, where a, b, c, e are constants:

$$T(n) = \begin{cases} d & \text{if } n \leq \text{some constant} \\ aT(n/b) + e \cdot n^c & \text{otherwise} \end{cases}$$

a = 4 e = 1

b = 2 c = 2

Use Master Theorem to find closed form

Original recurrence: $$T(N) = \begin{cases} 1 & \text{if } N \le 1 \\ 4T(\frac{N}{2}) + \frac{N(N-1)}{2} & \text{otherwise} \end{cases}$$

Step 2: Calculate $\log_b(a)$ and compare it to c

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

a = 4 e = 1

b = 2 c = 2

$\log_b(a)$ = $\log_2(4)$

$\log_2(4)$ = 2

c = 2 so $\log_b(a)$ = c

Ta-da!

$\log_b(a)$ = c so $T(n) \in \Theta(n^2 \log n)$

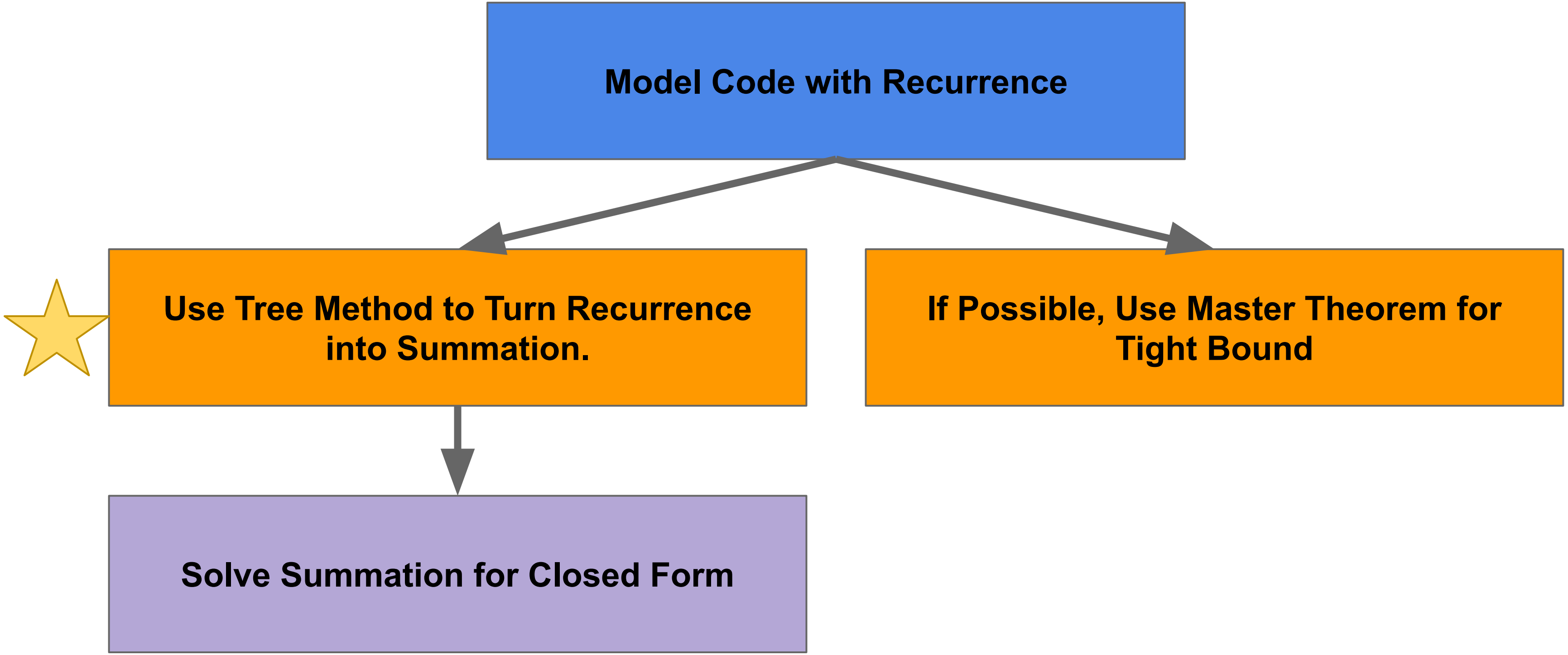
4. Master Theorem

Use the cheat sheet attached to your section handout!

MicroTeach: Core Tree Method

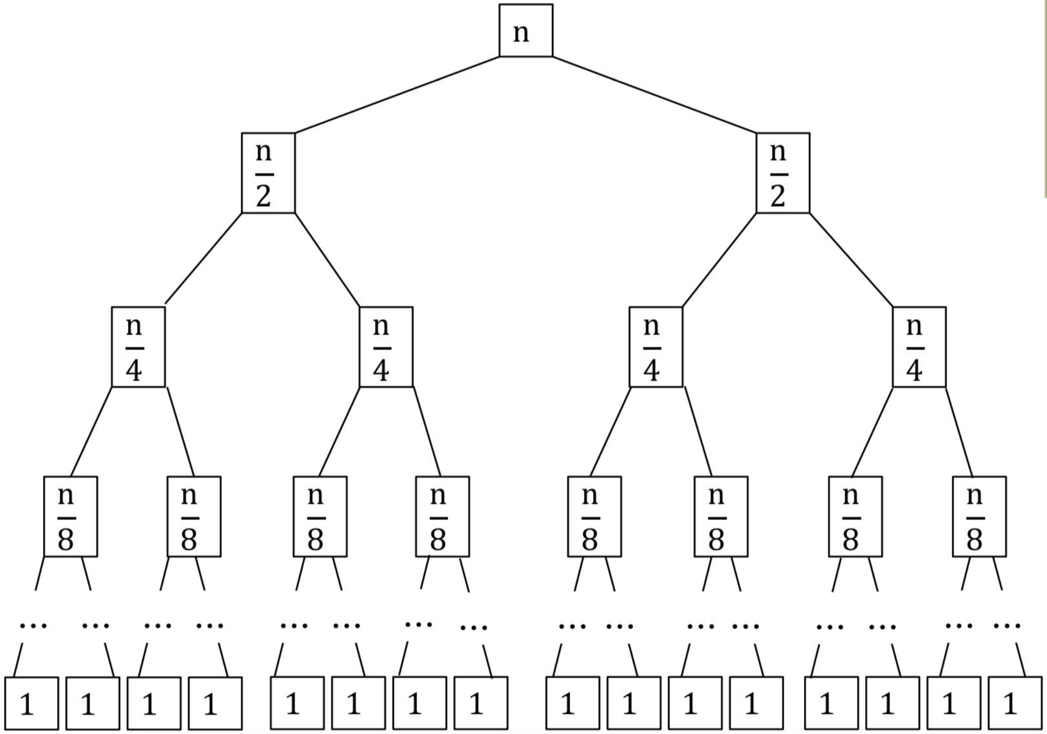
Where are we?

Three* steps in solving for the runtime of a recursive function.



Tree Method: Big Idea

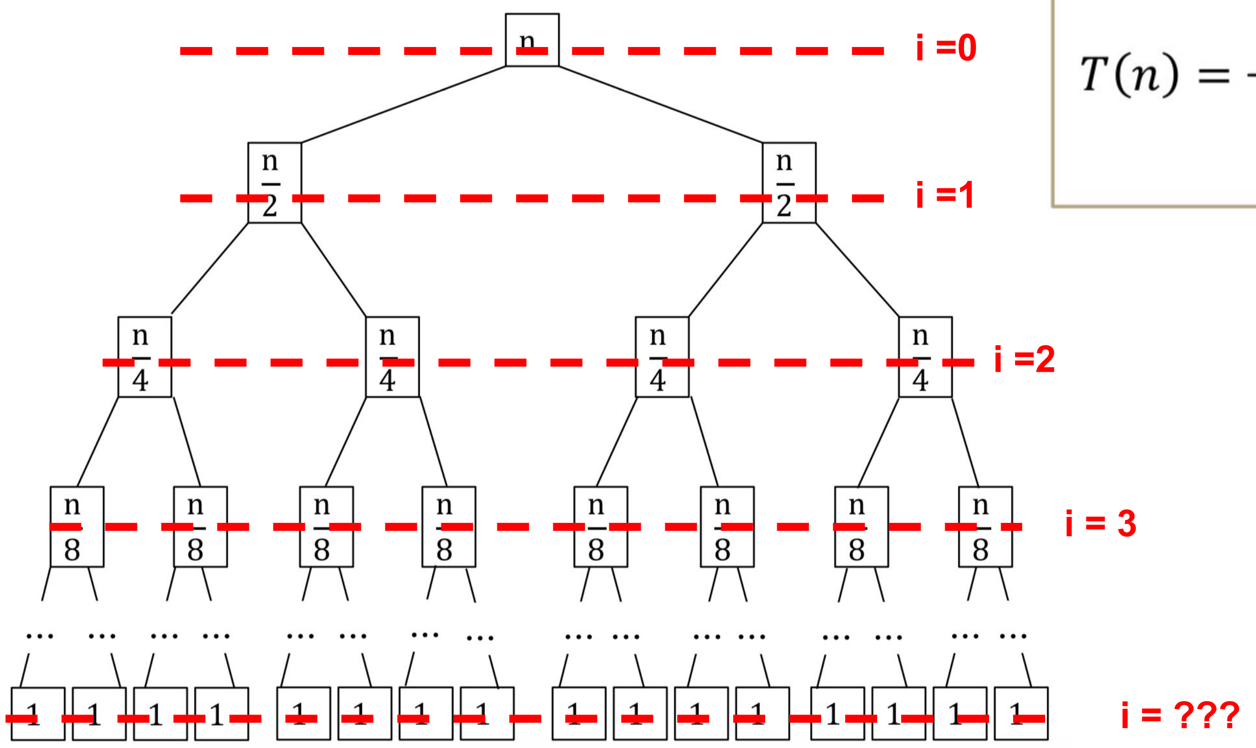
This is a **drawing** approach that turns **recurrences** into **summations**.



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

Tree Method: Big Idea

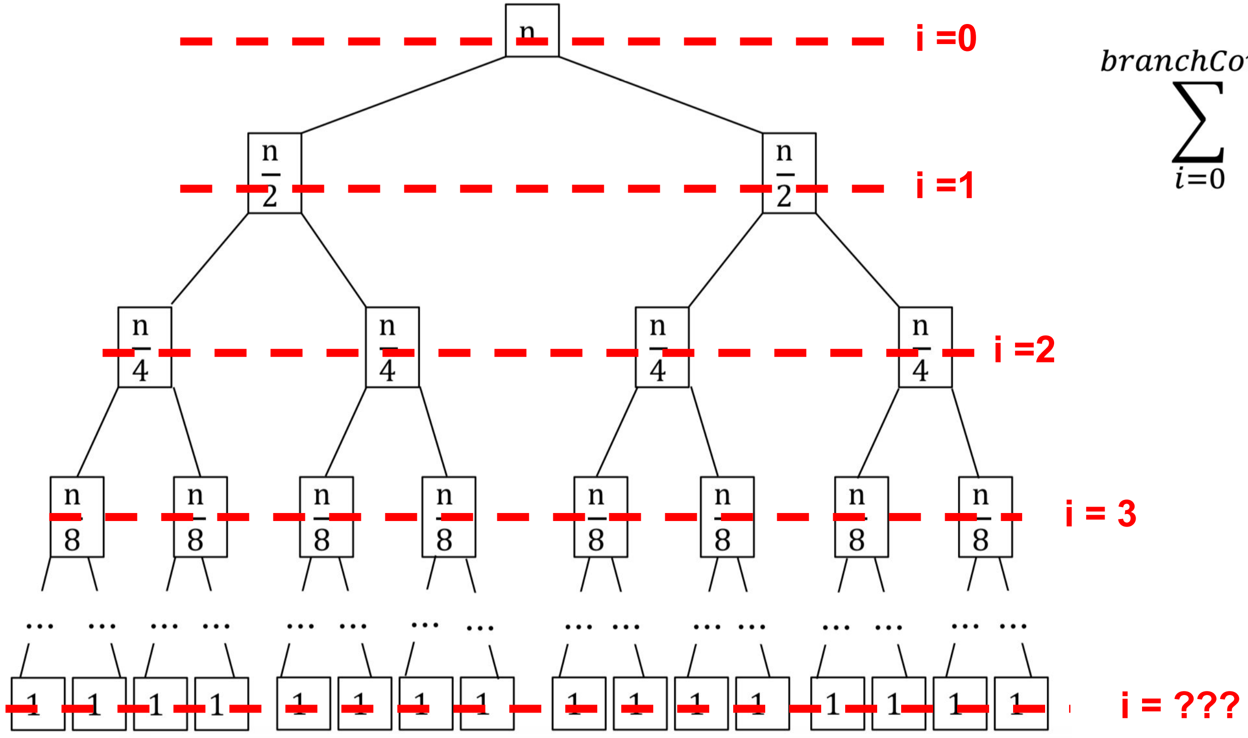
This is a **drawing** approach that turns **recurrences** into **summations**.



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

Tree Method: Big Idea

This is a **drawing** approach that turns **recurrences** into **summations**.

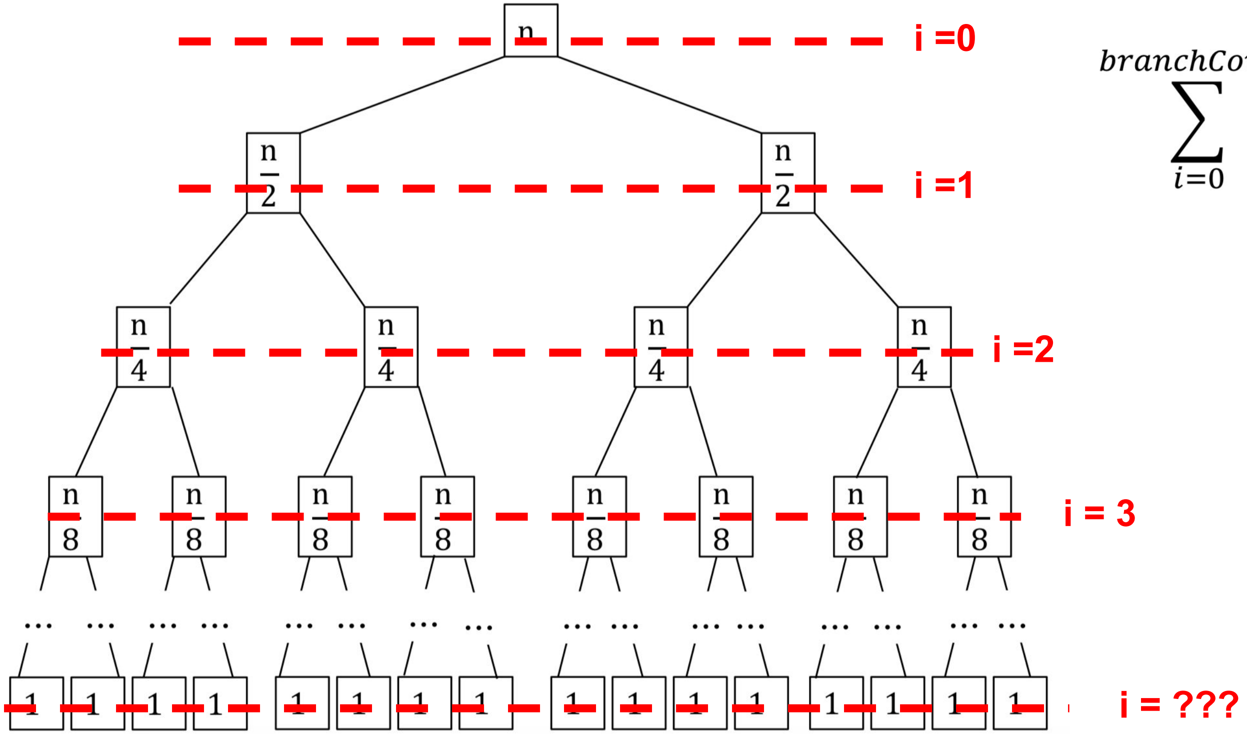


Total Work:

$$\sum_{i=0}^{branchCount} branchNum(i)branchWork(i)$$

Tree Method: Big Idea

This is a **drawing** approach that turns **recurrences** into **summations**.



Total Work:

$$\sum_{i=0}^{branchCount} branchNum(i)branchWork(i)$$

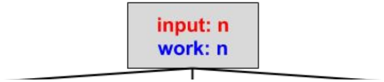
But to actually compute this, there are multiple sub-questions we have to ask!

(See Problem 5)

Problem 5A-G: Recurrence to Summation

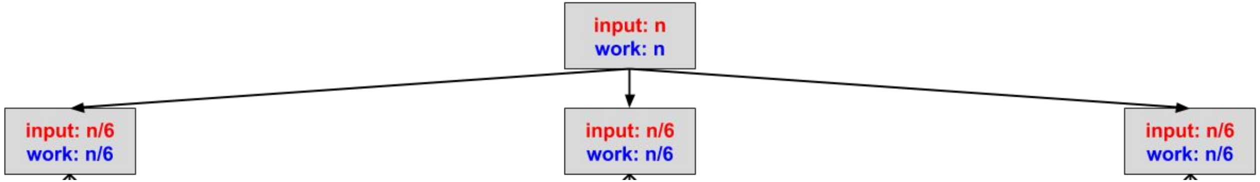
The Tree

$$A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$



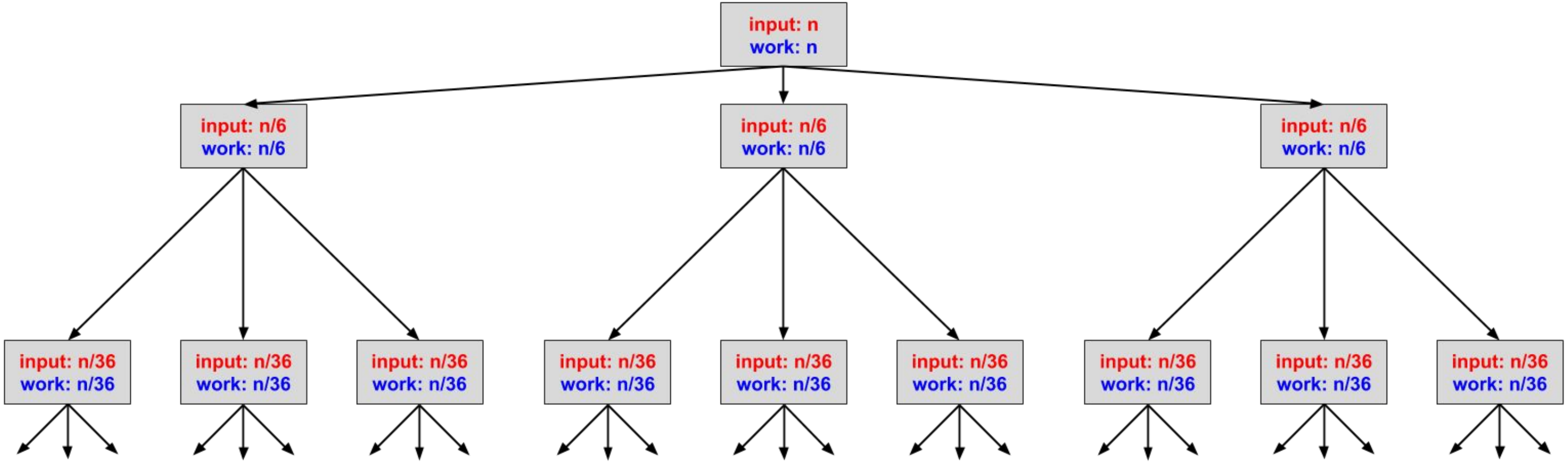
The Tree

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$



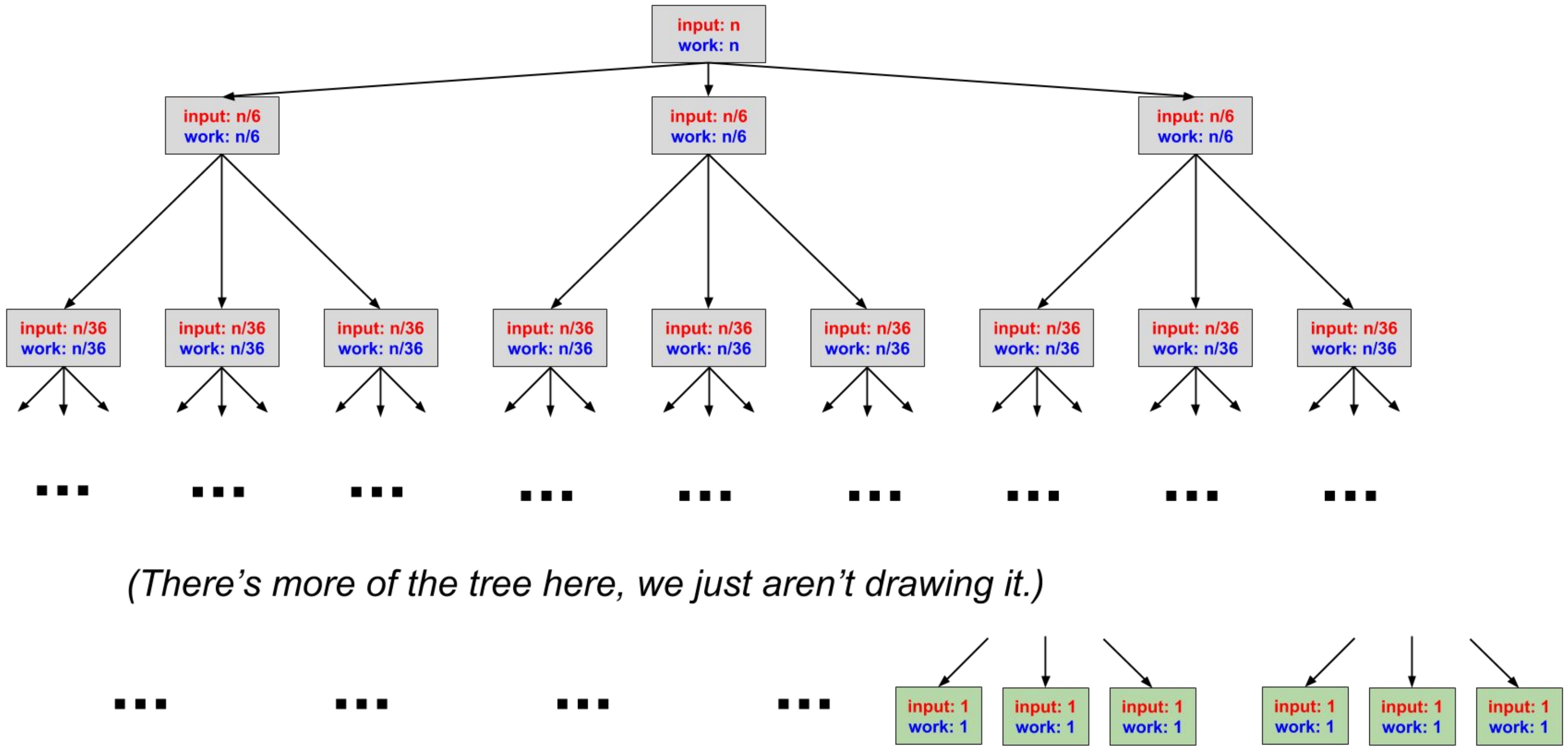
The Tree

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$



The Tree

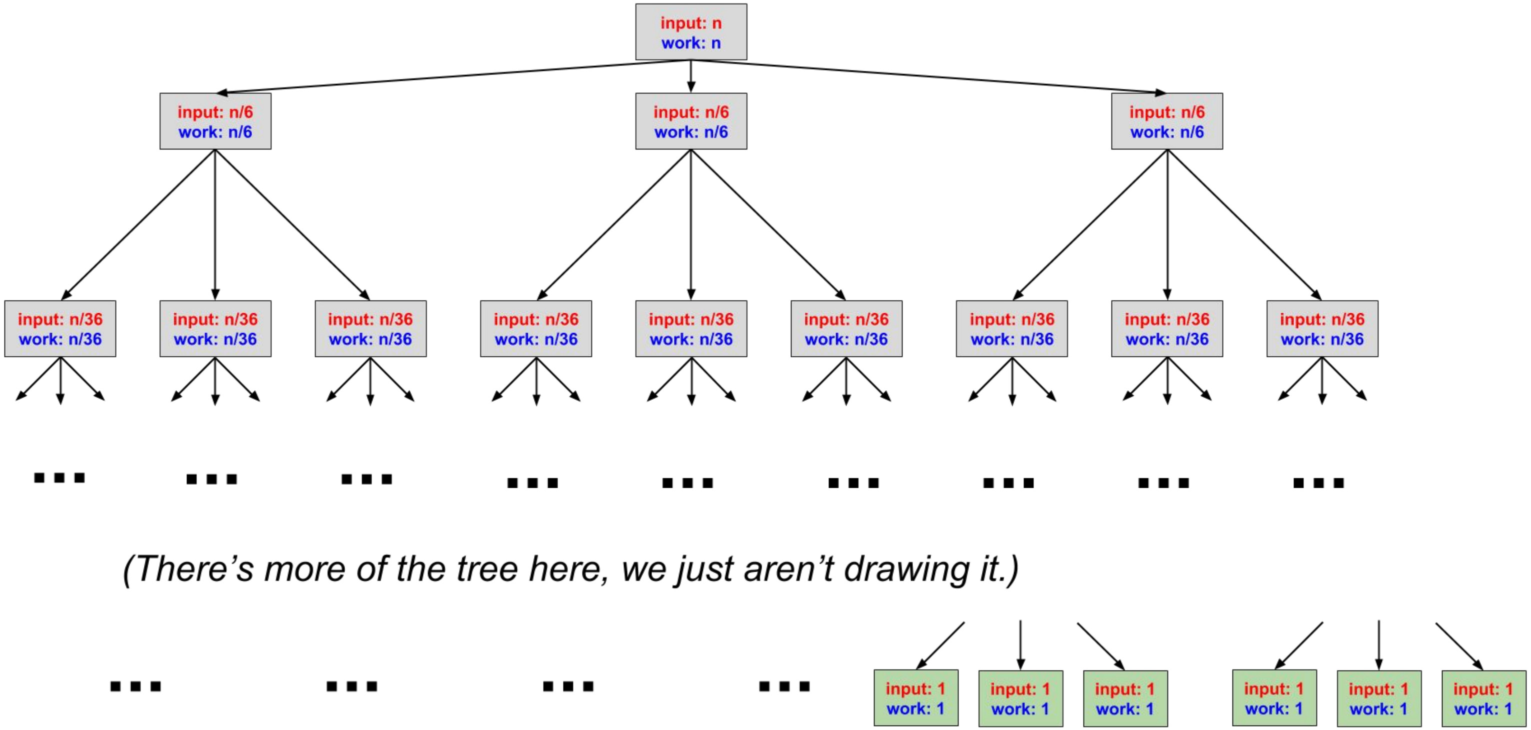
$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$



5A: Input to a node at level i

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

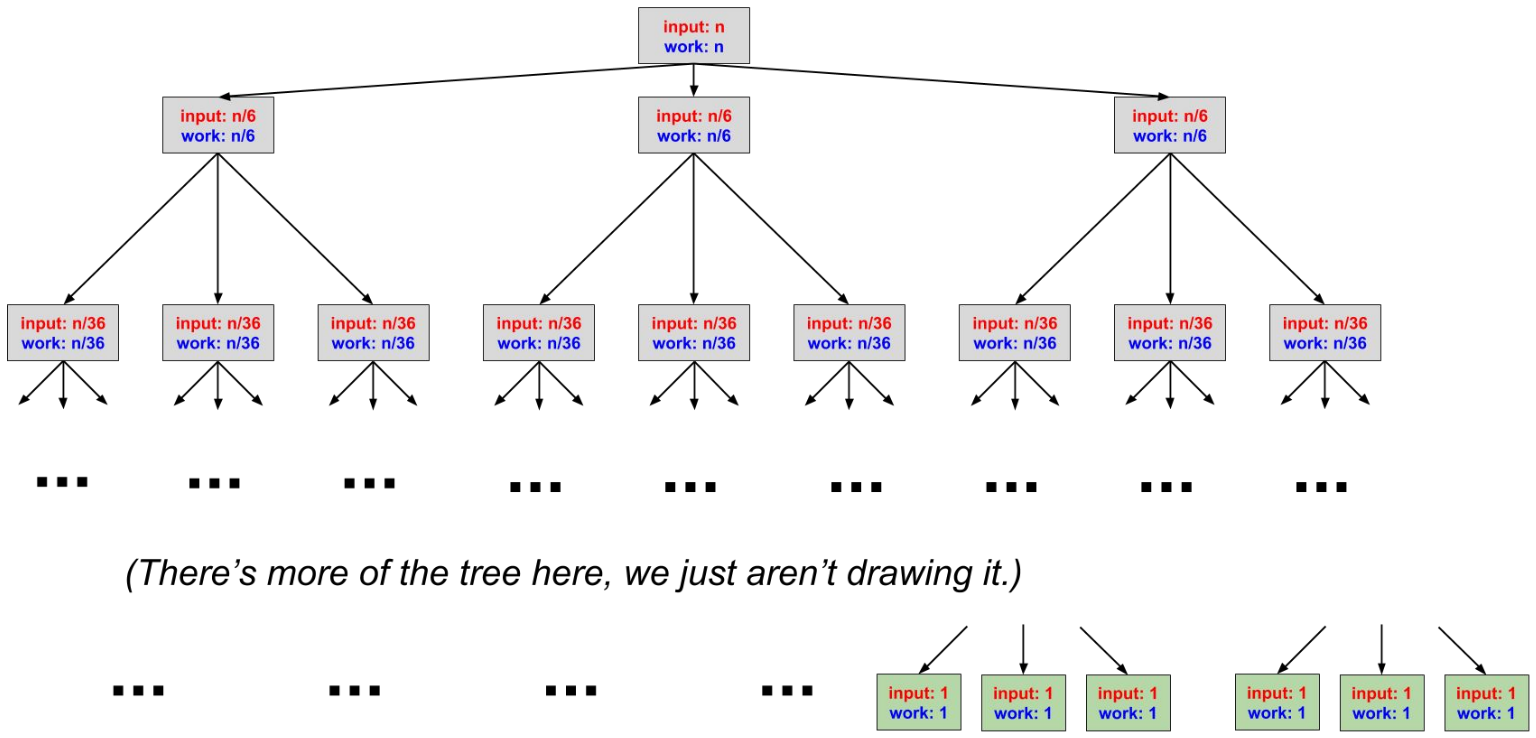
We divide by 6 at each level, so the input at level i is **$n/6^i$**.



5B: Work per node at (recursive) level i?

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

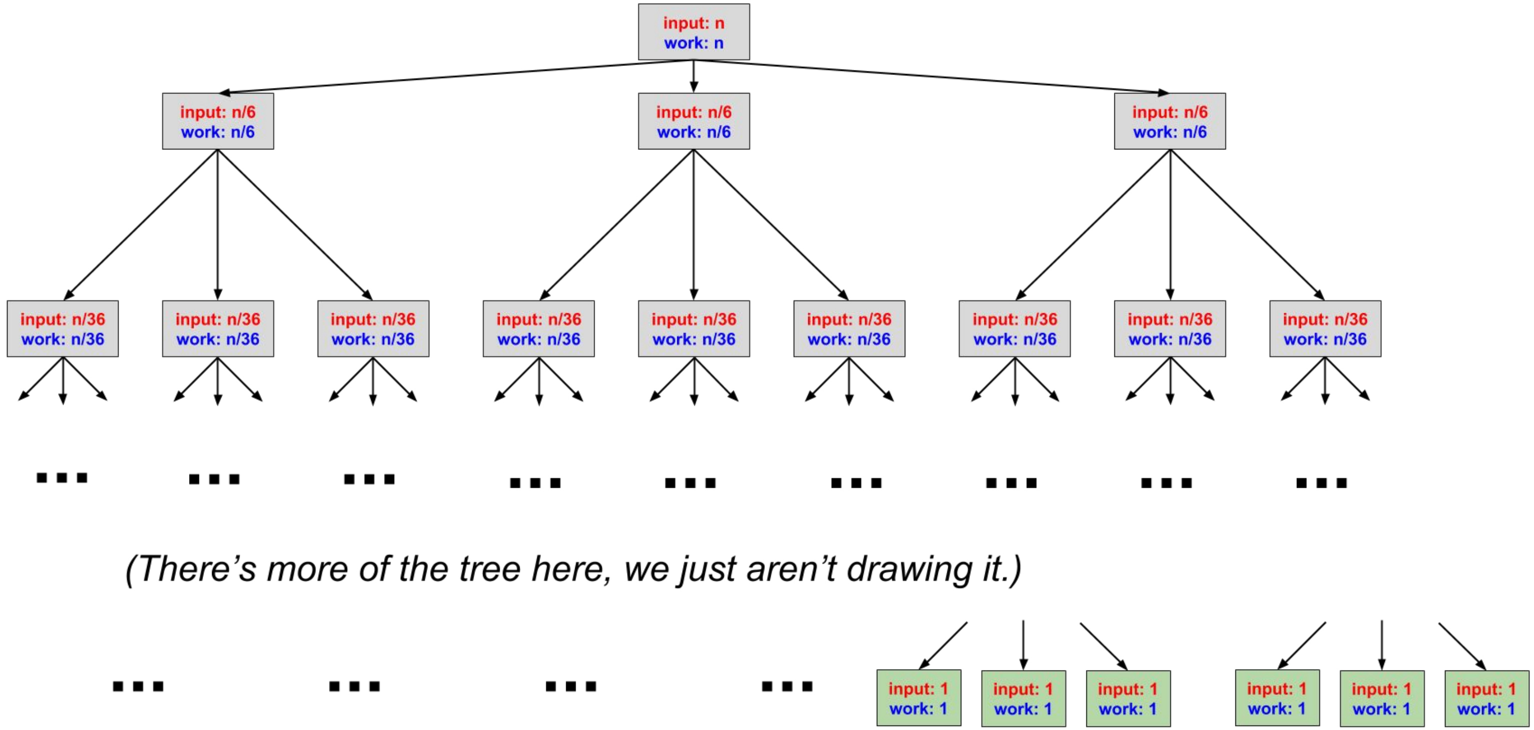
Same as the input to a node (in **this** case), so also **$n/6^i$**.



5C: Number of nodes at level i

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

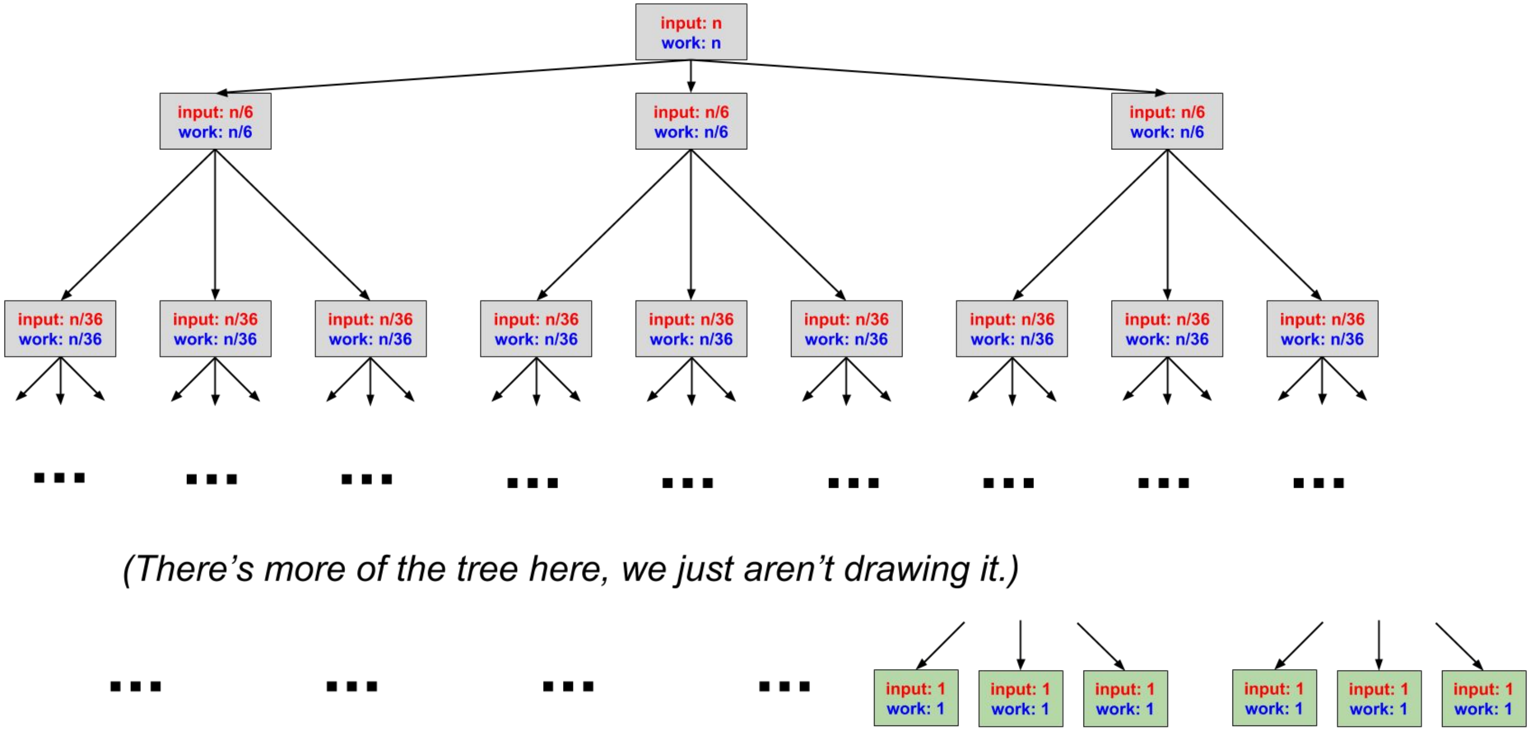
Each (non-base-case) node produces 3 more nodes, so at level i we have **3^i** nodes.



5D: Total work at the ith recursive level

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

(number of nodes in level) x (work per node at level)

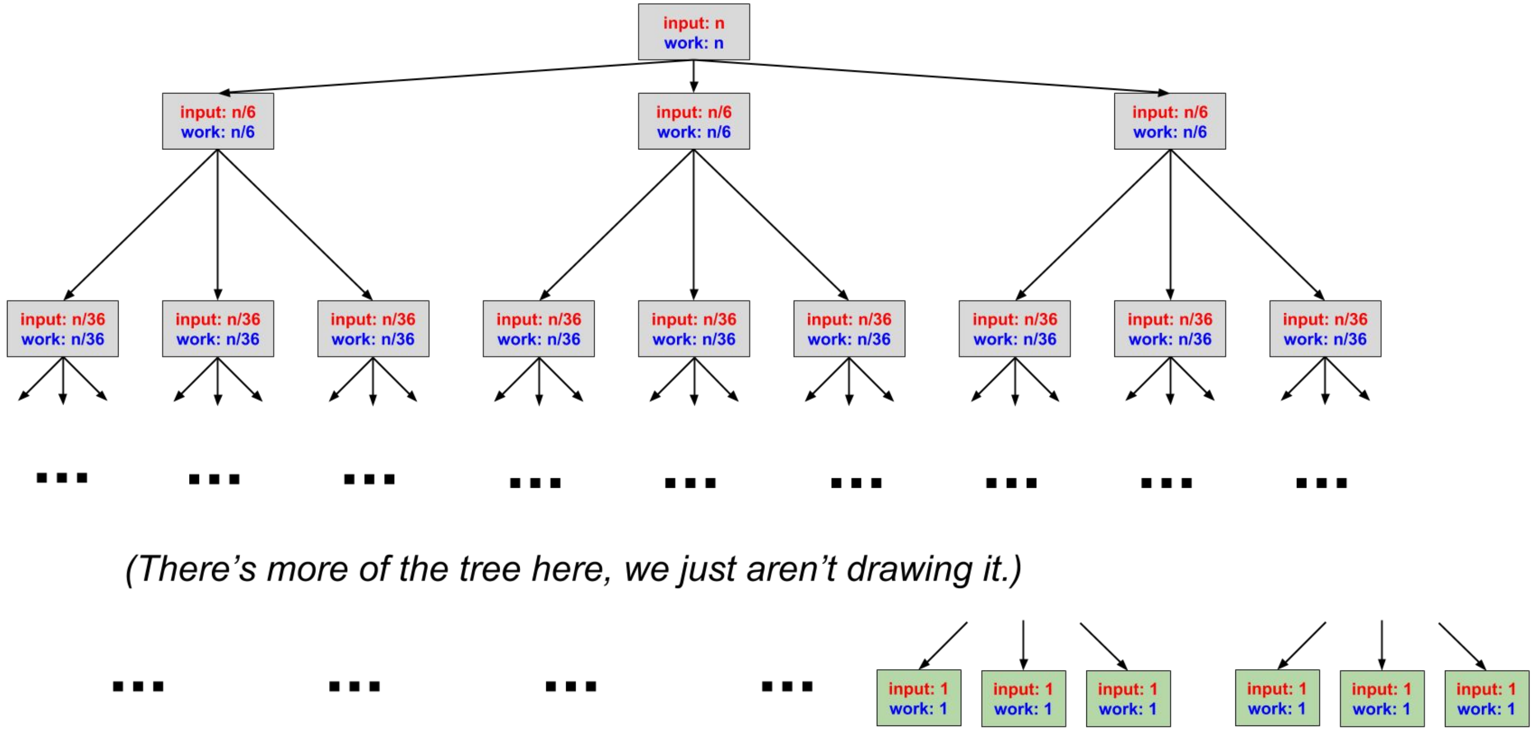


5D: Total work at the ith recursive level

$$A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

(number of nodes in level) x (work per node at level)

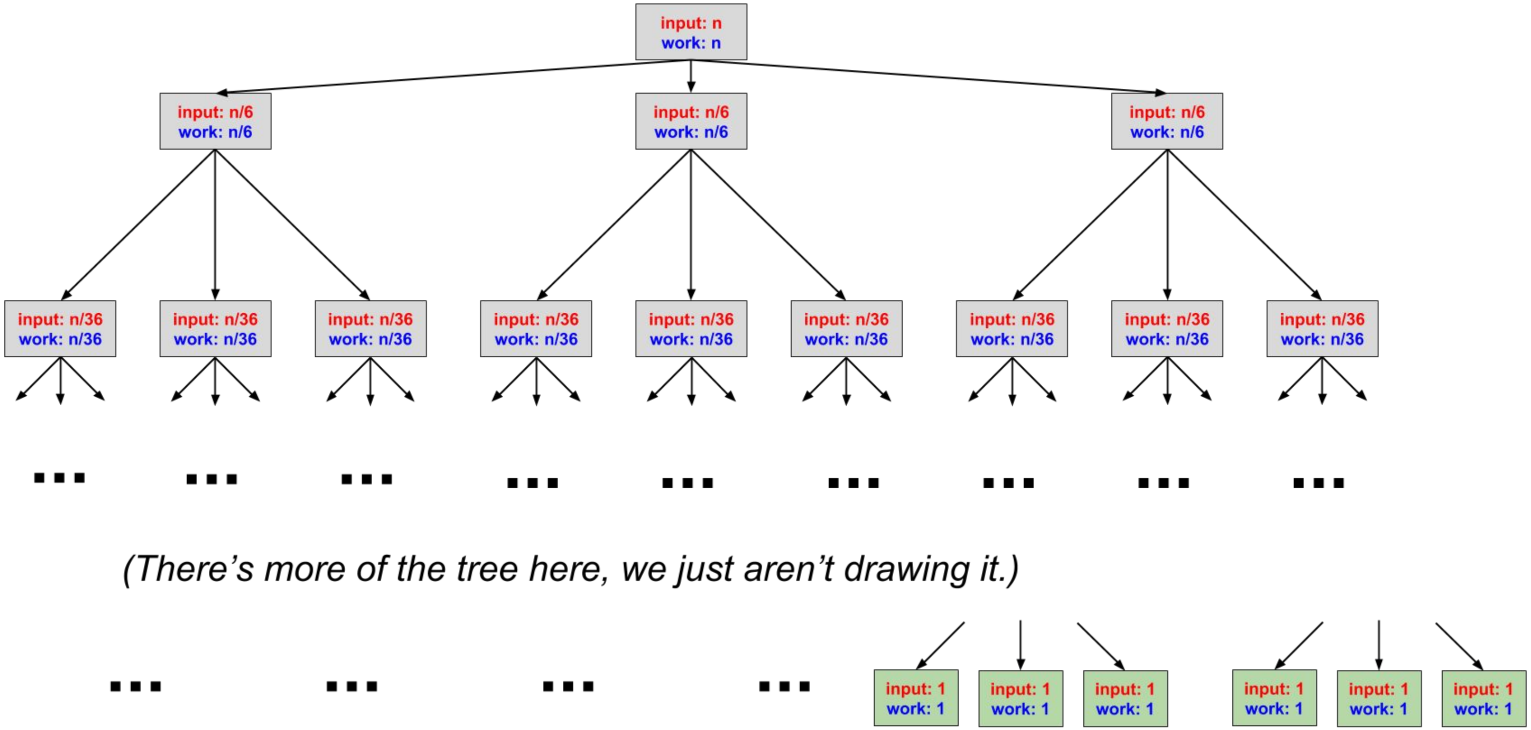
$3^i \times n/6^i$



5E: Last Level of the Tree

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

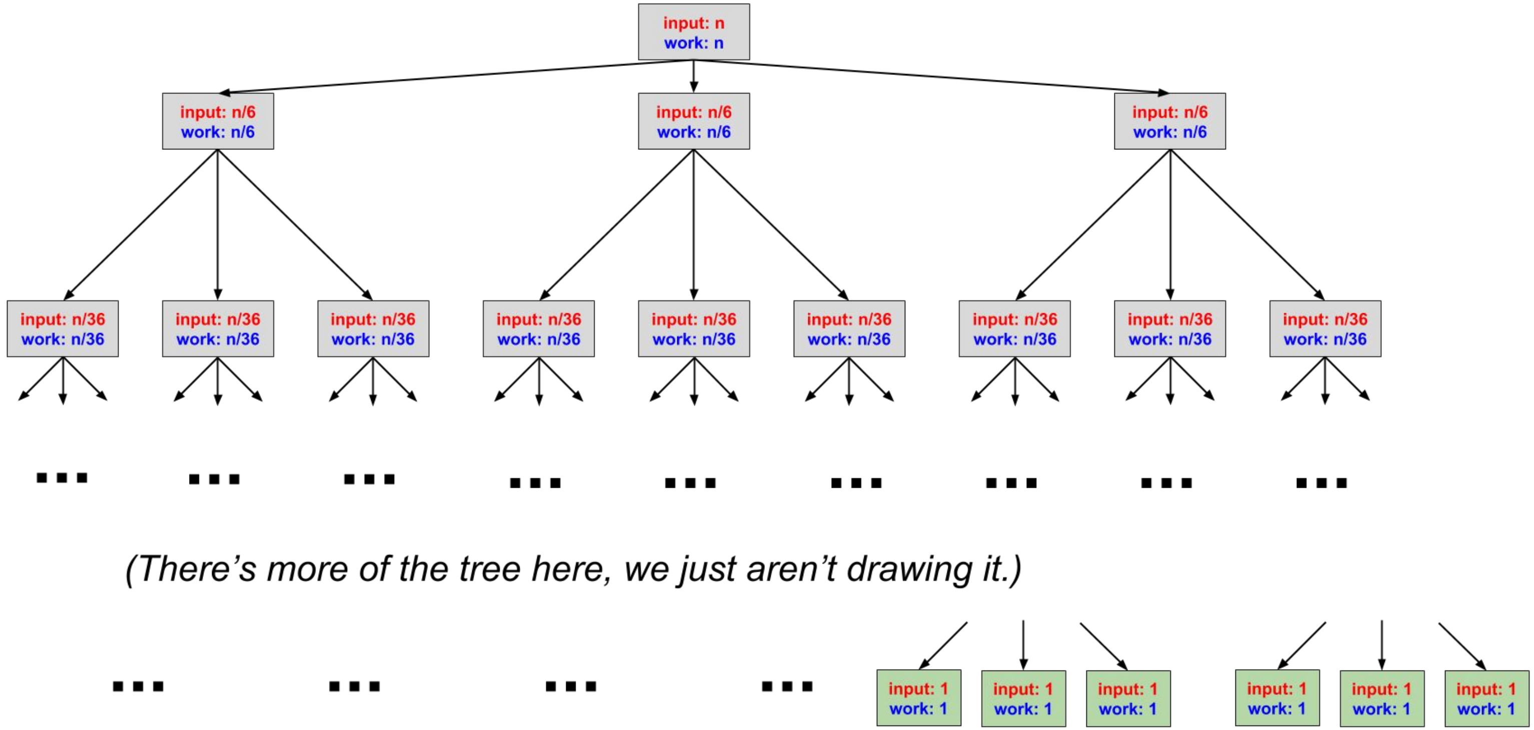
We hit our base case when **$n/6^i = 1$**.



5E: Last Level of the Tree

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

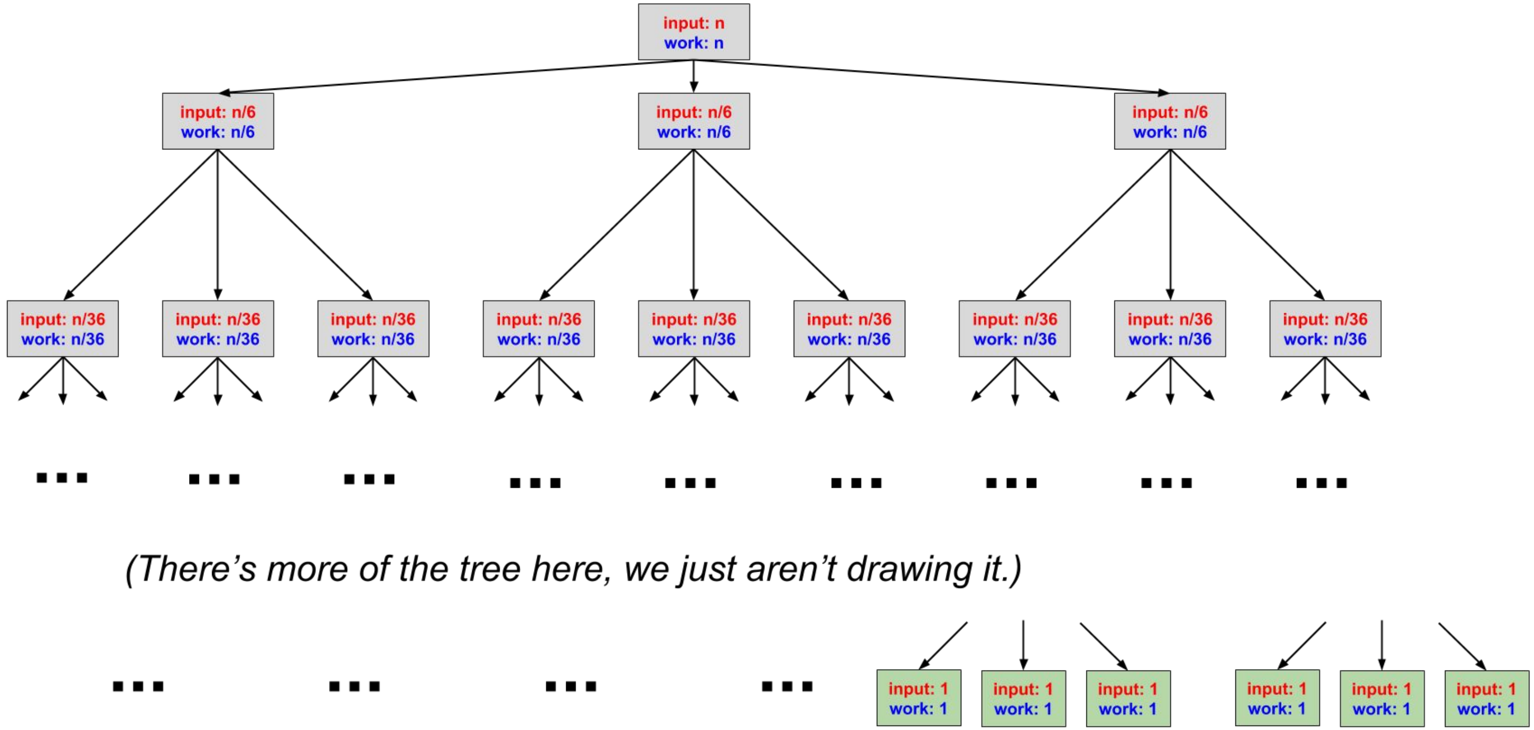
$$n/6^i = 1$$



5E: Last Level of the Tree

$$A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

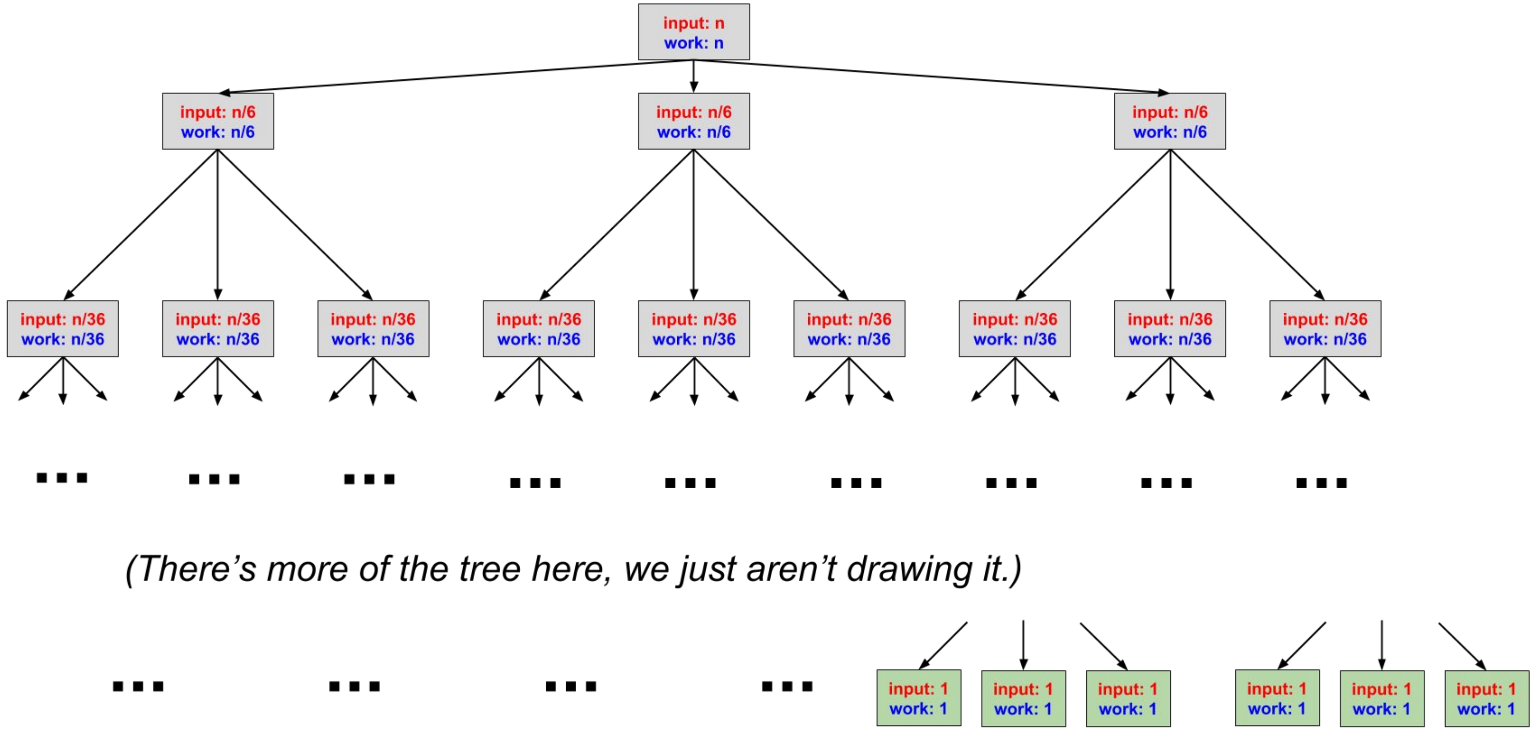
$n = 6^i$



5E: Last Level of the Tree

$$A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

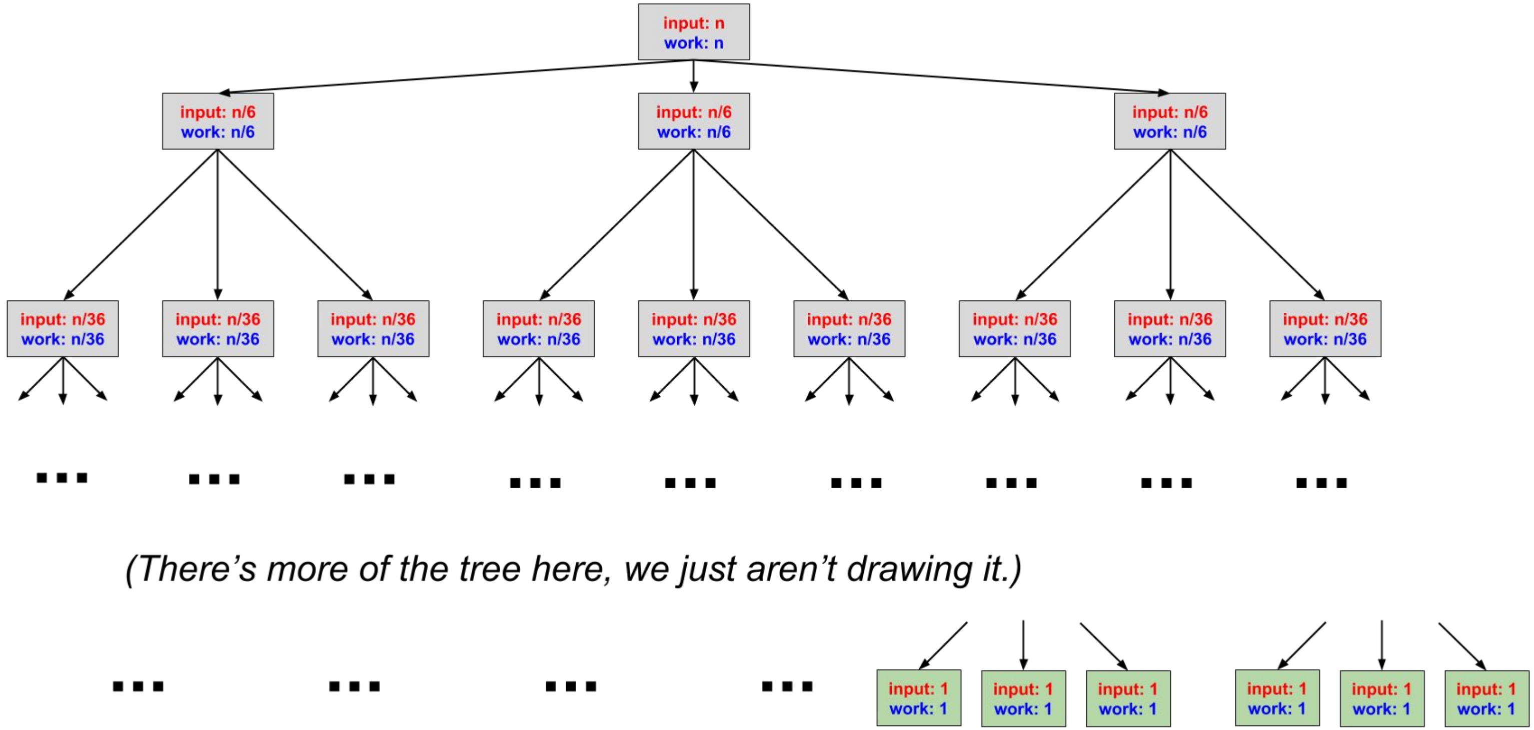
$$\log_6(n) = i$$



5F: Total Work Done in the Base Case

$$A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

(number of nodes in base case level) x (work per node in base case level)

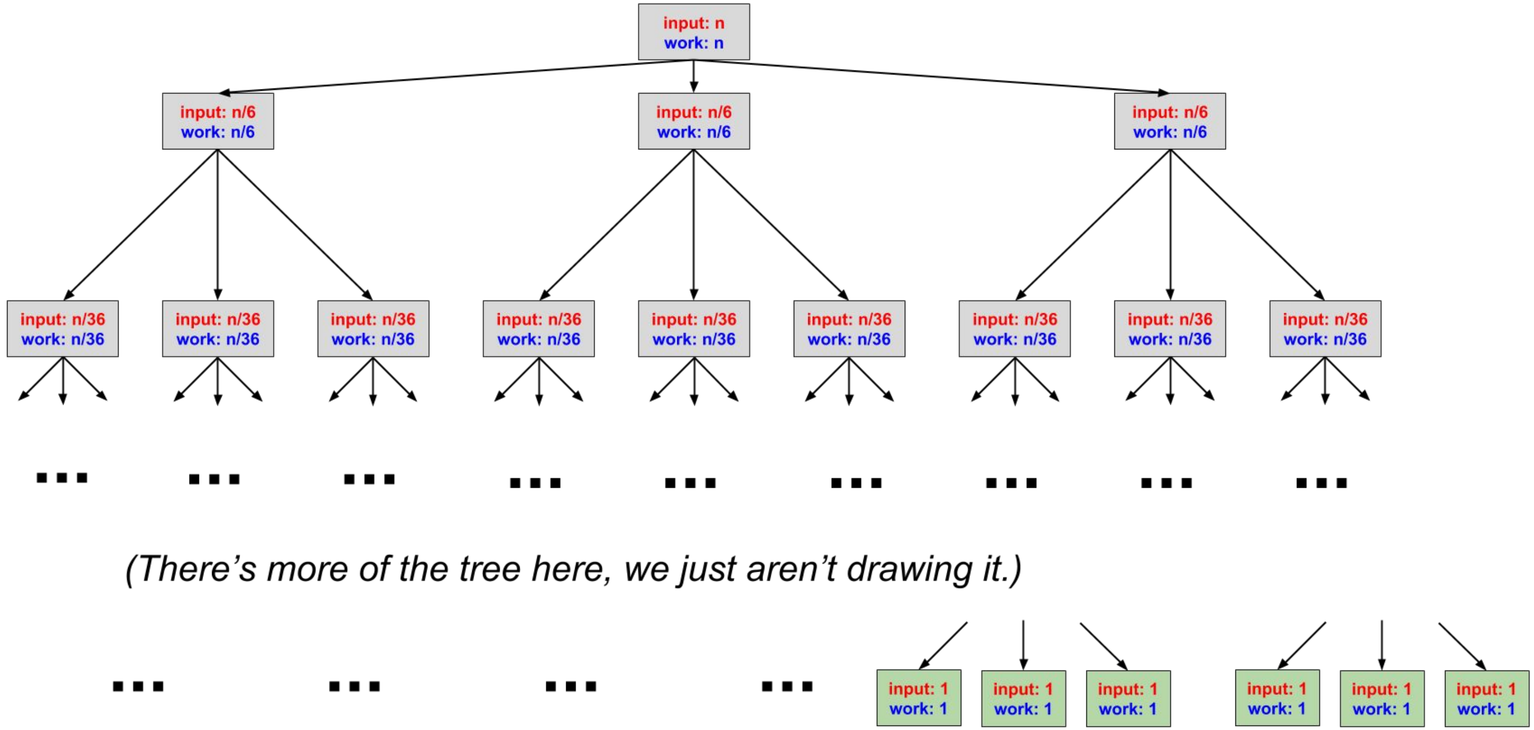


5F: Total Work Done in the Base Case

$$A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

(number of nodes in base case level) x (work per node in base case level)

$$3^i \cdot 1$$

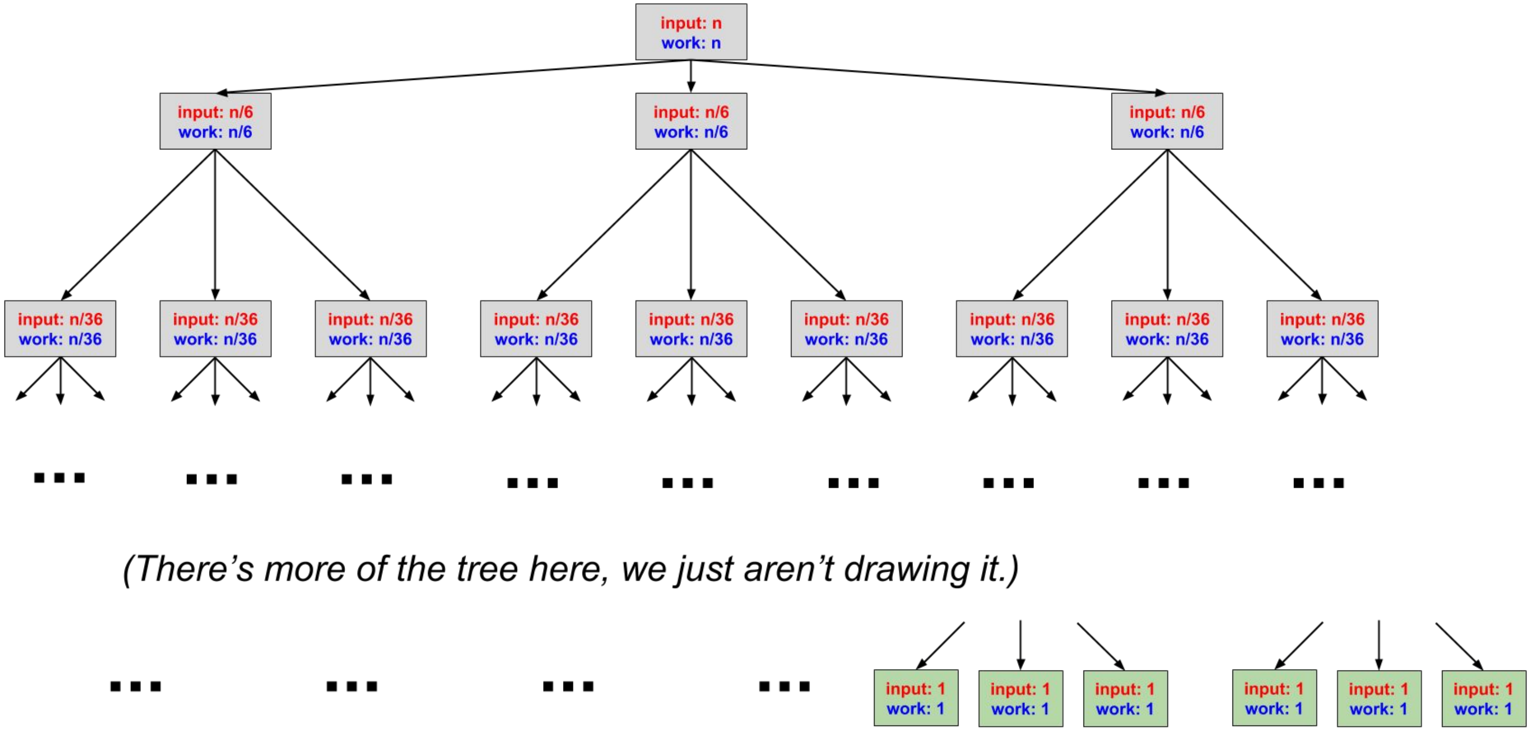


5F: Total Work Done in the Base Case

$$A(n) = \begin{cases} 1 & \text{if } n \le 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

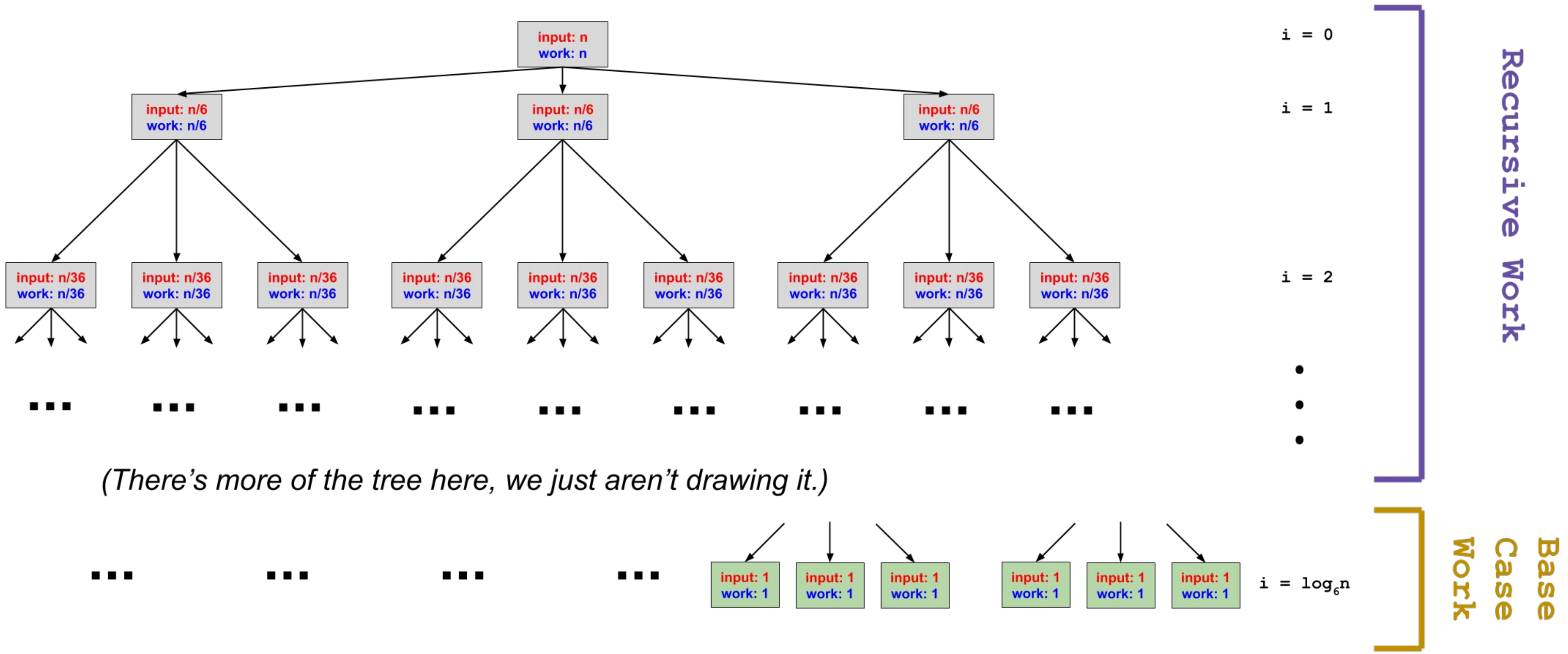
(number of nodes in base case level) x (work per node in base case level)

$$3^{log_6(n)} \cdot 1$$



5G: Total Work Summation

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

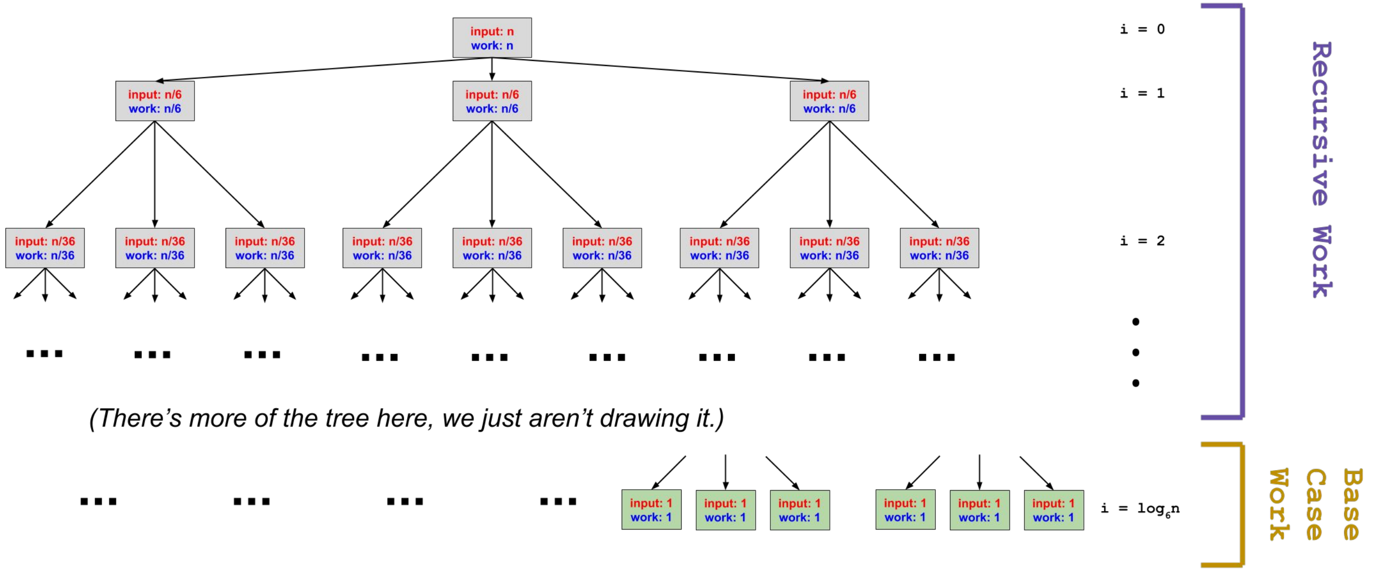


(There's more of the tree here, we just aren't drawing it.)

5G: Total Work Summation

$$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

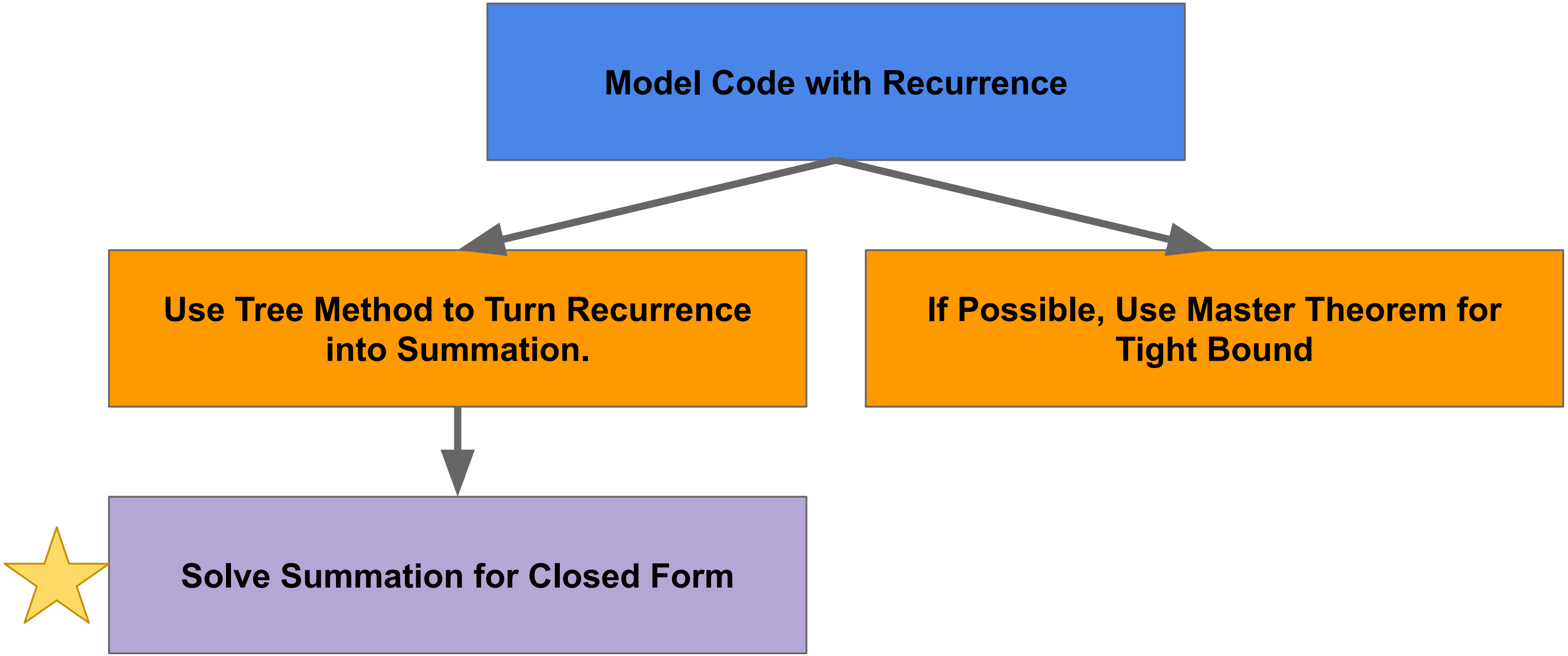
$$\sum_{i=0}^{\log_6(n)-1} [3^i \cdot \frac{n}{6^i}] + 3^{\log_6(n)} \cdot 1$$



Problem 5H: Getting the Closed Form

Where are we?

Three* steps in solving for the runtime of a recursive function.



5H: Simplify to a closed form

Simplify the summation if possible (look for terms that can be pushed out of the summation)

$$\sum_{i=0}^{\log_6(n)-1} [3^i \cdot \frac{n}{6^i}] + 3^{\log_6(n)} \cdot 1$$

$$= \sum_{i=0}^{\log_6(n)-1} [\frac{n}{2^i}] + 3^{\log_6(n)} = n \sum_{i=0}^{\log_6(n)-1} \left(\frac{1}{2}\right)^i + 3^{\log_6(n)}$$

Does this match any of our identities?

5H: Simplify to a closed form

$$= n \sum_{i=0}^{\log_6(n)-1} \left(\frac{1}{2}\right)^i + 3^{\log_6(n)}$$

$$= n * \frac{\left(\frac{1}{2}\right)^{\log_6(n)} - 1}{\frac{1}{2} - 1} + 3^{\log_6(n)}$$

$$= n * \frac{\left(\frac{1}{2}\right)^{\log_6(n)} - 1}{\frac{1}{2} - 1} + n^{\log_6(3)}$$

Finite geometric series

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

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

5H: Simplify to a closed form

$$n * \frac{\left(\frac{1}{2}\right)^{\log_6(n)} - 1}{\frac{1}{2} - 1} + n^{\log_6(3)}$$

NOTE: You don't have to simplify further, but if you were, you would get the following: (See section solutions for steps)

$$= 2n - n^{\log_6(3)}$$

Problem 5I: Master Theorem

5I: Use Master Theorem to find closed form

Original recurrence: $$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

Step 1: Does A(n) match Master Theorem form?

For recurrences in this form, where a, b, c, e are constants:

$$T(n) = \begin{cases} d & \text{if } n \leq \text{some constant} \\ aT(n/b) + e \cdot n^c & \text{otherwise} \end{cases}$$

a = 3

e = 1

b = 6

c = 1

5I: Use Master Theorem to find closed form

Original recurrence: $$A(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 3A(n/6) + n & \text{otherwise} \end{cases}$$

Step 2: Calculate $\log_b(a)$ and compare it to c

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

$a = 3$ $e = 1$

$b = 6$ $c = 1$

$\log_b(a) = \log_6(3)$

$\log_6(3) = x$

$6^x = 3$ so $x < 1$

$c = 1$ so $x < c$

Ta-da!

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