Recurrences

CSE 332 - Section 3

Recurrence Relations

Recurrence Relations

- Describes the time complexity of recursive algorithms, often uses T(n)
  - Same way that f(n) and g(n) described time complexity of non recursive algorithms last week
- Generally in the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$ "Divide & Conquer"

<u>OR</u>

$$T(n) = aT(n-b) + f(n)$$ "Chip & Conquer"

Recurrence Relations

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad \underline{\text{OR}} \quad T(n) = aT(n-b) + f(n)$$

- n = input size
- $T(n)$ = runtime for input size n
- b = how input shrinks for next recursive call(s) (reduction factor/ constant)
- a = number of recursive calls made per function call (branching factor)

```
bar(n) {
    if (n <= 1) {
        return 1;
    }
    return 2 * bar(n/2);
}
```

$a = 1$

$b = 2$

```
foo(n) {
    if (n <= 1) {
        return 1;
    }
    return foo(n-1) + foo(n-1);
}
```

$a = 2$

$b = 1$

Problem 0a

Recurrence relation forms:

- $T(n) = aT(\frac{n}{b}) + f(n)$
- $T(n) = aT(n - b) + f(n)$

Find a recurrence *T(n)* modelling the *worst-case runtime complexity* of *f(n)*

```
f(n) {
  if (n <= 0) {
    return 1
  }
  return 2 * f(n - 1) + 1
}
```

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 0 \\ T(n-1) + c_1 & \text{otherwise} \end{cases}$$

- When does the base case occur? $n \leq 0$
- What is the branching factor a? $a = 1$ since we only make one recursive call
- What is the reduction factor/constant b? $b = 1$ since we always reduce input size by 1
- What is the amount of non-recursive work $f(n)$? constant, which we can denote as c_1

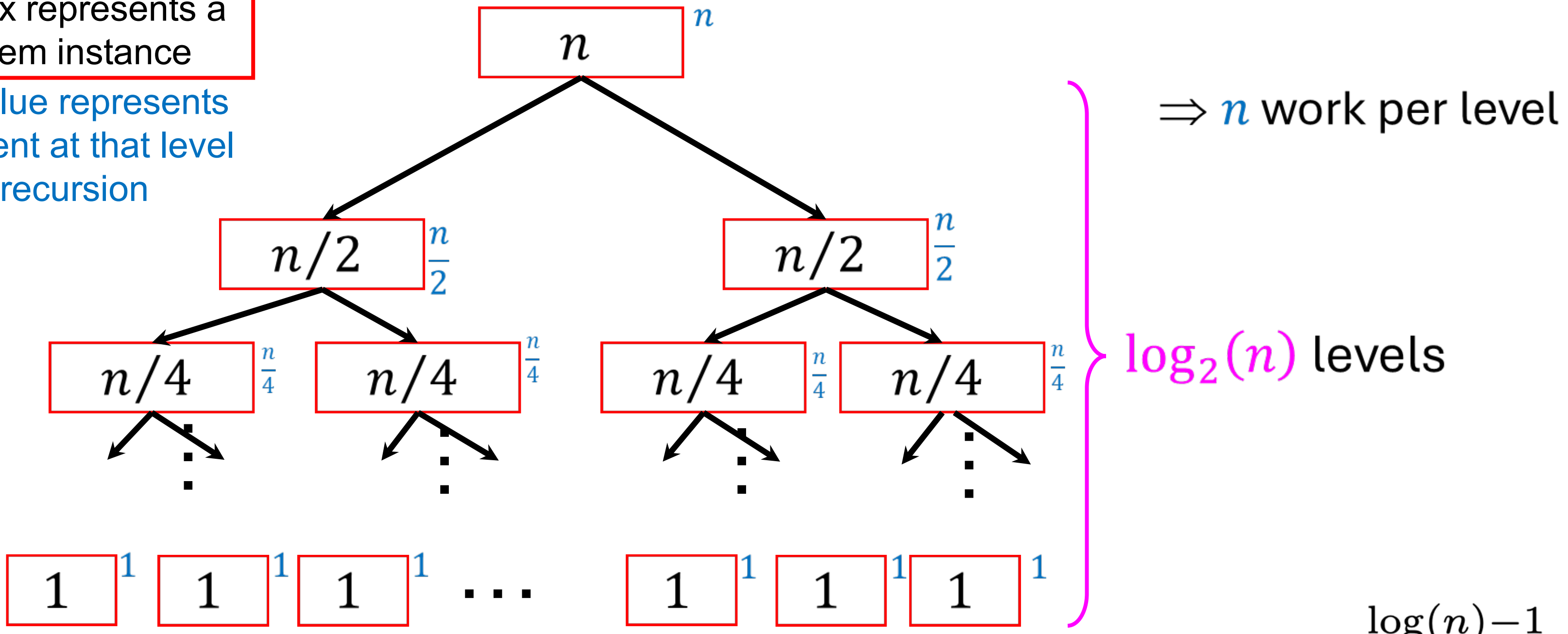
Tree Method Overview

Tree Method Example

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

Red box represents a problem instance

Blue value represents time spent at that level of recursion



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

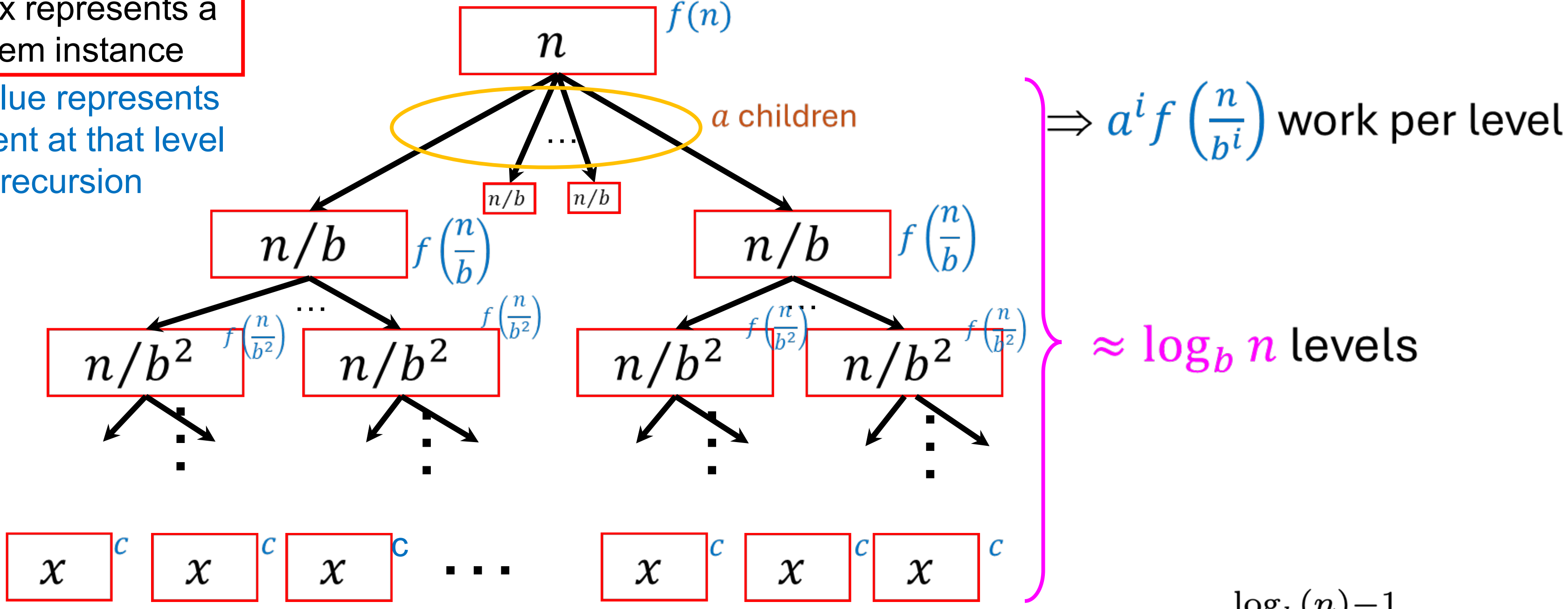
Big Idea: T(n/b)

Asymptotically, these never matter!

$$T(n) = \begin{cases} \boxed{c \quad \text{if } n \le x} \\ a\,T\left(\frac{n}{b}\right) + f(n) & \textit{otherwise} \end{cases}$$

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$$T(n) = \sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right)$$

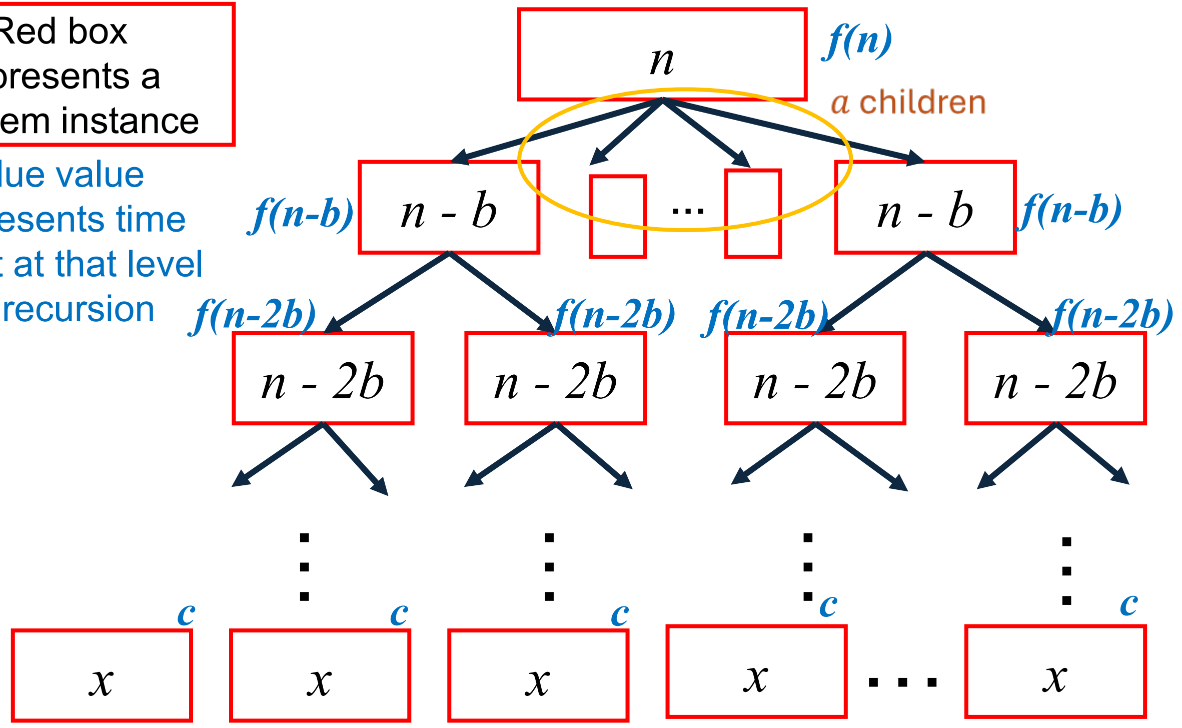
Big Idea: T(n- b)

Asymptotically, these never matter!

$$T(n) = \begin{cases} \boxed{c \qquad \text{if } n \leq x} \\ aT(n-b) + f(n) \quad otherwise \end{cases}$$

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$\Rightarrow a^i f(n - bi)$ work per level

$\approx n/b$ levels

$$T(n) = \sum_{i=0}^{n/b-1} a^i f(n - bi)$$

What Parts Matter?

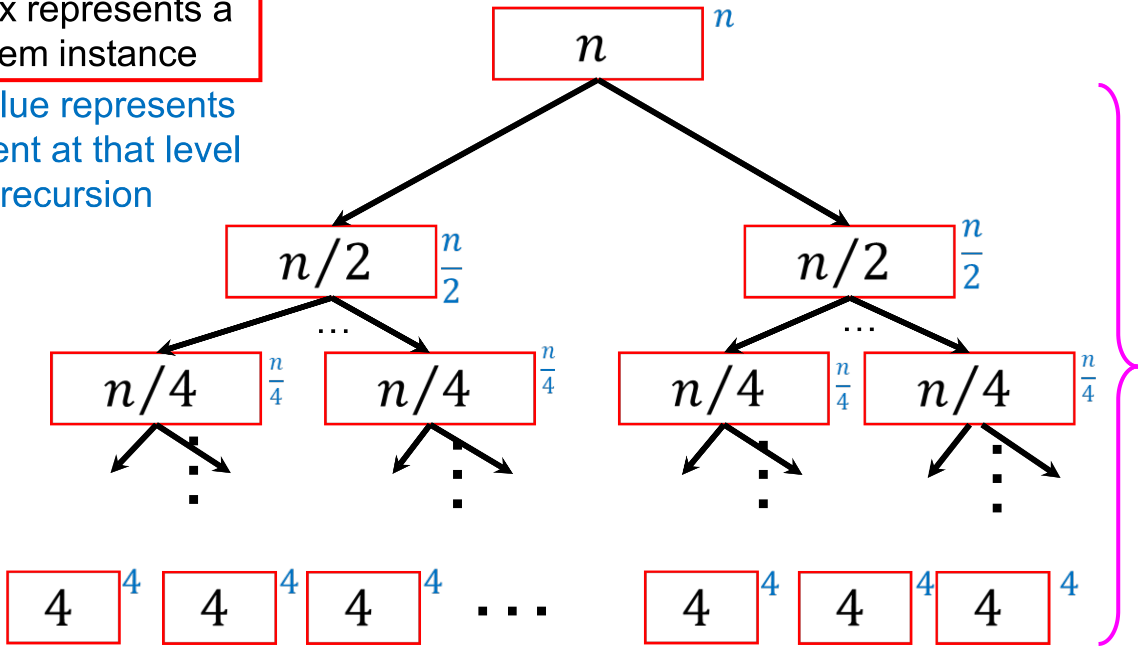
Asymptotically Speaking

Base Case Doesn't Matter!

$$T(n) = \begin{cases} 4 & \text{if } n \leq 4 \\ 2T\left(\frac{n}{2}\right) + n & otherwise \end{cases}$$

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$\Rightarrow n$ work per level

$\log_2(n) - 2$ levels

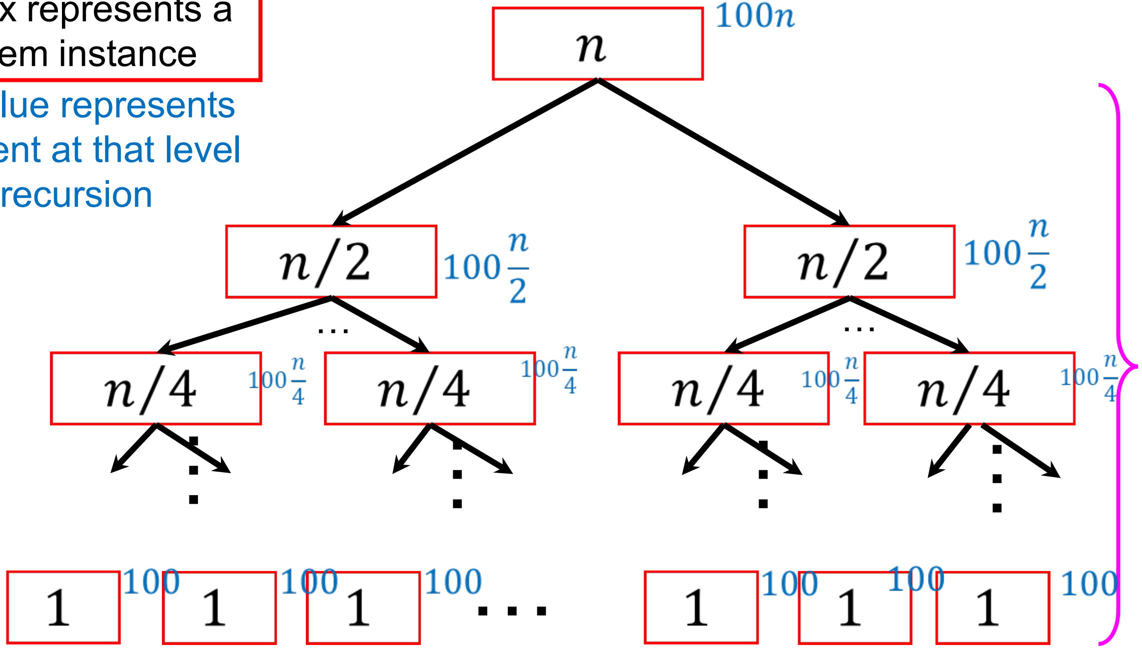
$$T(n) = \sum_{i=0}^{\log_2(n)-3} n$$

Constants for f(n) Don't Matter!

$$T(n) = \begin{cases} 100 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + 100n & otherwise \end{cases}$$

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$\Rightarrow$ $100n$ work per level

$\log_2(n)$ levels

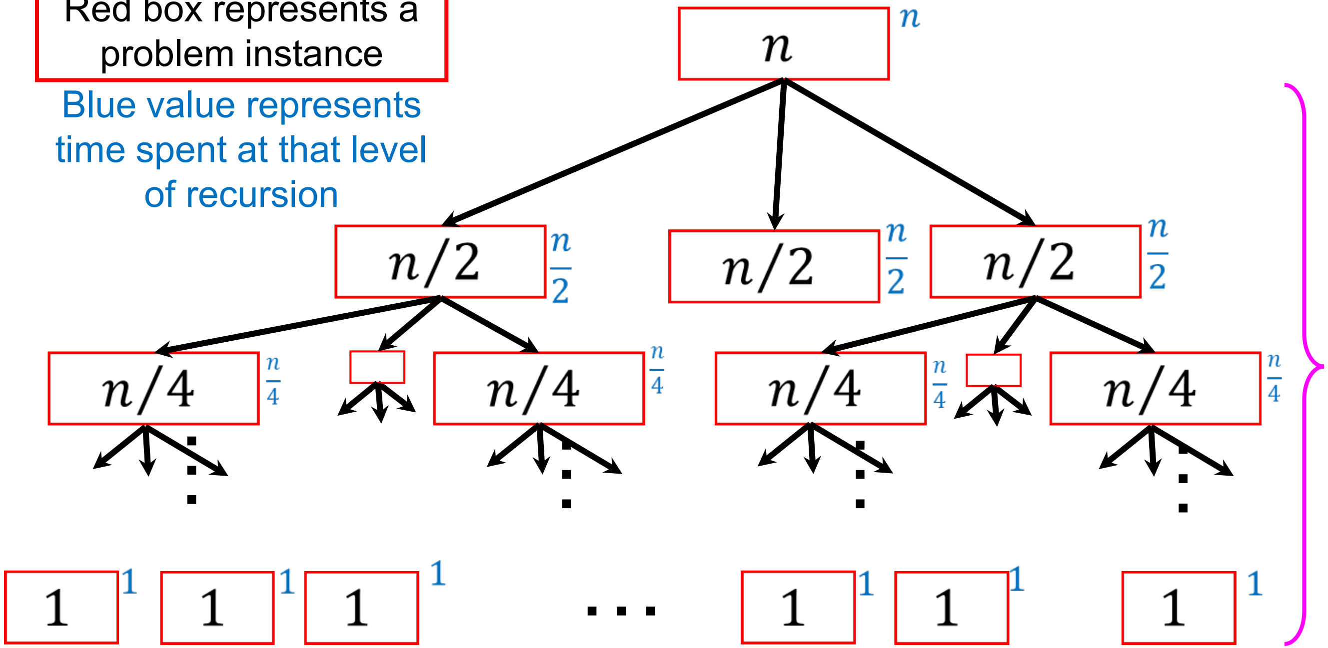
$$T(n) = \sum_{i=0}^{\log_2(n)-1} 100n$$

Branching Factor (a) Matters!

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

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$\Rightarrow \left(\frac{3}{2}\right)^i n$ work per level

$\log_2(n)$ levels

$$T(n) = \sum_{i=0}^{\log_2(n)-1} \left(\frac{3}{2}\right)^i n$$

Solving the Summation

$$T(n) = \sum_{i=0}^{\log_2(n)-1} \left(\frac{3}{2}\right)^i n$$

$$= n \sum_{i=0}^{\log_2(n)-1} \left(\frac{3}{2}\right)^i$$ can move the n using the constant multiple rule

$$= n \frac{1 - (\frac{3}{2})^{\log_2 n}}{1 - \frac{3}{2}}$$ Geometric Series Sum Rule $\boxed{\sum_{i=0}^{n} x^i = \frac{1-x^{n+1}}{1-x}}$

$$= n \frac{1 - (\frac{3^{log_2(n)}}{n})}{-\frac{1}{2}}$$ simplification + props. of log & exponents: $(\frac{3}{2})^{log_2(n)} = \frac{3^{log_2(n)}}{2^{log_2(n)}} = \frac{3^{log_2(n)}}{n}$

$$= 2 \cdot 3^{\log_2 n} - 2n$$ multiplied by -2 and distributed our n

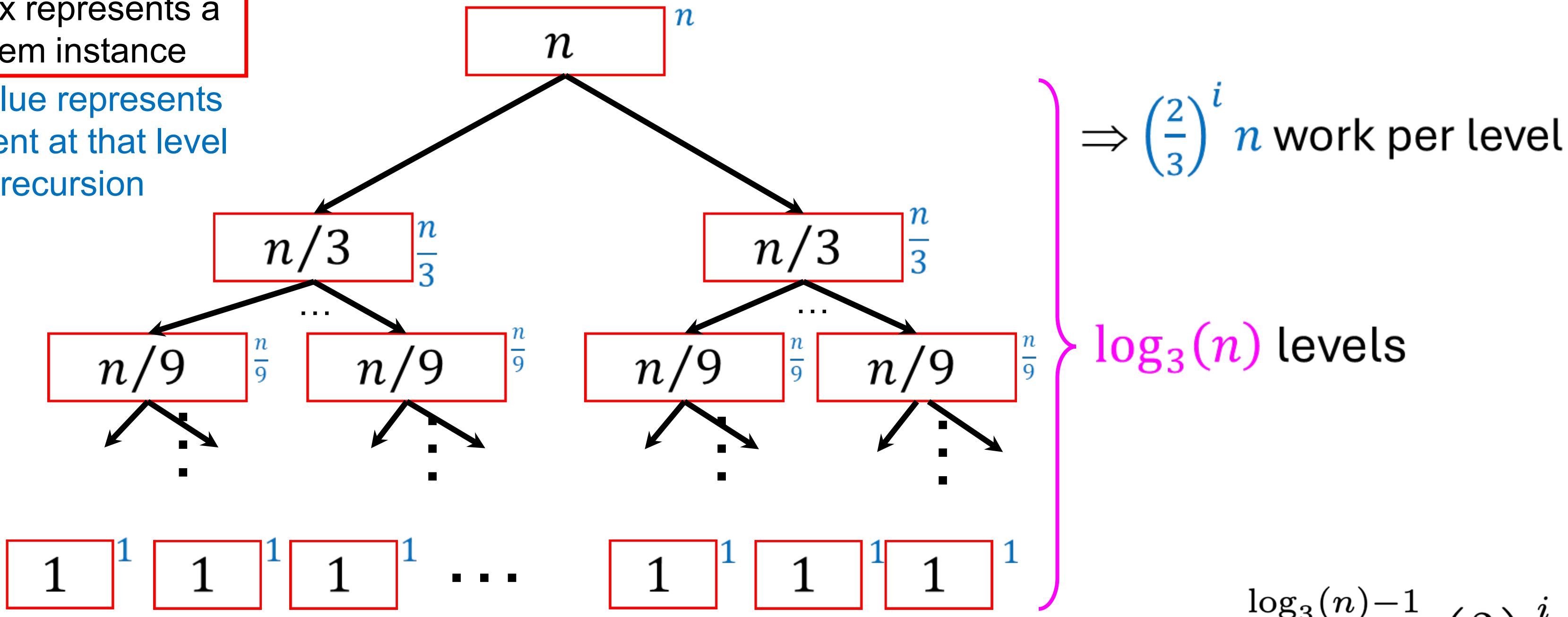
$$= 2 \cdot n^{\log_2 3} - 2n$$ log rules: $3^{\log_2 n} = n^{\log_2 3}$ $(a^{\log_b c} = c^{\log_b a})$

Reduction Factor (/b) Does Matter!

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

Red box represents a problem instance

Blue value represents time spent at that level of recursion



$$T(n) = \sum_{i=0}^{\log_3(n)-1} \left(\frac{2}{3}\right)^i n$$

Solving the Summation

$$T(n) = \sum_{i=0}^{\log_3(n)-1} \left(\frac{2}{3}\right)^i n$$

$$= n \sum_{i=0}^{\log_3(n)-1} \left(\frac{2}{3}\right)^i$$ can move the n using the constant multiple rule

This is a geometric series with a ratio < 1, so it converges to a constant!

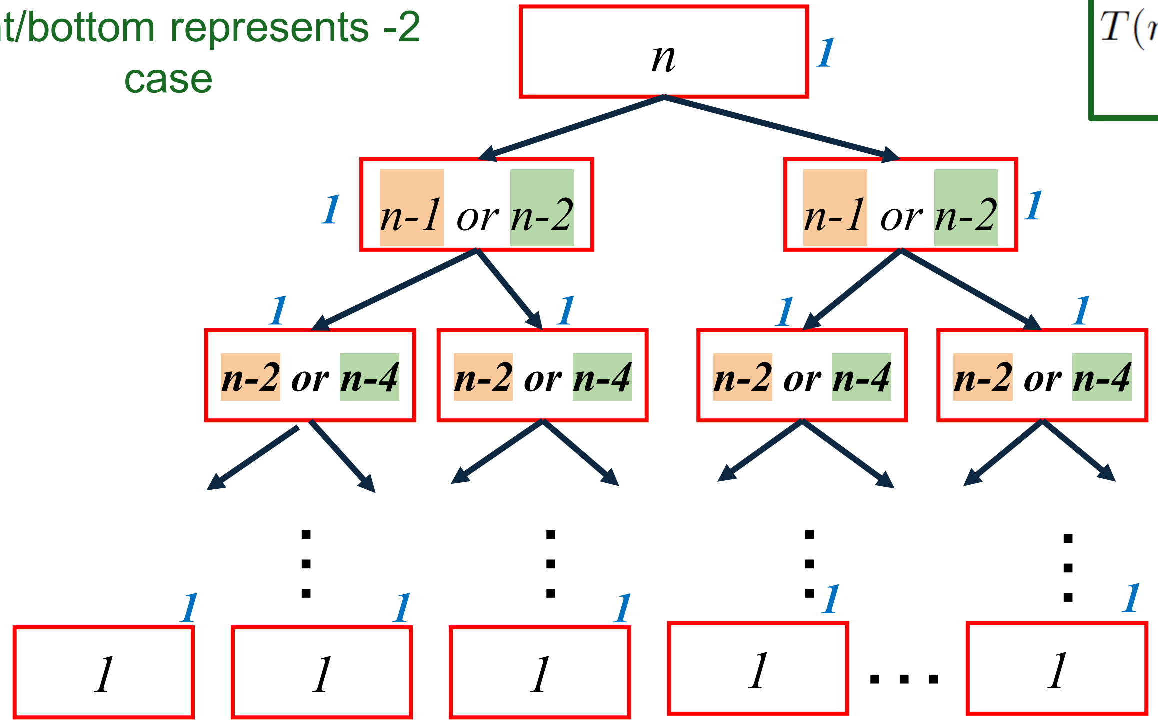
$$\boxed{T(n) \in \Theta(n)}$$

Reduction Constant (-b) Matters!

Left/top represents -1 case
Right/bottom represents -2 case

$$T(n) = \begin{cases} 1 & n \le 1 \\ 2T(n-1)+1 & otherwise \end{cases}$$

$$T(n) = \begin{cases} 1 & n \le 1 \\ 2T(n-2)+1 & otherwise \end{cases}$$



$\Rightarrow$ 2^i
work per level for both cases

$\approx$ n levels for -1
$\approx$ n/2 levels for -2

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

$$T(n) = \sum_{i=0}^{n/2-1} 2^i \approx \sqrt{2}^n$$

Hint: Use the Finite Geometric Series (#7 on Math Identities) to solve these summations!

General Advice

Recursive Running Times- Guidance

- Identify the number of subproblems you will have a recursive call for
  - This gives a
- Identify the size of each of the subproblems
  - This gives b
- Identify (asymptotically) the non-recursive running time
  - You can ignore constants and non-dominant terms!
  - This gives $f(n)$
- Express running time as $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

OR

$$T(n) = aT(n-b) + f(n)$$

Solving $T(n)$ Using The Tree method

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- Draw a tree such that:
  - Each node has a children
  - The "size" of each node is $\frac{1}{b}$ times the size of its parent
  - The "work" for each node is f applied to its size
  - The height of the tree is $\log_b n$
- Sum the tree horizontally
  - i.e. identify the total work done at each level
- Sum the levels' work vertically
  - Gives the sum of all work in the entire tree

Solving $T(n)$ Using The Tree method

$$T(n) = aT(n-b) + f(n)$$

Only differences between /b cases highlighted in yellow

- Draw a tree such that:
  - Each node has a children
  - The "size of each node is $-b$ times the size of its parent
  - The "work" for each node is f applied to its size
  - The height of the tree is n/b
- Sum the tree horizontally
  - I.e. identify the total work done at each level
- Sum the levels' work vertically
  - Given the sum of all work in the entire tree

Putting it All Together

Problem 2(a)

(a) Find a recurrence *T(n)* modeling the *worst-case runtime complexity* of *f(n)*.

```
f(n) {
    if (n <= 1) {
        return 0
    }
    int result = f(n/2)
    for (int i = 0; i < n; i++) {
        result *= 4
    }
    return result + f(n/2)
}
```

$$T(n) = \begin{cases} c_0 & n = 1 \\ \boxed{?} & otherwise \end{cases}$$

Problem 2(a)

(a) Find a recurrence *T(n)* modeling the *worst-case runtime complexity* of *f(n)*.

```
f(n) {
    if (n <= 1) {
        return 0
    }
    int result = f(n/2)
    for (int i = 0; i < n; i++) {
        result *= 4
    }
    return result + f(n/2)
}
```

$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$

$$T(n) = \boxed{a}T\left(\boxed{\frac{n}{b}}\right) + f(n)$$

- 2 function calls -> a = 2
- Reducing input size by half -> (n / 2)
- Non-recursive work has loop with n iterations and some constant work -> f(n) = c_2n + c_1

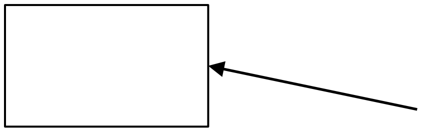
Problem 2(b)

(b) Find a closed form to your answer for (a).

$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$

$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$

$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$



Our first call to T(n)

$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$

Input: n

Our first call to T(n)

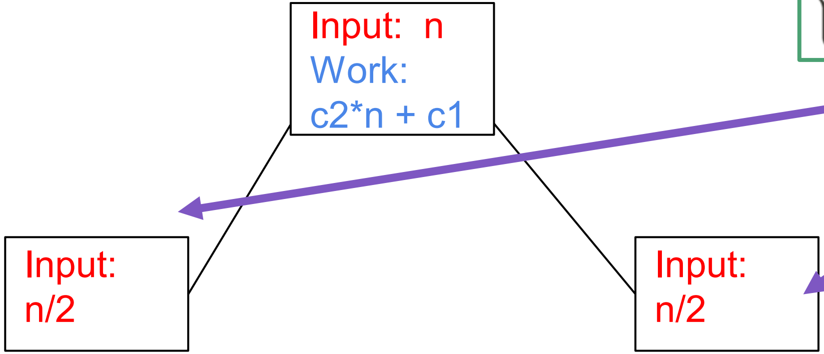
Input: n
Work:
c2*n + c1

$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) \boxed{+ c_2 n + c_1} & otherwise \end{cases}$$

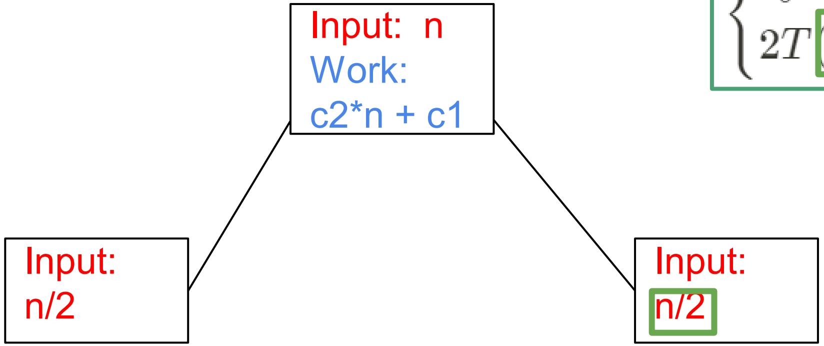
Our first call to T(n)

$$T(n) = \begin{cases} c_0 & n = 1 \\ \boxed{2T(\frac{n}{2})} + c_2 n + c_1 & otherwise \end{cases}$$

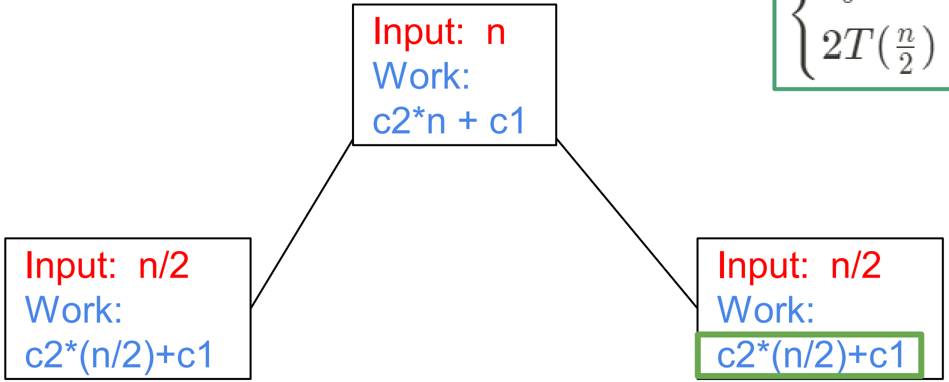
“2T(...)” = 2 recursive calls



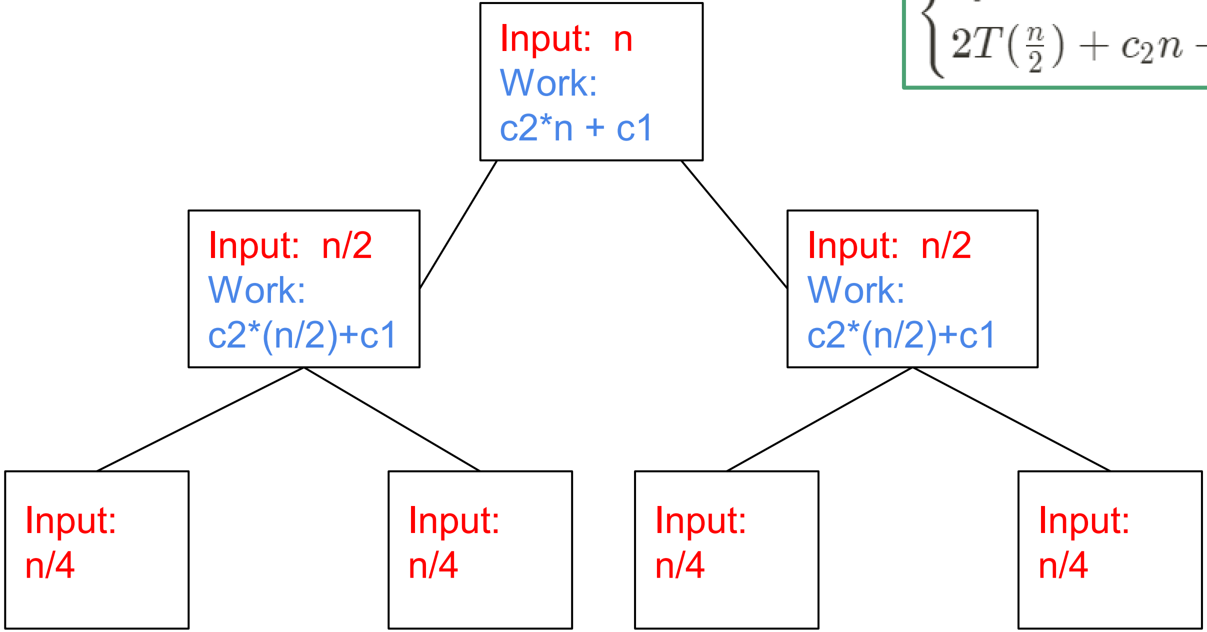
$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T\boxed{\left(\frac{n}{2}\right)} + c_2 n + c_1 & otherwise \end{cases}$$



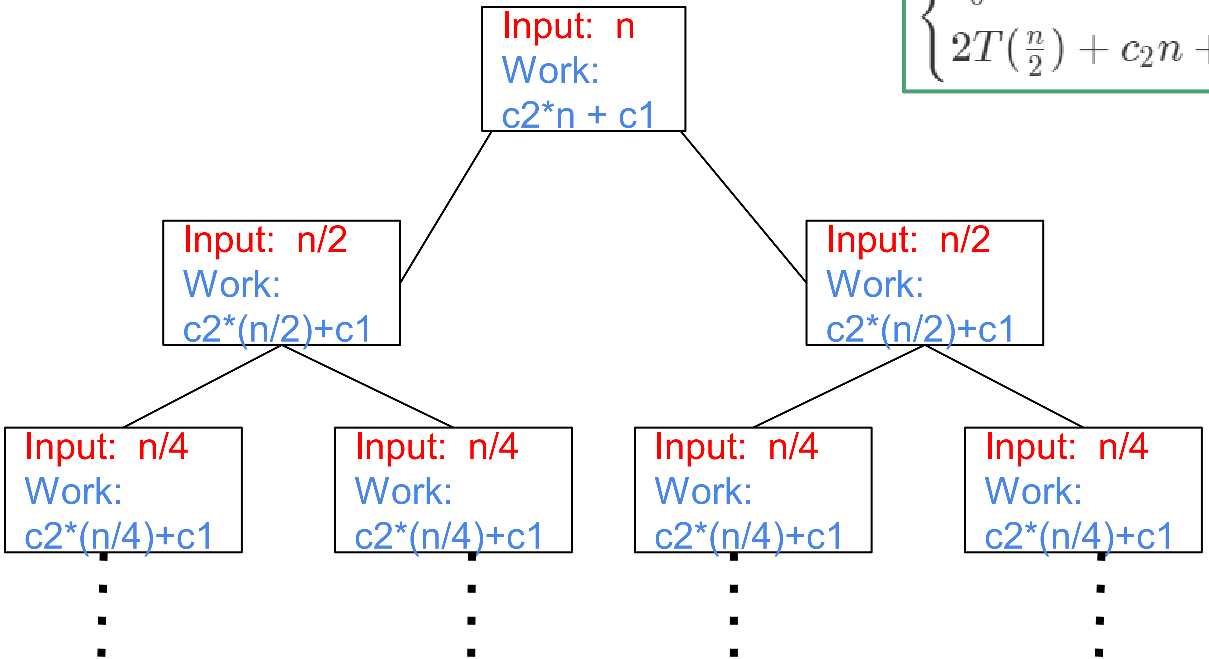
$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + \boxed{c_2 n + c_1} & otherwise \end{cases}$$



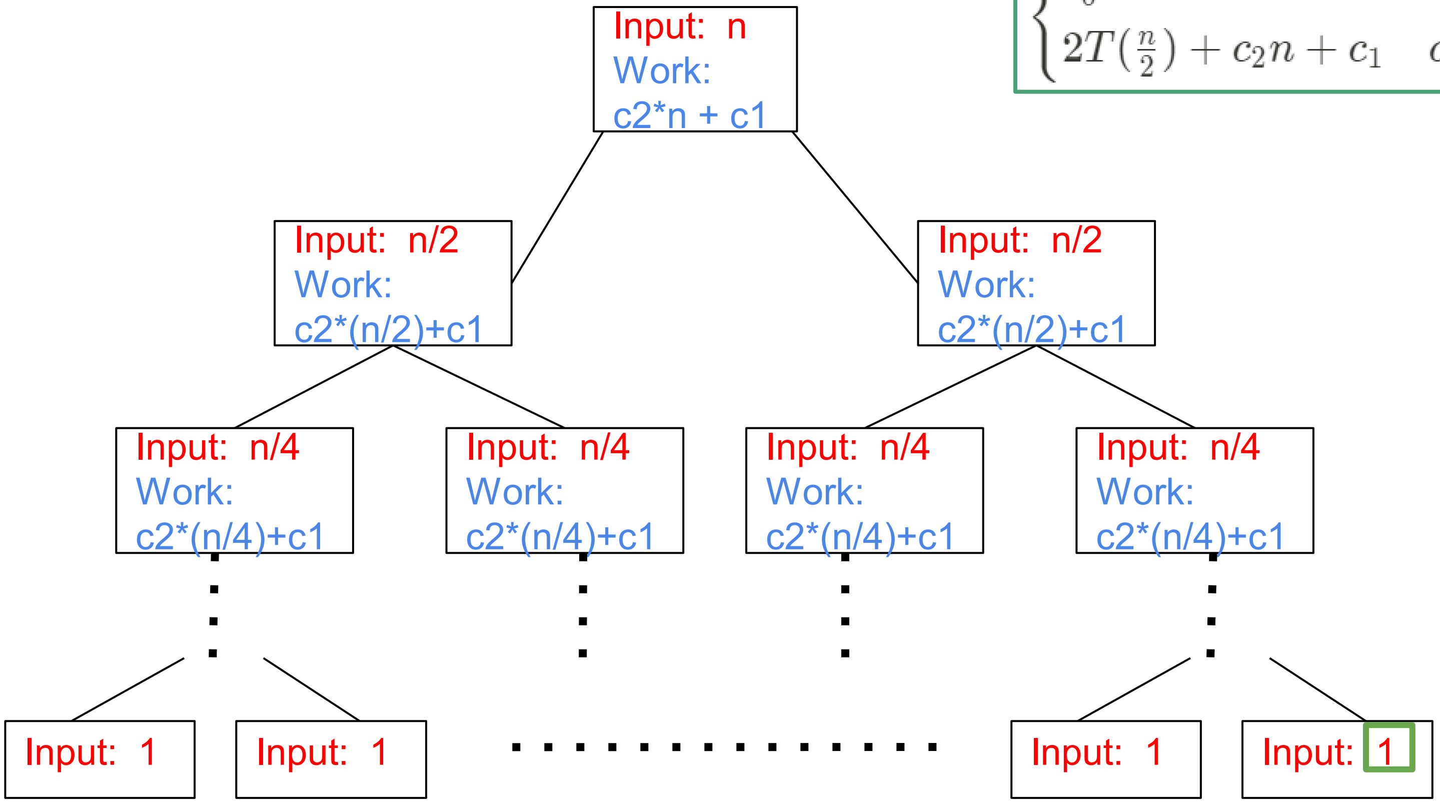
$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$



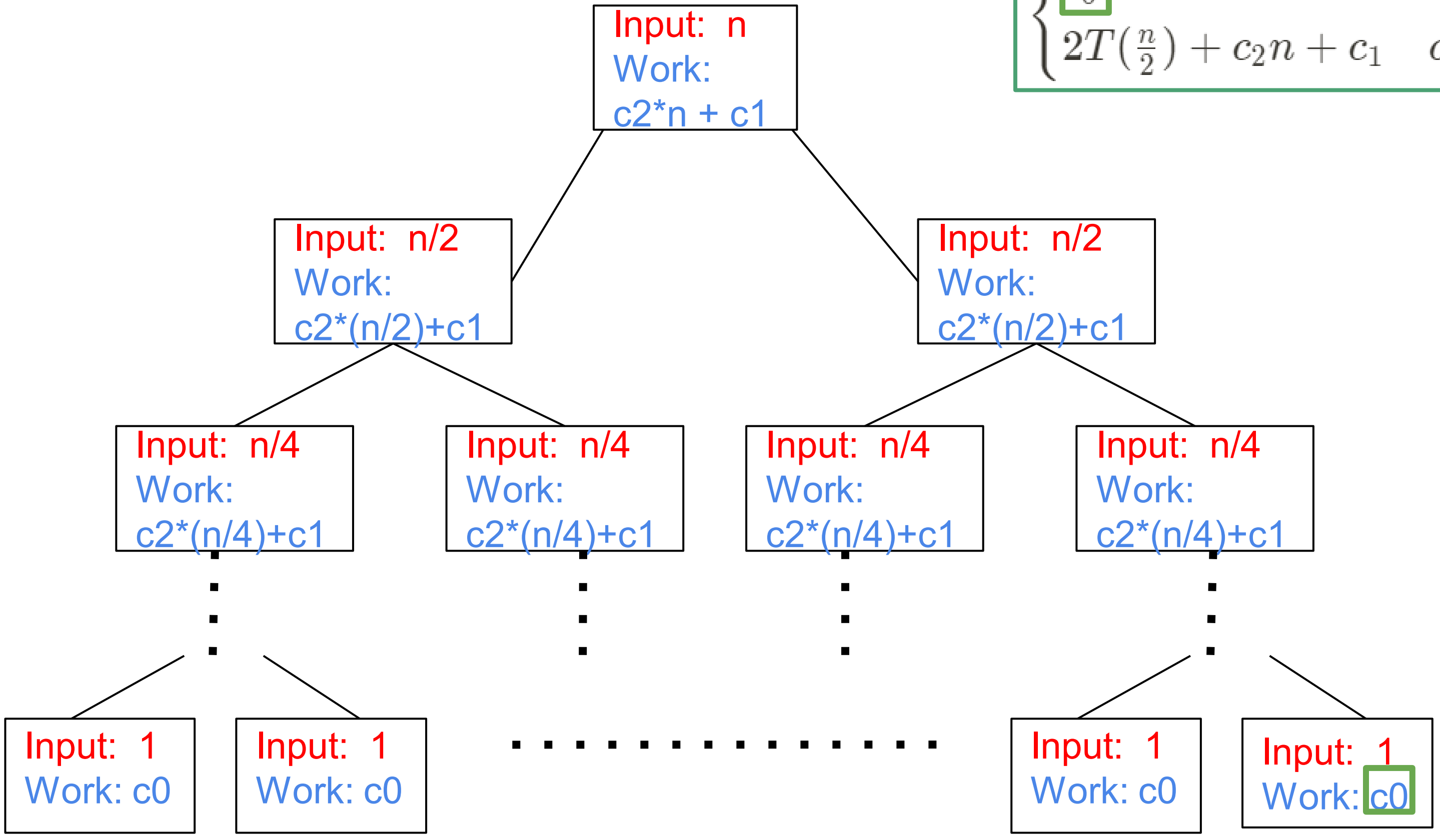
$$T(n) = \begin{cases} c_0 & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$

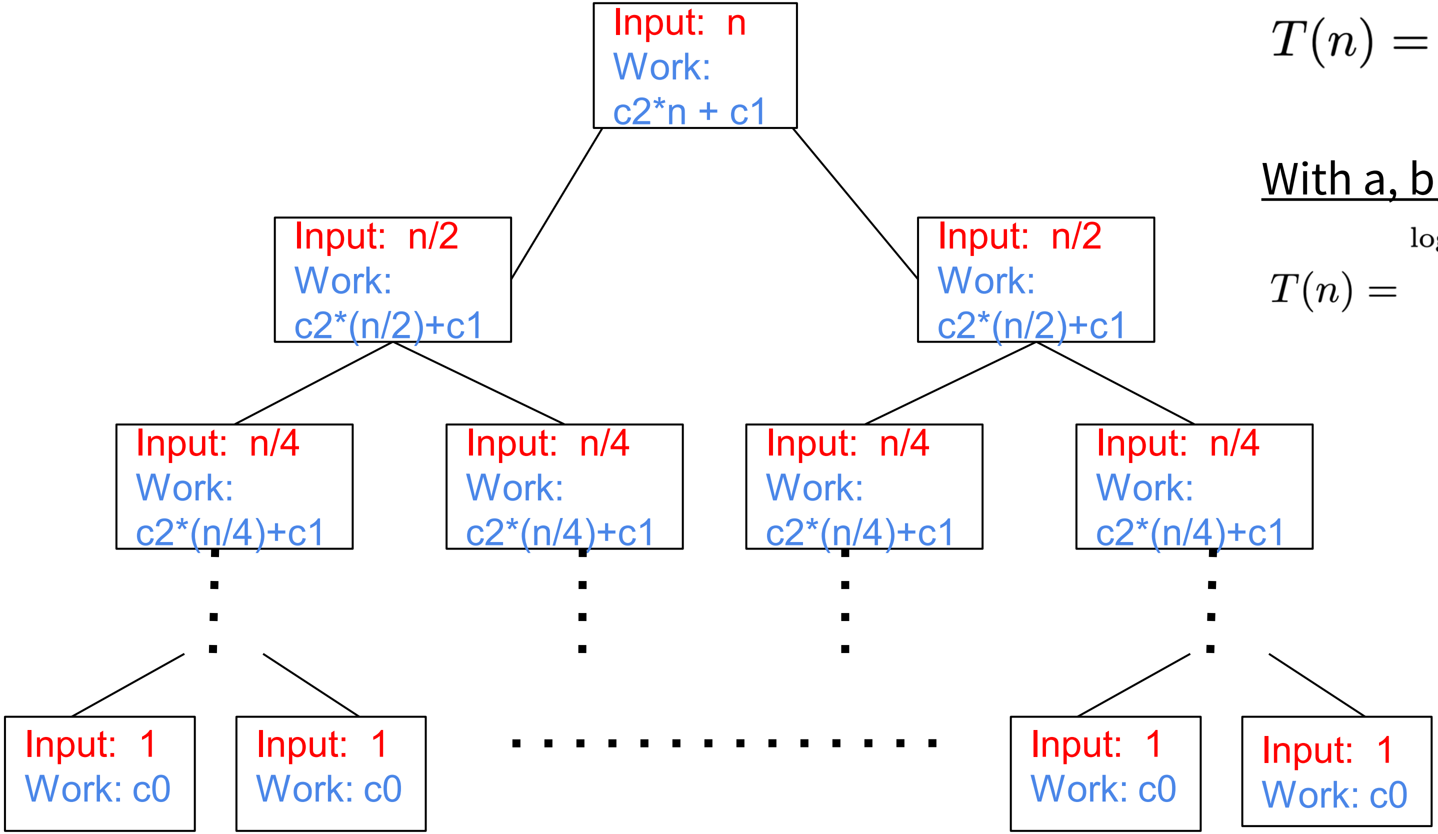


$$T(n) = \begin{cases} c_0 & n = \boxed{1} \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$



$$T(n) = \begin{cases} \boxed{c_0} & n = 1 \\ 2T(\frac{n}{2}) + c_2 n + c_1 & otherwise \end{cases}$$





Since we're in /b case:

$$T(n) = \sum_{i=0}^{\log_b(n)-1} a^i f\left(\frac{n}{b^i}\right)$$

With a, b, and f(n) plugged in:

$$T(n) = \sum_{i=0}^{\log_2(n)-1} 2^i \left(c_2\left(\frac{n}{2^i}\right) + c_1\right)$$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_2(n)-1} 2^i \left(c_2\left(\frac{n}{2^i}\right) + c_1\right) \\
&= \sum_{i=0}^{\log_2(n)-1} \left(c_2 n + 2^i \cdot c_1\right) \\
&= c_2 n \log_2(n) + c_1 \sum_{i=0}^{\log_2(n)-1} 2^i \\
&= c_2 n \log_2(n) + c_1 \frac{1 - 2^{\log_2(n)}}{1-2} \\
&= c_2 n \log_2(n) + c_1 \left(n - 1\right) \\
&\in \Theta(n \log n)
\end{aligned}
$$

Finite Geometric Series:

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

Thank You!