

Recurrences

CSE 332 - Section 3



Part 1: Recurrence Relations



Recurrence Relations Overview

- 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:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad \text{“Divide \& Conquer”}$$

OR

$$T(n) = aT(n - b) + f(n) \quad \text{“Chip \& Conquer”}$$

Recurrence Relations Examples

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

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

```
foo(L) { // n = L.size  
    if (L.size <= 1) {  
        return 1;  
    }  
    L.remove(0); // n = n-1  
    return foo(L) + foo(L);  
}
```

Problem 0a

Recurrence relation forms:

$$\bullet T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

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

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

```
1  f(stack) { // stack.size = n
2      if (stack.size == 0) {
3          return 0
4      }
5      stack.pop // stack.size --
6      return 2*f(stack)+1
7  }
```

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

- When does the base case occur?
- What is the branching factor a ?
- What is the reduction factor / constant b ?
- What is the amount of non-recursive work $f(n)$?

Problem 0a

Recurrence relation forms:

$$\bullet T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

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

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

```
1  f(stack) { // stack.size = n
2      if (stack.size == 0) {
3          return 0
4      }
5      stack.pop // stack.size --
6      return 2*f(stack)+1
7  }
```

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

- 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

Problem 0b

Recurrence relation forms:

$$\bullet T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

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

Find a recurrence $T(n)$ modelling the *worst-case runtime complexity* of $\text{bar}(n)$

```
1 bar(A, start, n) {
2   if n ≤ 10000 {
3     return 1000
4   }
5   if (bar(A, start, n/3) > 5) {
6     for (int i=0; i<n; i++) {
7       println("Yay")
8     }
9     return 5*bar(A, start, n/3)
10  } else {
11    for (int i=0; i<n*n; i++) {
12      println("Yay")
13    }
14    return 4 * bar(A, start, n/3)
15  }
```

- When does the base case occur? $n \leq 10000$
- What is the branching factor a ? $a = 2$
- What is the reduction factor b ? $b = 3$
- What is the amount of non-recursive work $f(n)$? $c_1 * n + c_2$

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 10000 \\ \text{[redacted]} & \text{otherwise} \end{cases}$$

Tree Method Overview



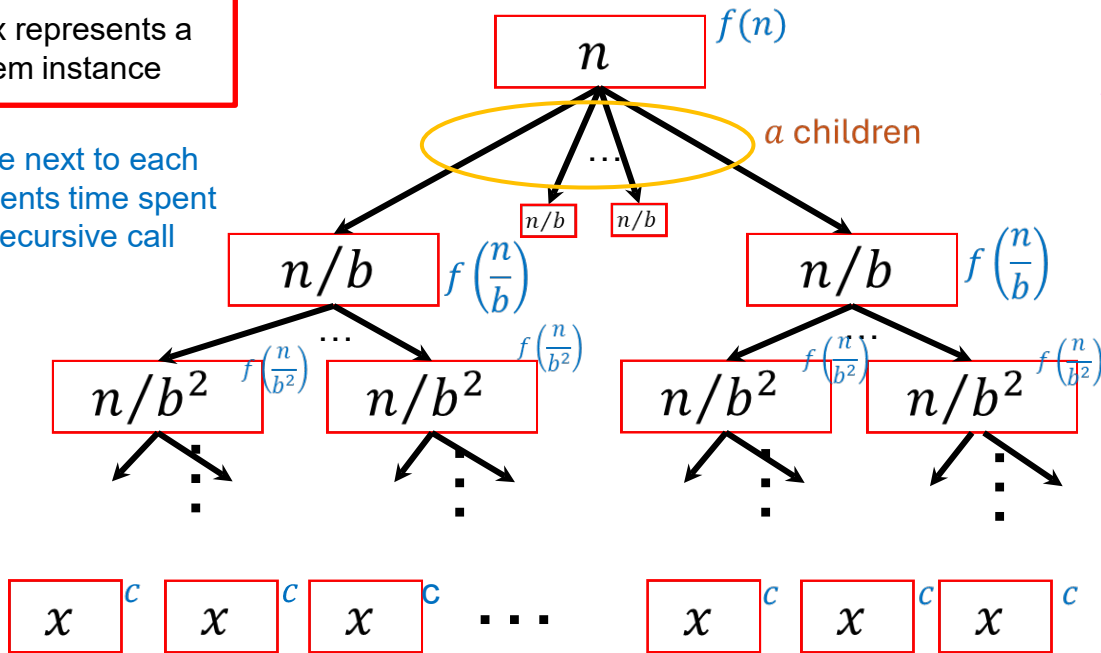
Big Idea: $T(n/b)$

Asymptotically, these never matter!

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

Red box represents a problem instance

Blue value next to each box represents time spent at that recursive call



$$\Rightarrow a^i f\left(\frac{n}{b^i}\right) \text{ work per level}$$

$$\approx \log_b n \text{ levels}$$

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

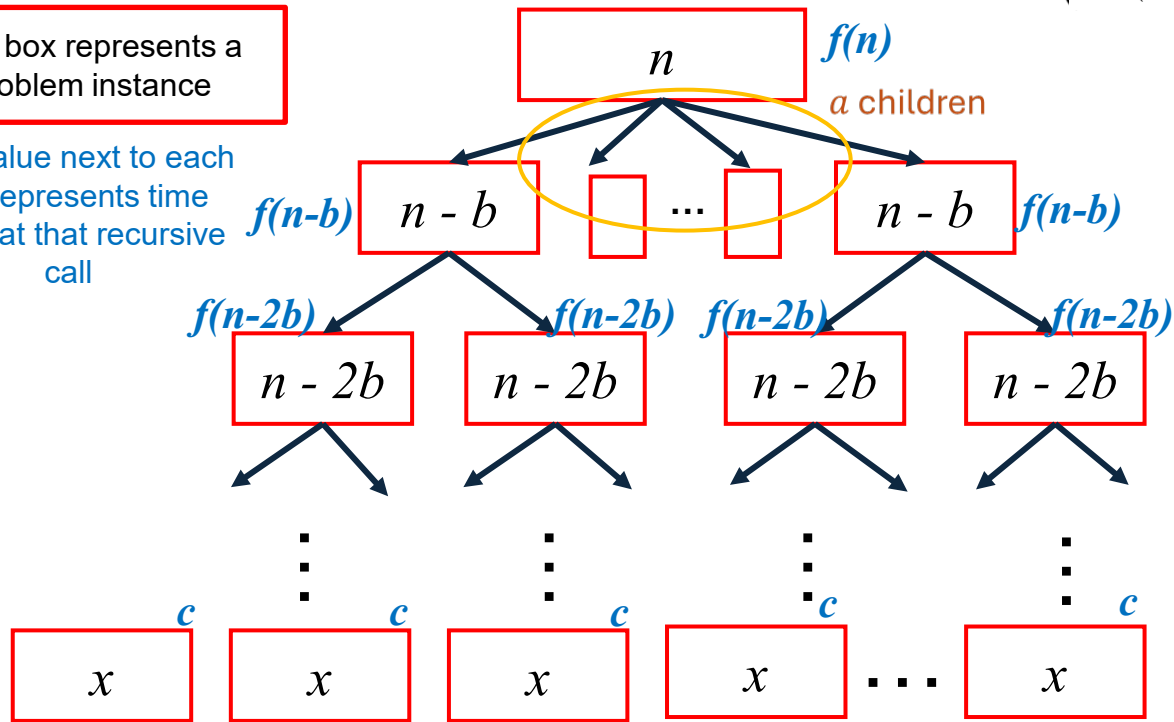
Big Idea: $T(n-b)$

Asymptotically, these never matter!

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

Red box represents a problem instance

Blue value next to each box represents time spent at that recursive call



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

$\approx \frac{n}{b}$ levels

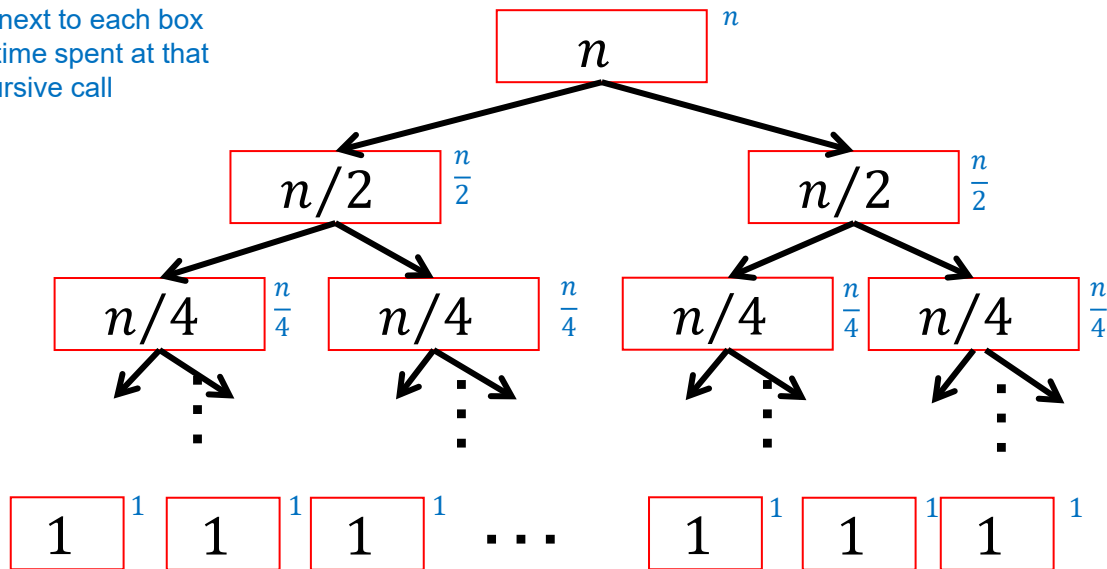
$$T(n) = \sum_{i=0}^{\frac{n}{b}-1} a^i f(n - bi)$$

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

Red box represents a problem instance

Blue value next to each box represents time spent at that recursive call

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



$$\Rightarrow 2^i * \frac{n}{2^i} = n$$

work per level
(root level is 0)

$\log_2 n$ levels
of recursion

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

Q1(a) Solving the Summation

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

$$= [(\log_2(n) - 1) + 1] \cdot n$$

$$= n \cdot \log_2(n)$$

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

Your Turn!

Try problems 1b-1f

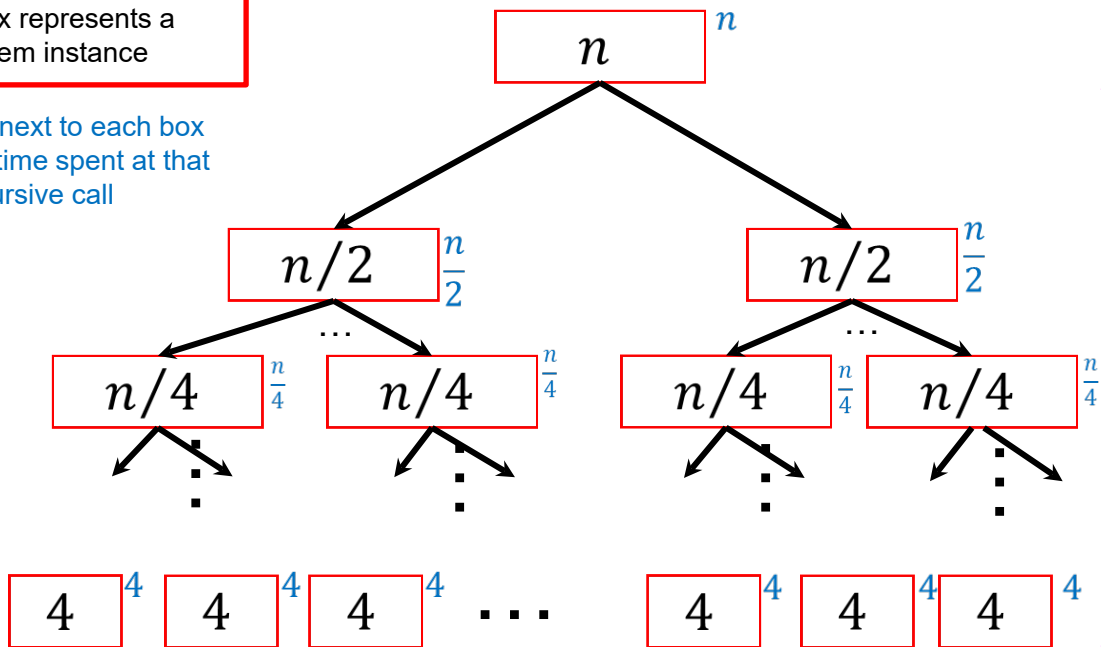


Q1(b) Base Case Doesn't Matter!

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

Red box represents a problem instance

Blue value next to each box represents time spent at that recursive call



$$\Rightarrow 2^i * \frac{n}{2^i} = n \text{ work per level}$$

$\log_2(n) - 2$ levels of recursion

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

Q1(b) Solving the Summation

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

$$= [(\log_2(n) - 3) + 1] \cdot n$$

$$n \cdot \log_2(n) - 2n$$

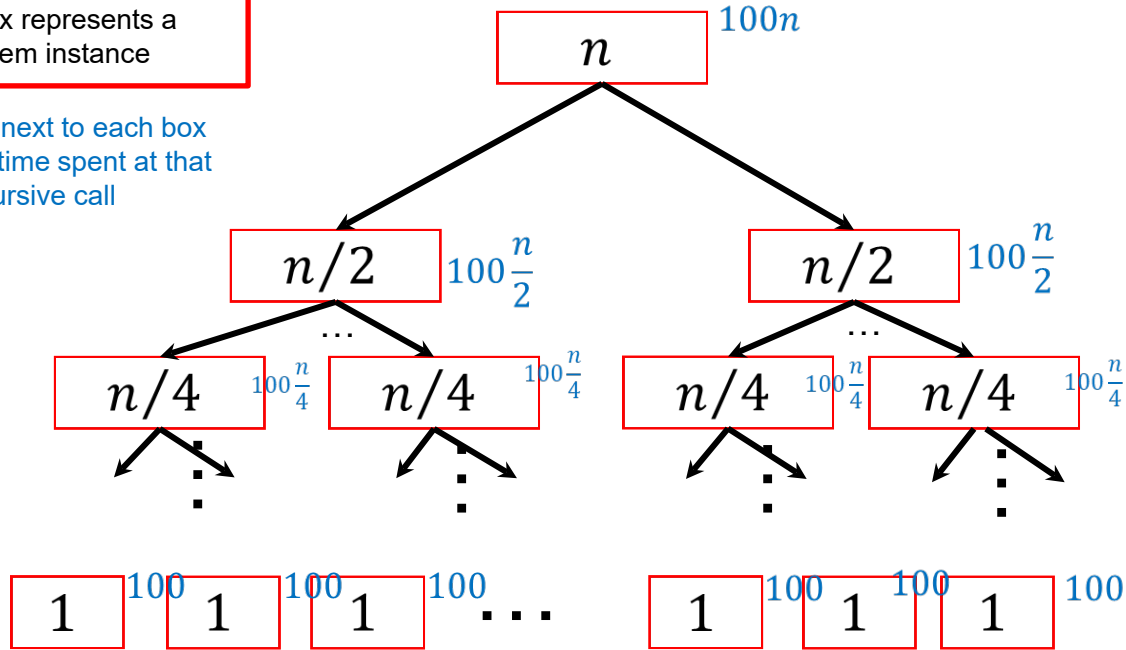
$$\in \Theta(n \log n)$$

Q1(c) Constants for $f(n)$ Don't Matter!

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

Red box represents a problem instance

Blue value next to each box represents time spent at that recursive call



$$\Rightarrow 2^i * \frac{100n}{2^i} = 100n$$

work per level

$\log_2(n)$ levels of recursion

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

Q(1c) Solving the Summation

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

$$= [(\log_2(n) - 1) + 1] \cdot 100n$$

$$100n \cdot \log_2(n)$$

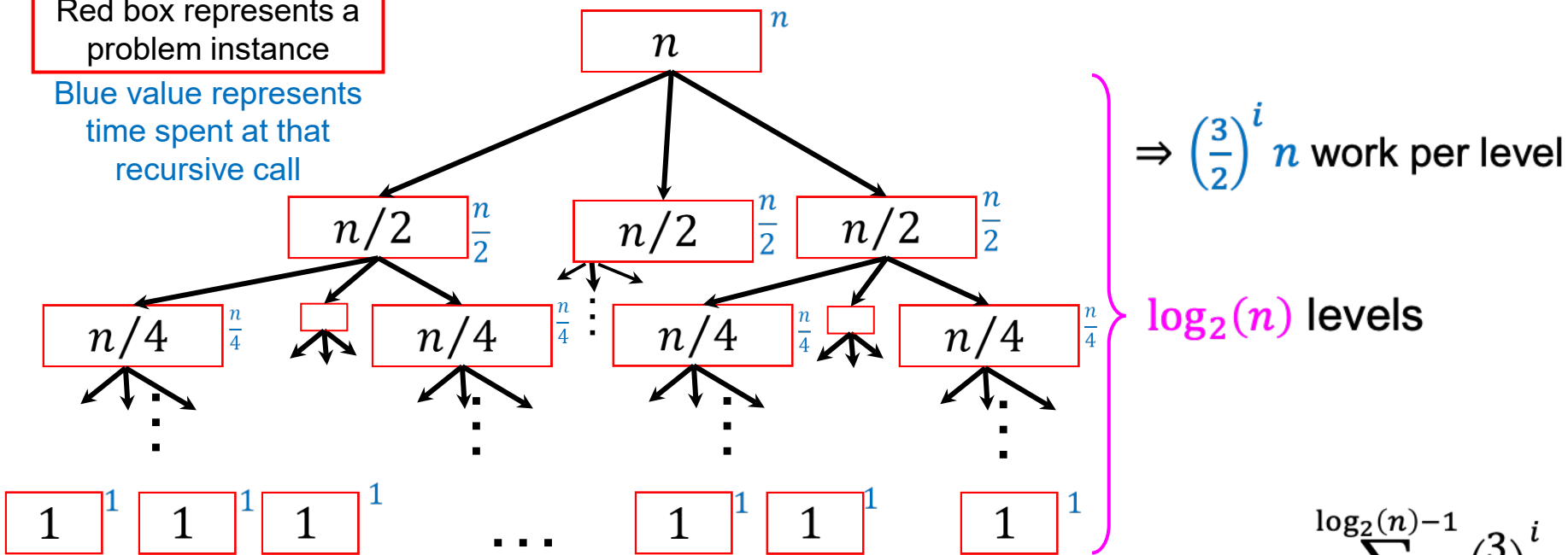
$$\in \Theta(n \log n)$$

Q1(d) Branching Factor (a) Matters!

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

Red box represents a problem instance

Blue value represents time spent at that recursive call



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

Q(1d) 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

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

Geometric Series Sum Rule:

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

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

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

$$= 2 \cdot 3^{\log_2 n} - 2n$$

multiplied by -2 and distributed our n

$$= 2 \cdot n^{\log_2 3} - 2n$$

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

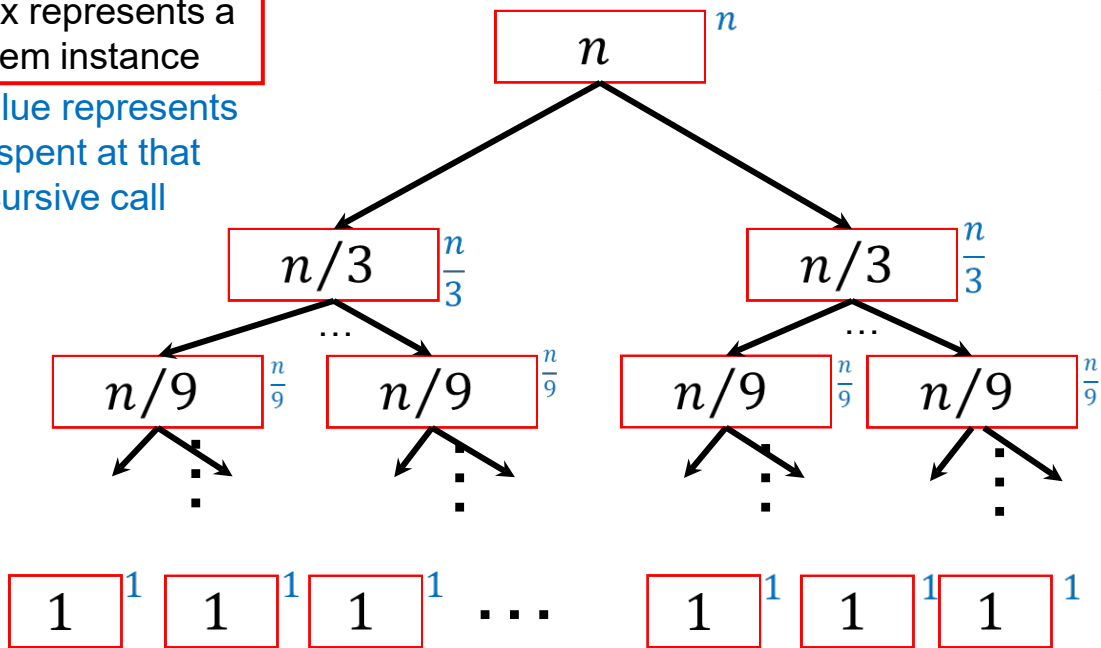
$$\in \Theta(n^{\log_2 3})$$

Q1(e) Reduction Factor (/b) Does Matter!

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

Red box represents a problem instance

Blue value represents time spent at that recursive call



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

$\log_3(n)$ levels

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

Q(1e) 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!

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

Q1(f): Tree method

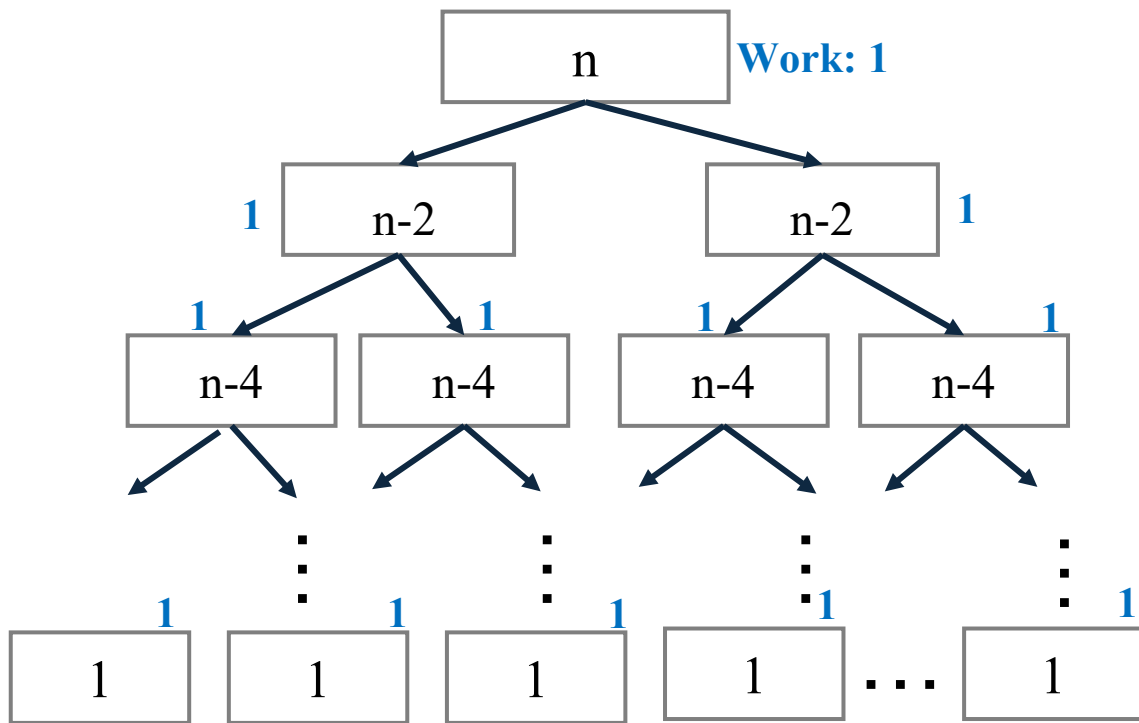
$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2T(n-1) + 1 & \text{otherwise} \end{cases}$$

#children #levels work

$\Rightarrow 2^i$ work per level

$\approx n/2$ levels

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



Q1(f): Solving the Summation

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

$$= 2^0 \cdot \frac{1 - 2^{\frac{n}{2}}}{1 - 2}$$

(Sum of a finite geometric series):

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

$$= \frac{1 - 2^{\frac{n}{2}}}{-1}$$

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

$$= 2^{\frac{n}{2}} - 1$$

$$= \sqrt{2}^n - 1$$

$$\Rightarrow \in \Theta\left(\sqrt{2}^n\right)$$

Note: formula like this will be provided for exams

Reduction Constant (-b) Matters!

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

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

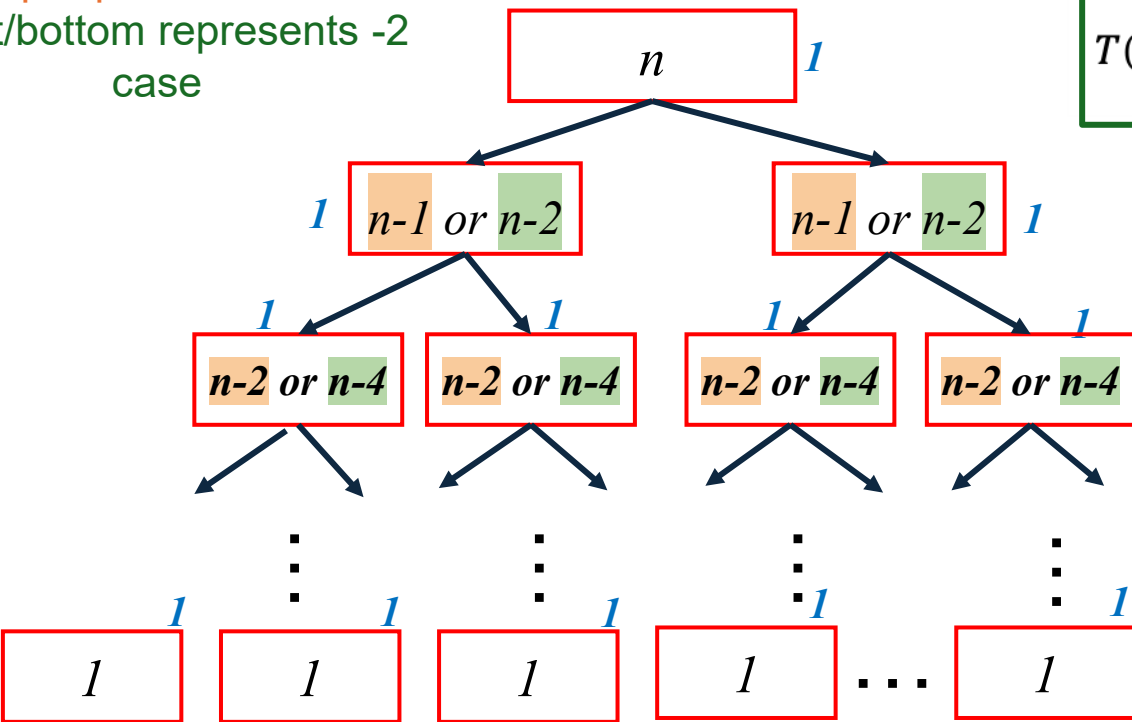
$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2T(n-2) + 1 & \text{otherwise} \end{cases}$$

$$\Rightarrow 2^i$$

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 $1/b$ 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
 - Identify the total work done at each level
- Sum the levels’ work vertically
 - This 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 divide and conquer cases highlighted in yellow

- Draw a tree such that:
 - Each node has a children
 - The “size of each node is b less than 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 $f(n)$.

```
1  f(L, start, n) {  
2      if (n - start <= 1) {  
3          return 0  
4      }  
5      int res = f(L, start, n/2)  
6      for (int i=0; i<n; i++) {  
7          result *= 4  
8      }  
9      return res + f(L, start, n/2)  
10 }
```

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

Problem 2(a)

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

```
1  f(L, start, j) { //n = j - start
2      if (j - start <= 1) {
3          return 0
4      }
5      int res = f(L, start, j/2)
6      for (int i=0; i<j; i++) {
7          result *= 4
8      }
9      return res + f(L, start, j/2)
10 }
```

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

$$T(n) = \begin{cases} c_0, n = 1 \\ 2T\left(\frac{n}{2}\right) + c_2n + c_1, \text{otherwise} \end{cases}$$

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

Problem 2(b)

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

$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$

$$T(n) = \begin{cases} c_0, n = 1 \\ 2T(n/2) + c_2n + c_1, otherwise \end{cases}$$

$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$



Our first call to $T(n)$

$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$

Input: n

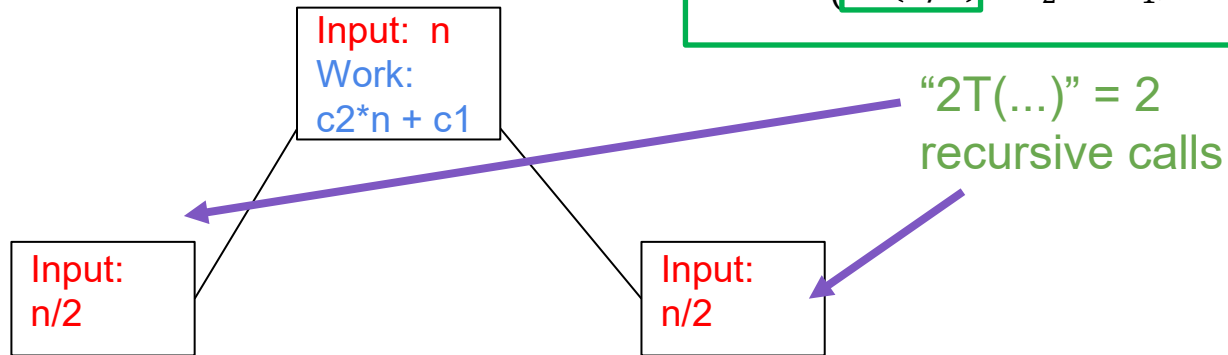
Our first call to $T(n)$

Input: n
Work:
 $c_2 \cdot n + c_1$

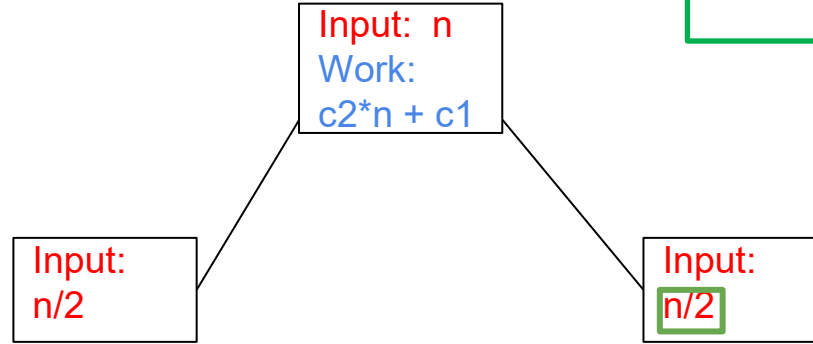
$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2 n + c_1, & \text{otherwise} \end{cases}$$

Our first
call to $T(n)$

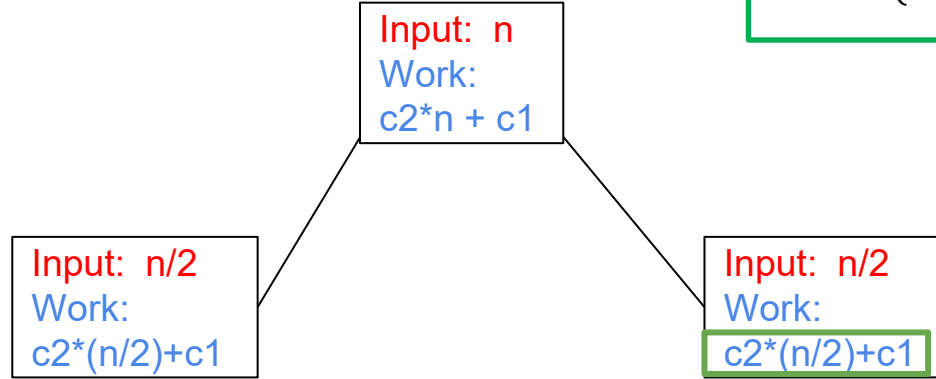
$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$



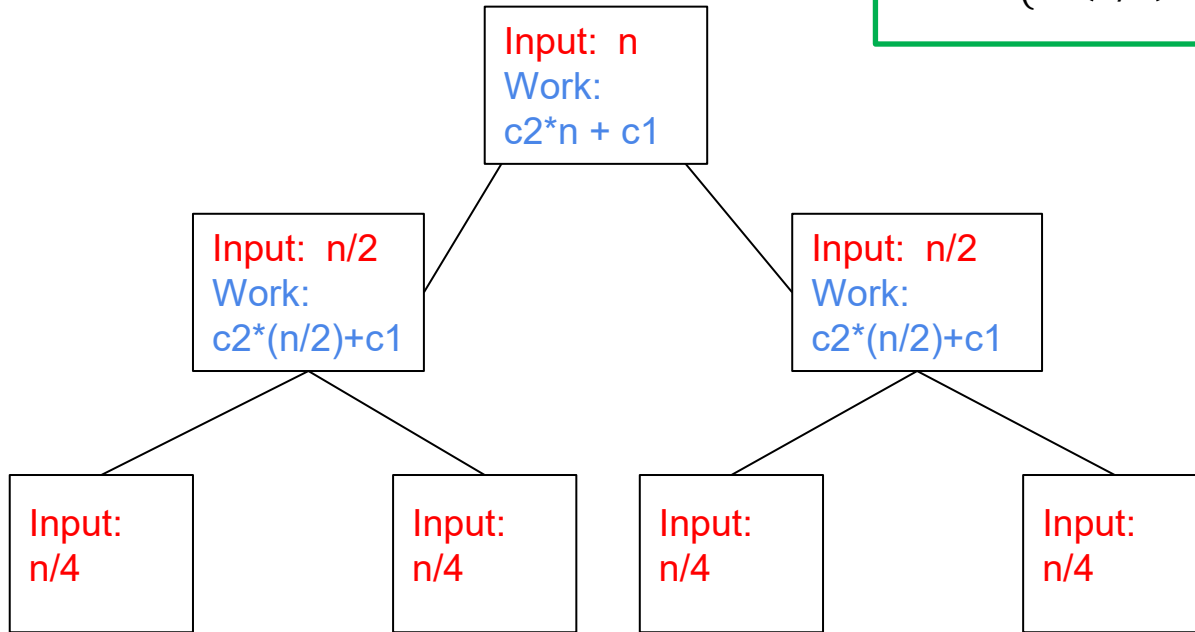
$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$



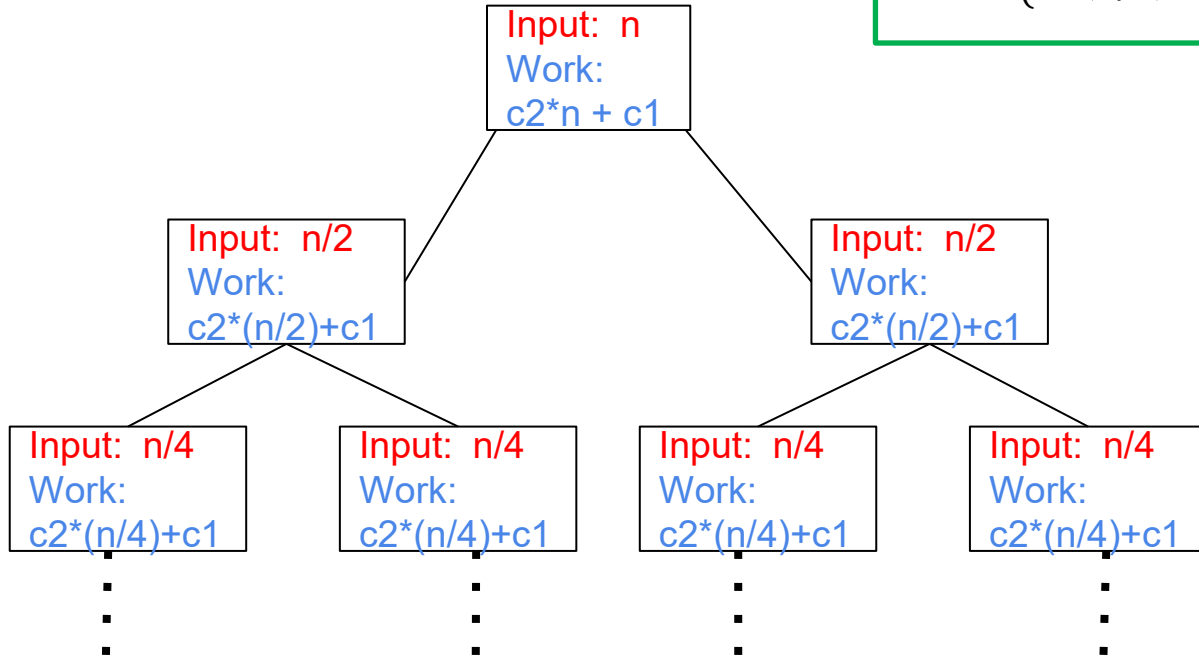
$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$



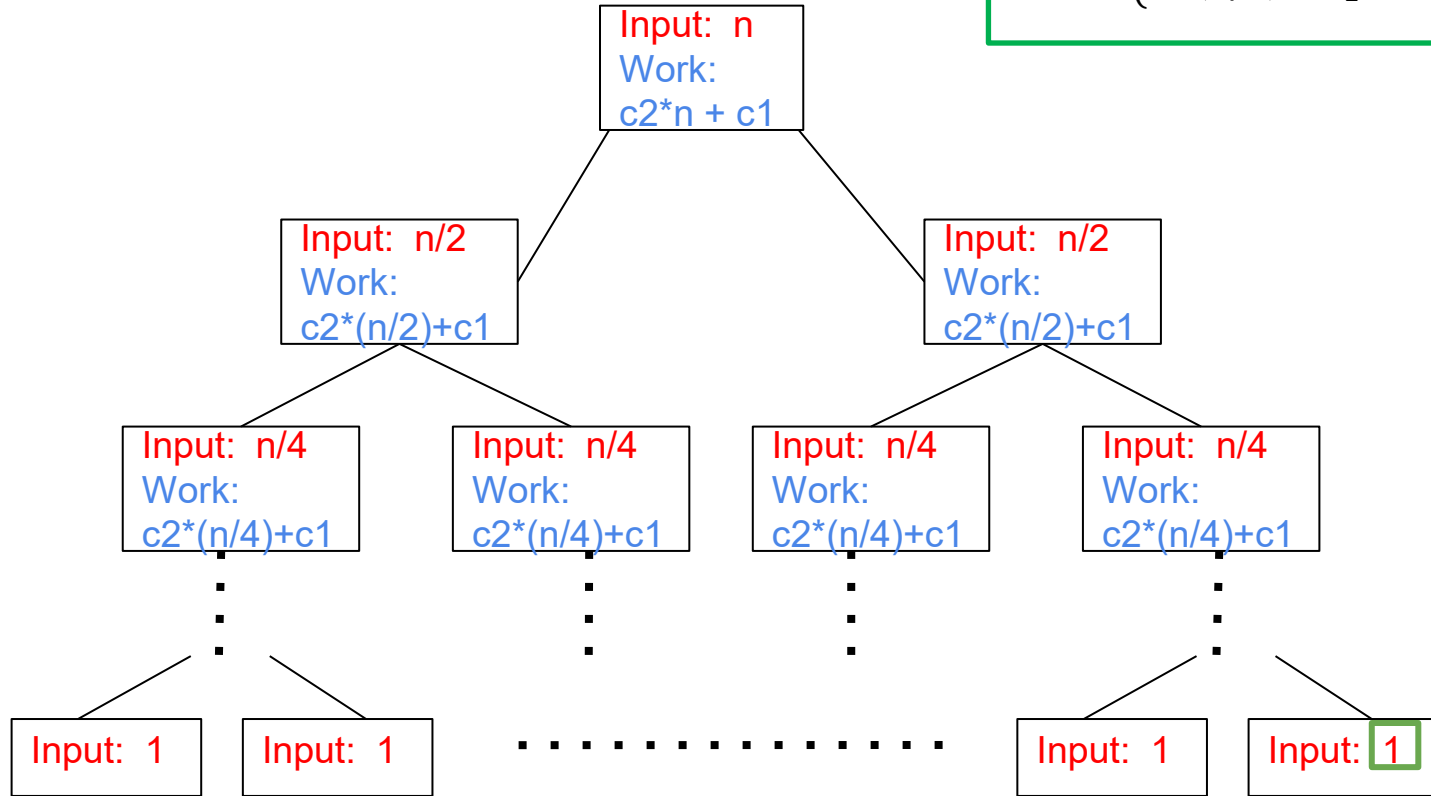
$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$



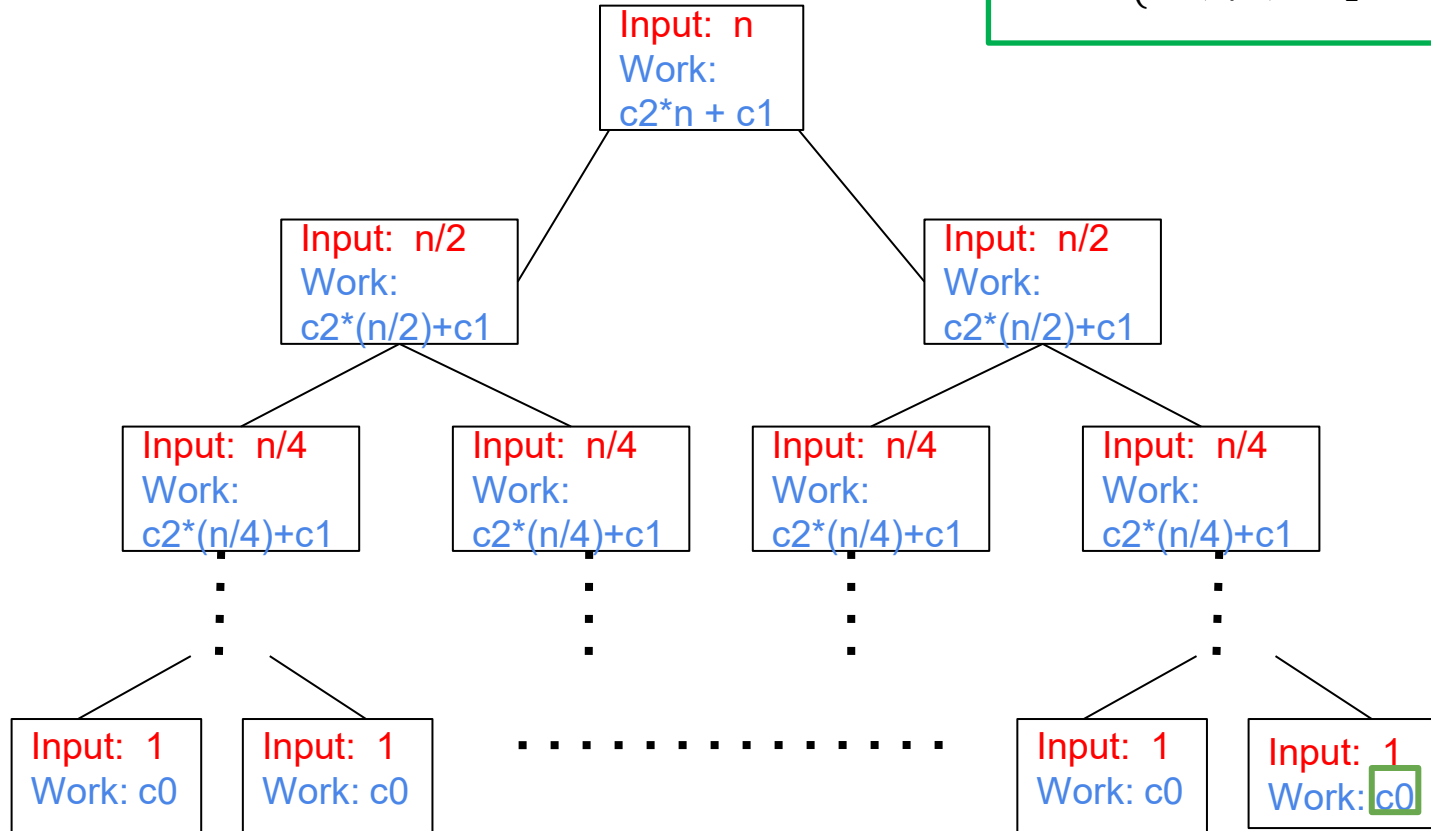
$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$



$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$



$$T(n) = \begin{cases} c_0, & n = 1 \\ 2T(n/2) + c_2n + c_1, & \text{otherwise} \end{cases}$$

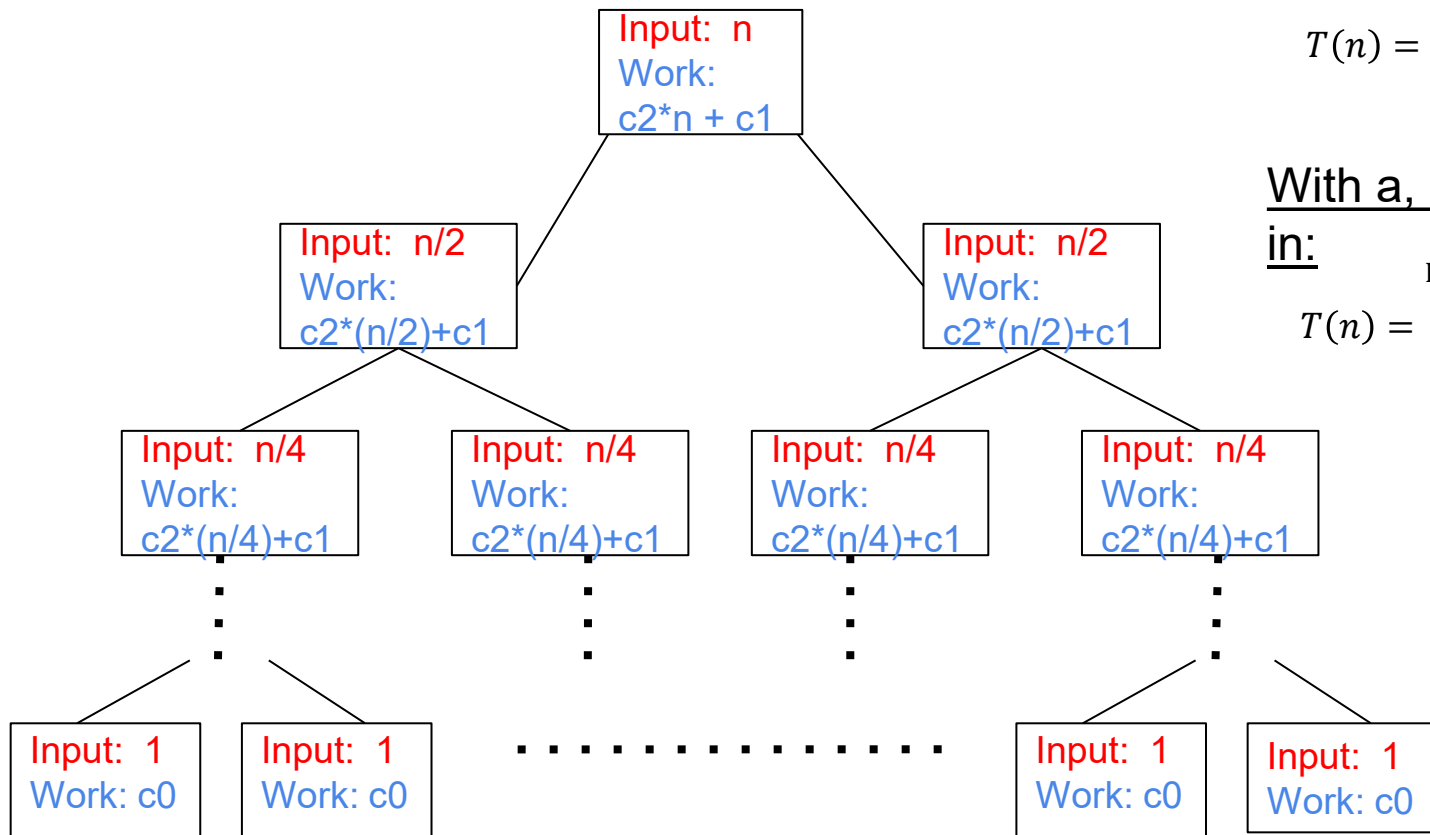


Since we're in /b case:

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

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

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



Thank You!

