# Data Structures and Algorithms

## *CS245-2017S-03*

## *Recursive Function Analysis*

David Galles

Department of Computer Science
University of San Francisco

# 03-0: Algorithm Analysis

```
for (i=1; i<=n*n; i++)
  for (j=0; j<i; j++)
    sum++;
```

# 03-1: Algorithm Analysis

```
for (i=1; i<=n*n; i++)      Executed n*n times
  for (j=0; j<i; j++)        Executed <= n*n times
    sum++;                    O(1)
```

Running Time: $O(n^4)$

# 03-2: Algorithm Analysis

```
for (i=1; i<=n*n; i++)
  for (j=0; j<i; j++)
    sum++;
```

Exact # of times `sum++` is executed:

$$
\begin{aligned}
\sum_{i=1}^{n^2} i &= \frac{n^2(n^2+1)}{2} \\
&= \frac{n^4+n^2}{2} \\
&\in \Theta(n^4)
\end{aligned}
$$

# 03-3: Recursive Functions

```
long power(long x, long n) {
  if (n == 0)
    return 1;
  else
    return x * power(x, n-1);
}
```

# 03-4: Recurrence Relations

$T(n)$ = Time required to solve a problem of size $n$

Recurrence relations are used to determine the running time of recursive programs – recurrence relations themselves are recursive

$$T(0) = \text{time to solve problem of size } 0$$

– Base Case

$$T(n) = \text{time to solve problem of size } n$$

– Recursive Case

# 03-5: Recurrence Relations

```
long power(long x, long n) {
  if (n == 0)
    return 1;
  else
    return x * power(x, n-1);
}
```

$T(0) = c_1$ for some constant $c_1$

$T(n) = c_2 + T(n-1)$ for some constant $c_2$

# 03-6: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(n) = T(n-1) + c_2$$

If we knew $T(n-1)$, we could solve $T(n)$.

$$T(n) \quad = T(n-1) + c_2$$

# 03-7: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(n) = T(n-1) + c_2$$

If we knew $T(n-1)$, we could solve $T(n)$.

$$\begin{aligned} T(n) &= T(n-1) + c_2 \\ &= T(n-2) + c_2 + c_2 \\ &= T(n-2) + 2c_2 \end{aligned}$$

$$T(n-1) = T(n-2) + c_2$$

# 03-8: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(n) = T(n-1) + c_2$$

If we knew $T(n-1)$, we could solve $T(n)$.

$$\begin{aligned}
T(n) &= T(n-1) + c_2 \\
&= T(n-2) + c_2 + c_2 \\
&= T(n-2) + 2c_2 \\
&= T(n-3) + c_2 + 2c_2 \\
&= T(n-3) + 3c_2
\end{aligned}$$

$$T(n-1) = T(n-2) + c_2$$

$$T(n-2) = T(n-3) + c_2$$

# 03-9: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(n) = T(n-1) + c_2$$

If we knew $T(n-1)$, we could solve $T(n)$.

$$\begin{aligned} T(n) &= T(n-1) + c_2 \\ &= T(n-2) + c_2 + c_2 \\ &= T(n-2) + 2c_2 \\ &= T(n-3) + c_2 + 2c_2 \\ &= T(n-3) + 3c_2 \\ &= T(n-4) + 4c_2 \end{aligned}$$

$$T(n-1) = T(n-2) + c_2$$

$$T(n-2) = T(n-3) + c_2$$

$$T(n-3) = T(n-4) + c_2$$

# 03-10: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(n) = T(n-1) + c_2$$

If we knew $T(n-1)$, we could solve $T(n)$.

$$\begin{aligned} T(n) &= T(n-1) + c_2 & T(n-1) = T(n-2) + c_2 \\ &= T(n-2) + c_2 + c_2 & \\ &= T(n-2) + 2c_2 & T(n-2) = T(n-3) + c_2 \\ &= T(n-3) + c_2 + 2c_2 & \\ &= T(n-3) + 3c_2 & T(n-3) = T(n-4) + c_2 \\ &= T(n-4) + 4c_2 & \\ &= \ldots & \\ &= T(n-k) + kc_2 & \end{aligned}$$

# 03-11: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(n) = T(n-k) + k * c_2 \quad \text{for all } k$$

If we set $k = n$, we have:

$$\begin{aligned} T(n) &= T(n-n) + nc_2 \\ &= T(0) + nc_2 \\ &= c_1 + nc_2 \\ &\in \Theta(n) \end{aligned}$$

# 03-12: Building a Better Power

```
long power(long x, long n) {
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(x*x, n/2);
  else
    return power(x*x, n/2) * x;
}
```

# 03-13: Building a Better `Power`

```
long power(long x, long n) {
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(x*x, n/2);
  else
    return power(x*x, n/2) * x;
}
```

$T(0) = c_1$

$T(1) = c_2$

$T(n) = T(n/2) + c_3$

(Assume n is a power of 2)

# 03-14: Solving Recurrence Relations

$$T(n) = T(n/2) + c_3$$

# 03-15: Solving Recurrence Relations

$$T(n) = T(n/2) + c_3$$
$$= T(n/4) + c_3 + c_3$$
$$= T(n/4) + 2c_3$$

$$T(n/2) = T(n/4) + c_3$$

# 03-16: Solving Recurrence Relations

$$\begin{aligned} T(n) &= T(n/2) + c_3 \\ &= T(n/4) + c_3 + c_3 \\ &= T(n/4) + 2c_3 \\ &= T(n/8) + c_3 + 2c_3 \\ &= T(n/8) + 3c_3 \end{aligned}$$

$$T(n/2) = T(n/4) + c_3$$

$$T(n/4) = T(n/8) + c_3$$

# 03-17: Solving Recurrence Relations

$$\begin{aligned} T(n) &= T(n/2) + c_3 \\ &= T(n/4) + c_3 + c_3 \\ &= T(n/4) + 2c_3 \\ &= T(n/8) + c_3 + 2c_3 \\ &= T(n/8) + 3c_3 \\ &= T(n/16) + c_3 + 3c_3 \\ &= T(n/16) + 4c_3 \end{aligned}$$

$$T(n/2) = T(n/4) + c_3$$

$$T(n/4) = T(n/8) + c_3$$

$$T(n/8) = T(n/16) + c_3$$

# 03-18: Solving Recurrence Relations

$$
\begin{aligned}
T(n) &= T(n/2) + c_3 \\
&= T(n/4) + c_3 + c_3 \\
&= T(n/4) + 2c_3 \\
&= T(n/8) + c_3 + 2c_3 \\
&= T(n/8) + 3c_3 \\
&= T(n/16) + c_3 + 3c_3 \\
&= T(n/16) + 4c_3 \\
&= T(n/32) + c_3 + 4c_3 \\
&= T(n/32) + 5c_3
\end{aligned}
$$

$$T(n/2) = T(n/4) + c_3$$

$$T(n/4) = T(n/8) + c_3$$

$$T(n/8) = T(n/16) + c_3$$

$$T(n/16) = T(n/32) + c_3$$

# 03-19: Solving Recurrence Relations

$$
\begin{aligned}
T(n) &= T(n/2) + c_3 \\
&= T(n/4) + c_3 + c_3 \\
&= T(n/4) + 2c_3 \\
&= T(n/8) + c_3 + 2c_3 \\
&= T(n/8) + 3c_3 \\
&= T(n/16) + c_3 + 3c_3 \\
&= T(n/16) + 4c_3 \\
&= T(n/32) + c_3 + 4c_3 \\
&= T(n/32) + 5c_3 \\
&= \ldots \\
&= T(n/2^k) + kc_3
\end{aligned}
$$

$$T(n/2) = T(n/4) + c_3$$

$$T(n/4) = T(n/8) + c_3$$

$$T(n/8) = T(n/16) + c_3$$

$$T(n/16) = T(n/32) + c_3$$

# 03-20: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(1) = c_2$$
$$T(n) = T(n/2) + c_3$$

$$T(n) = T(n/2^k) + kc_3$$

We want to get rid of $T(n/2^k)$. Since we know $T(1)$ ...

$$\begin{aligned} n/2^k &= 1 \\ n &= 2^k \\ \lg n &= k \end{aligned}$$

# 03-21: Solving Recurrence Relations

$T(1) = c_2$
$T(n) = T(n/2^k) + kc_3$

Set $k = \lg n$:

$$
\begin{aligned}
T(n) &= T(n/2^{\lg n}) + (\lg n)c_3 \\
&= T(n/n) + c_3 \lg n \\
&= T(1) + c_3 \lg n \\
&= c_2 + c_3 \lg n \\
&\in \Theta(\lg n)
\end{aligned}
$$

# 03-22: `Power` Modifications

```
long power(long x, long n) {
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(x*x, n/2);
  else
    return power(x*x, n/2) * x;
}
```

# 03-23: `Power` Modifications

```
long power(long x, long n) {
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(power(x,2), n/2);
  else
    return power(power(x,2), n/2) * x;
}
```

This version of power will not work. Why?

# 03-24: Power Modifications

```
long power(long x, long n) {
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(power(x,n/2), 2);
  else
    return power(power(x,n/2), 2) * x;
}
```

This version of power also will not work. Why?

# 03-25: `Power` Modifications

```
long power(long x, long n) {
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(x,n/2) * power(x,n/2);
  else
    return power(x,n/2) * power(x,n/2) * x;
}
```

This version of power does work.

What is the recurrence relation that describes its running time?

# 03-26: Power Modifications

```
long power(long x, long n) {
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(x,n/2) * power(x,n/2);
  else
    return power(x,n/2) * power(x,n/2) * x;
}
```

$$
\begin{aligned}
T(0) &= c_1 \\
T(1) &= c_2 \\
T(n) &= T(n/2) + T(n/2) + c_3 \\
&= 2T(n/2) + c_3
\end{aligned}
$$

(Again, assume n is a power of 2)

# 03-27: Solving Recurrence Relations

$$
\begin{aligned}
T(n) &= 2T(n/2) + c_3 \\
&= 2[2T(n/4) + c_3]c_3 \\
&= 4T(n/4) + 3c_3 \\
&= 4[2T(n/8) + c_3] + 3c_3 \\
&= 8T(n/8) + 7c_3 \\
&= 8[2T(n/16) + c_3] + 7c_3 \\
&= 16T(n/16) + 15c_3 \\
&= 32T(n/32) + 31c_3 \\
&\ldots \\
&= 2^kT(n/2^k) + (2^k - 1)c_3
\end{aligned}
$$

$$T(n/2) = 2T(n/4) + c_3$$

$$T(n/4) = 2T(n/8) + c_3$$

# 03-28: Solving Recurrence Relations

$$T(0) = c_1$$
$$T(1) = c_2$$
$$T(n) = 2^k T(n/2^k) + (2^k - 1)c_3$$

Pick a value for $k$ such that $n/2^k = 1$:

$$\begin{aligned}
n/2^k &= 1 \\
n &= 2^k \\
\lg n &= k \\
T(n) &= 2^{\lg n} T(n/2^{\lg n}) + (2^{\lg n} - 1)c_3 \\
&= nT(n/n) + (n-1)c_3 \\
&= nT(1) + (n-1)c_3 \\
&= nc_2 + (n-1)c_3 \\
&\in \Theta(n)
\end{aligned}$$

# 03-29: Recursion Trees

- We can also do this substitution visually, leads to Recursion Trees
- Consider:

$$\begin{aligned} T(n) &= 2T(n/2) + Cn \\ T(1) &= C_2 \\ T(0) &= C_2 \end{aligned}$$

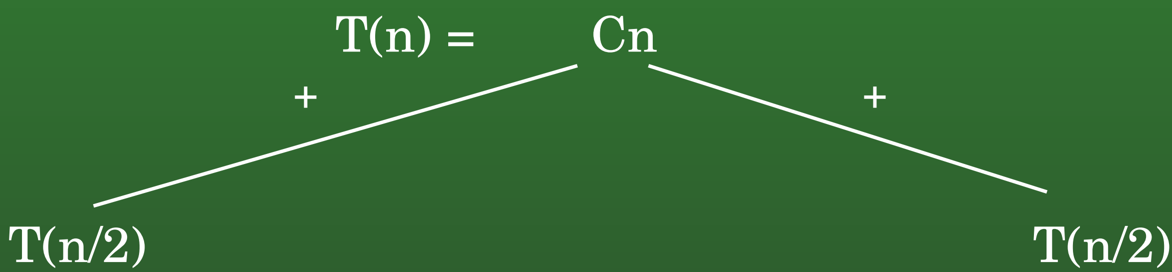
# 03-30: Recursion Trees

- Start with the recursive definition

$$T(n) = Cn + 2T(n/2)$$

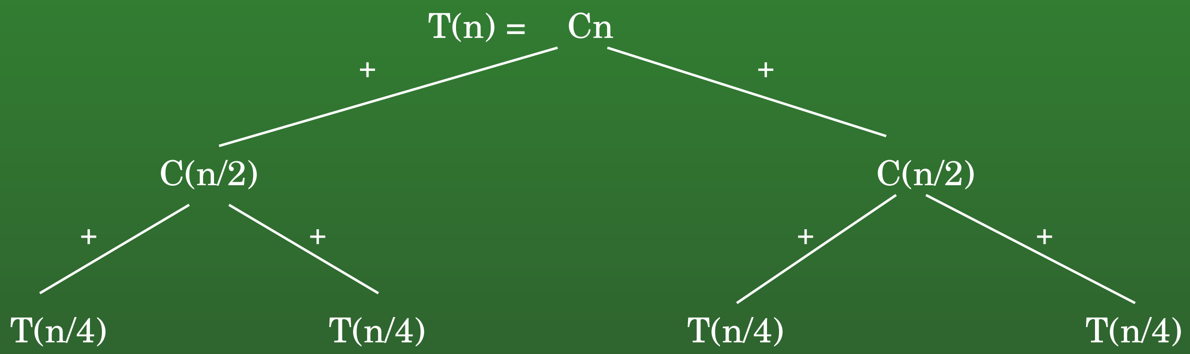
# 03-31: Recursion Trees

- Move the equation around a bit to get:



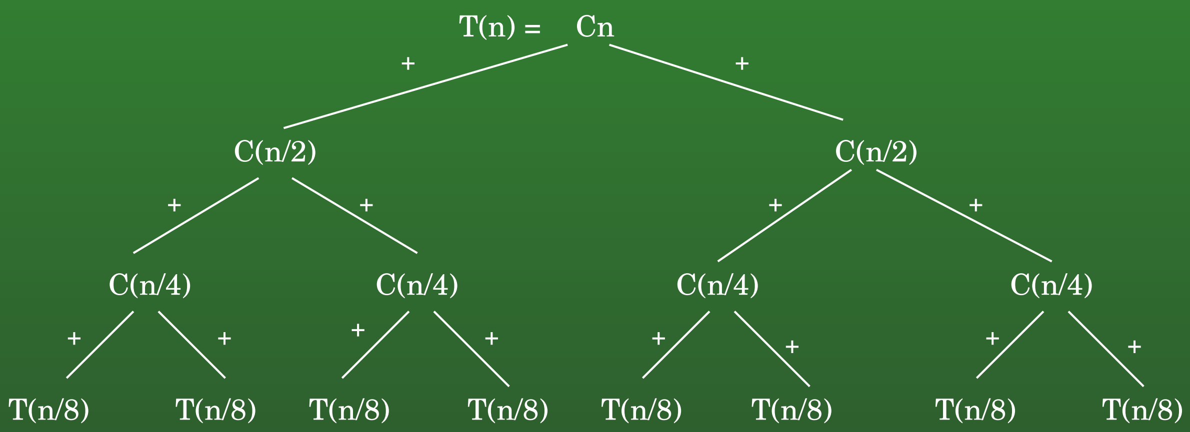

- Repalce each occurance of $T(n/2)$ with $T(n/4) + T(n/4) + C(n/2)$

# 03-32: Recursion Trees



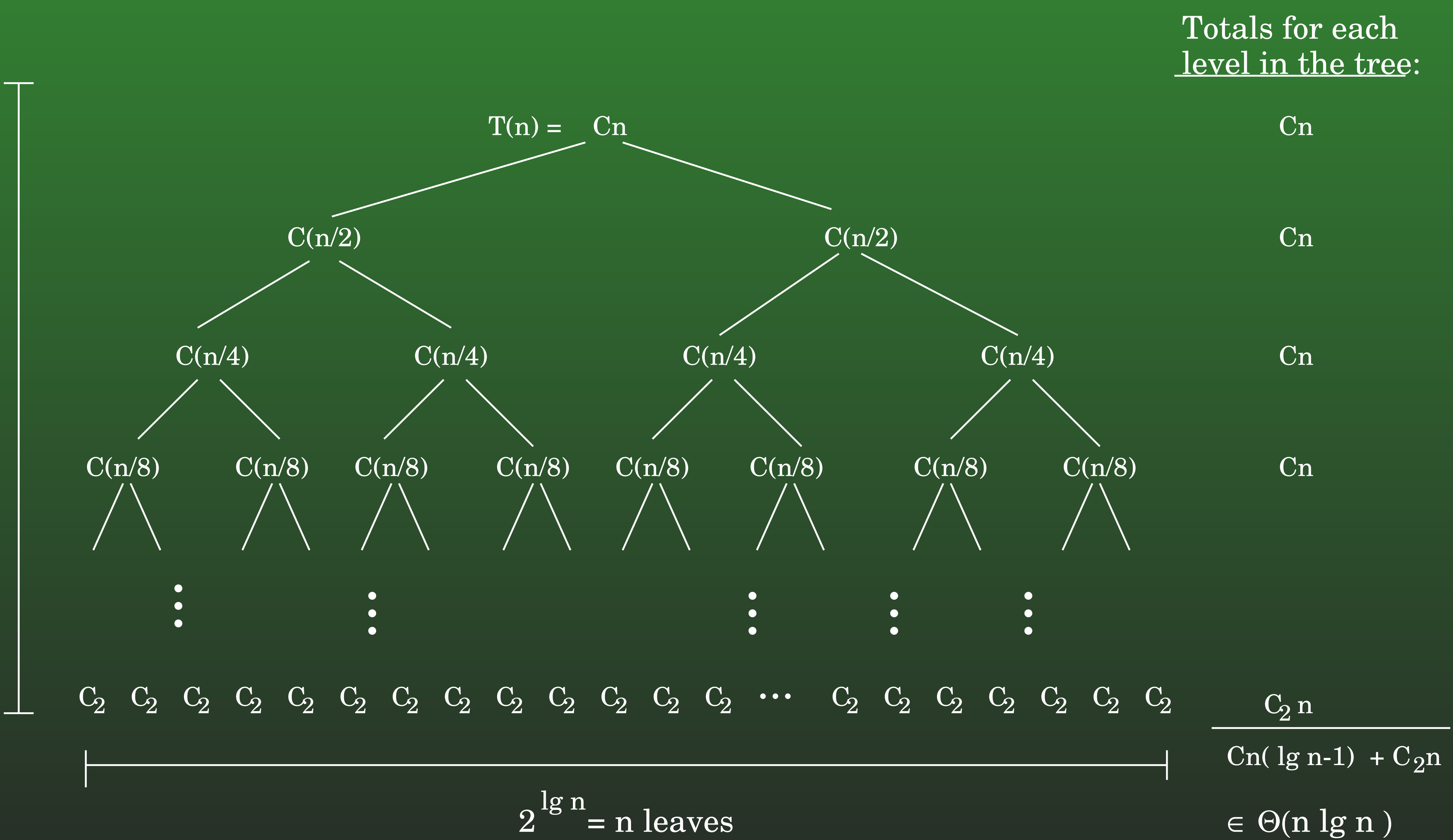

- Replace again, using $T(n) = 2T(n/2) + Cn$

# 03-33: Recursion Trees




- If we continue replacing ...

# 03-34: Recursion Trees



# 03-35: Recursion Trees

$$\begin{aligned} T(1) &= C_1 \\ T(n) &= T(n-1) + C_2 \end{aligned}$$

# 03-36: Recursion Trees

$$
\begin{aligned}
T(0) &= C_1 \\
T(1) &= C_1 \\
T(n) &= T(n/2) + C_2
\end{aligned}
$$

# Substitution Method

- We can prove that a bound is correct using induction, this is the substituion method

$$\begin{aligned} T(1) &= C_1 \\ T(n) &= T(n-1) + C_2 \end{aligned}$$

Show: $T(n) \in O($ ? $)$

# 03-38: Substitution Method

- We can prove that a bound is correct using induction, this is the substituion method

$$\begin{aligned} T(1) &= C_1 \\ T(n) &= T(n-1) + C_2 \end{aligned}$$

Show: $T(n) \in O(n)$, that is:
$T(n) \leq C * n$ for all $n > n_0$,
for some pair of constants $C, n_0$

# 03-39: Substitution Method

$$T(1) = C_1$$
$$T(n) = T(n-1) + C_2$$

Show: $T(n) \in O(n)$, that is, $T(n) \leq C * n$

- Base case: $T(1) = C_1 \leq C * 1$ for some constant $C$

  This is true as long as $C \geq C_1$.

# 03-40: Substitution Method

$$T(1) = C_1$$
$$T(n) = T(n-1) + C_2$$

Show: $T(n) \in O(n)$, that is, $T(n) \leq C * n$

- Recursive case:

$$T(n) = T(n-1) + C_2 \quad \text{Recurrence definition}$$

# 03-41: Substitution Method

$$
\begin{aligned}
T(1) &= C_1 \\
T(n) &= T(n-1) + C_2
\end{aligned}
$$

Show: $T(n) \in O(n)$, that is, $T(n) \leq C * n$

- Recursive case:

$$
\begin{aligned}
T(n) &= T(n-1) + C_2 && \text{Recurrence definition} \\
&\leq C(n-1) + C_2 && \text{Inductive hypothesis}
\end{aligned}
$$

# 03-42: Substitution Method

$$
\begin{aligned}
T(1) &= C_1 \\
T(n) &= T(n-1) + C_2
\end{aligned}
$$

Show: $T(n) \in O(n)$, that is, $T(n) \leq C * n$

- Recursive case:

$$
\begin{aligned}
T(n) &= T(n-1) + C_2 && \text{Recurrence definition} \\
&\leq C(n-1) + C_2 && \text{Inductive hypothesis} \\
&\leq Cn + (C_2 - C) && \text{Algebra} \\
&\leq Cn && \text{If } C > C_2
\end{aligned}
$$

This is true as long as $C \geq C_1$.

# 03-43: Substitution Method

- We can prove that a bound is correct using induction, this is the substituion method

$$
\begin{aligned}
T(1) &= C_1 \\
T(n) &= T(n-1) + C_2
\end{aligned}
$$

Show: $T(n) \in \Omega(n)$

$T(n) \geq C * n$ for all $n > n_0$,

for some pair of constants $C, n_0$

# Substitution Method

$$\begin{aligned} T(1) &= C_1 \\ T(n) &= T(n-1) + C_2 \end{aligned}$$

Show: $T(n) \in \Omega(n)$, that is, $T(n) \geq C * n$

- Base case: $T(1) = C_1 \geq C * 1$ for some constant $C$

  This is true as long as $C \leq C_1$.

# 03-45: Substitution Method

$$T(1) = C_1$$
$$T(n) = T(n-1) + C_2$$

Show: $T(n) \in \Omega(n)$, that is, $T(n) \geq C * n$

- Recursive case:

$$T(n) = T(n-1) + C_2$$ Recurrence definition

# Substitution Method

$$
\begin{aligned}
T(1) &= C_1 \\
T(n) &= T(n-1) + C_2
\end{aligned}
$$

Show: $T(n) \in \Omega(n)$, that is, $T(n) \geq C * n$

- Recursive case:

$$
\begin{aligned}
T(n) &= T(n-1) + C_2 && \text{Recurrence definition} \\
&\geq C(n-1) + C_2 && \text{Inductive hypothesis}
\end{aligned}
$$

# 03-47: Substitution Method

$$T(1) = C_1$$
$$T(n) = T(n-1) + C_2$$

Show: $T(n) \in \Omega(n)$, that is, $T(n) \geq C * n$

- Recursive case:

$$\begin{aligned} T(n) &= T(n-1) + C_2 && \text{Recurrence definition} \\ &\geq C(n-1) + C_2 && \text{Inductive hypothesis} \\ &\geq Cn + (C_2 - C) && \text{Algebra} \\ &\geq Cn && \text{If } C \leq C_2 \end{aligned}$$

This is true as long as $C \leq C_1$.

# 03-48: Substitution Method

$$
\begin{aligned}
T(0) &= C_2 \\
T(1) &= C_2 \\
T(n) &= 2T(n/2) + C_1 n
\end{aligned}
$$

Show: $T(n) \in O(n \lg n)$, that is, $T(n) \leq C * n \lg n$

# Substitution Method

$$T(0) = C_2$$
$$T(1) = C_2$$
$$T(n) = 2T(n/2) + C_1 n$$

Show: $T(n) \in O(n \lg n)$, that is, $T(n) \leq C * n \lg n$

- Base cases:
  - $T(0) = C_1 \leq C * 0 \lg 0$ for some constant $C$
  - $T(1) = C_1 \leq C * 1 \lg 1$ for some constant $C$

Hmmm....

# 03-50: Substitution Method

$$T(0) = C_2$$
$$T(1) = C_2$$
$$T(n) = 2T(n/2) + C_1 n$$

Show: $T(n) \in O(n \lg n)$, that is, $T(n) \leq C * n \lg n$

- Only care about $n > n_0$. We can pick 2, 3 as base cases (why?)
  - $T(2) = C_1 \leq C * 2 \lg 2$ for some constant $C$
  - $T(3) = C_1 \leq C * 3 \lg 3$ for some constant $C$

# 03-51: Substitution Method

$$T(0) = C_2$$

$$T(1) = C_2$$

$$T(n) = 2T(n/2) + C_1 n$$

$T(n) = 2T(n/2) + C_1 n$ Recurrence Definition

# 03-52: Substitution Method

$$
\begin{aligned}
T(0) &= C_2 \\
T(1) &= C_2 \\
T(n) &= 2T(n/2) + C_1 n
\end{aligned}
$$

$$
\begin{aligned}
T(n) &= 2T(n/2) + C_1 n && \text{Recurrence Definition} \\
&\leq 2C(n/2)\lg(n/2) + C_1 n && \text{Inductive hypothesis}
\end{aligned}
$$

# 03-53: Substitution Method

$$T(0) = C_2$$
$$T(1) = C_2$$
$$T(n) = 2T(n/2) + C_1 n$$

| | | | |
|---|---|---|---|
| $T(n)$ | $=$ | $2T(n/2) + C_1 n$ | Recurrence Definition |
| | $\leq$ | $2C(n/2)\lg(n/2) + C_1 n$ | Inductive hypothesis |
| | $\leq$ | $Cn \lg n/2 + C_1 n$ | Algebra |
| | $\leq$ | $Cn \lg n - Cn \lg 2 + C_1 n$ | Algebra |
| | $\leq$ | $Cn \lg n - Cn + C_1 n$ | Algebra |

# 03-54: Substitution Method

$T(0)$ = $C_2$

$T(1)$ = $C_2$

$T(n)$ = $2T(n/2) + C_1 n$

| | | | |
|---|---|---|---|
| $T(n)$ | $=$ | $2T(n/2) + C_1 n$ | Recurrence Definition |
| | $\leq$ | $2C(n/2)\lg(n/2) + C_1 n$ | Inductive hypothesis |
| | $\leq$ | $Cn \lg n/2 + C_1 n$ | Algebra |
| | $\leq$ | $Cn \lg n - Cn \lg 2 + C_1 n$ | Algebra |
| | $\leq$ | $Cn \lg n - Cn + C_1 n$ | Algebra |
| | $\leq$ | $Cn \lg n$ | If $C > C_1$ |

# 03-55: Substitution Method

- Sometimes, the math doesn't work out in the substitution method:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T\left(\frac{n}{2}\right) + 1 \end{aligned}$$

(Work on board)

 
# Substitution Method

Try $T(n) \leq cn$:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) 1 \\ &\leq 2c\left(\frac{n}{2}\right) + 1 \\ &\leq cn + 1 \end{aligned}$$

We did not get back $T(n) \leq cn$ – that extra +1 term means the proof is not valid. We need to get back *exactly* what we started with (see invalid proof of $\sum_{i=1}^{n} i \in O(n)$ for why this is true)

# 03-57: Substitution Method

Try $T(n) \leq cn - b$:

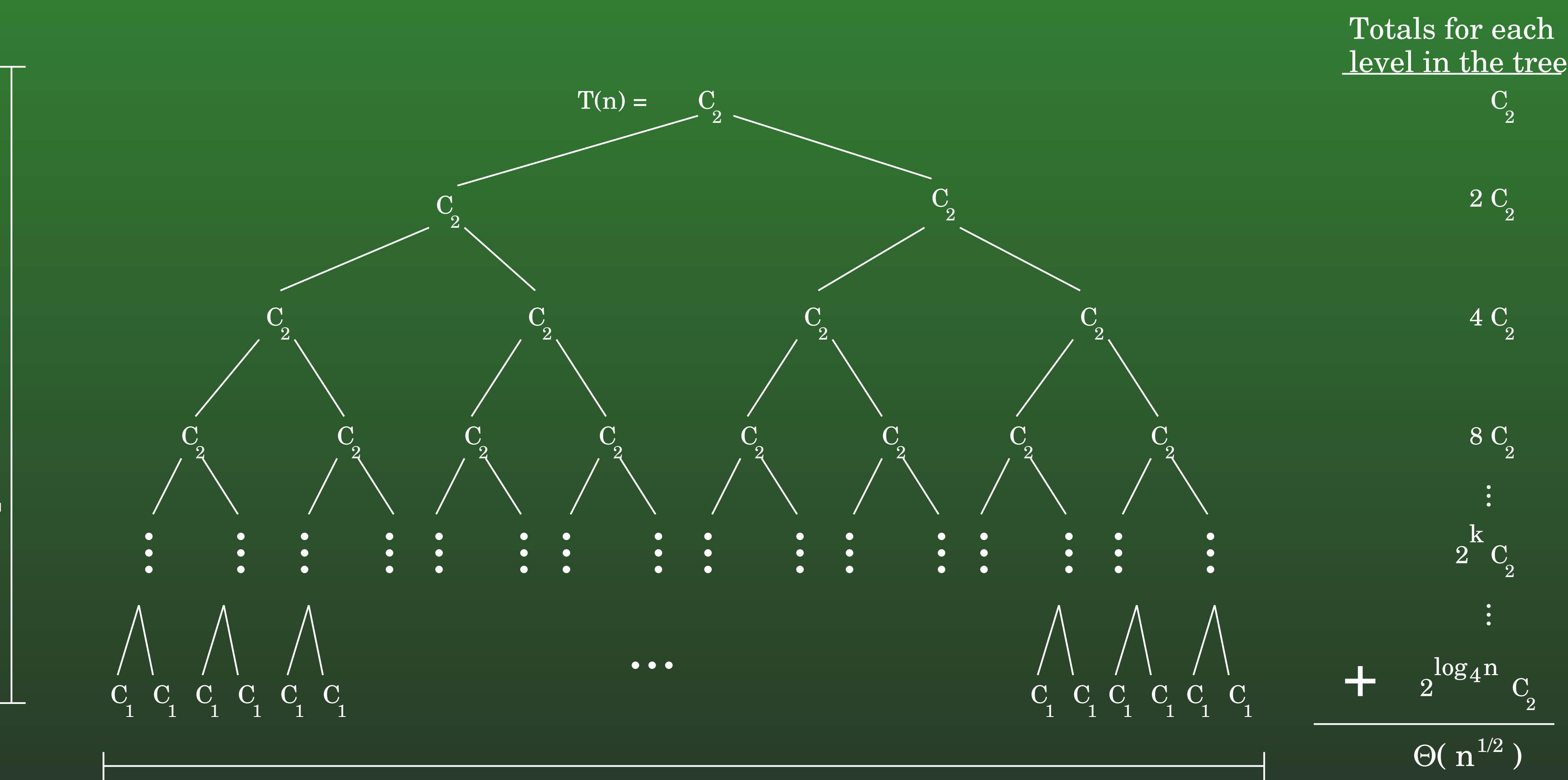
$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + 1 \\ &\leq 2\left(c\left(\frac{n}{2}\right) - b\right) + 1 \\ &\leq cn - 2b + 1 \\ &\leq cn - b \end{aligned}$$

As long as $b \geq 1$

# 03-58: Master Method

Recursion Tree for: $T(n) = 2T(n/4) + C_2$

# 03-59: Master Method

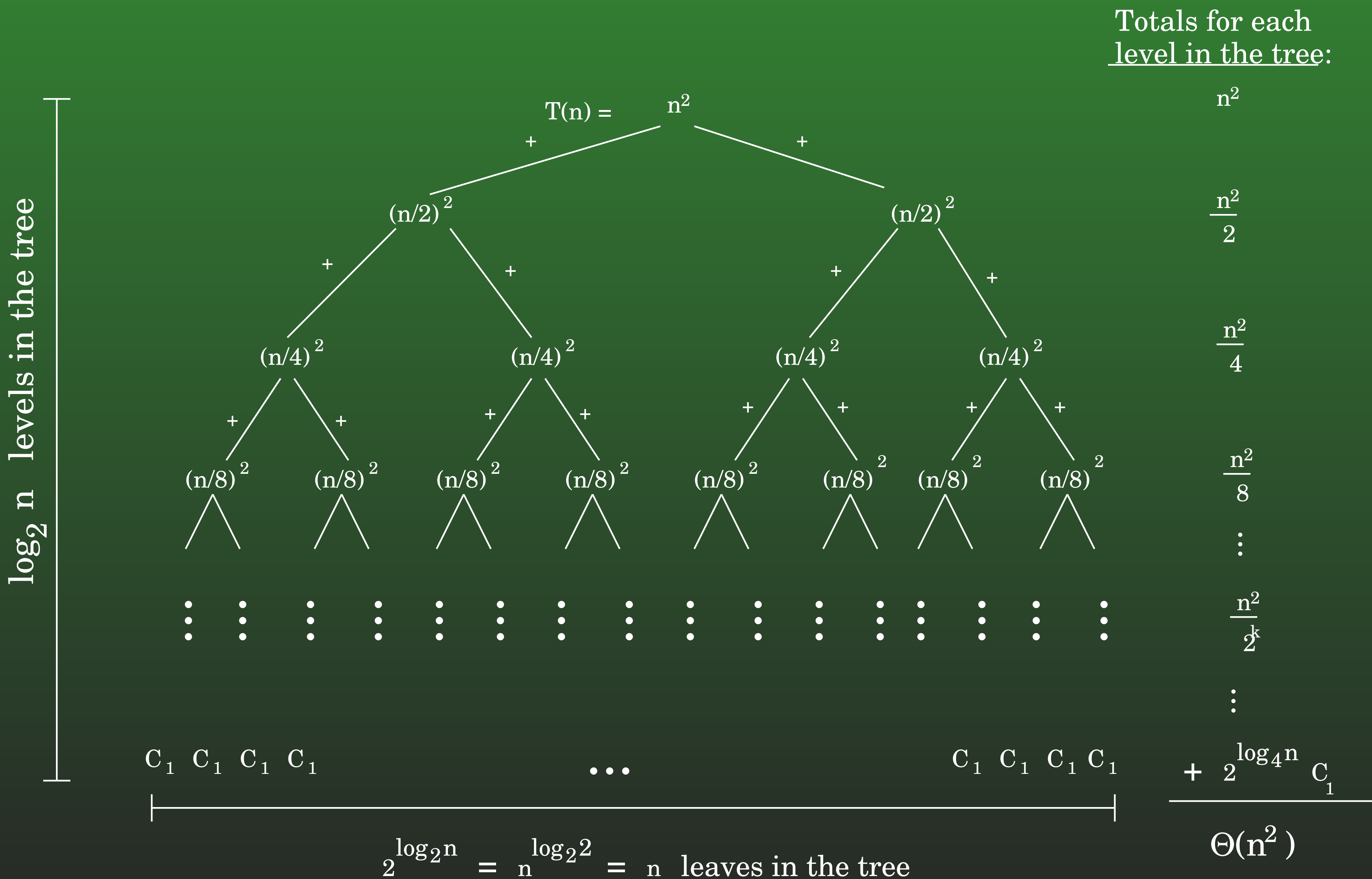



$$2^{\log_4 n} = n^{\log_4 2} = n^{1/2}$$ leaves in the tree

# Master Method

Recursion Tree for: $T(n) = 2T(n/2) + n^2$

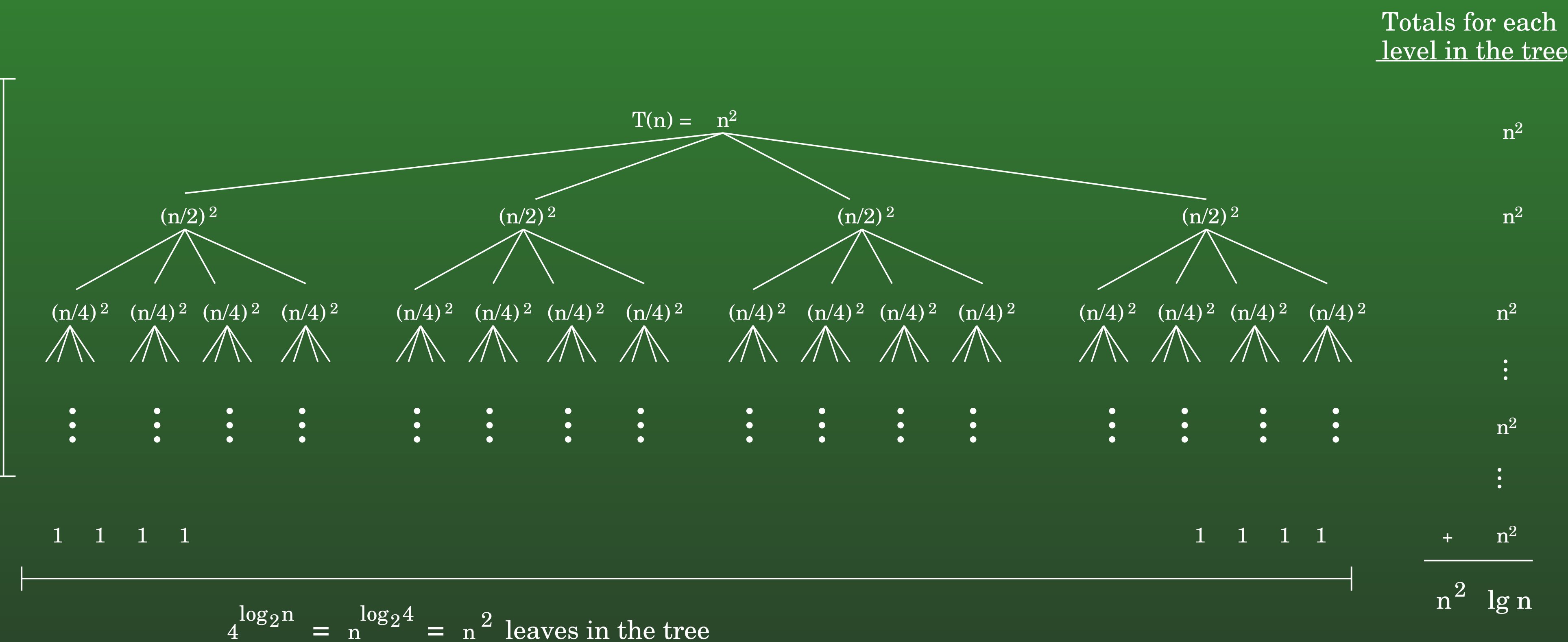
# Master Method



# 03-62: Master Method

Recursion Tree for: $T(n) = 4T(n/2) + n^2$

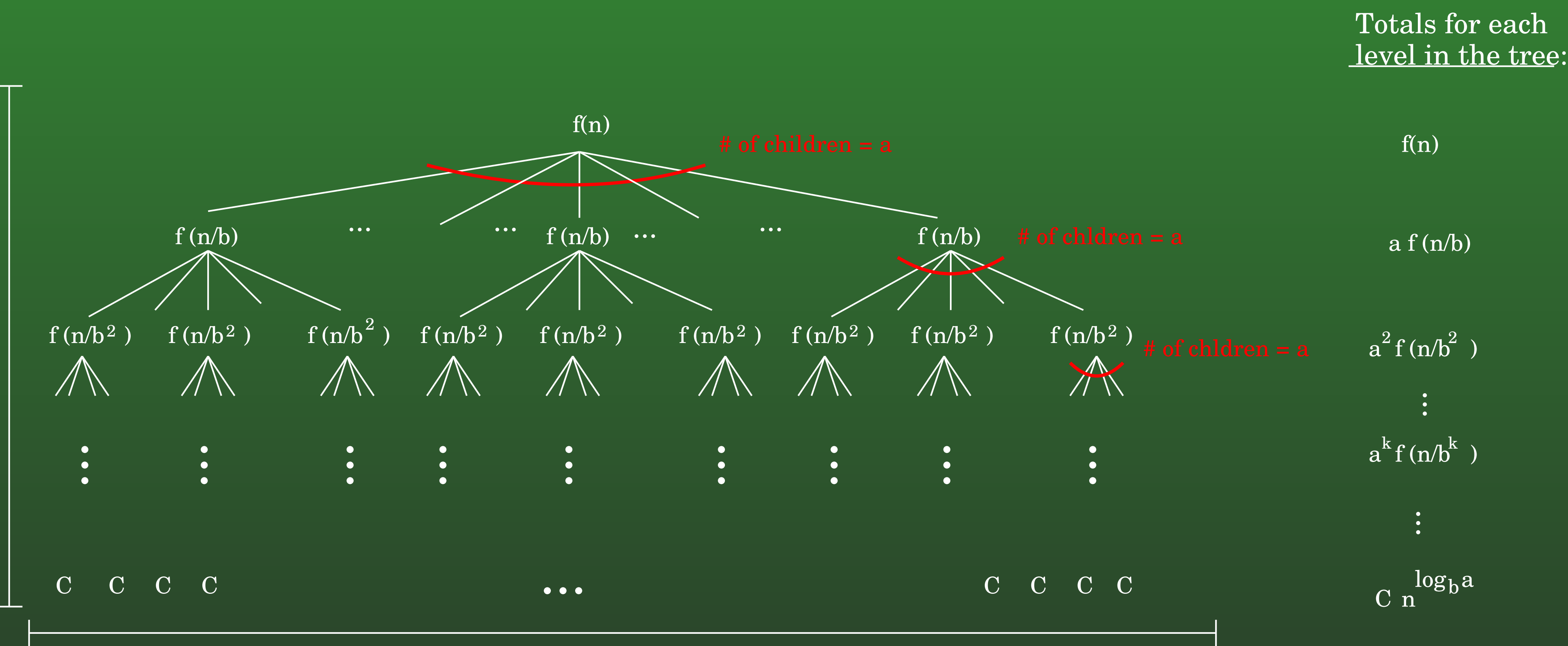
# 03-63: Master Method



# Master Method

Recursion Tree for: $T(n) = aT(n/b) + f(n)$

# Master Method




$a^{\log_b n} = n^{\log_b a}$ leaves in the tree

# Master Method

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

1. if $f(n) \in O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$
2. if $f(n) \in \Theta(n^{\log_b a})$ then $T(n) \in \Theta(n^{\log_b a} * \lg n)$
3. if $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and large $n$, then $T(n) \in \Theta(f(n))$

# 03-67: Master Method

$$T(n) = 9T(n/3) + n$$

# 03-68: Master Method

$$T(n) = 9T(n/3) + n$$

- $a = 9, b = 3, f(n) = n$
- $n^{\log_b a} = n^{\log_3 9} = n^2$
- $n \in O(n^{2-\epsilon})$

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

## 03-69: Master Method

$$T(n) = T(2n/3) + 1$$

# Master Method

$$T(n) = T(2n/3) + 1$$

- $a = 1, b = 3/2, f(n) = 1$
- $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
- $1 \in O(1)$

$$T(n) = \Theta(1 * \lg n) = \Theta(\lg n)$$

# 03-71: Master Method

$$T(n) = 3T(n/4) + n \lg n$$

# 03-72: Master Method

$$T(n) = 3T(n/4) + n \lg n$$

- $a = 3, b = 4, f(n) = n \lg n$
- $n^{\log_b a} = n^{\log_4 3} = n^{0.792}$
- $n \lg n \in \Omega(n^{0.792+\epsilon})$
- $3(n/4) \lg(n/4) \leq c * n \lg n$

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

# Master Method

$$T(n) = 2T(n/2) + n \lg n$$

# 03-74: Master Method

$$T(n) = 2T(n/2) + n \lg n$$

- $a = 2, b = 2, f(n) = n \lg n$
- $n^{\log_b a} = n^{\log_2 2} = n^1$

Master method does not apply!

$n^{1+\epsilon}$ grows faster than $n \lg n$ for *any* $\epsilon > 0$

Logs grow *incredibly* slowly! $\lg n \in o(n^\epsilon)$ for any $\epsilon > 0$