Analyzing Merge Sort + Recurrence

Instructor: Krishna Venkatasubramanian

CSC 212

Announcements

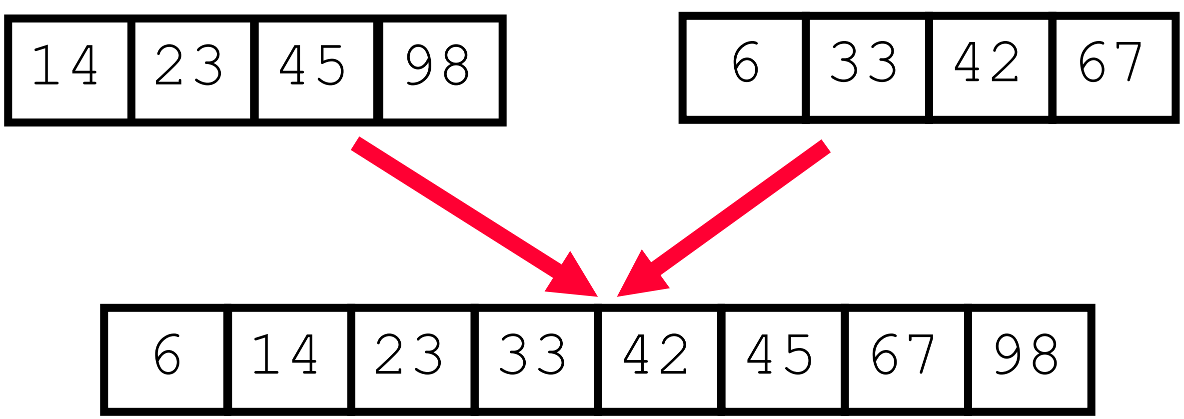
- Assignment 1 OUT
  - Due in three weeks (Oct 24, 11:59pm)
  - Find the link on the Schedule on the course webpage
- Quiz 2 on Tuesday
  - Will cover all the materials covered until today

Recap: Merge Sort

```
MergeSort(A)
  if A's size > 1
   Divide array A in halves
   Call MergeSort on first half.
   Call MergeSort on second half.
   Merge two results (combine).
```

Recap: How to Merge?

- **Merge the sub-problem solutions together:**
  - **Compare the sub-array's first elements**
  - **Remove the smallest element and put it into the result array**
  - **Continue the process until all elements have been put into the result array**



Recurrence Relations

- When computing the complexity of Divide and Conquer algorithms, we have to consider:
  - The **sub-problems created**
  - The **size of the sub-programs**
  - **Effort needed to create** the sub-problems
  - **Effort needed to combine** the sub-problems

- Formally
  - $T(n) = aT(n/b) + D(n) + C(n)$

Note that there a T(..) in the equation of T(n)?

- Where
  - a is the number of sub-problems that n is divided into
  - b is the size of the sub-problem
  - D(n) is the complexity it takes to divide the problem
  - C(n) is the complexity of combining the solutions

Merge Sort Recurrence (1)

- Recurrence relation for MergeSort(A)
  - *T(n) = ???*

```
MergeSort(A)
  if A's size > 1
      Divide array A in halves
      Call MergeSort on first half.
      Call MergeSort on second half.
      Merge two results (combine).
```

- Every time MergeSort(A) is called:
  - How many sub-problems are created?
    - a = ?
  - What is the size of each sub-problem?
    - b = ?
  - What is the complexity of creating the sub-problems?
    - D(n) = O(?)
  - What is the complexity of merging them?
    - C(n) = O(?)

Merge Sort Recurrence (1)

- Recurrence relation for MergeSort(A)
  - $T(n) = 2T(n/2) + O(1) + O(n)$

- OR
  - $T(n) = 2T(n/2) + c*n$

- How do we solve T(n) and find out the overall complexity?
  - **Unrolling the recursion** (iterative method)
  - **Substitution method** (guess the answer and prove by induction)
  - **Master method** (memorize a few rules and apply them)

Unrolling the Recursion

- *T(n) = 2 T(n/2) + cn*

  = 2 (2 T (n/4) + n/2) + cn = ***4 T(n/4) + 2cn***

  = 4(2T(n/8) + n/4) + 2cn = ***8 T(n/8) + 3cn***

  …….

  = $\boldsymbol{2^k\, T(n/2^k) + k^*cn}$

- If $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$. Then:

  $= 2^{lgn}\, T(1) + c^*nlgn$

  $= n + c^*n\ lg\ n$

Why do we do this?

OR

- $T(n) = O(n \lg n)$

Example: Unrolling Recursion

- $T(n) = T(n-1) + c$

$$= T(n\text{-}2) + c + c$$
$$= T(n\text{-}3) + c + c + c$$
$$\ldots$$
$$= \boldsymbol{T(n\text{-}k) + k^*c}$$

If $n\text{-}k = 1$, *then* $k = n\text{-}1$. *Therefore*

$$= \quad T(n\text{-}(n\text{-}1)) + (n\text{-}1)^* c = T(1) + nc - c$$
$$= \; nc - c$$

$$T(n) = O(n)$$

One more:

- $T(n) = 3T(n/2) + n^2$

$= 3(3T(n/4) + (n/2)^2) + n^2 =$ **$3^2T(n/4) + 3/4\ n^2 + n^2$**

$= 3^2(3T(n/8)+(n/4)^2) + ¾\ n^2 + n^2 =$ **$3^3T(n/8) + 9/16\ n^2 + 3/4\ n^2 + n^2$**

...

$= 3^k\ T(n/2^k) + n^2\ (1+3/4+9/16+27/64+....)$

If $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$. Then:

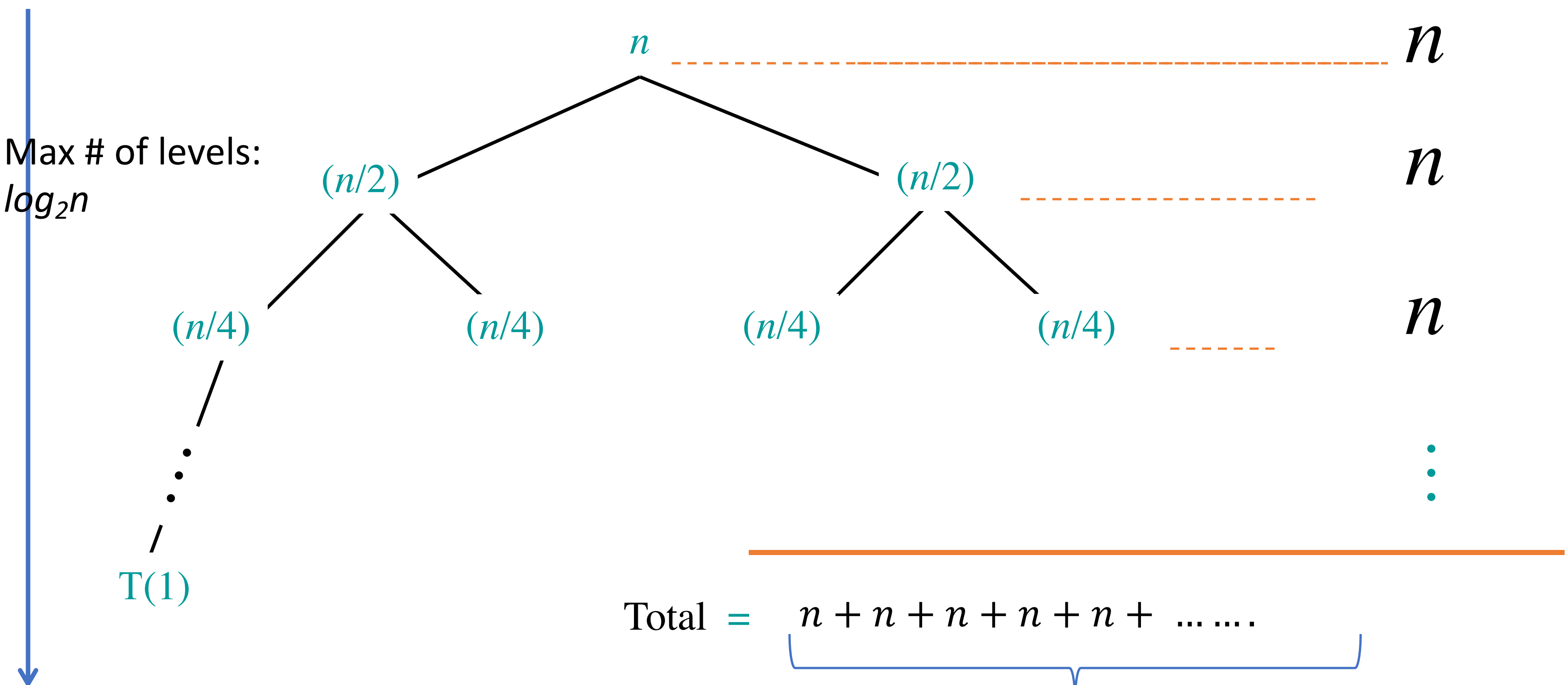
$3^{\lg n}\ T(1) + n^2\ \underbrace{(1+(3/4)+(3/4)^2 + ...(3/4)^{k-1})}_{\text{Infinitely decreasing geometric series}}$

$\sim 3^{\lg n} + n^2$

$= O(n^2)$

Unspooling Visually

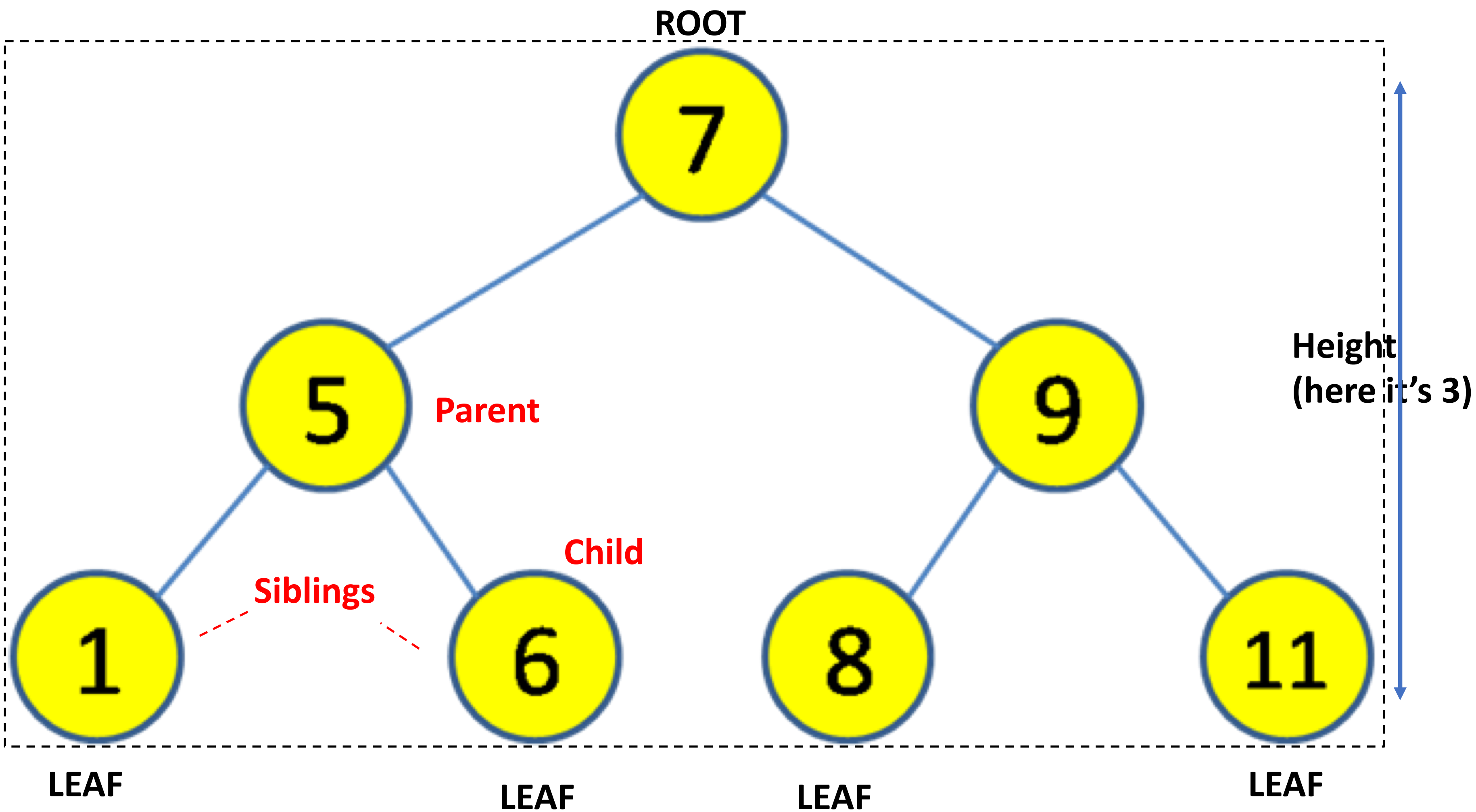
Solve $T(n) = 2T(n/2) + n$:



$$\text{Total} = \underbrace{n + n + n + n + n + \ \dots \dots}_{\log_2 n}$$

$= O(n \log_2 n)$

Quick Note: A TREE Structure



(All leaf nodes need not be at the same level)

https://computersciencesource.wordpress.com/2010/01/08/year-2-algorithms-trees/

Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

n^2

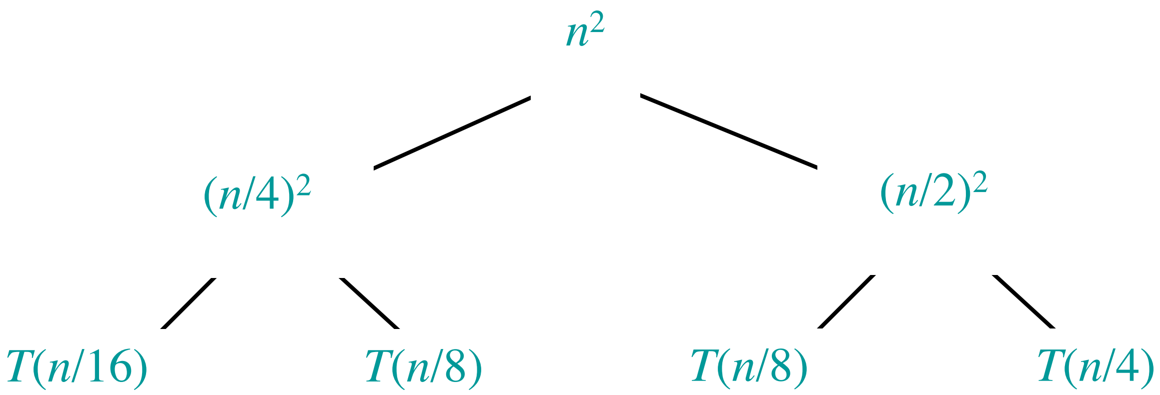
Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$
\begin{array}{ccc}
 & n^2 & \\
\swarrow & & \searrow \\
T(n/4) & & T(n/2)
\end{array}
$$

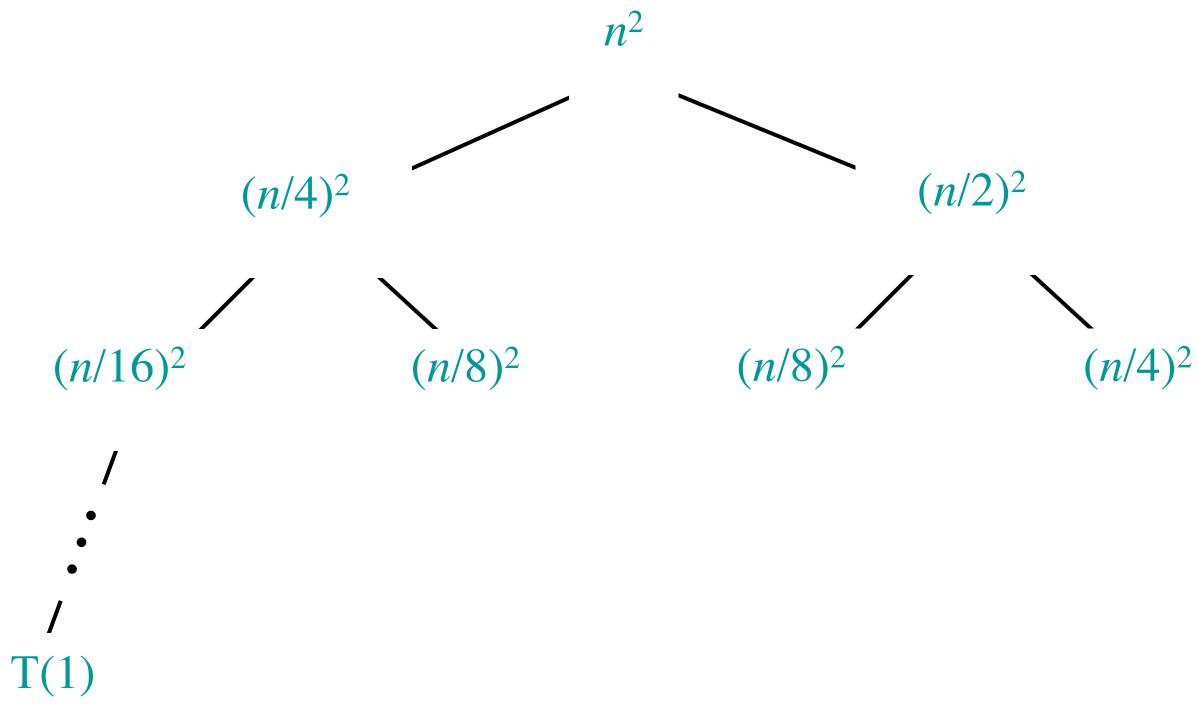
Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



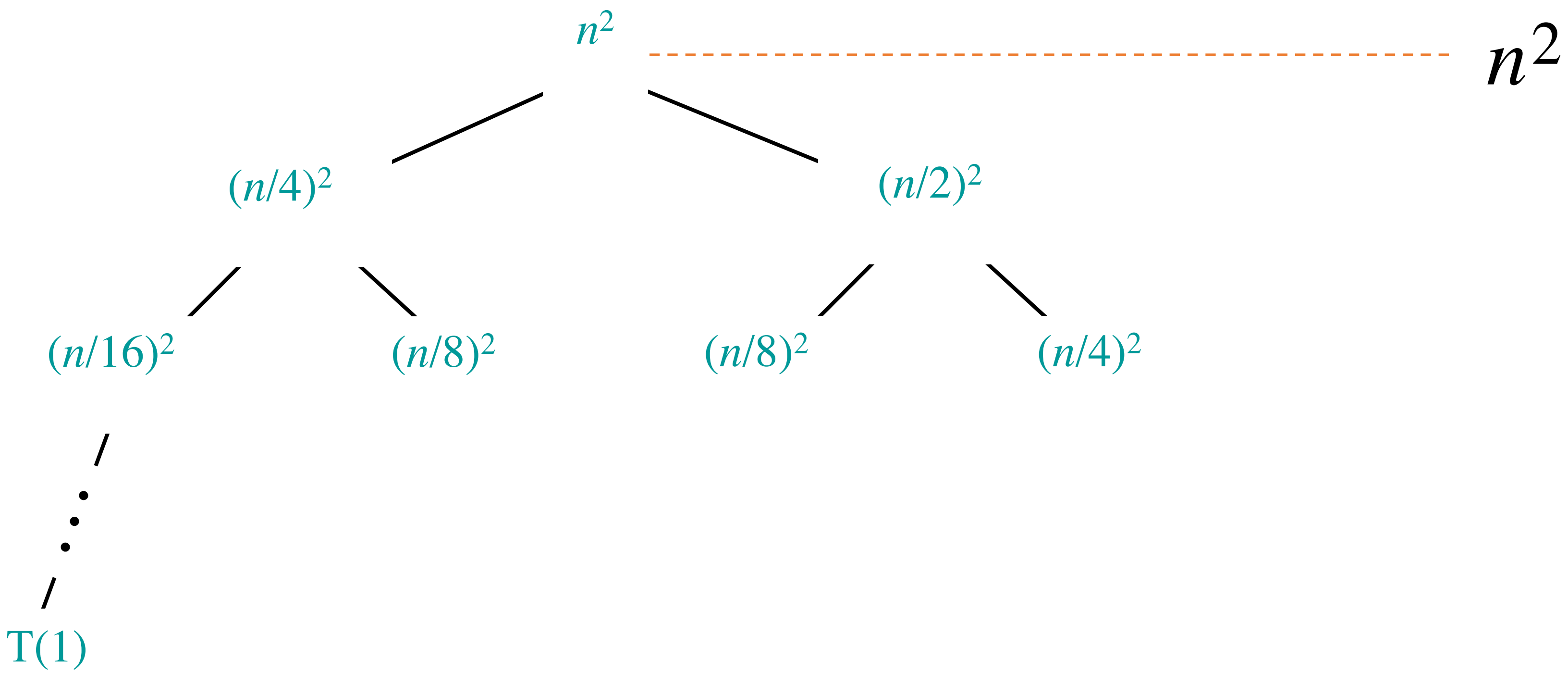
Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



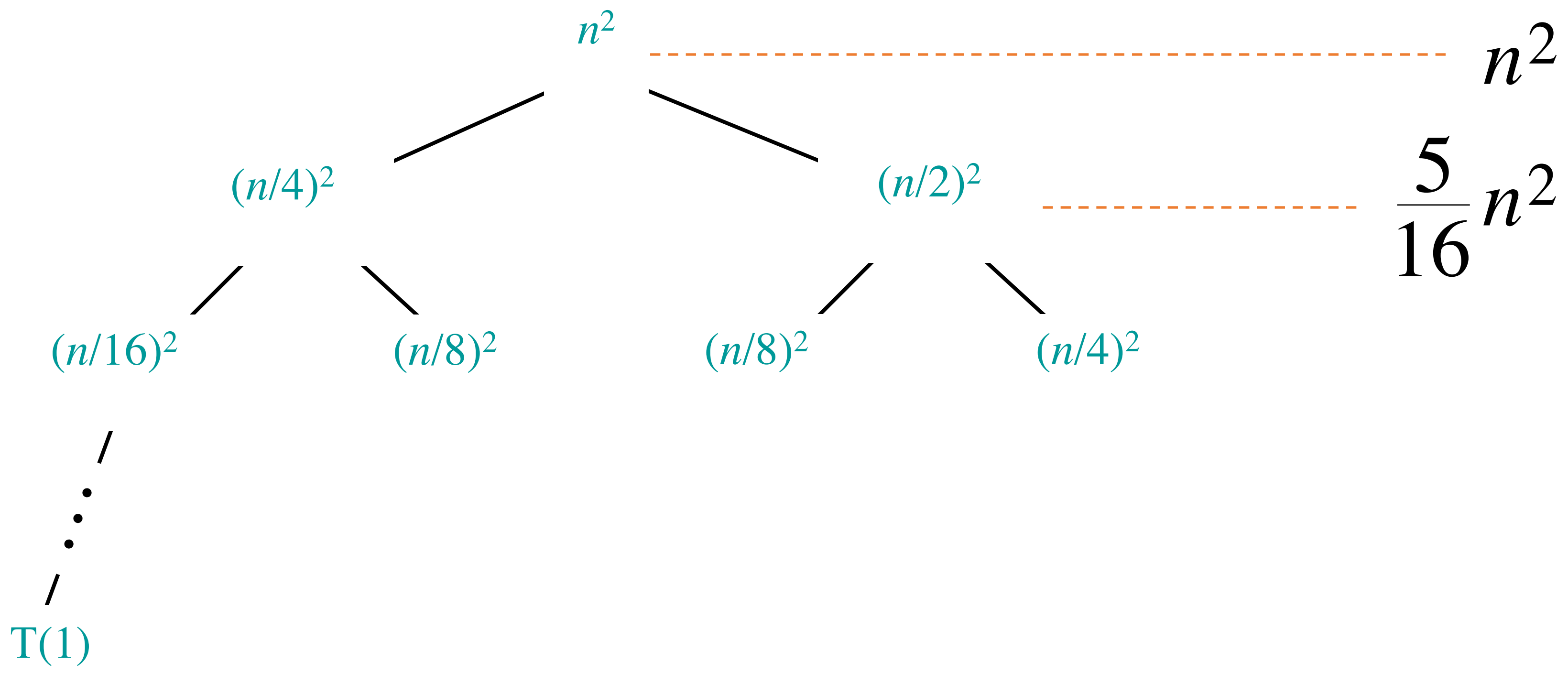
Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



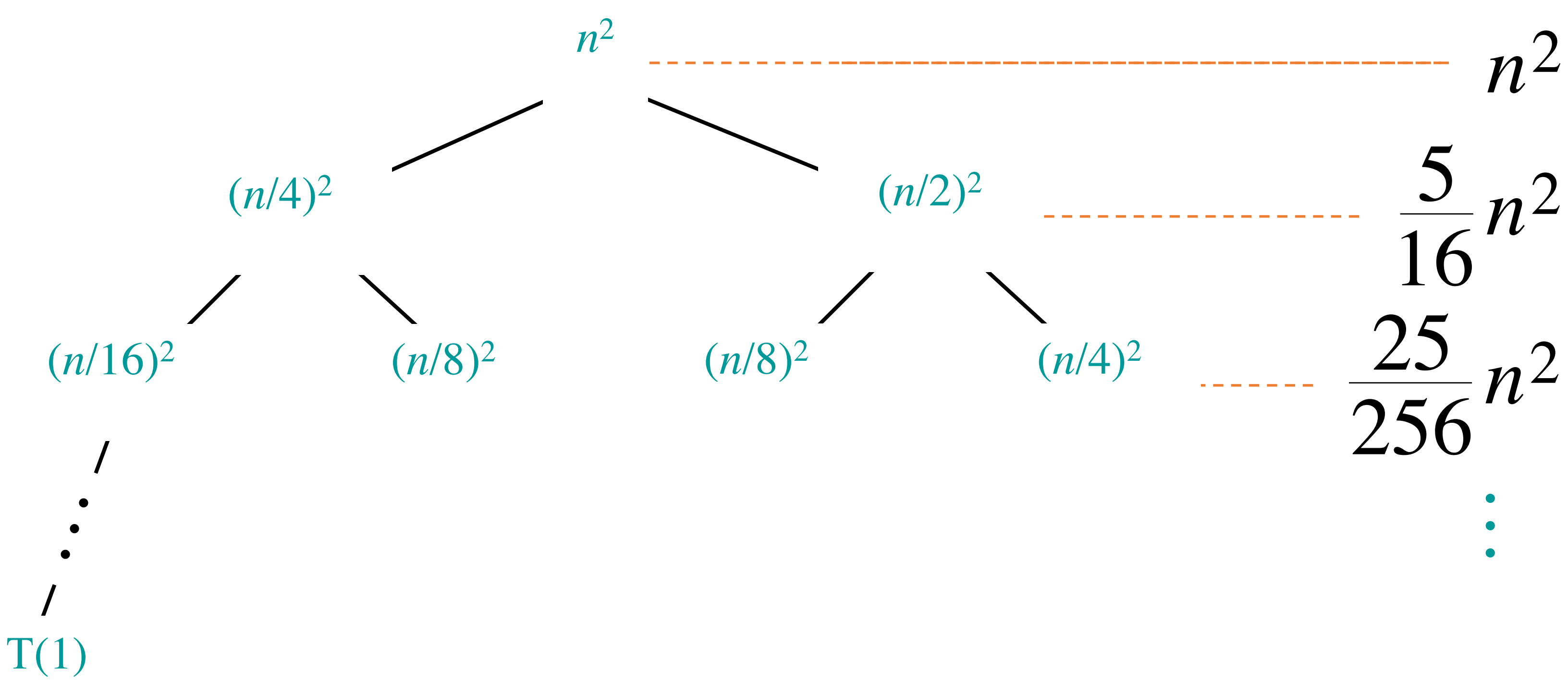
Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



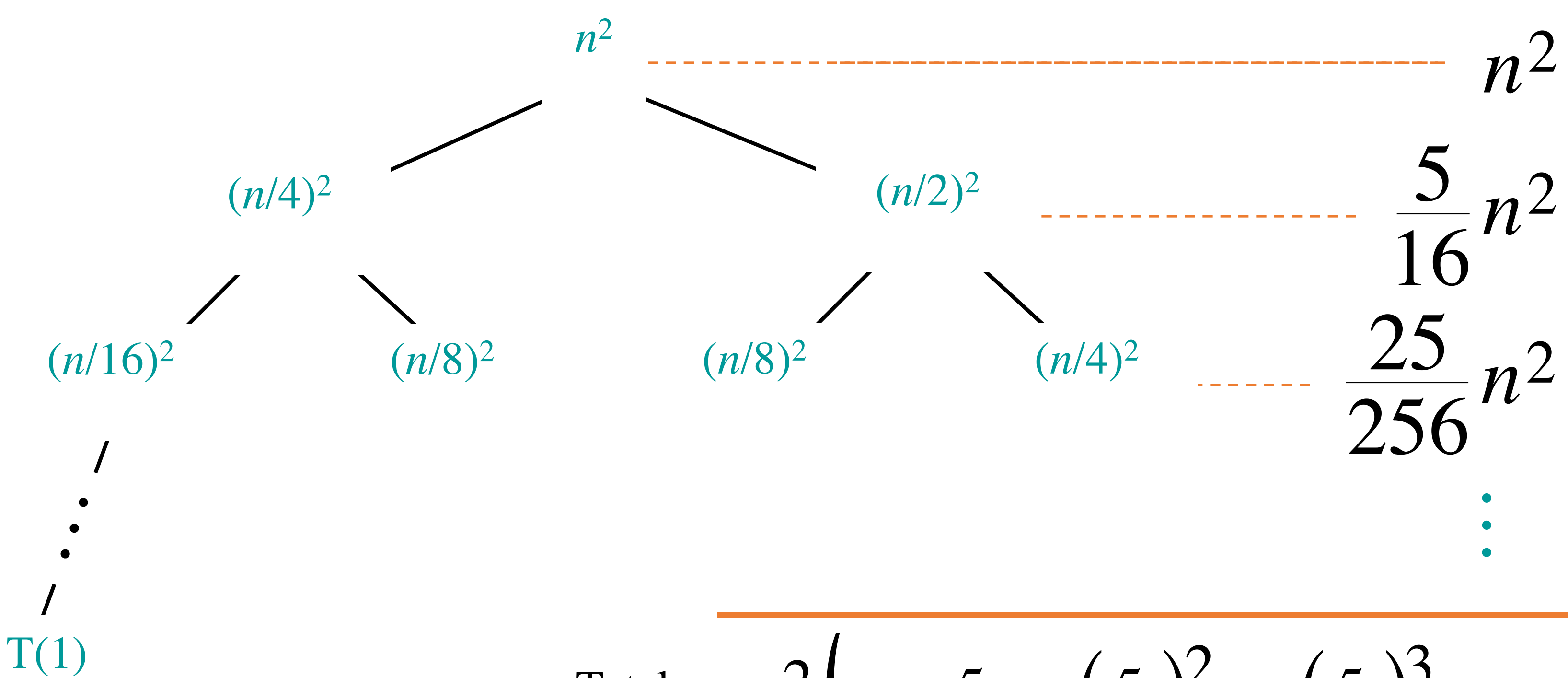
Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Another Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$

$= O(n^2)$ *geometric series*

Imagine Another Merge Sort

```
ImaginaryMergeSort(A)
  if A's size > 1
    Divide array A into 1/3s and 2/3s
    Call ImaginaryMergeSort on first 1/3.
    Call ImaginaryMergeSort on second 2/3.
    Merge two results (combine).
```

- Recurrence relation for `ImaginaryMergeSort`(A)
  - *$T(n) =??$*

Another MergeSort

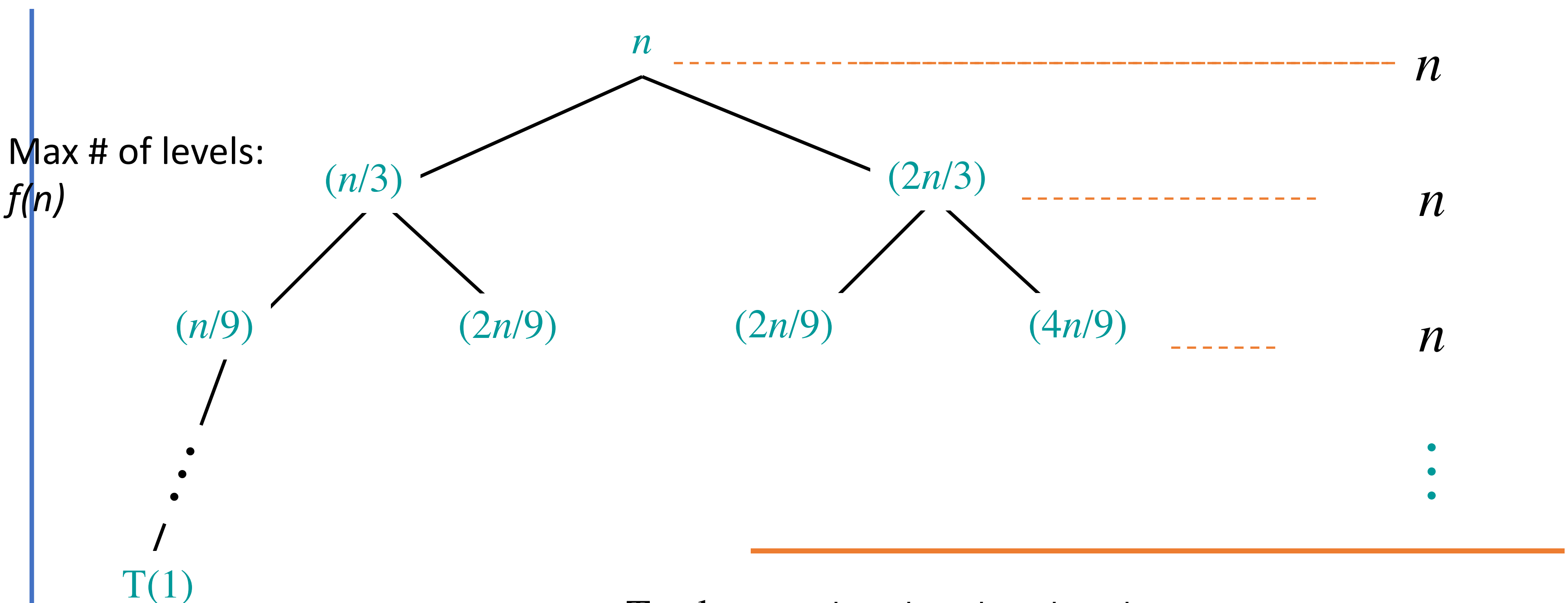
```
MergeSort ( A, r, s )
      if ( r ≥ s)  return;
      m  = r+(s-r) / 3;
      A1 = MergeSort ( A, r, m );
      A2 = MergeSort ( A, m+1, s );
      Merge (A1, A2);
```

- Recurrence relation for MergeSort(A,1,n)
  - $T(n) = T(n/3) + T(2n/3) + cn$

We will ignore the ‘c’ going forward. It’s sloppy but makes things easier to see

Another Example of Recursion Tree

Solve $T(n) = T(n/3) + T(2n/3) + n$:



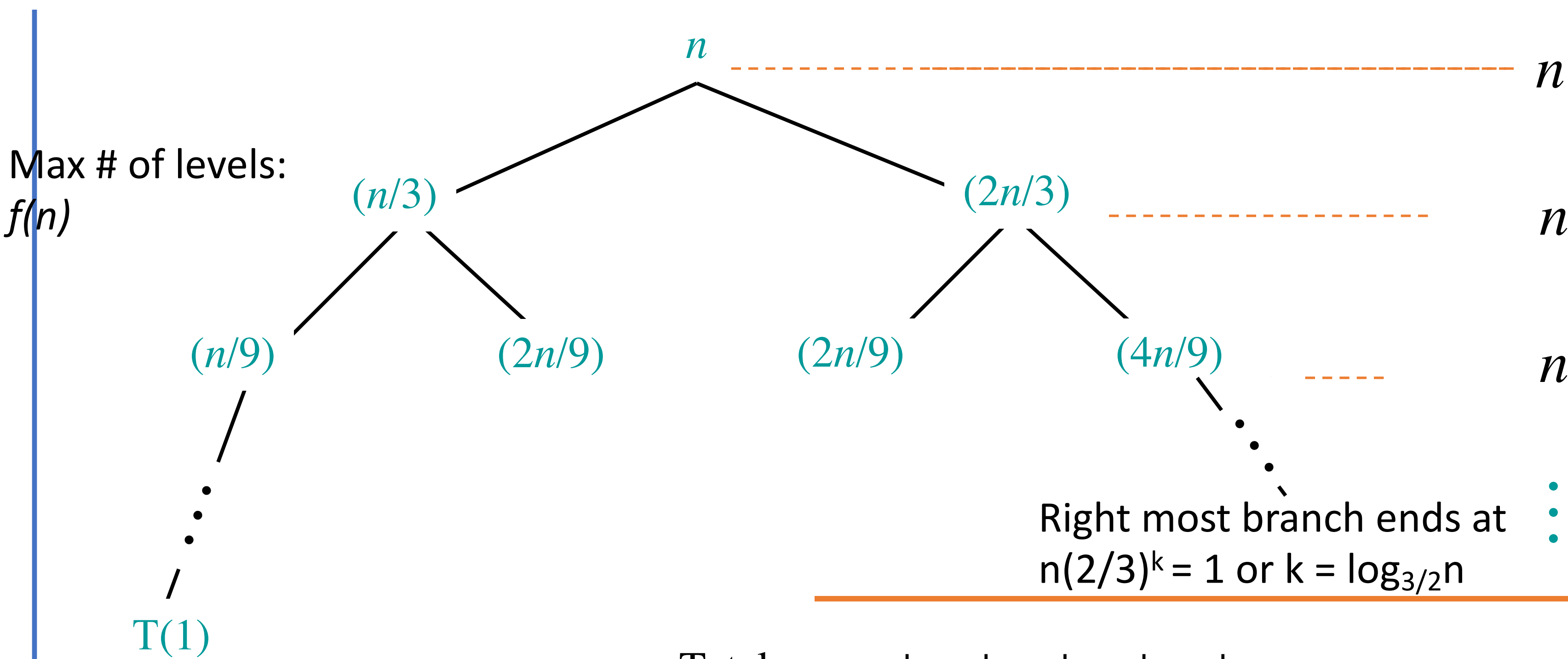
$$\text{Total} = \underbrace{n + n + n + n + n + \ \ldots\ldots}_{\text{Some function f(n)}}$$

What is f(n)???

$= O(n*f(n))$

Another Example of Recursion Tree

Solve $T(n) = T(n/3) + T(2n/3) + n$:



Right most branch ends at
$n(2/3)^k = 1$ or $k = \log_{3/2} n$

Left most branch ends at
$n/3^k = 1$ or $k = \log_3 n$

$$\text{Total} = \underbrace{n + n + n + n + n + \ldots\ldots}_{f(n) = \log_{3/2} n \quad \text{(longest path)}}$$

$= O(n \log_{3/2} n) = O(n \lg n)$

Same as before!!!

Practice for home

- $T(n) = T(n-1) + cn$
  - $T(n) = O(n^2)$
- $T(n) = T(n-1) + cn^2$
  - $T(n) = O(n^3)$
- $T(n) = T\left(\frac{n}{2}\right) + c$
  - $T(n) = O(\lg n)$
- $T(n) = T\left(\frac{n}{3}\right) + cn$
  - $T(n) = O(n)$
- $T(n) = 2T\left(\frac{n}{2}\right) + cn^2$
  - $T(n) = O(n^2)$
- $T(n) = 4T\left(\frac{n}{2}\right) + cn$
  - $T(n) = O(n^2)$
- $T(n) = 4T\left(\frac{n}{2}\right) + cn^2$
  - $T(n) = O(\mathrm{n}^2 \lg n)$

- **Practice as home through unspooling + drawing recursion trees**


That's all Folks!
Any Question ?