

Lecture 3

Recurrences

Md. Asif Bin Khaled

Recurrences

— — —

Recurrences go hand in hand with the **divide and conquer** paradigm because they give us a natural way to characterize the running times of divide and conquer algorithms.

A **recurrence** is an equation or inequality that describes a function in terms of its value on smaller inputs.

Recurrences are used when an algorithm has recursive calls to itself.

Recurrences Continued.

— — —

Recurrences can take many forms. For example, a recursive algorithm might divide subproblems into unequal sizes, such as $\frac{2}{3}$ to $\frac{1}{3}$ split. If the divide and combine steps take linear time, such an algorithm would give rise to the recurrence,

$$T(n) = T(2n/3) + T(n/3) + \Theta(n)$$

Subproblems are not necessarily constrained to being a constant fraction of the original problem size. For example, a recursive linear search would create just one subproblem containing only one element fewer than the original problem,

$$T(n) = T(n-1) + \Theta(1)$$

Recurrences Continued.

Occasionally, we shall see recurrences that are not equalities but rather inequalities, such as,

$$T(n) \leq 2T(n/2) + O(n)$$

Because such a recurrence states only an upper bound on $T(n)$, we will couch its solution using O notation rather than Θ notation. Similarly if the inequality were reversed to,

$$T(n) \geq 2T(n/2) + O(n)$$

Then because the recurrence gives only a lower bound on $T(n)$, we would use the Ω notation in its solution.

Technicalities in Recurrences

In practice, we neglect certain technical detail when we state and solve recurrences. For example, if we call merge sort on n elements when n is odd, we end up with subproblems of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$. Technically, the recurrence describing the worst case running time of merge sort is really,

$$T(n) = \Theta(1) \text{ if } n=1$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) \text{ if } n>1$$

Boundary conditions represent another class of details that we typically ignore.

$$T(n) = 2T(n/2) + \Theta(n), n>1$$

Binary Search

— — —

```
Binary_Search(A, l, r, key):  
    if (r >= l)  
        int mid =(l + r)/2;  
        if (A[mid] == key):  
            return mid  
        if (A[mid] > key):  
            return Binary_Search(A, l, mid-1, key);  
    return Binary_Search(A, mid+1, r, key);
```

Simulation of Binary Search

— — —

	1						8	
A	1	2	3	3	4	5	7	9

key=7

Simulation of Binary Search

— — —

$$\text{mid} = (1+8)/2 = 4$$

A

1							8
1	2	3	3	4	5	7	9

key=7

Simulation of Binary Search

— — —

$\text{mid} = (1+8)/2 = 4$
 $A[\text{mid}] < \text{key}$

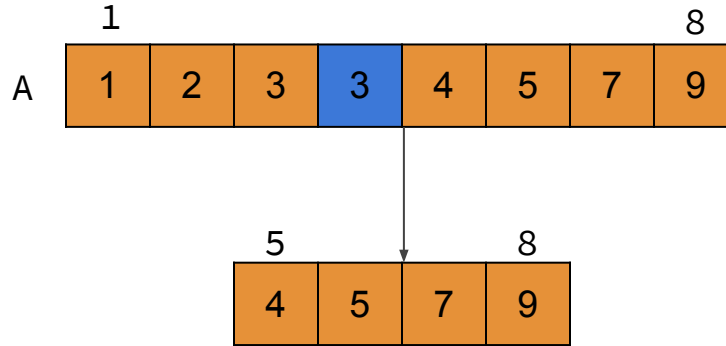
	1						8	
A	1	2	3	3	4	5	7	9

key=7

Simulation of Binary Search

— — —

$$\text{mid} = (1+8)/2 = 4$$

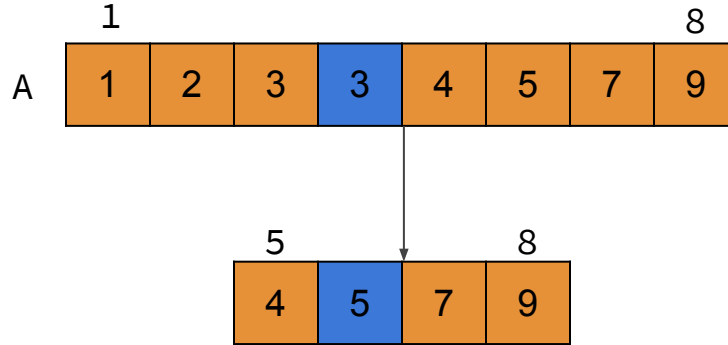


key=7

Simulation of Binary Search

— — —

$\text{mid} = (5+8)/2 = 6$
 $A[\text{mid}] < \text{key}$

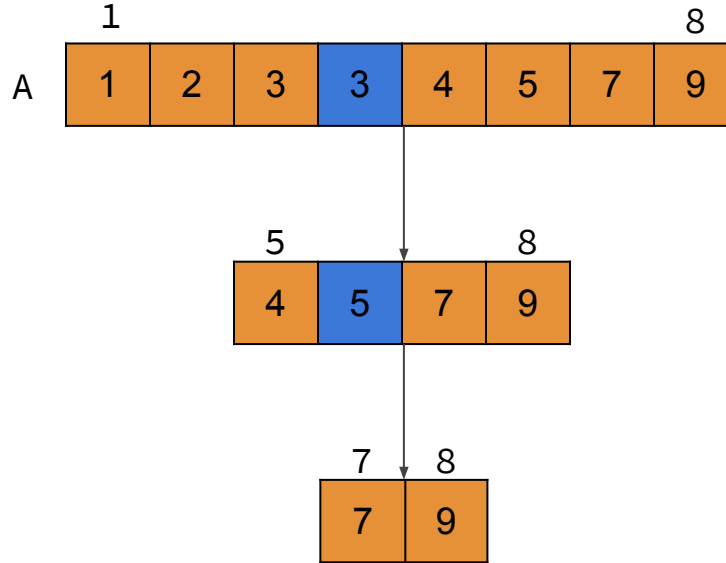


key=7

Simulation of Binary Search

— — —

$$\text{mid} = (7+8)/2 = 7$$

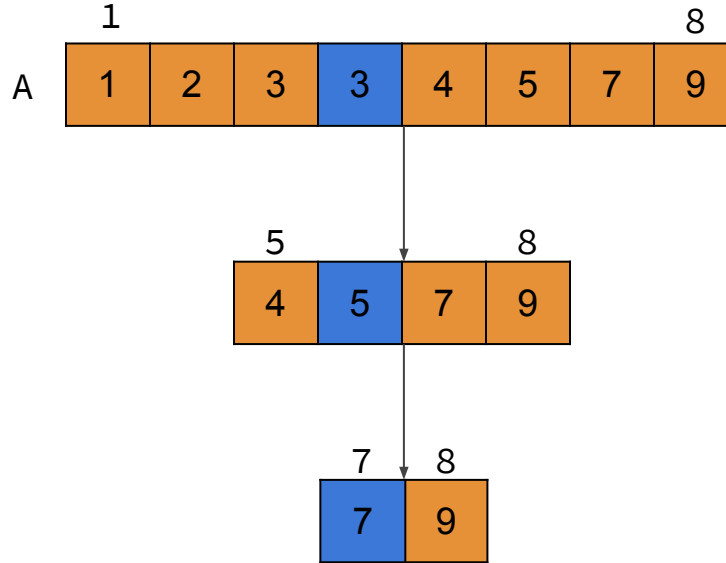


key=7

Simulation of Binary Search

— — —

$$\text{mid} = (7+8)/2 = 7$$

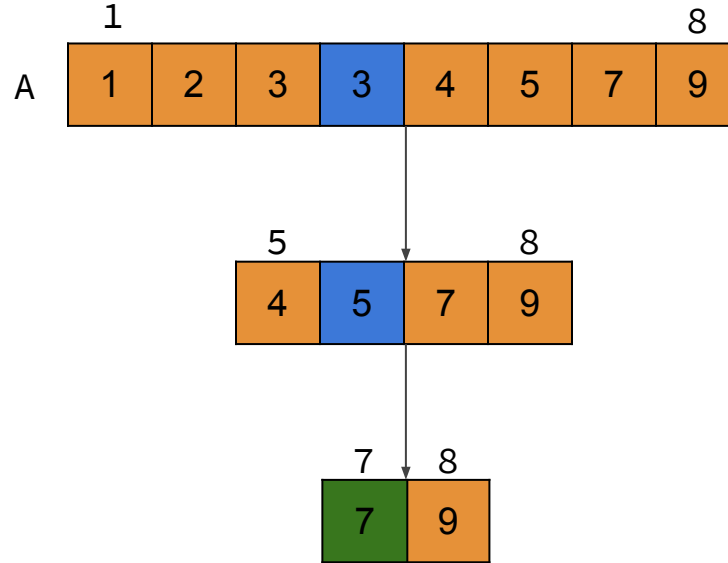


key=7

Simulation of Binary Search

— — —

$\text{mid} = (7+8)/2 = 7$
 $A[\text{mid}] = \text{key}$
found!!



key=7

Solving Recurrences

There are three main methods for solving recurrences, that is , for obtaining asymptotic “ Θ ” or “ O ” bounds on the solution.

- 1. Substitution Method:** we guess a bound and then use mathematical induction to prove our guess correct.
- 2. Recursion Tree Method:** It converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion. We use techniques for bounding summations to solve the recurrence.
- 3. Master Method:** Provides bounds for recurrence of the form, $T(n) = aT(n/b) + f(n)$ where $a \geq 1$, $b > 1$ and $f(n)$ is a given function.

Substitution Method

— — —

The substitution method for solving recurrences involves guessing the form of the solution and then using mathematical induction to find the constants and show that the solution works. The name comes from the substitution of the guessed answer for the function when the inductive hypothesis is applied to smaller values. This method is powerful, but it obviously can be applied only in cases when it is easy to guess the form of the answer.

Substitution Method Example

— — —

$$\begin{aligned}
 1. \quad T(n) &= 2T(n/2) + \theta(n), & T(1) &= 1 \\
 &= 2T(n/2) + cn \\
 &= 2(2T(n/4) + cn/2) + cn && \text{[Substitution]} \\
 &= 4T(n/4) + 2*cn/2 + cn \\
 &= 4T(n/4) + cn + cn \\
 &= 4(2T(n/8) + cn/4) + cn + cn && \text{[Substitution]} \\
 &= 8T(n/8) + 4*cn/4 + cn + cn \\
 &= 8T(n/8) + cn + cn + cn \\
 &\vdots
 \end{aligned}$$

Substitution Method Example Continued

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

Let,

$$n/2^k = 1$$

$$n = 2^k$$

$$\lg n = \lg 2^k$$

$$\lg n = k \quad \lg 2 = 1$$

$$\lg n = k$$

$$\begin{aligned} \text{So, } T(n) &= 2^{\lg n} T(1) + \lg n * cn \\ &= n^{\lg 2} * 1 + \lg n * cn \end{aligned}$$

$$= \mathbf{\Theta(n \lg n)}$$

Substitution Method Example Continued

— — —

$$\begin{aligned}
 2. \quad T(n) &= 3T(2n/3) + \theta(1), \quad T(1) = 1 \\
 &= 3T(n/(3/2)) + c \\
 &= 3(3T(n/(9/4)) + c) + c \quad [\text{Substitution}] \\
 &= 9T(n/(9/4)) + 3c + c \\
 &= 9(3T(n/(27/8)) + c) + 3c + c \quad [\text{Substitution}] \\
 &= 27T(n/(27/8)) + 9c + 3c + c \\
 &\vdots \\
 &= 3^k T(n/(3/2)^k) + c \sum_{i=0}^{\text{till } k-1} 3^i
 \end{aligned}$$

Substitution Method Example Continued

Let,

$$n/(3/2)^k = 1$$

$$n = (3/2)^k$$

$$\lg(3/2)_n = \lg(3/2)_{(3/2)^k}$$

$$\lg(3/2)_n = k \cdot \lg(3/2)_{(3/2)}$$

$$\lg(3/2)_n = k$$

$$\text{So, } T(n) = 3^{\lg(3/2)_n} T(1) + c \sum_{i=0}^{\lg(3/2)_n - 1} 3^i$$

$$= n^{\lg(3/2)_3} \cdot 1 + c \sum_{i=0}^{\lg(3/2)_3 - 1} 3^i$$

$$= \Theta(n^{\lg(3/2)_3}) = \Theta(n^{2.709})$$

Substitution Method Example Continued

— — —

$$\begin{aligned}
 3. \quad T(n) &= 3T(n/2) + \theta(n), \quad T(1) = 1 \\
 &= 3T(n/2) + cn \\
 &= 3(3T(n/4) + cn/2) + cn \quad [\text{Substitution}] \\
 &= 9T(n/4) + 3cn/2 + cn \\
 &= 9(3T(n/8) + cn/4) + 3cn/2 + cn \quad [\text{Substitution}] \\
 &= 27T(n/8) + 9cn/4 + 3cn/2 + cn \\
 &\vdots \\
 &= 3^k T(n/2^k) + cn \sum_{i=0}^{\text{till } k-1} (3/2)^i
 \end{aligned}$$

Substitution Method Example Continued

Let,

$$n/2^k = 1$$

$$n = 2^k$$

$$\lg n = \lg 2^k$$

$$\lg n = k \lg 2$$

$$\lg n = k$$

$$\text{So, } T(n) = 3^{\lg n} T(1) + cn \sum_{i=0}^{\lg n - 1} 3^i / 2^i$$

$$= n^{\lg 3} * 1 + cn \sum_{i=0}^{\lg n - 1} 3^i / 2^i$$

$$= \Theta(n^{\lg 3}) \approx \Theta(n^{1.58})$$

Recursion Tree Method

In **recursion tree** each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.

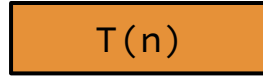
We sum the cost within each level of the tree to obtain a set of per-level costs, and then we sum all the per-level costs to determine the total cost of all the levels of the recursion.

A recursion tree is best used to generate a good guess, which can be then verified using the substitution method. The recursion-tree method can be unreliable, just like any method that uses ellipses.

Recursion Tree Method Example Continued

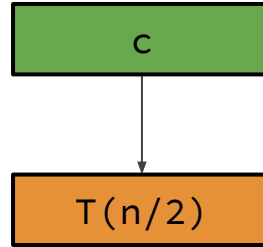
— — —

$$T(n) = T(n/2) + \theta(1)$$


$$T(n)$$

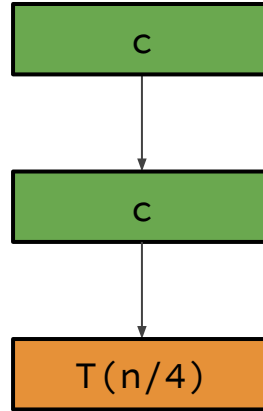
Recursion Tree Method Example Continued

$$T(n) = T(n/2) + \theta(1)$$



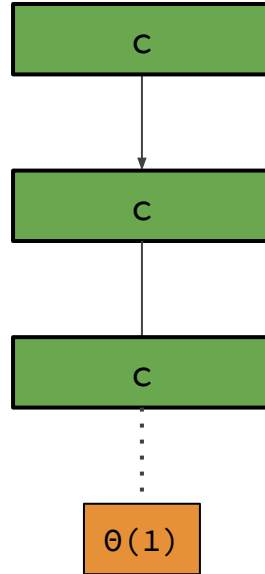
Recursion Tree Method Example Continued

$$T(n) = T(n/2) + \theta(1)$$



Recursion Tree Method Example Continued

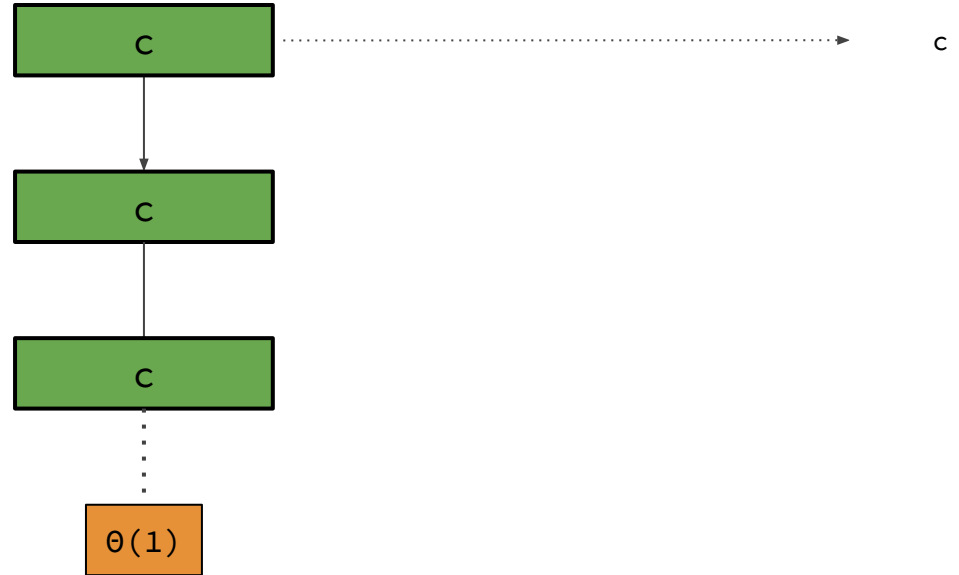
$$T(n) = T(n/2) + \theta(1)$$



Recursion Tree Method Example Continued

— — —

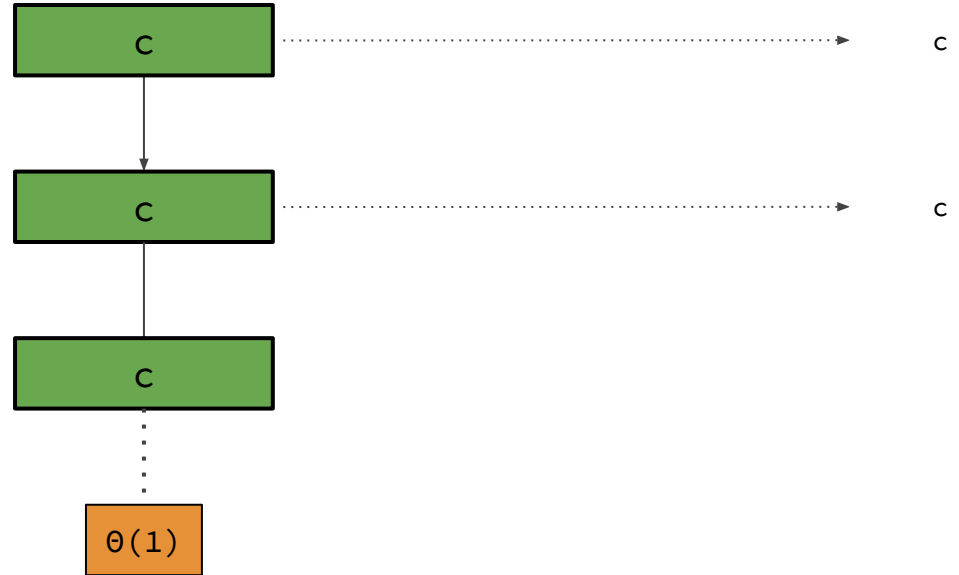
$$T(n) = T(n/2) + \theta(1)$$



Recursion Tree Method Example Continued

— — —

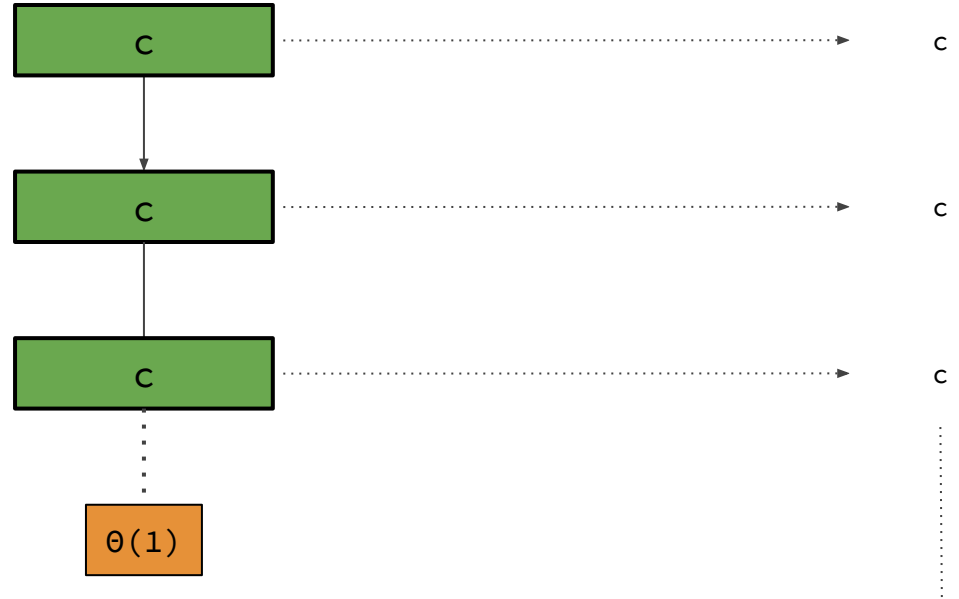
$$T(n) = T(n/2) + \theta(1)$$



Recursion Tree Method Example Continued

— — —

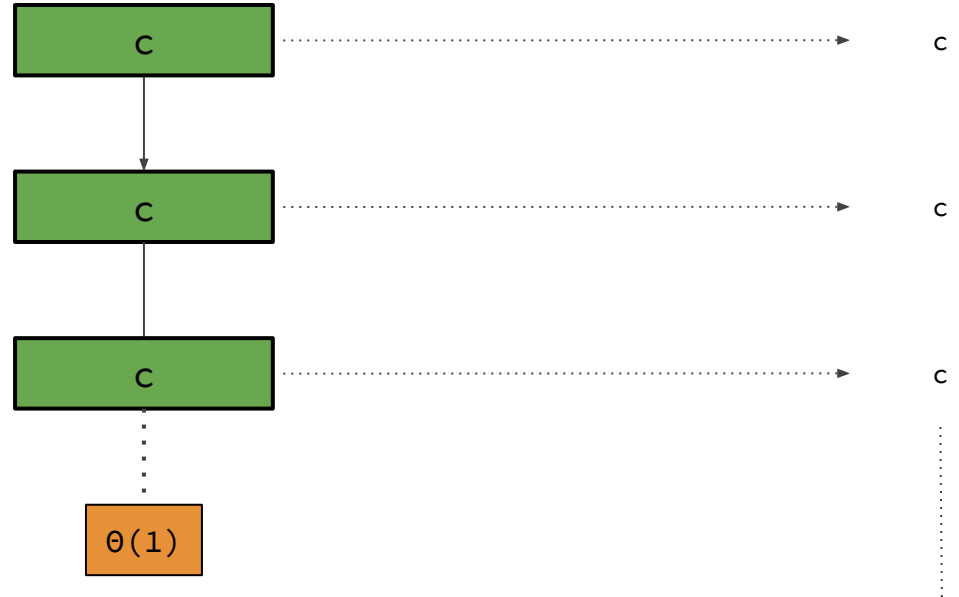
$$T(n) = T(n/2) + \theta(1)$$



Recursion Tree Method Example Continued

— — —

$$T(n) = T(n/2) + \theta(1)$$



$$\begin{aligned} T(n) &= \lg n * c \\ &= \theta(\lg n) \end{aligned}$$

Recursion Tree Method Example Continued

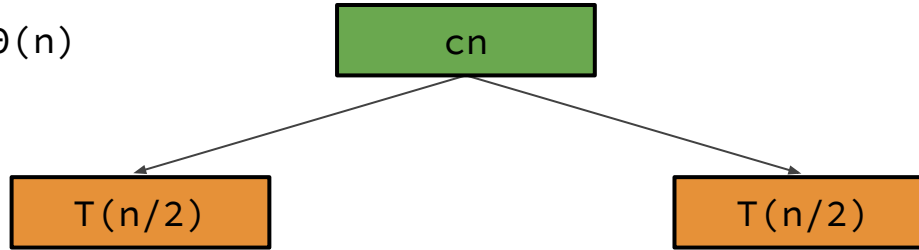
— — —

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

$T(n)$

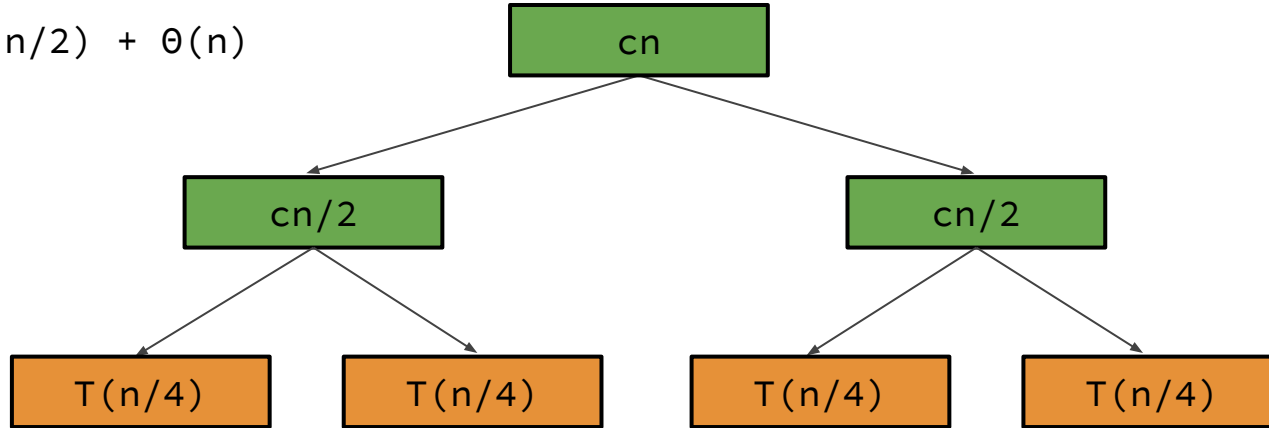
Recursion Tree Method Example Continued

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



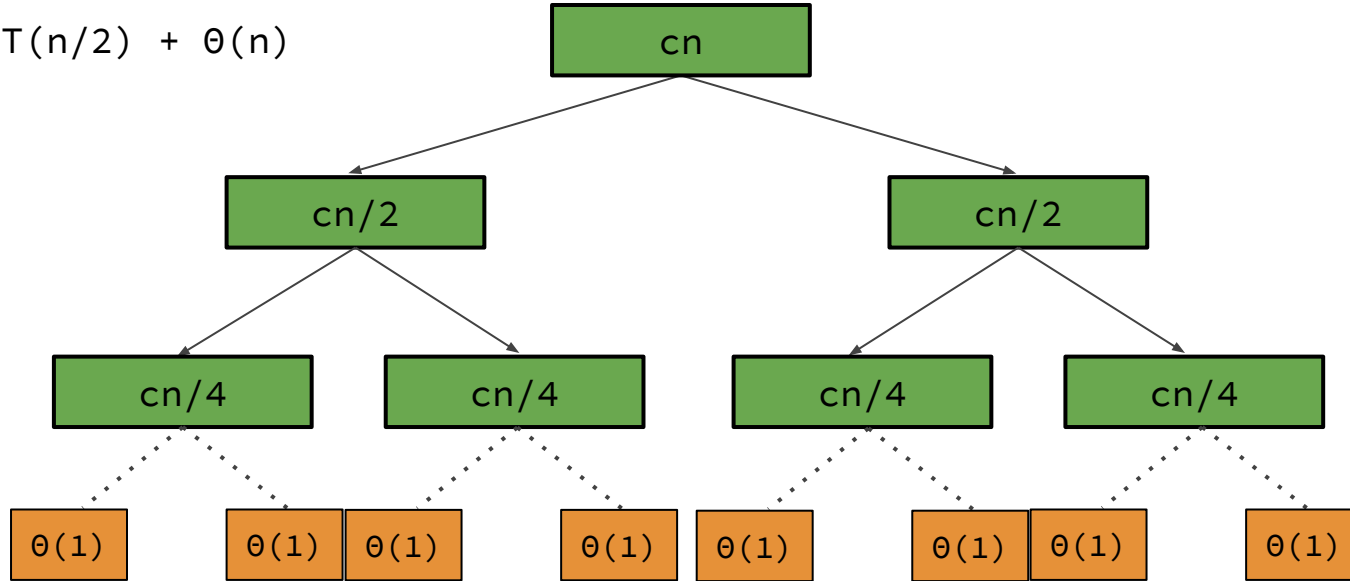
Recursion Tree Method Example Continued

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



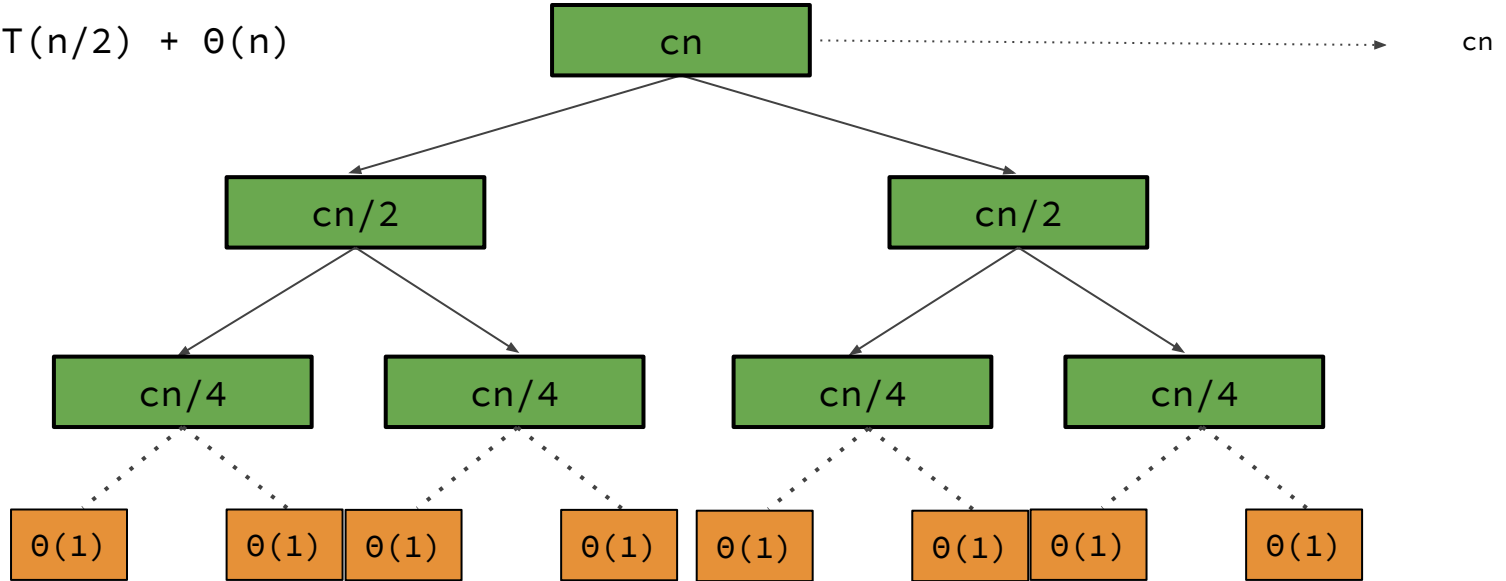
Recursion Tree Method Example Continued

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



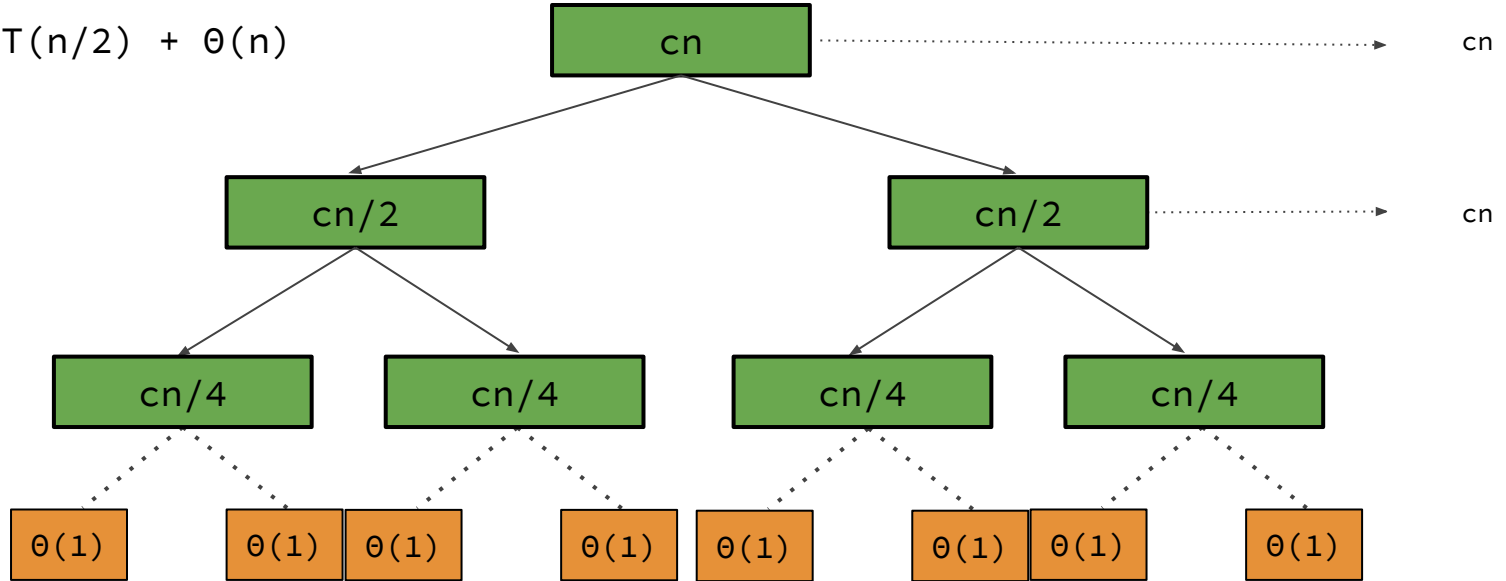
Recursion Tree Method Example Continued

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



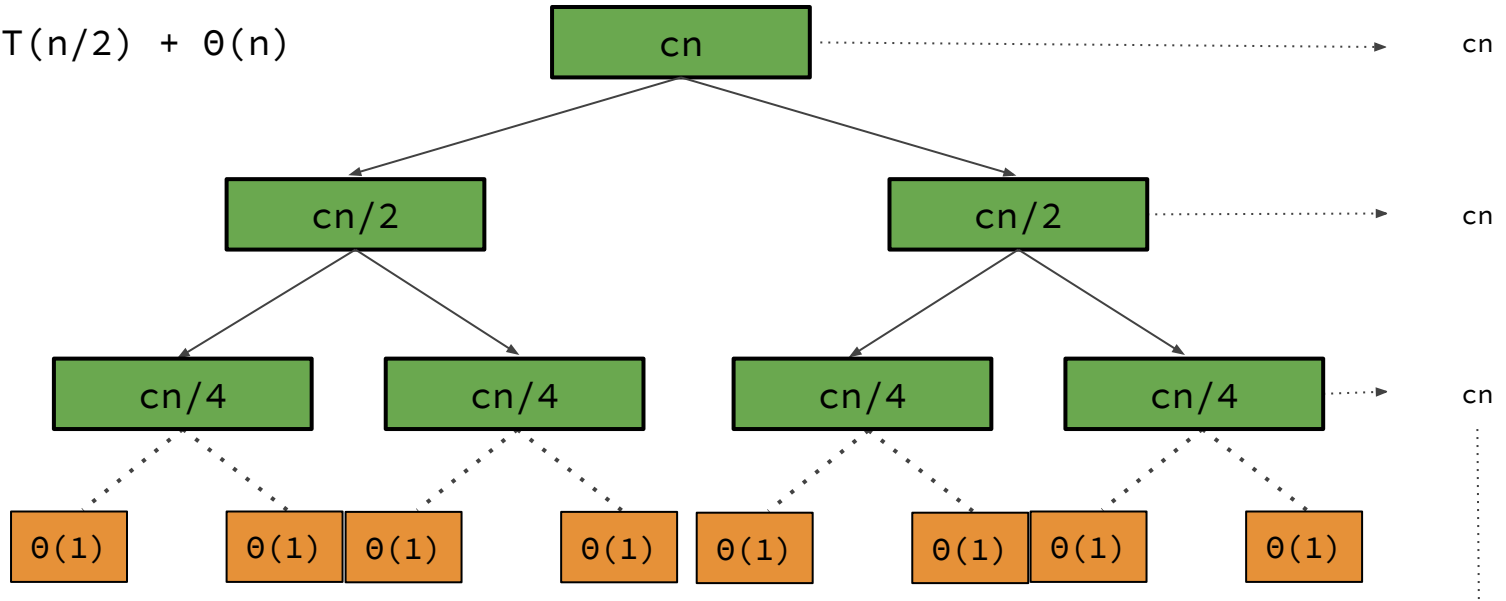
Recursion Tree Method Example Continued

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



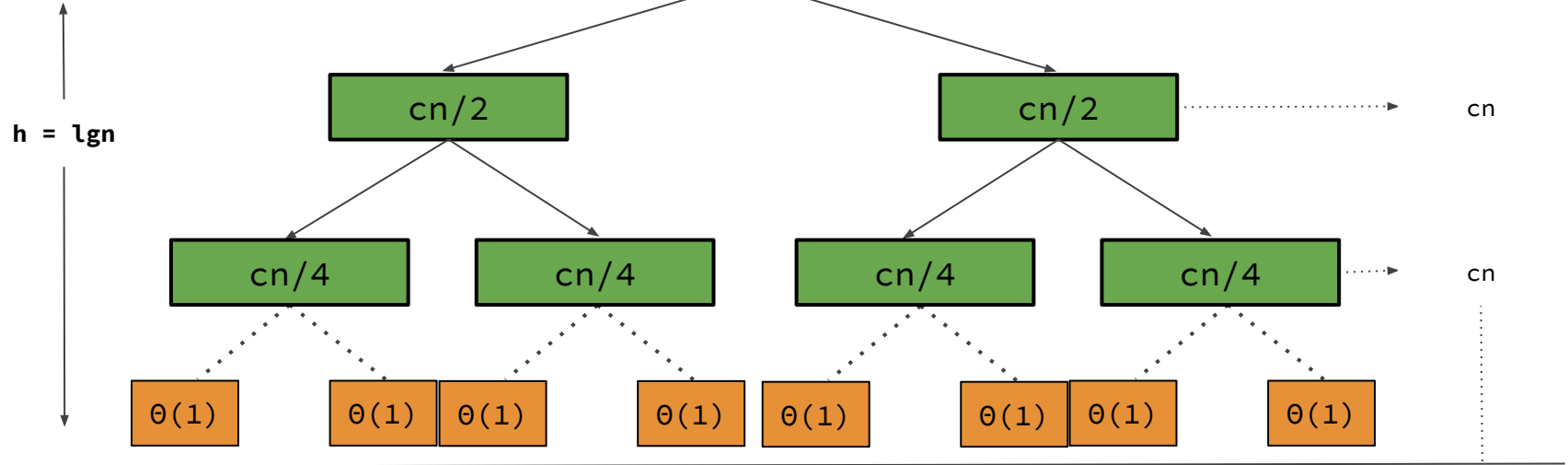
Recursion Tree Method Example Continued

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



Recursion Tree Method Example Continued

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



$$\begin{aligned} T(n) &= \lg n * cn \\ &= \theta(n \lg n) \end{aligned}$$

Recursion Tree Method Example

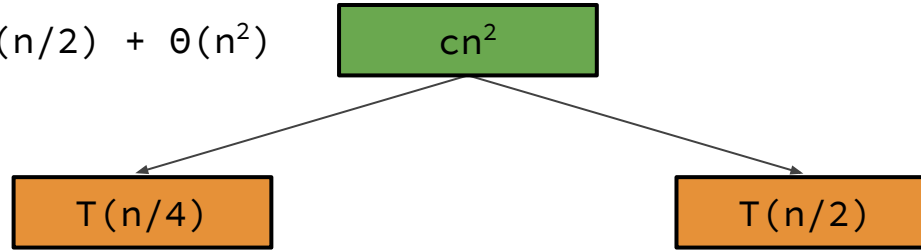
— — —

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

$T(n)$

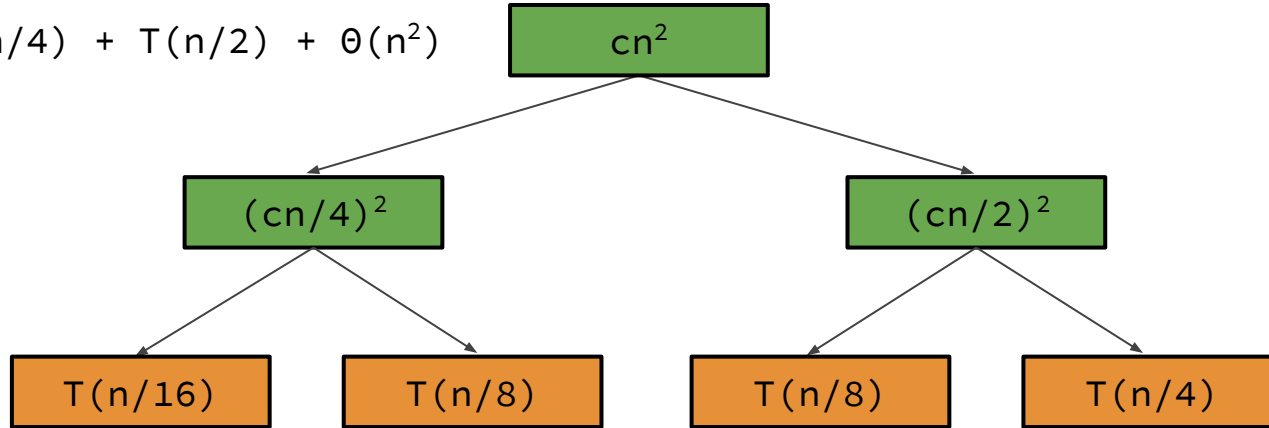
Recursion Tree Method Example Continued

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



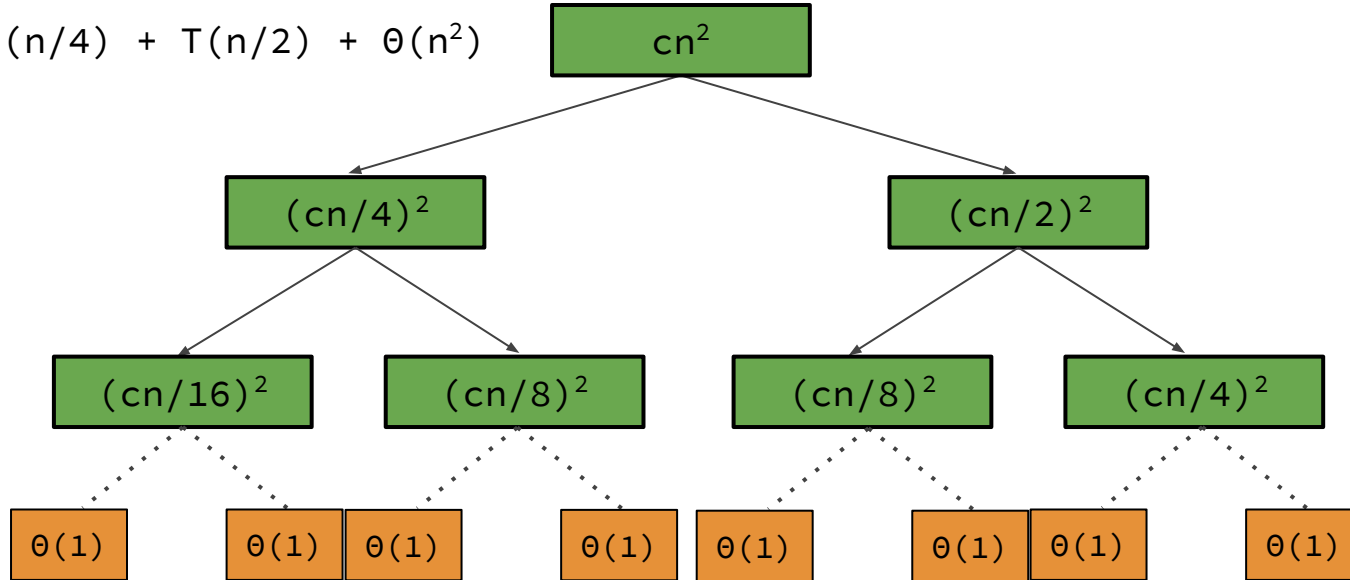
Recursion Tree Method Example Continued

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



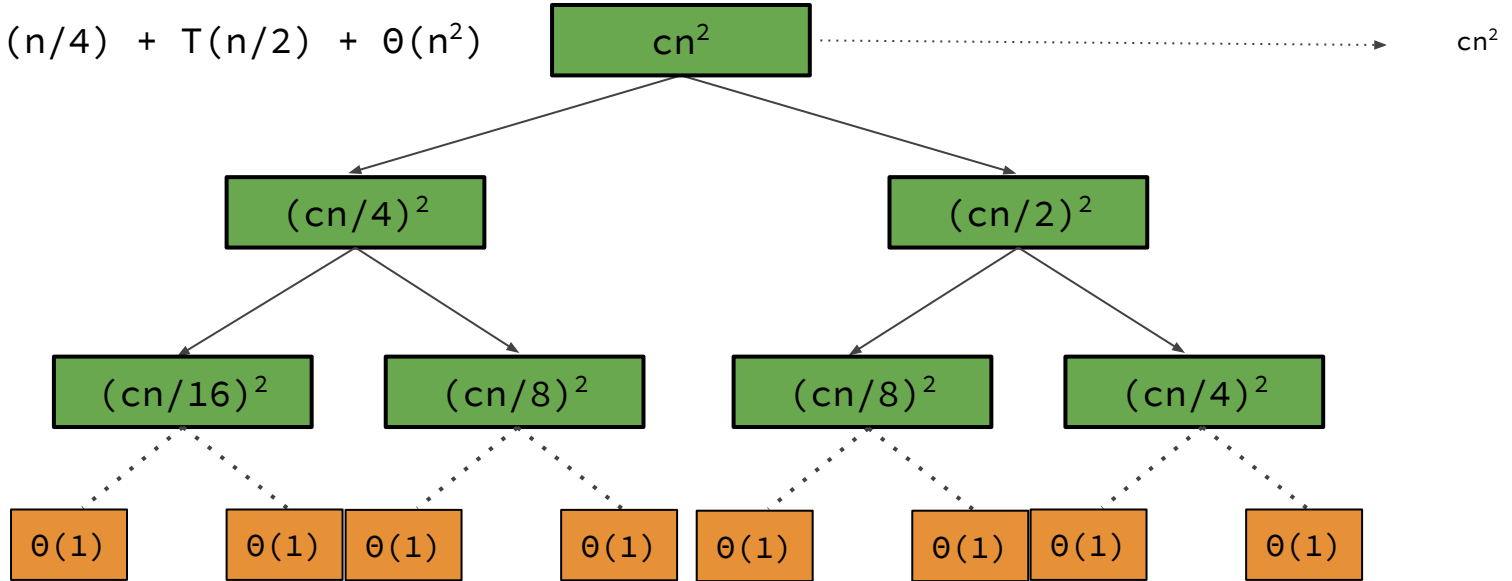
Recursion Tree Method Example Continued

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



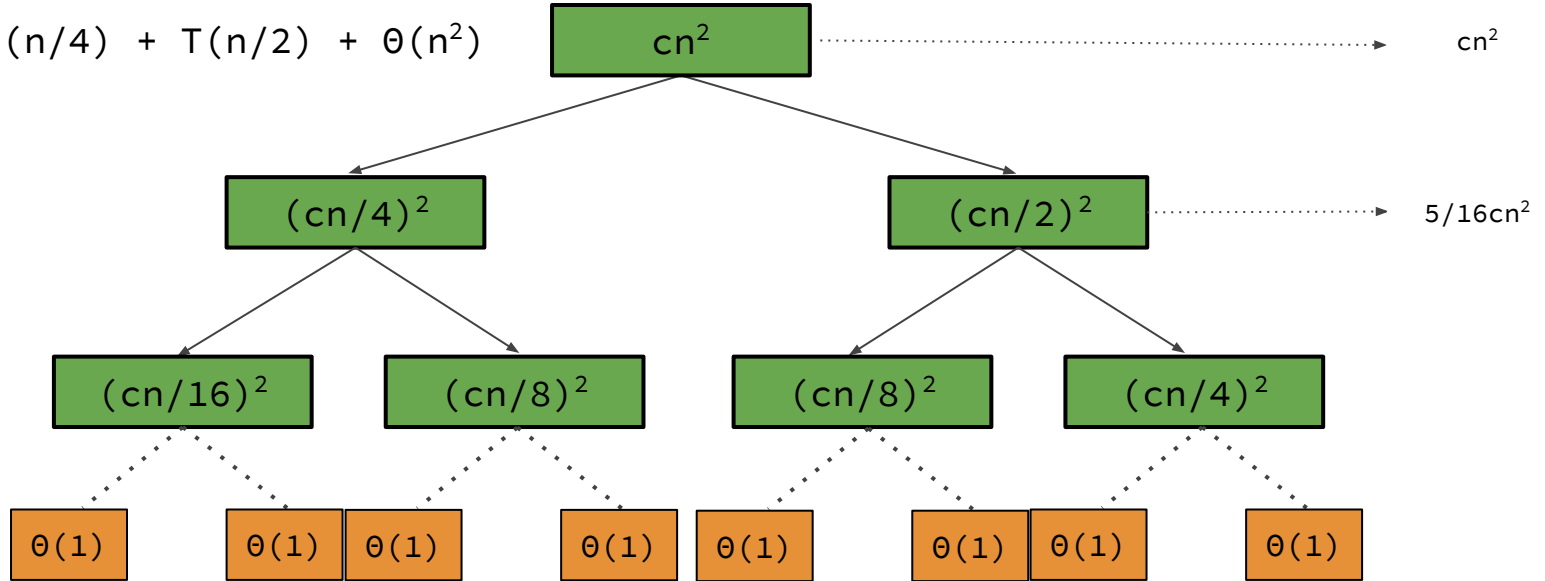
Recursion Tree Method Example Continued

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



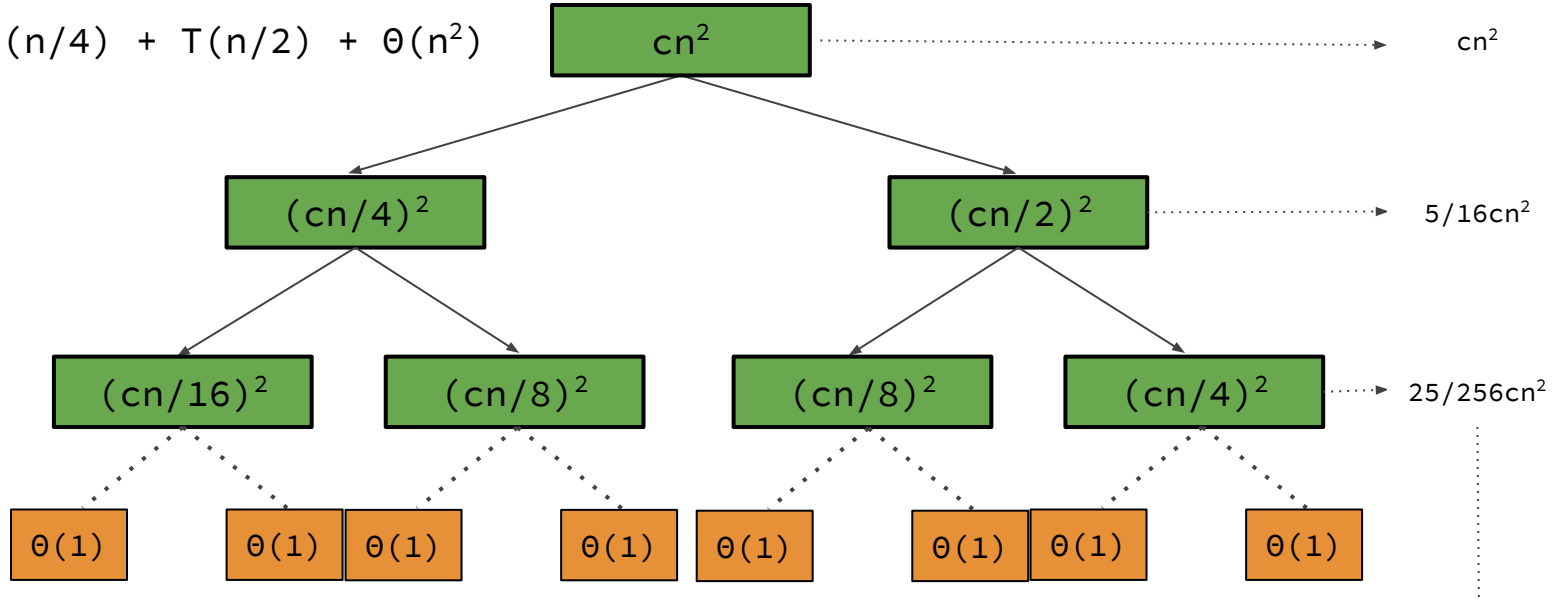
Recursion Tree Method Example Continued

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



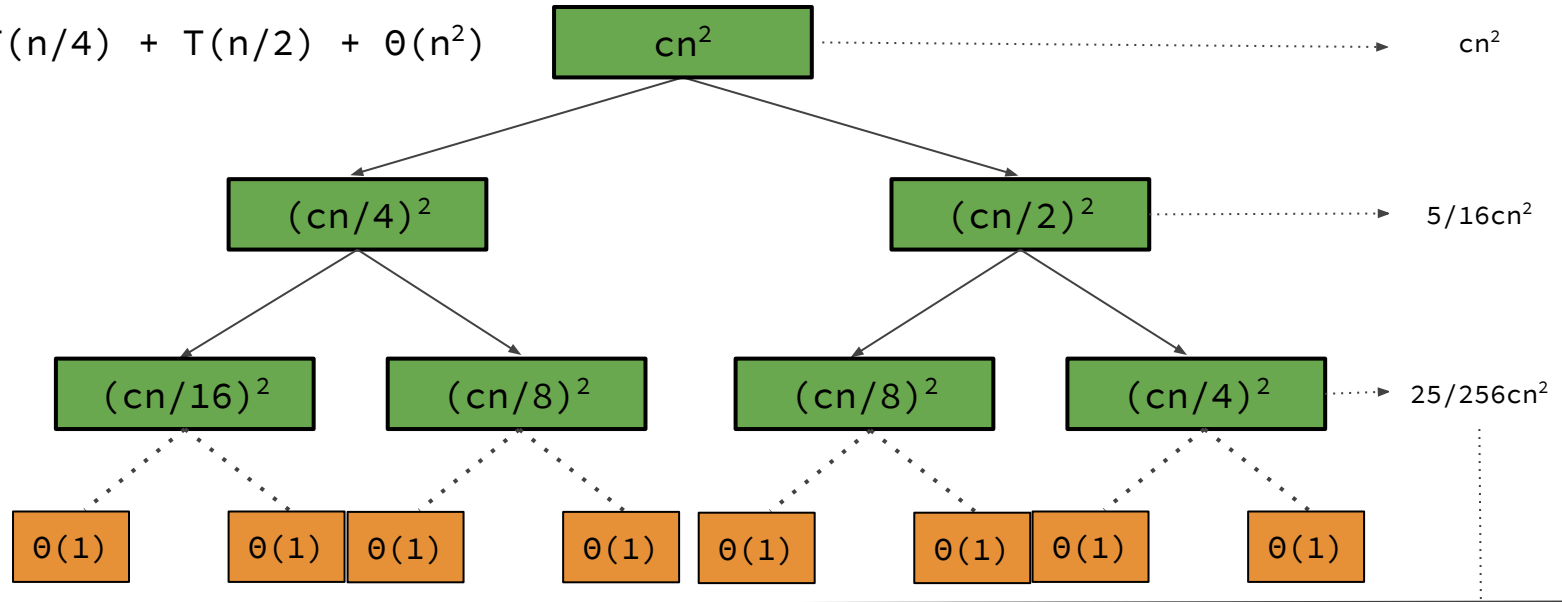
Recursion Tree Method Example Continued

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



Recursion Tree Method Example Continued

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



$$\begin{aligned} T(n) &\leq cn^2(1 + 5/16 + 25/256 + \dots) \\ &\leq cn^2 * (\text{a geometric series, a constant}) \\ &\leq \mathbf{O(n^2)} \end{aligned}$$

Master Method

— — —

The master method provides a “**cookbook**” method for solving recurrences of the form

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

Where, **$a \geq 1$** and **$b > 1$** are given positive constants and **$f(n)$** is an asymptotically positive function. This recurrence describes the running time of an algorithm that divides a problem of size n into **a** subproblems, each of size n/b . The function **$f(n)$** encompasses the cost of dividing the problem and combining the results of the subproblems.

Master Theorem

The master method depends on the following theorem.

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

Where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds,

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

Master Theorem Examples

— — —

1. $T(n) = 2T(n/2) + \theta(n)$

Here,

$a=2$, $b=2$ and $f(n)=n$

So,

$$n^{\log_b a} = n^{\log_2 2} = n^1 = n$$

$n=n$

Thus, case 2

So, $T(n) = \theta(n \lg^{(0+1)} n) = \theta(n \lg^1 n) = \theta(n \lg n)$

Master Theorem Examples Continued.

— — —

$$2. \quad T(n) = T(n/2) + \theta(1)$$

Here,

$$a=1, \quad b=2 \quad \text{and} \quad f(n)=1$$

So,

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$$
$$1=1$$

Thus, case 2

$$\text{So, } T(n) = \theta(1 \cdot \lg^{(0+1)} n) = \theta(\lg^1 n) = \theta(\lg n)$$

Master Theorem Examples Continued.

— — —

$$3. \quad T(n) = 4T(n/2) + \theta(n)$$

Here,

$$a=4, \quad b=2 \quad \text{and} \quad f(n)=n$$

So,

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$n < n^2, \quad f(n) = O(n^{2-\epsilon}) \quad \text{for} \quad \epsilon=1$$

Thus, case 1

$$\text{So, } T(n) = \theta(n^2)$$

Master Theorem Examples Continued.

— — —

$$4. \quad T(n) = 4T(n/2) + \theta(n^2)$$

Here,

$$a=4, \quad b=2 \quad \text{and} \quad f(n)=n^2$$

So,

$$n^{\log_b a} = n^{\log_2 4} = n^2$$
$$n^2 = n^2$$

Thus, case 2

$$\text{So, } T(n) = \theta(n^2 \lg^{(0+1)} n) = \theta(n^2 \lg^1 n) = \theta(n^2 \lg n)$$

Master Theorem Examples Continued.

— — —

5. $T(n) = 4T(n/2) + \Theta(n^3)$

Here,

$a=4$, $b=2$ and $f(n)=n^3$

So,

$n^{\log_b a} = n^{\log_2 4} = n^2$

$n^3 > n^2$, $f(n) = O(n^{2+\epsilon})$ for $\epsilon=1$

Thus, case 3

and, $af(n/b) \leq cf(n)$ for $c < 1$

Master Theorem Examples Continued.

— — —

$$4 \star (n/2)^3 \leq cn^3$$

$$4 \star n^3 / 8 \leq cn^3$$

$$1/2 \leq c$$

$$1/(2c) \leq 1$$

taking $c=1/2$

$$1/(2 \star 1/2) \leq 1$$

$$1 \leq 1 \quad [\text{true}]$$

So, $T(n) = \mathbf{\Theta(n^3)}$

Master Theorem Examples Continued.

— — —

$$6. \quad T(n) = T(2n/3) + \theta(1) = T(n/(3/2)) + \theta(1)$$

Here,

$$a=1, \quad b=3/2 \quad \text{and} \quad f(n)=1$$

So,

$$n^{\log_b a} = n^{\log_{3/2} 1} = n^{\log_{3/2} (3/2)^0} = n^0 = 1$$
$$1=1$$

Thus,

case 2

$$\text{So,} \quad T(n) = \theta(1 \times \lg^{(0+1)} n) = \theta(\lg^1 n) = \theta(\lg n)$$

Muster Theorem**

— — —