Solving Recurrence

Tanjina Helaly

Recurrence

- Recurrences are a major tool for analysis of algorithms
- Divide and Conquer algorithms which are analyzable by recurrences.

Recurrences and Running Time

- An equation or inequality that describes a function in terms of its value on smaller inputs.

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

- Recurrences arise when an algorithm contains recursive calls to itself
- What is the actual running time of the algorithm?
- Need to solve the recurrence
  - Find an explicit formula of the expression
  - Bound the recurrence by an expression that involves n

Example Recurrences

- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\lg n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Methods for Solving Recurrences

- Iteration method
- Substitution method
- Recursion tree method
- Master method

Iteration method

The Iteration Method

- Convert the recurrence into a summation and try to bound it using known series
  - Iterate the recurrence until the initial condition is reached.
  - Use back-substitution to express the recurrence in terms of n and the initial (boundary) condition.

The Iteration Method

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

$$\begin{aligned}
T(n) &= c + T(n/2) = c+T(n/2^1) \\
&= c + c + T(n/4) = c+c+T(n/2^2) \\
&= c + c + c + T(n/8) = c+c+c+T(n/2^3) \\
&= \ldots\ldots \\
&= c+c+\ldots+c+T(n/2^k)
\end{aligned}$$

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

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

Assume $n = 2^k \Rightarrow k = \log n$

$$\text{So, } T(n) = c \underbrace{+ c + \ldots + c + T}_{\text{k times}}(1)$$

$$\begin{aligned}
&= c\lg n + T(1) \\
&= \Theta(\lg n) \qquad [T(1) = \text{constant}]
\end{aligned}$$

Iteration Method – Example

$$\mathbf{T(n) = n + 2T(n/2)}$$

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

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

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

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

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

$= in + 2^iT(n/2^i)$

$= kn + 2^kT(1)$

$= n\lg n + nT(1) = \Theta(n\lg n)$ [T(1) = constant]

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

Assume 2^k

$n/2^k = 1 \Rightarrow k = \log n$

Substitution method

THE SUBSTITUTION METHOD

- Guess a bound/solution
- Use mathematical induction to prove our guess correct.

Substitution method

- Guess a solution
  - $T(n) = O(g(n))$
  - Induction goal: apply the definition of the asymptotic notation
    - $T(n) \leq d\, g(n)$, for some $d > 0$ and $n \geq n_0$ (strong induction)
  - Induction hypothesis: $T(k) \leq d\, g(k)$ for all $k < n$
- Prove the induction goal
  - Use the **induction hypothesis** to find some values of the constants d and n_0 for which the **induction goal** holds

Substitution method – Example(Binary search)

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

- Guess: $T(n) = O(\lg n)$
  - Induction goal: $T(n) \le d \lg n$, for some d and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le d \lg(n/2)$
- Proof of induction goal:

$$T(n) = T(n/2) + c \le d \lg(n/2) + c$$

$$= d \lg n - d + c \le d \lg n$$

if: $-d + c \le 0$, $d \ge c$

Substitution method – Example 2

$$\mathbf{T(n) = T(n\text{-}1) + n}$$

- Guess: $T(n) = O(n^2)$
  - Induction goal: $T(n) \leq c\ n^2$, for some c and $n \geq n_0$
  - Induction hypothesis: $T(n-1) \leq c(n-1)^2$ for all $k < n$
- Proof of induction goal:

$$T(n) = T(n-1) + n \leq c\ (n-1)^2 + n$$

$$= cn^2 - (2cn - c - n) \leq cn^2$$

if: $2cn - c - n \geq 0 \Leftrightarrow c \geq n/(2n-1) \Leftrightarrow c \geq 1/(2 - 1/n)$

For $n \geq 1 \Rightarrow 2 - 1/n \geq 1 \Rightarrow$ any $c \geq 1$ will work

Substitution method – Example 3

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

- Guess: $T(n) = O(n\lg n)$
  - Induction goal: $T(n) \le cn \lg n$, for some c and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le cn/2 \lg(n/2)$
- Proof of induction goal:

$$T(n) = 2T(n/2) + n \le 2c\,(n/2)\lg(n/2) + n$$

$$= cn \lg n - cn + n \le cn \lg n$$

if: $-cn + n \le 0 \Rightarrow c \ge 1$

Recursion Tree method

Recursion Tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- Convert the recurrence into a tree:
  - Each node represents the cost incurred at various levels of recursion
  - Sum up the costs of all levels

- The recursion tree method is good for generating guesses for the substitution method.

Example – Merge sort

Time Complexity, $T(n) = 2T(n/2) + cn$

$T(n)$

Example – Merge sort Cont…

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

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

Example – Merge sort Cont…

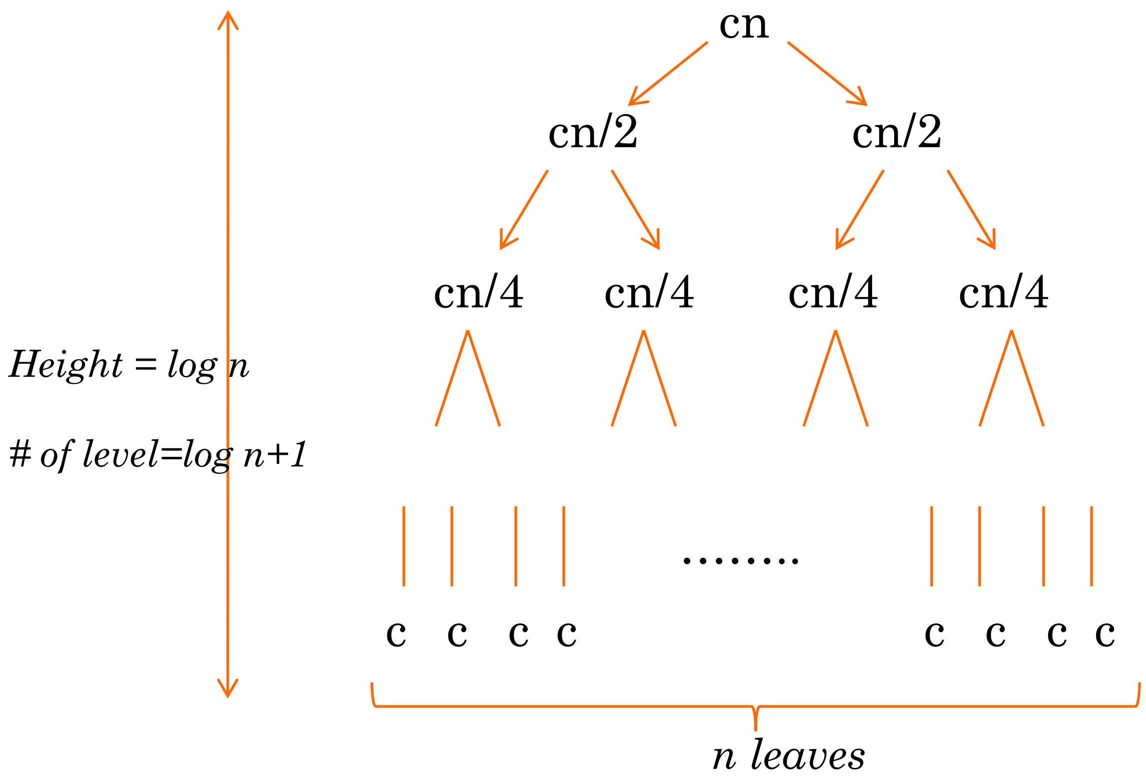
$T(n) = 2T(n/2) + cn$

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

$$
\begin{tikzcd}
& & & cn \arrow[dll] \arrow[drr] & & & \\
& cn/2 \arrow[dl] \arrow[dr] & & & & cn/2 \arrow[dl] \arrow[dr] & \\
T(n/4) & & T(n/4) & & T(n/4) & & T(n/4)
\end{tikzcd}
$$

Example – Merge sort Cont…

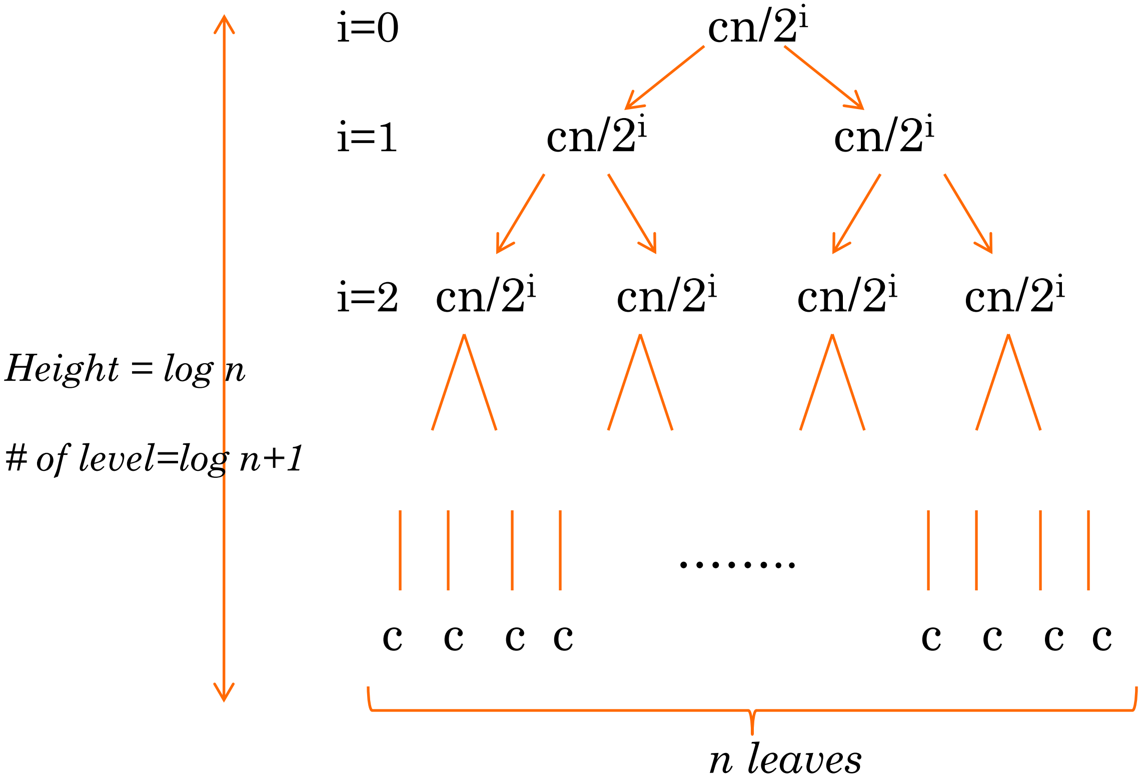
T(n) = 2T(n/2) + cn



EXAMPLE – MERGE SORT CONT…

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

Cost for each node = $cn/2^i$.
Cost at each level = $2^i * cn/2^i$
= cn

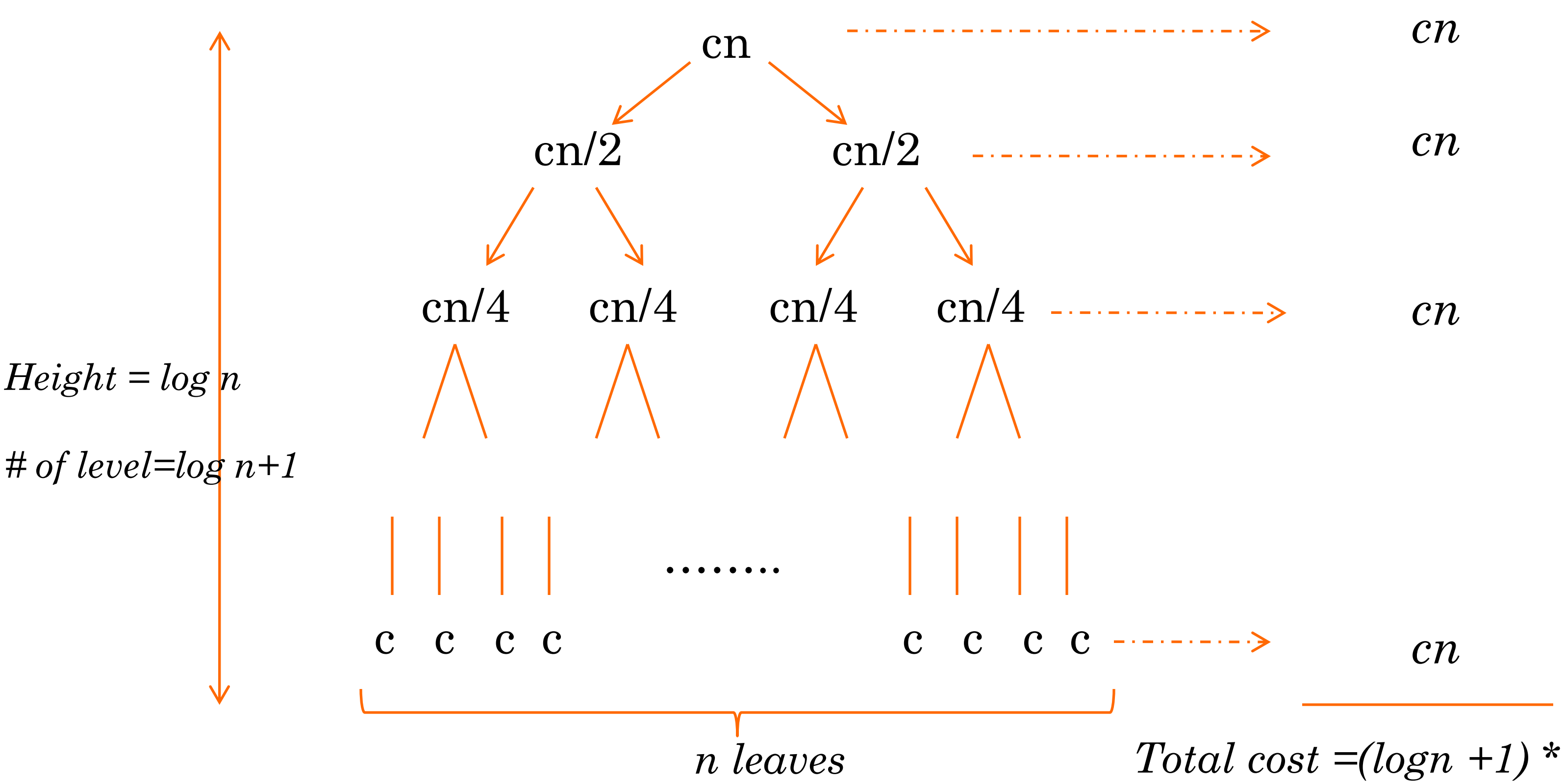


At last level, $n/2^i = 1$
So, $2^i = n$
=> $i = \log n$

EXAMPLE – MERGE SORT CONT…

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

Cost



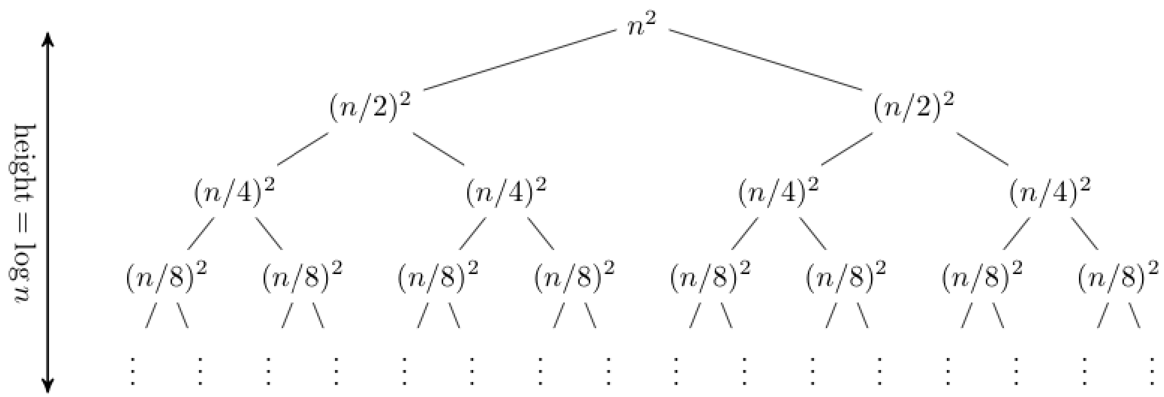
*Total cost =(logn +1) * cn*

= cnlogn + cn

= θ(n log n)

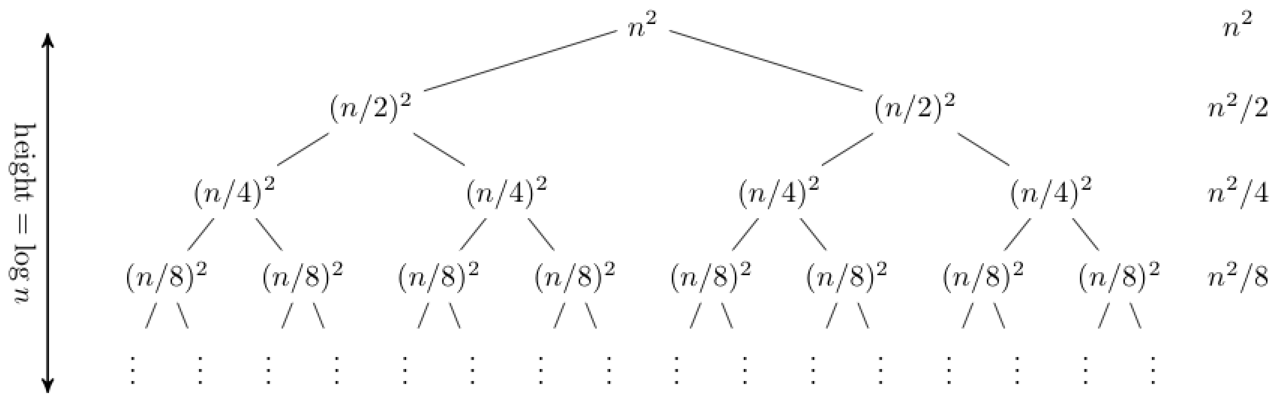
Recursion tree – Example 2

$$T(n) = 2T(n/2) + cn^2$$



Recursion tree – Example 2

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



- Total cost = $n^2 *(1+1/2+(1/2)^2+(1/2)^3 + \ldots\ldots +(1/2)^{\log n})$

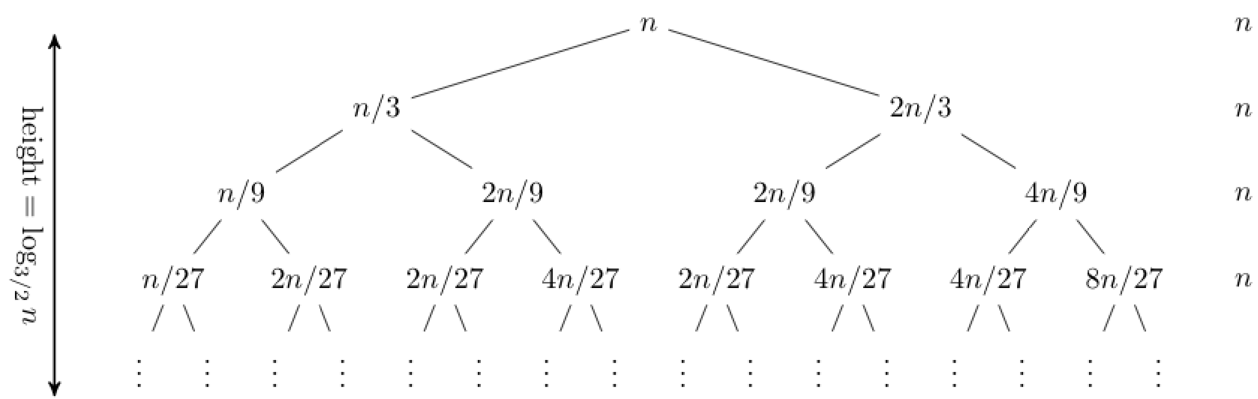
  $= n^2 *(1-(1/2)^{\log n+1})/(1-1/2)$ [using $\sum_{k=0}^{n-1} ar^k = \frac{a(1-r^n)}{1-r}$]

  $= 2n^2$ *[as n become large $(1/2)^{\log n+1}$ will be ~0.]*

  $=\Theta(n^2)$

Recursion tree – Example 3

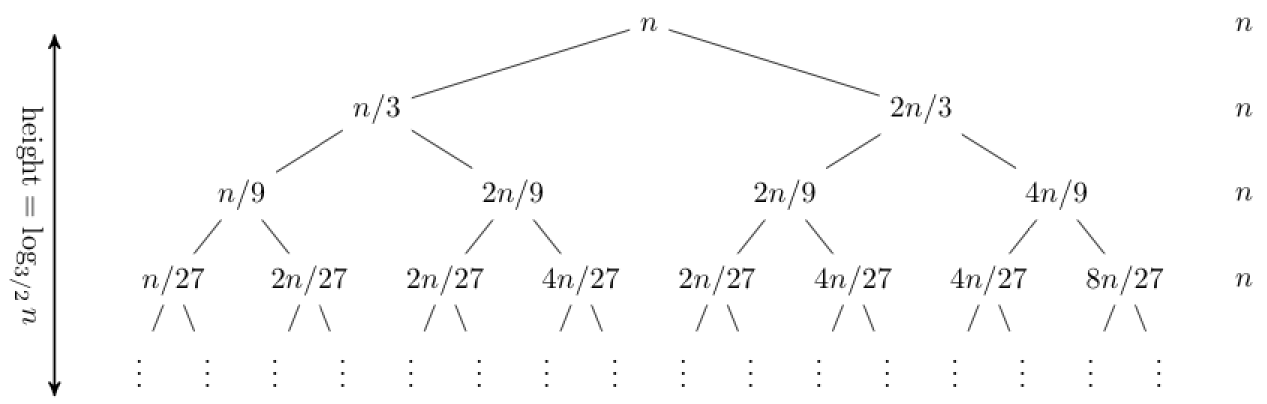
- $$T(n) = T(n/3) + T(2n/3) + n.$$



- As the tree is not balance, to find the height we need to take the subtree that has more levels, in our case $\frac{n}{(\frac{3}{2})^i}$
  - So at base case $\frac{n}{(\frac{3}{2})^i} = 1 \Rightarrow (\frac{3}{2})^i = n \Rightarrow i = log_{3/2}n$

Recursion tree – Example 3

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



- So, total cost = *Total cost* $=(\log_{3/2} n + 1) * n$

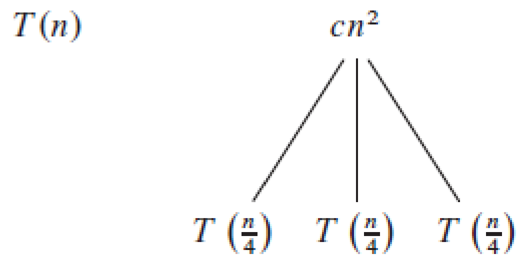
$$= n\log_{3/2} n + n$$

$$= n\log n / \log(3/2) + n$$

$$= \theta(n \log n)$$

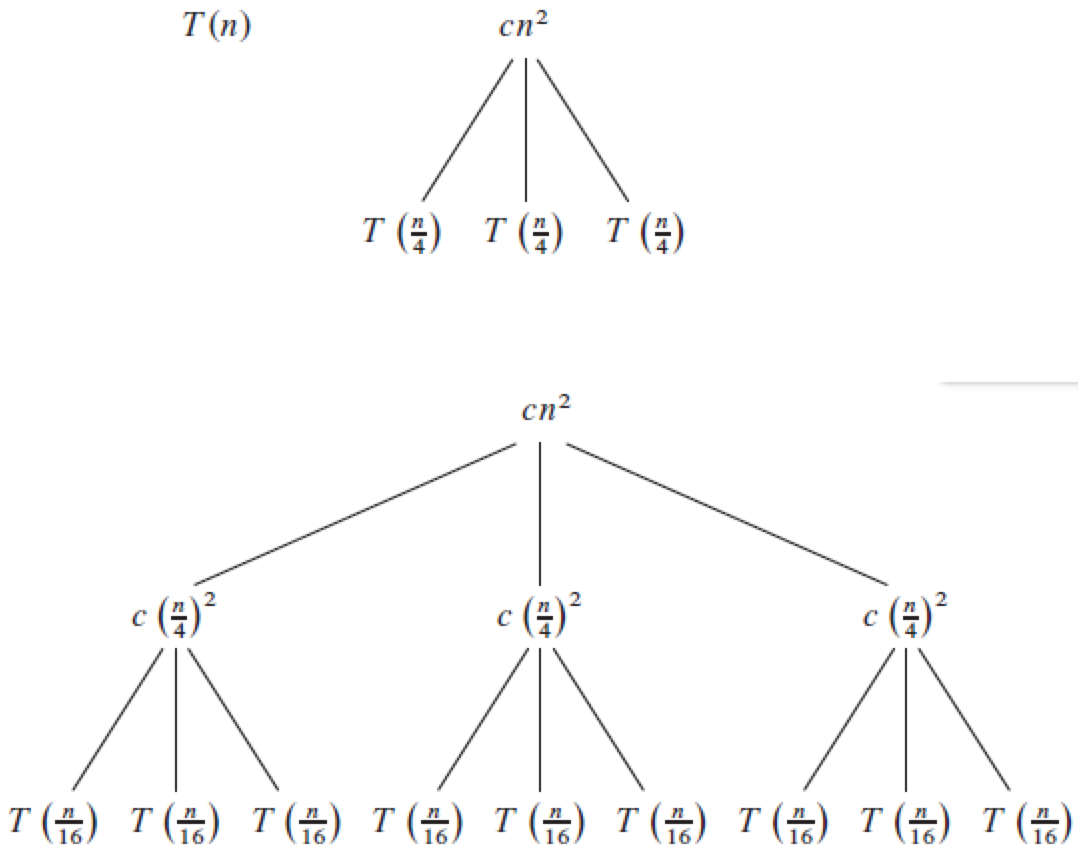
Recursion tree – Example 4

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

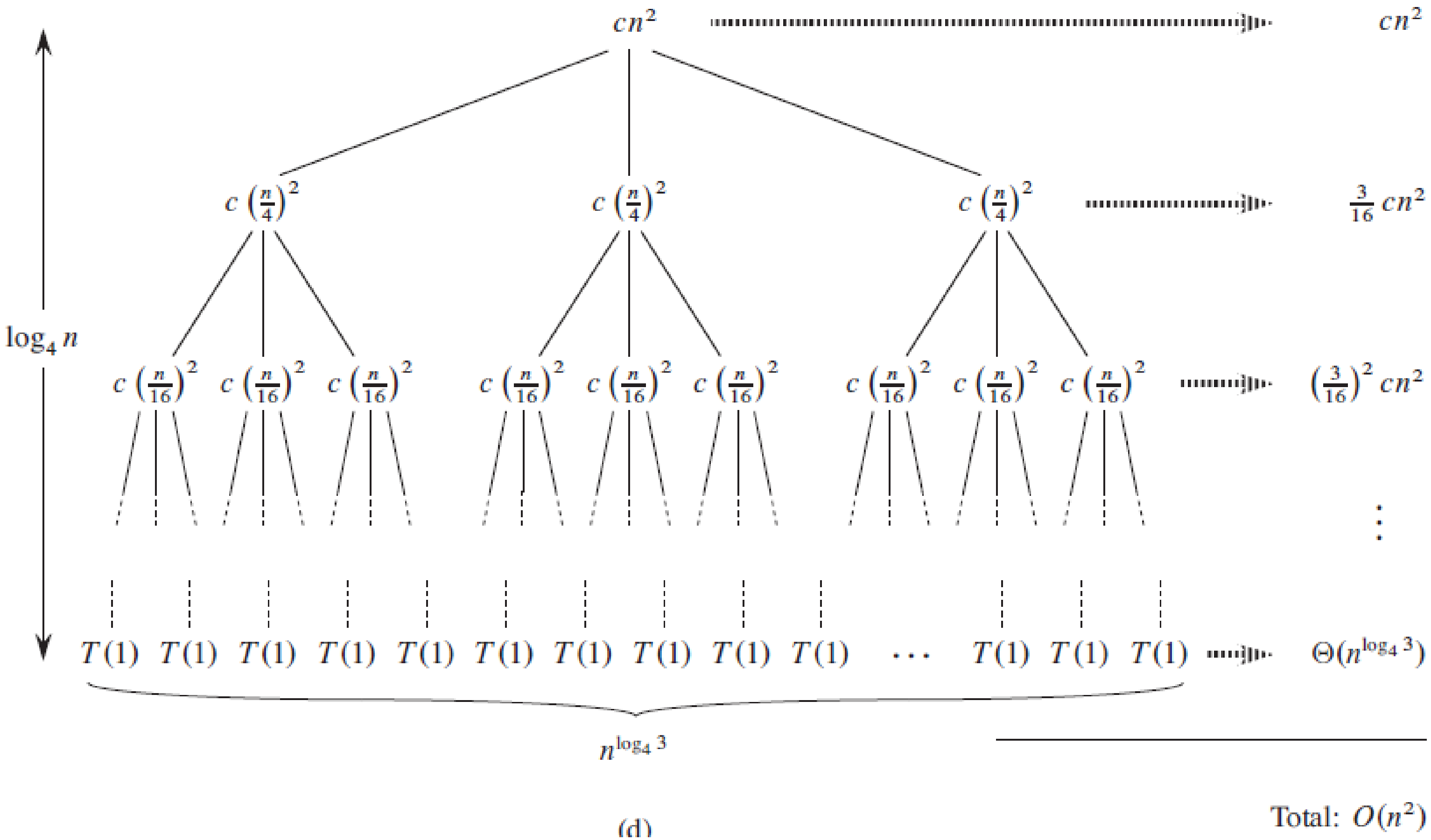


Recursion tree – Example 4 cont…

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



ANOTHER EXAMPLE – CONT…



(d)

Recursion tree – Example 4 cont…

- # of node at each step $= 3^i$
- # of node at last step $= 3^{\log_4 n} \quad as\ i = \log_4 n$

$$= n^{\log_4 3}$$

- As we assumed T(1) is constant
  - Cost at leaf level $= n^{\log_4 3} T(1)$

$$= \Theta(n^{\log_4 3})$$

Recursion tree – Example 4 cont…

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots\ldots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1 - 3/16} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

Recursion tree – Example 4 cont…

- Or solve the following way

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \ldots\ldots + (\frac{3}{16})^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n - 1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1-(3/16)^{\log n}}{1-3/16} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \qquad [as\ \ n \to \infty, (3/16)^{\log n} \to 0] \\
&= \Theta(n^2)
\end{aligned}
$$

Master's method

Master's method

- "Cookbook" for solving recurrences of the form:

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

where, $a \geq 1$, $b > 1$, and $f(n) > 0$

Idea: compare $f(n)$ with $n^{\log_b a}$

- $f(n)$ is asymptotically smaller or larger than $n^{\log_b a}$ by a polynomial factor n^{ε}
- $f(n)$ is asymptotically equal with $n^{\log_b a}$

Master's method

- "Cookbook" for solving recurrences of the form:

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

where, $a \geq 1$, $b > 1$, and $f(n) > 0$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then: $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then: $T(n) = \Theta(n^{\log_b a} \lg n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$, and if

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

then:

regularity condition $T(n) = \Theta(f(n))$

Why $n^{\log_b a}$?

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

$$a^2T\left(\frac{n}{b^2}\right)$$

$$a^3T\left(\frac{n}{b^3}\right)$$

$$\vdots$$

$$T(n) = a^iT\left(\frac{n}{b^i}\right) \quad \forall i$$

- Assume $n = b^k \Rightarrow k = \log_b n$
- At the end of iteration $i = k$:

$$T(n) = a^{\log_b n}T\left(\frac{b^i}{b^i}\right) = a^{\log_b n}T(1) = \Theta\left(a^{\log_b n}\right) = \Theta\left(n^{\log_b a}\right)$$

Master's method – Example 1

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

$a = 2$, $b = 2$, $\log_2 2 = 1$

Compare $g(n) = n^{\log_2 2} = n$ with $f(n) = n$

$\Rightarrow f(n) = \Theta(n) \Rightarrow$ **Case 2**

$\Rightarrow T(n) = \Theta(n \lg n)$

Master's method – Example 2

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

$a = 2$, $b = 2$, $\log_2 2 = 1$

Compare n with $f(n) = n^2$

$\Rightarrow f(n) = \Omega(n^{1+\varepsilon})$ **Case 3** $\Rightarrow$ verify regularity cond.

$a\, f(n/b) \leq c\, f(n)$

$\Leftrightarrow 2\, n^2/4 \leq c\, n^2 \Rightarrow c = ½$ is a solution ($c<1$)

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

Master's method – Example 3

- $$T(n) = 2T(n/2) + \sqrt{n}$$

  a = 2, b = 2, $\log_2 2 = 1$

  Compare n with $f(n) = n^{1/2}$

  $\Rightarrow f(n) = O(n^{1-\varepsilon})$ **Case 1**

  $\Rightarrow T(n) = \Theta(n)$

Reference

- Chapter 4 (Cormen)
- *https://www.cse.unr.edu/~bebis/CS477/Lect/Recurrences.ppt*
- *https://courses.csail.mit.edu/6.046/spring04/lectures/l2.ppt*
- https://www.cs.cornell.edu/courses/cs3110/2012sp/lectures/lec20-master/lec20.html