CENG 218
Design and Analysis of Algorithms

Izmir Institute of Technology

Lecture 4: Solving recurrences

Slides were mostly prepared using the material provided by Prof. Charles E. Leiserson and Prof. Erik Demaine from MIT

Solving recurrences

- The analysis of merge sort required us to solve a recurrence, that is obtaining asymptotic bounds on the solution.

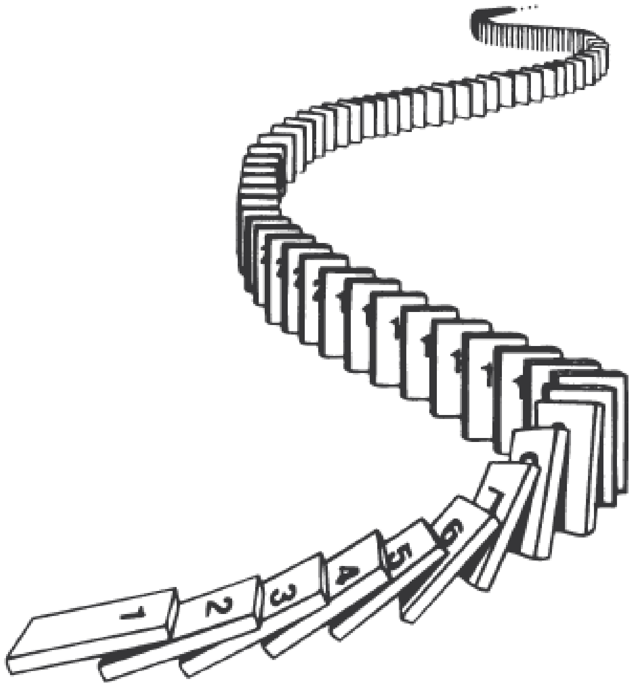
- Recurrences are a major tool for analysis of algorithms. We will learn a few methods:
    - ***Substitution method***: We guess a bound and then use mathematical induction to prove it.
    - ***Recurrence tree method*** converts the recurrence into a tree.
    - ***Master method*** uses a formula.

Reminder on mathematical induction

Mathematical induction is a powerful proof technique based on the fact that
if you show:

1) Base case: $P(1)$ is true.
2) Inductive step:
   $\forall k \geq 1\ (P(k) \rightarrow P(k+1))$ is true.

then, this implies $\forall n\ P(n)$.



Mathematical induction example

Prove that sum of first n positive integers is $n(n+1)/2$

$$\forall n \geq 1 : \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Base case: P(1) is True. For n=1 equality holds.

Inductive step: $\forall k \geq 1$ Assume $P(k)$, and prove $P(k+1)$.

$$\sum_{i=1}^{k+1} i = \sum_{i=1}^{k} i + (k+1) = \frac{k(k+1)}{2} + k + 1 = \frac{(k+1)(k+2)}{2}$$

By inductive hypothesis P(k)

Thus, if P(k) holds, P(k+1) holds.

Substitution method

- It comprises two steps:

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show the solution works.

- We can use substitution method to establish either upper or lower bounds on a recurrence.
- Let us determine an upper bound for $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- This is the recurrence relation for merge sort.

Actually it is $T(\lceil n/2 \rceil)+T(\lfloor n/2 \rfloor)$, but we neglect the difference.

Substitution method

- We guess that the solution is $T(n)=O(n \lg n)$.
- The method requires us to prove $T(n) \leq cn \lg n$.
  Also, for $n/2$, $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg (\lfloor n/2 \rfloor)$ should hold.
- Substituting into recurrence yields

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

$$T(n) \leq 2 (c \lfloor n/2 \rfloor \lg (\lfloor n/2 \rfloor)) + n$$

$$\leq cn \lg (n/2) + n$$

$$= cn \lg n - cn \lg 2 + n$$

$$= cn \lg n - (cn - n) \quad \longleftarrow \textit{desired} - \textit{residual}$$

$$\leq cn \lg n$$

desired ... *residual* ≥ 0 as long as $c \geq 1$

Substitution method

- Mathematical induction now requires us to show our solution holds for the base case.
- We expect that $T(1) = \Theta(1)$ (as it should take a constant time).
- We choose n=1 as the base case.
  However $T(1) \le c\ 1 \lg 1 = 0$. Not good!
- But, we are free to choose the base case (n_0).
  Let us choose n_0=2, then $T(2) \le c\ 2 \lg 2 = 2c$.
  $T(2)$ is a constant time. (Inductive step requires $c \ge 1$)
- Thus, with n_0=2 & $c \ge 1$, we proved $T(n)=O(n \lg n)$

Substitution method: Example 2

- Consider $T(n) = 2T(n/2) + 1$
- We guess $T(n) = O(n)$. So, we try to show $T(n) \leq cn$.

$T(n) \leq 2c(n/2) + 1$

$= cn + 1$

$\not\leq cn$

We are off only by the constant 1.

$O(n)$ is correct but the math did not work out!

IDEA: Strengthen the inductive hypothesis by *subtracting* a low-order term.

Substitution method: Example 2

Let us subtract a constant d and try: $T(n) \le cn - d$.

$T(n) \le 2(c(n/2) - d) + 1$

$= cn - 2d + 1$

$\le \underbrace{cn - d}_{\text{desired}}$ ……… *as long as* $d \ge 1$

What about the base case?

We expect $T(1)=\Theta(1)$ because it takes constant time.

For n=1:

$T(1) \le c\text{-}d$ …choose c larger than d

Thus, with n_0=1, $d\ge 1$ and $c>d$, we proved $T(n)=O(n)$

Substitution method: Example 3

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

Guess $O(n^3)$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^3 + n \\
&= (c/2)n^3 + n \\
&= cn^3 - ((c/2)n^3 - n) \quad \leftarrow \textit{desired} - \textit{residual} \\
&\le cn^3 \quad \leftarrow \textit{desired}
\end{aligned}
$$

whenever $\underbrace{(c/2)n^3 - n}_{\textit{residual}} \ge 0$, e.g. $c \ge 2$ and $n \ge 1$.

Substitution method: Example 3

- Check the base case.
- ***Base:*** $T(n) \le cn^3$ for $n=1$.
- $T(1) = \Theta(1) \le cn^3 = c$, no problem if we pick c big enough.

This bound is not tight!

A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(n) \le cn^2$:

$$\begin{aligned} T(n) &= 4T(n/2)+n \\ &\le 4c(n/2)^2+n \\ &\le cn^2+n \\ &\le cn^2 \end{aligned}$$

true for ***no*** choice of $c > 0$. Lose!

Solution: Strengthen the inductive hypothesis by *subtracting* a low-order term.

A tighter upper bound?

New inductive hypothesis: $T(n) \le c_1 n^2 - c_2 n$

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

$\le 4\,(c_1(n/2)^2 - c_2(n/2)) + n$

$= c_1 n^2 - 2c_2 n + n$

$= c_1 n^2 - c_2 n - (c_2 n - n)$

$\le c_1 n^2 - c_2 n$ … true if $c_2 \ge 1$.

Pick c_1 big enough to handle the base case.

Base: $T(1) \le c_1 - c_2$, any $c_1 > c_2$ can be chosen.

Substitution method: Example 4

Show $T(n) = T(n\text{-}1) + n$ is $\mathbf{\Omega}(n^2)$

This time, inductive hypothesis is $T(n) \geq cn^2$

$$
\begin{aligned}
T(n) &= T(n\text{-}1) + n \\
&\geq c(n\text{-}1)^2 + n \\
&= cn^2 - 2cn + c + n \\
&= cn^2 + n(1\text{-}2c) + c \quad \longleftarrow \textit{desired + residual} \\
&\geq \underset{\uparrow \atop \textit{desired}}{cn^2} \quad \ldots \text{ as long as } n(1\text{-}2c) + c \geq 0
\end{aligned}
$$

holds when $n \geq 0$ and $0 < c \leq 0.5$

Note: Substitution method requires $c > 0$

Base case: $T(1) \geq 0$ is trivial

Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable for exact estimations.
- It promotes intuitions, however.
- Usually we make a guess by recursion tree, then prove by substitution method.

Example of recursion tree

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

Example of recursion tree

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

$$T(n)$$

Example of recursion tree

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

$$n^2$$

$$T(n/4) \qquad\qquad\qquad\qquad T(n/2)$$

Example of recursion tree

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

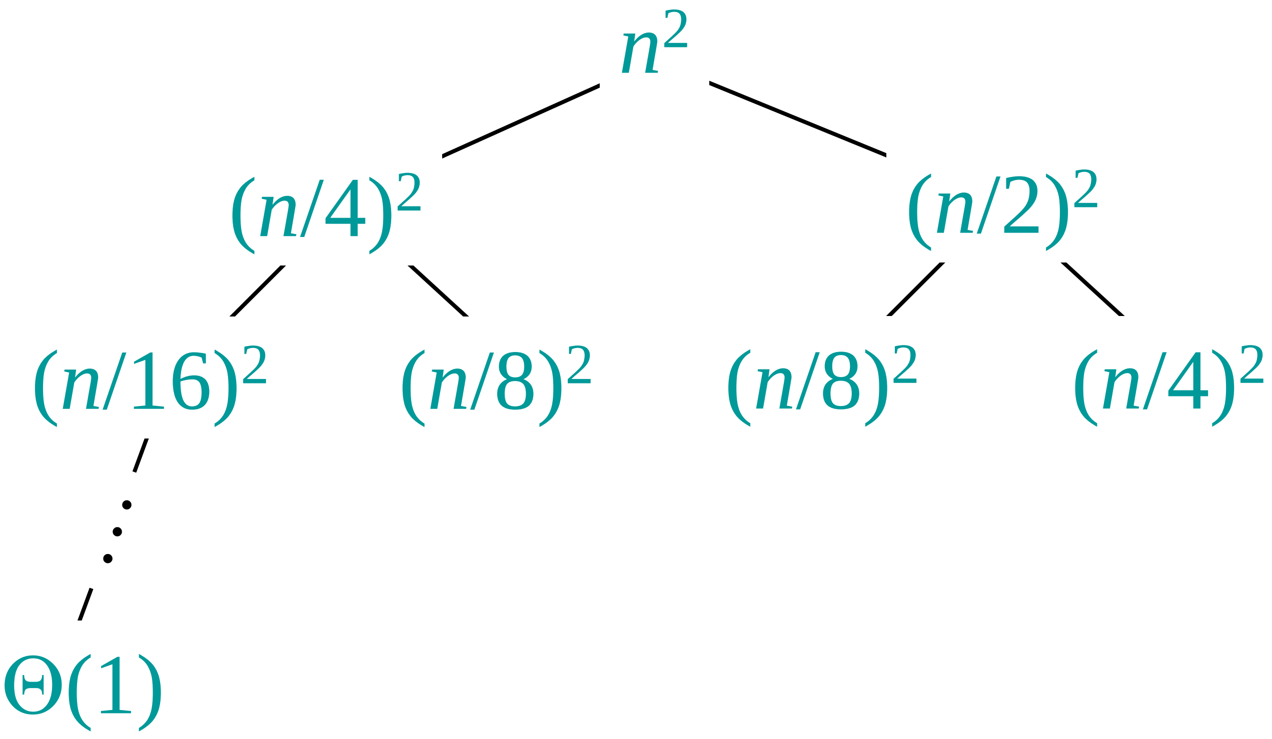
$$n^2$$

$$(n/4)^2 \qquad\qquad (n/2)^2$$

$$T(n/16) \quad T(n/8) \qquad T(n/8) \quad T(n/4)$$

Example of recursion tree

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



Example of recursion tree

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

$$n^2 \quad \cdots\cdots \quad n^2$$

$$(n/4)^2 \qquad\qquad (n/2)^2$$

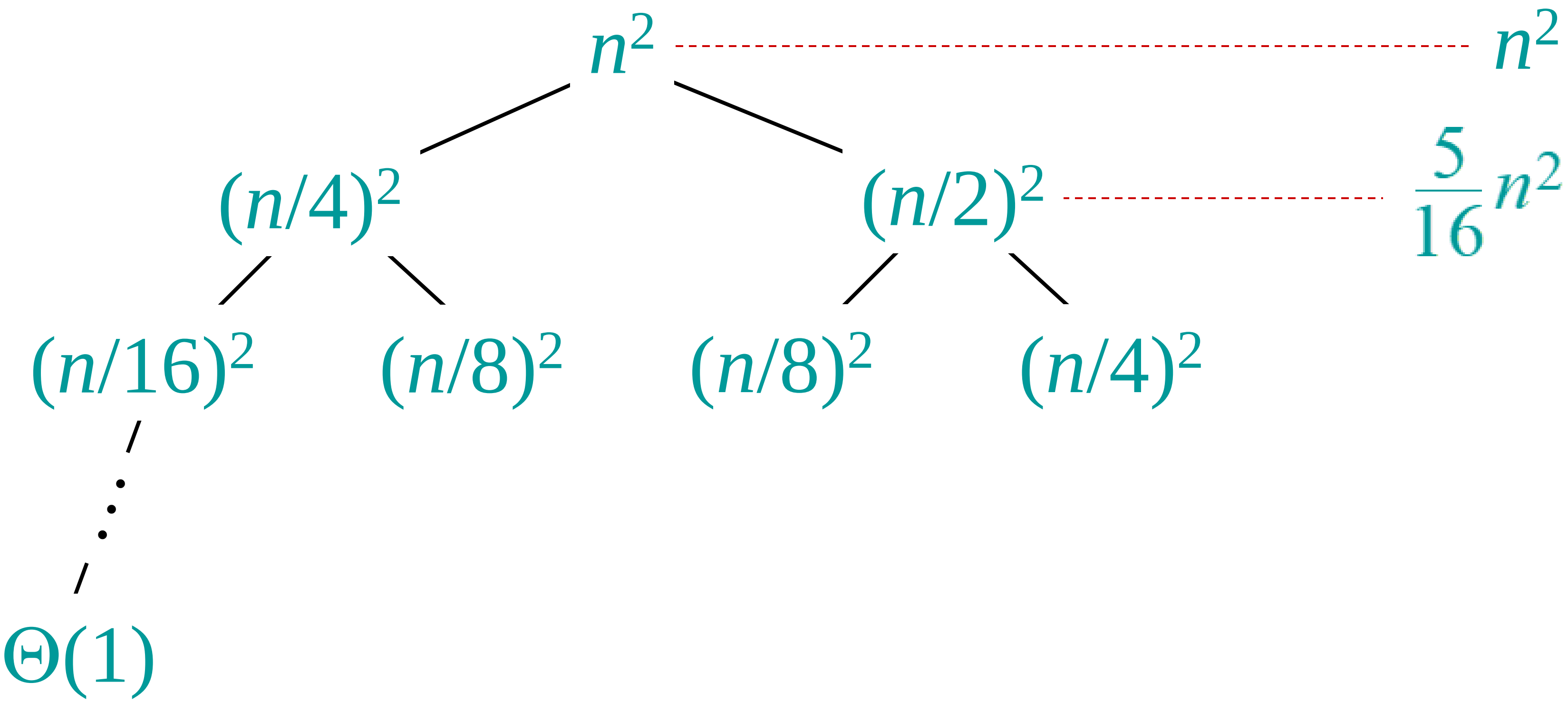
$$(n/16)^2 \qquad (n/8)^2 \qquad (n/8)^2 \qquad (n/4)^2$$

$$\vdots$$

$$\Theta(1)$$

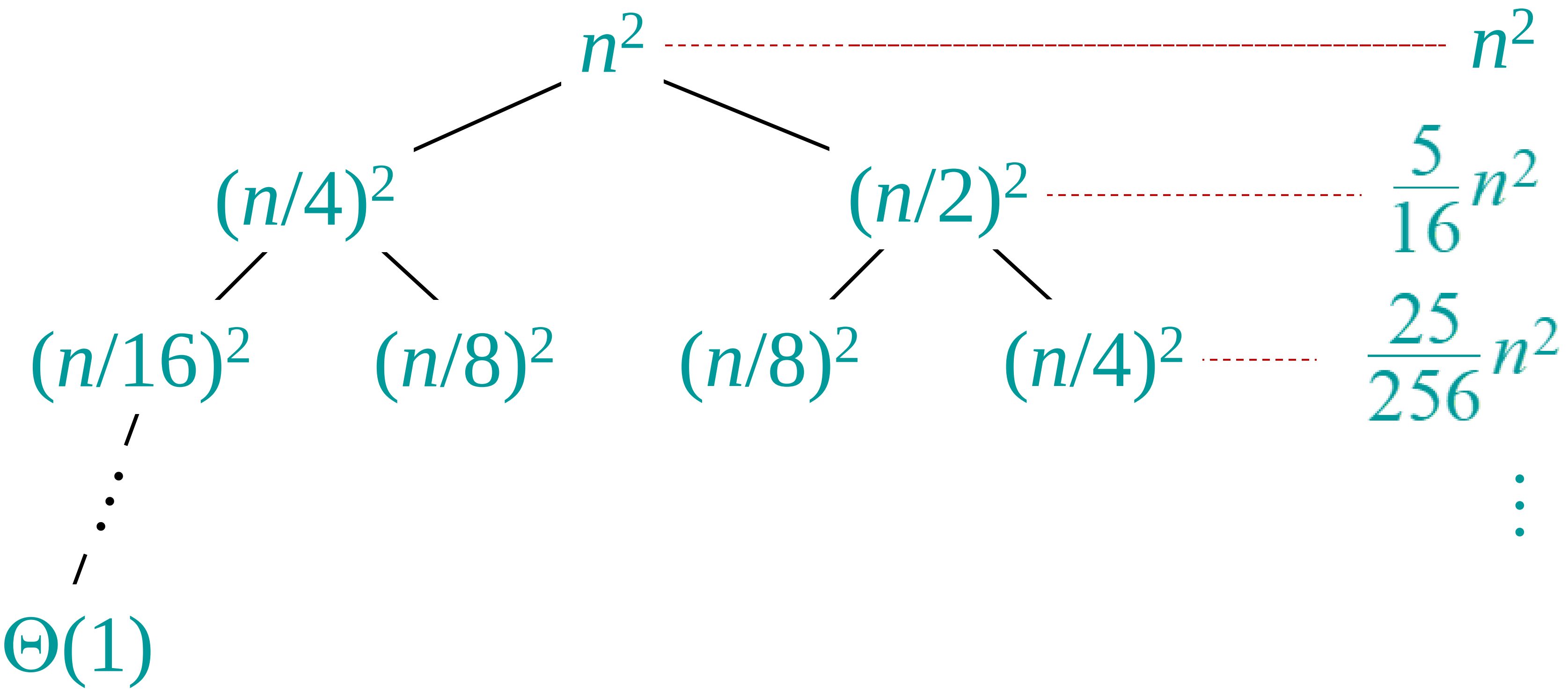
Example of recursion tree

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



Example of recursion tree

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



Example of recursion tree

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

$$n^2 \quad \cdots\cdots \quad n^2$$

$$(n/4)^2 \qquad (n/2)^2 \quad \cdots\cdots \quad \frac{5}{16}n^2$$

$$(n/16)^2 \qquad (n/8)^2 \qquad (n/8)^2 \qquad (n/4)^2 \quad \cdots\cdots \quad \frac{25}{256}n^2$$

$$\vdots \qquad\qquad \vdots$$

$$\Theta(1)$$

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

$$\le n^2\left(\frac{1}{1-\frac{5}{16}}\right) = O(n^2)$$

geometric series

Appendix: geometric series

$$1 + x + x^2 + \ldots + x^n = \frac{1 - x^{n+1}}{1 - x} \quad \text{for } x \neq 1$$

$$1 + x + x^2 + \ldots = \frac{1}{1 - x} \quad \text{for } |x| < 1$$

Recursion tree method as an initial guess to substitution method

Now use substitution method to prove

$T(n) = T(n/4) + T(n/2) + n^2$ is $O(n^2)$

Assume that $T(n) \leq cn^2$:

$$
\begin{aligned}
T(n) &\leq cn^2/16 + cn^2/4 + n^2 \\
&= (5/16)cn^2 + n^2 \\
&= cn^2 - (11/16)cn^2 + n^2 \\
&= cn^2 - ((11/16)c-1)n^2 \\
&\leq cn^2 \qquad \text{..choose } c \geq 16/11
\end{aligned}
$$

Recursion-tree: Example 2

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

Recursion-tree: Example 2

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

$$
\begin{array}{ccccccc}
 & & & cn & & & cn \\
 & c(n/3) & & & c(2n/3) & & cn \\
c(n/9) & & c(2n/9) & & c(2n/9) & c(4n/9) & cn \\
\vdots & & & & & & \vdots \\
\Theta(1) & & & & & &
\end{array}
$$

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

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

Recursion-tree: Example 2

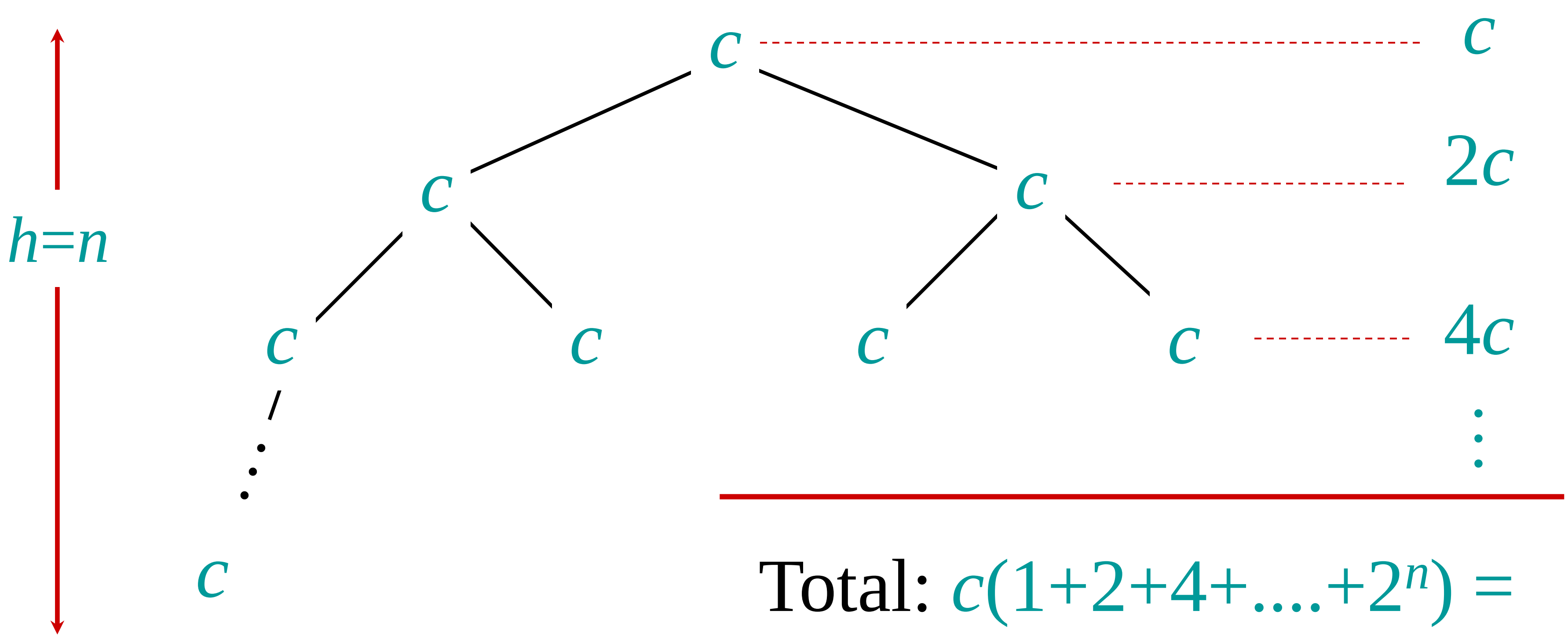
- Our estimate from recursion tree method is $O(n \lg n)$
- We can use it as an initial guess of substitution method.
- Please check Chapter 4.4 (4.2 in 2nd edition) to see the proof by substitution of $T(n) = T(n/3) + T(2n/3) + O(n)$ is $O(n \lg n)$

Recursion-tree: Example 3

Solve $T(n) = 2T(n\text{-}1) + O(1)$

Recursion-tree: Example 3

Solve $T(n) = 2T(n\text{-}1) + O(1)$



Total: $c(1+2+4+....+2^n) =$

$$= c\,\frac{1-2^{n+1}}{1-2} = c(2^{n+1}\text{-}1) = 2c2^n\text{-}c = \Theta(2^n)$$

geometric series

The master method

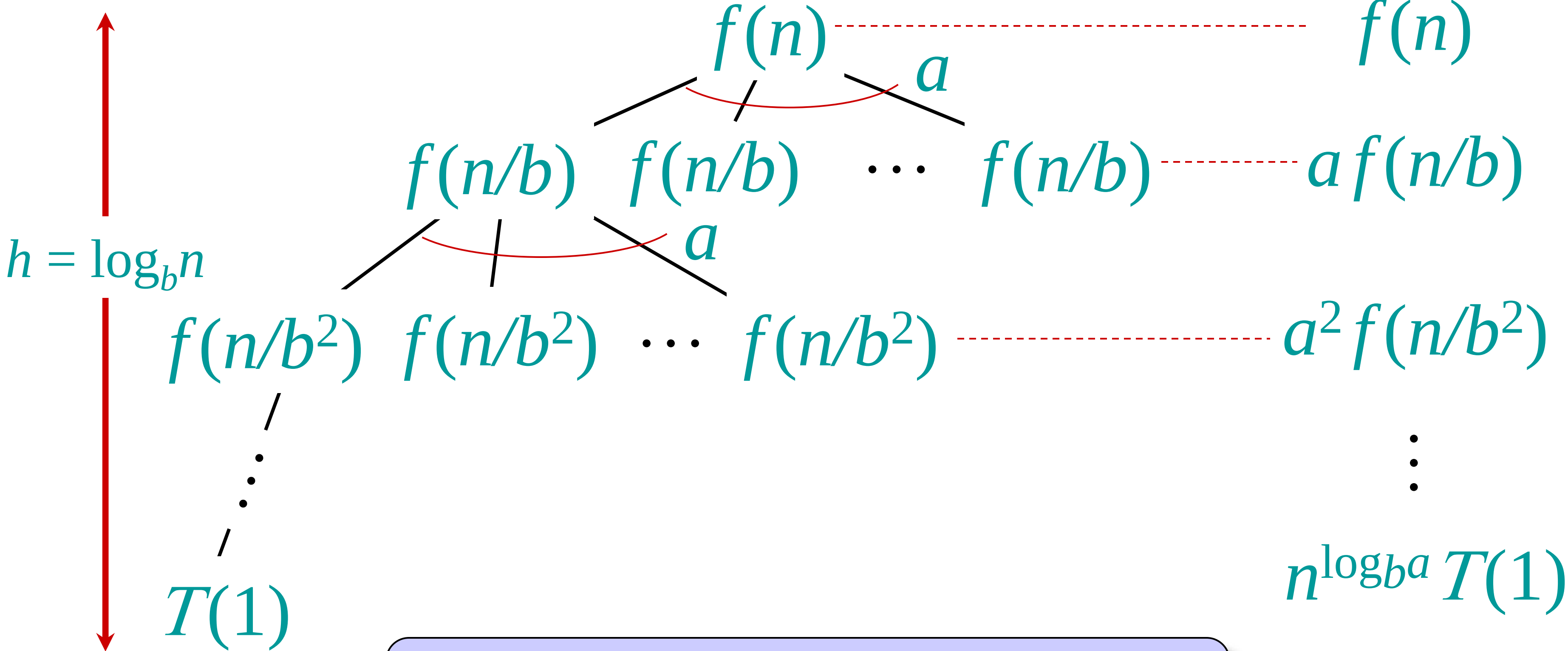
- The third method for solving recurrences is the master method. It is as easy as applying formulas given by ‘master theorem’.
- It applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and $f(n)$ is asymptotically positive.

Idea of master theorem

Recursion tree:



$$\#\text{leaves} = a^h = a^{\log_b n} = n^{\log_b a}$$

Master method: Case 1

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

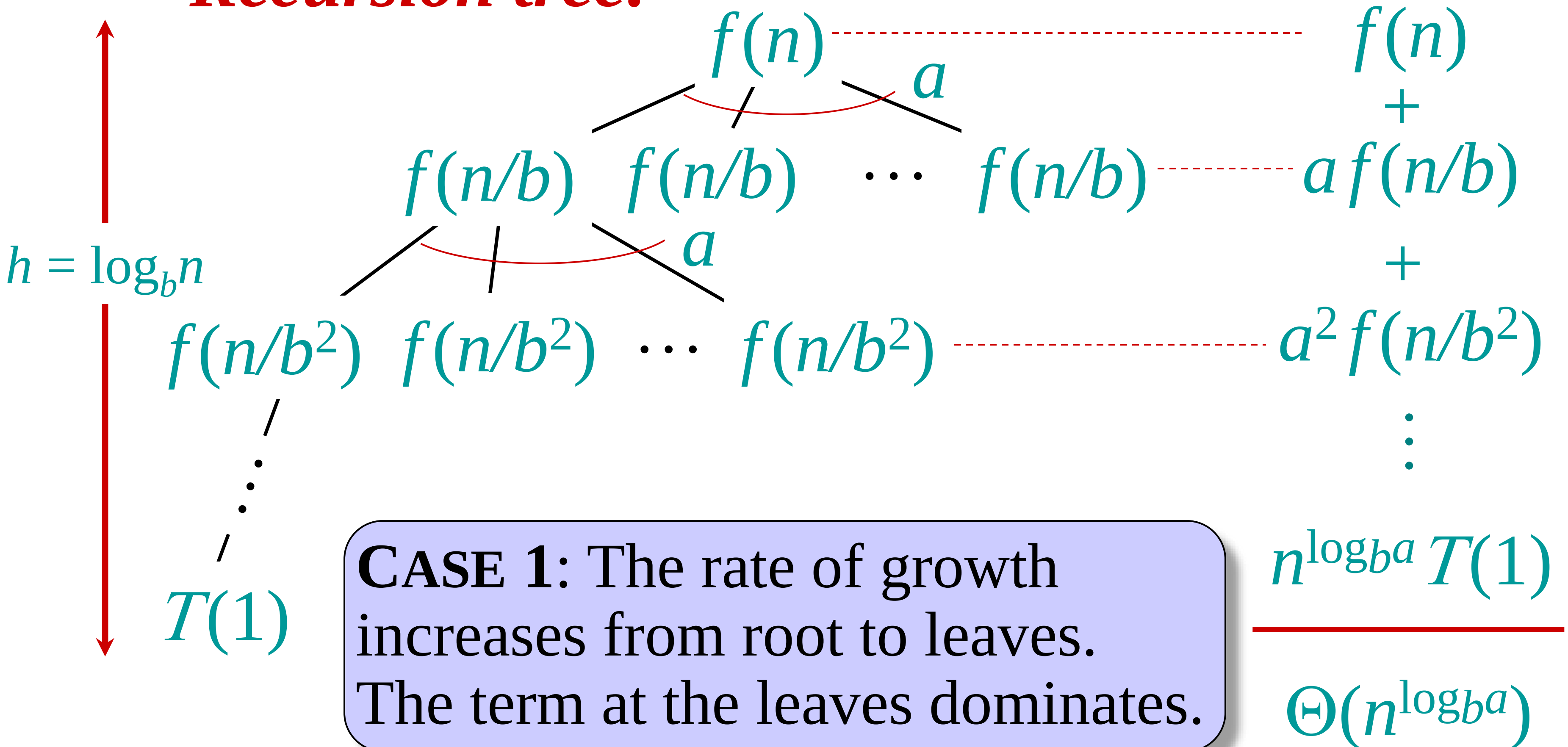
1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

   meaning: $f(n)$ grows polynomially slower than $n^{\log_b a}$.

   Solution: $T(n) = \Theta(n^{\log_b a})$.

Idea of master theorem

Recursion tree:



CASE 1: The rate of growth increases from root to leaves. The term at the leaves dominates.

Master method: Case 2

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

2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.

   meaning: $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Idea of master theorem

Recursion tree:

$f(n)$ ……………… $f(n)$

a

$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$ ……… $a\,f(n/b)$

$h = \log_b n$

a

$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$ ……… $a^2 f(n/b^2)$

$\vdots$

$T(1)$

CASE 2: ($k = 0$) The rate of growth is approximately the same on each of $\log_b n$ levels.

$$f(n) + a\,f(n/b) + a^2 f(n/b^2) + \cdots + n^{\log_b a}\,T(1)$$

$$\Theta(n^{\log_b a} \lg n)$$

Master method: Case 3

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

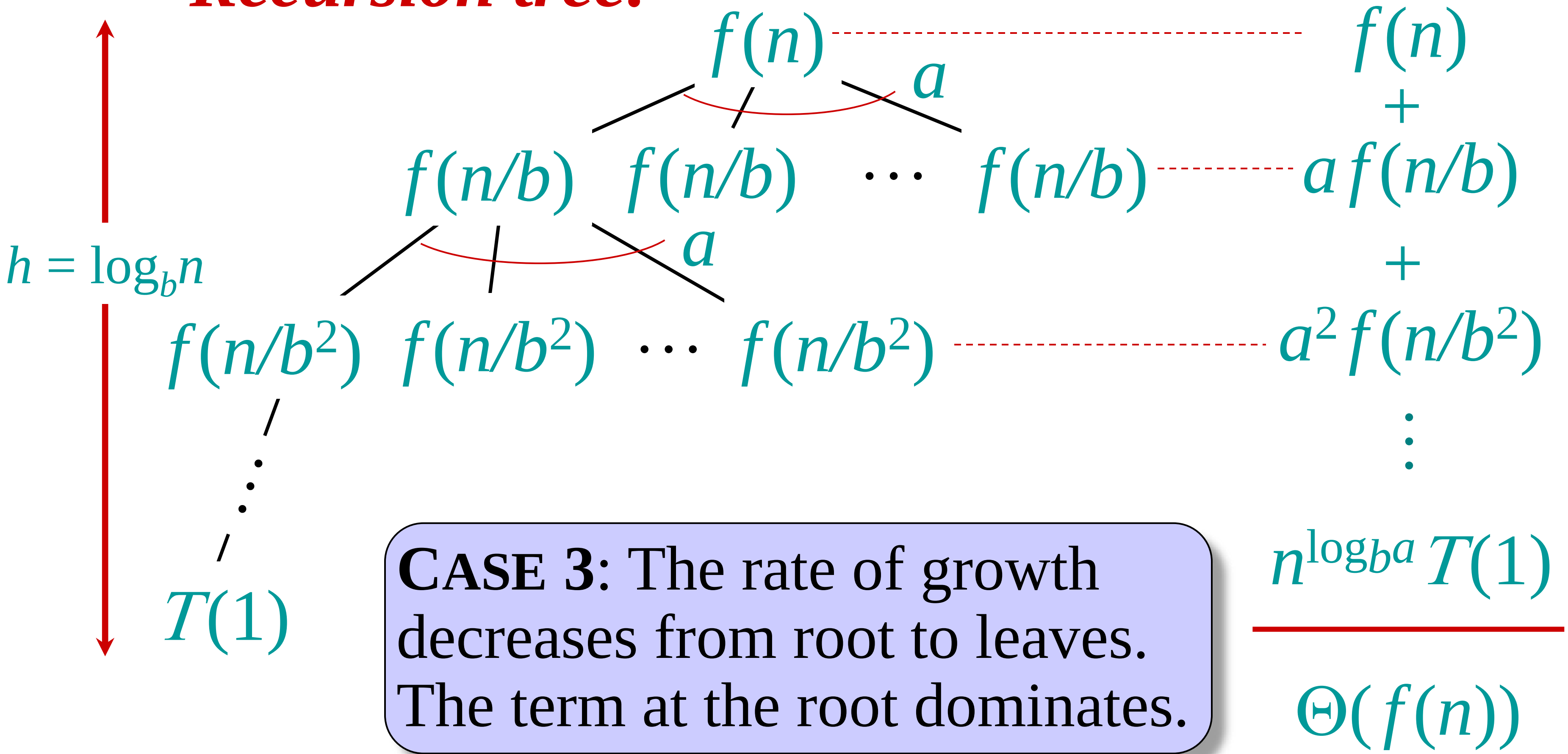
3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.

   meaning: $f(n)$ grows polynomially faster than $n^{\log_b a}$.

   Solution: $T(n) = \Theta(f(n))$.

Idea of master theorem

Recursion tree:



CASE 3: The rate of growth decreases from root to leaves. The term at the root dominates.

Master method examples

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

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n$.

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon}) = O(n^{2-\varepsilon})$ for $\varepsilon=1$.

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

Eg. $T(n) = 4T(n/2) + n^2$

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^2$.

CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^k n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

$\therefore\ T(n) = \Theta(n^2 \lg^{k+1} n) = \Theta(n^2 \lg n)$.

Master method examples

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

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^3$.

CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{2 + \varepsilon})$ for $\varepsilon = 1$.

$\therefore\ T(n) = \Theta(n^3)$.

Master method examples

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

$a = 2, b = 2 \Rightarrow n^{\log_b a} = n;\ f(n) = n \lg n$.

CASE ~~**3**: $f(n)$ is polyn~~omially larger than n.

NO!

$f(n) = n \lg n$ is asymptotically larger than n but not polynomially larger. I.e., $n\lg n \neq \Omega(n^{1+\varepsilon})$.

Actually it is **CASE 2**:

$f(n) = \Theta(n^{\log_b a} \lg^k n) = \Theta(n\lg n)$, that is, $k = 1$.

$\therefore\ T(n) = \Theta(n\lg^{k+1} n) = \Theta(n\lg^2 n)$.

Master method for Binary search

Binary search: Break list into one sub-problem of size $\leq \lceil n/2 \rceil$.

$$T(n) = T(n/2)+c \quad (c = \text{constant time})$$

So, a=1, b=2 $\Rightarrow n^{\log_b a} = 1$; $f(n) = 1$.

CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^k n) = \Theta(\lg^0 n)$, $k = 0$.

$$\therefore\ T(n) = \Theta(\lg^{k+1} n) = \Theta(\lg n).$$

Please also try to apply recursion tree method.

Master method for Merge sort

Merge sort: Break the list into 2 sublists, each of size $\le \lceil n/2 \rceil$, then merge in $\Theta(n)$ time.

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

So, a=2, b=2 $\Rightarrow n^{\log_b a} = n$; $f(n) = \Theta(n)$.

CASE 2: $f(n) = \Theta(n \lg^0 n)$, that is, $k = 0$.

$$\therefore\ T(n) = \Theta(n \lg n).$$

Please also try to apply recursion tree method.

The End

- Next time: Analysis of some divide and conquer algorithms
- Read and try to solve the exercises of
  Chapters 4.3, 4.4, 4.5 (3^{rd} Edition)
  Chapters 4.1, 4.2, 4.3 (2^{nd} Edition)
- Also check problems of Chapter 4.