Analysis and Design of Algorithms

CS3230

Week 2

Recurrence and Master theorem

Ken Sung

Arnab Bhattacharyya

Properties of functions

Properties of functions

- Exponentials
- Logarithms
- Summations
- Limits

Exponentials

$$a^{-1} = 1/a$$

$$(a^m)^n = a^{mn}$$

$$a^m a^n = a^{m+n}$$

$$e^x \geq 1 + x$$

Any exponential function with base $a > 1$ grows faster than any polynomial

- Lemma: For any constants k>0 and a>1, $n^k = o(a^n)$.
- Proof: Need to identify c and n_0 s.t. $n^k < c \cdot a^n$ for $n \geq n_0$.
  - $\frac{n}{\ln n}$ is an increasing function.
  - There exists n_0 such that $\frac{k}{\ln a} < \frac{n}{\ln n}$ for $n \geq n_0$.
  - For $n \geq n_0$, $k \log_a n = \frac{k \ln n}{\ln a} < n$.
  - For $n \geq n_0$, $a^{k \log_a n} < a^n$.
  - For $n \geq n_0$, $n^k < a^n$.
  - Hence, $n^k = o(a^n)$.

Logarithms

- **Binary log:** $\lg n = \log_2 n$
- **Natural log:** $\ln n = \log_e n$
- **Exponentiation:** $\lg^k n = (\lg n)^k$
- **Composition:** $\lg \lg n = \lg(\lg n)$

$$
\begin{aligned}
a &= b^{\log_b a} \\
\log_c(ab) &= \log_c a + \log_c b \\
\log_b a^n &= n \log_b a \\
\log_b a &= \frac{\log_c a}{\log_c b} \\
\log_b(1/a) &= -\log_b a \\
\log_b a &= \frac{1}{\log_a b} \\
a^{\log_b c} &= c^{\log_b a}
\end{aligned}
$$

Base of logarithm does not matter in asymptotics

$$\lg n = \Theta(\ln n) = \Theta(\log_{10} n)$$

Exponentials of different bases differ by an exponential factor

$$4^n = \boxed{2^n} 2^n$$

Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$\log(n!) = \Theta(n \lg n)$$

Summations

Arithmetic Series

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n$$

$$= \frac{1}{2}n(n+1) = \Theta(n^2)$$

Geometric series

$$\sum_{k=1}^{n} x^k = 1 + x + x^2 + \cdots + x^n$$

$$= \frac{x^{n+1} - 1}{x - 1}$$

$$\sum_{k=1}^{\infty} x^k = \frac{1}{1-x} \text{ when } |x| < 1$$

Harmonic series

$$
\begin{aligned}
H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \\
&= \sum_{k=1}^{n} \frac{1}{k} \\
&= \ln n + O(1)
\end{aligned}
$$

Telescoping series

- For any sequence $a_0, a_1, \ldots, a_n$, $\sum_{k=0}^{n-1}(a_k - a_{k+1}) =$

$$
\begin{aligned}
&(a_0 - \cancel{a_1})+\\
&(\cancel{a_1} - \cancel{a_2})+\\
&(\cancel{a_2} - \cancel{a_3})+\\
&(\cancel{a_3} - \cancel{a_4})+\\
&\ldots\\
&(\cancel{a_{n-1}} - a_n)
\end{aligned}
\quad = a_0 - a_n
$$

- Example:

$$
\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} \left(\frac{1}{k} - \frac{1}{k+1} \right)
$$

$$
= 1 - \frac{1}{n}
$$

Limit

- Assume f(n), g(n)>0.

- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0 \rightarrow f(n) = o\big(g(n)\big)$
- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) < \infty \rightarrow f(n) = O\big(g(n)\big)$
- $0 < \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) < \infty \rightarrow f(n) = \Theta\big(g(n)\big)$
- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) > 0 \rightarrow f(n) = \Omega\big(g(n)\big)$
- $\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \infty \rightarrow f(n) = \omega\big(g(n)\big)$

$$\lim_{n \to \infty} \left(\frac{f(n)}{g(n)} \right) = 0 \rightarrow \quad f(n) = o\big(g(n)\big)$$

- Proof:
  - Since $\lim_{n \to \infty} \left(\frac{f(n)}{g(n)} \right) = 0$, by definition, we have:
    - For all ε>0, there exists δ>0 such that $\frac{f(n)}{g(n)} < \varepsilon$ for n> δ.
  - Set c= ε and n_0=δ. We have:
    - For all c>0, there exists n_0>0 such that $\frac{f(n)}{g(n)} < c$ for n> n_0.
    - Hence, for all c>0, there exists n_0>0 such that $f(n) < c \cdot g(n)$ for n> n_0.
    - By definition, $f(n) = o\big(g(n)\big)$.

L'Hopital's Rule

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = \lim_{x \to \infty} \frac{f'(x)}{g'(x)}$$

Example

$$\lim_{n\to\infty} \frac{n \log n}{n^2}$$

$$= \lim_{n\to\infty} \frac{\log n}{n}$$

$$= \lim_{n\to\infty} \frac{1/n}{1}$$

L'Hopital's rule

$$= \lim_{n\to\infty} \frac{1}{n} = 0$$

$$\implies n \log n \in o(n^2)$$

Example

- Question: By limit, show that $n^3 + 3n^2 + 4n + 1 = \omega(n^2)$.

- Proof:
  - $\lim_{n\to\infty}\left(\frac{n^3+3n^2+4n+1}{n^2}\right) = \lim_{n\to\infty}\left(n + 3 + \frac{4}{n} + \frac{1}{n^2}\right) = \infty.$

  - Hence, $n^3 + 3n^2 + 4n + 1 = \omega(n^2)$

Properties of bigO

- **Transitivity**

$$f(n) = \Theta(g(n)) \ \& \ g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \ \& \ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \ \& \ g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = o\,(g(n)) \ \& \ g(n) = o\,(h(n)) \Rightarrow f(n) = o\,(h(n))$$
$$f(n) = \omega(g(n)) \ \& \ g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

- **Reflexivity**

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

Properties of bigO

- **Symmetry**

  $f(n) = \Theta(g(n))$ *iff* $g(n) = \Theta(f(n))$

- **Complementarity**

  $f(n) = O(g(n))$ *iff* $g(n) = \Omega(f(n))$

  $f(n) = o(g(n))$ *iff* $g(n) = \omega((f(n))$

Recurrences and Master Theorem

How to analyze the running time of a recursive algorithm?

1. Derive a recurrence

2. Solve the recurrence

Merge sort

MERGE-SORT $A[1 \,..\, n]$

1. If $n = 1$, done.
2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$.
3. "***Merge***" the 2 sorted lists.

Analyzing merge sort

$T(n)$	**MERGE-SORT** $A[1 \,..\, n]$
$\Theta(1)$	1. If $n = 1$, done.
$2T(n/2)$	2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$.
$\Theta(n)$	3. ***"Merge"*** the 2 sorted lists

Abuse

Sloppiness: Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.

Recurrence for merge sort

$$T(n) = \begin{cases} \Theta(1) & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & if\ n > 1 \end{cases}$$

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.
- Below, we describe a few ways to solve the recurrence to find a good upper bound on $T(n)$.

Solving recurrence

How to solve an recurrence?

- Substitution method
- Telescoping method
- Recursion tree
- Master method

Substitution method

- The most general method:

1. Guess the form of the solution
2. Verify by induction

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

- [Assume T(1)=q where q is a constant.]
- Step 1: Guess $T(n) = O(n^3)$.
  - I.e. there exists a constant c such that $T(n) \leq c \cdot n^3$. for $n \geq n_0$.
- Step 2: Verify by induction.
  - Set $c=\max\{2,q\}$ and $n_0=1$.
  - Base case ($n=n_0=1$): $T(1) = q \leq c(1)^3$.
  - Recursive case ($n>1$):
    - By strong induction, assume $T(k) \leq c \cdot k^3$ for $n > k \geq 1$.
    - $T(n) = 4\ T(n/2) + n \leq 4\ c\ (n/2)^3 + n = (c/2)\ n^3 + n \leq c\ n^3$.
  - Hence, $T(n) \leq c\ n^3$ for $n \geq 1$.
- Conclusion: $T(n) = O(n^3)$.

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

- Is $T(n) = O(n^3)$ a tight bound?

- Answer: No.
- The tight bound is $T(n) = O(n^2)$.

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

- A possible solution to prove that $T(n) = O(n^2)$.
  - i.e. we show that $T(n) \leq c\,n^2$ for $n \geq n_0$.

- Set $c=\max\{2,q\}$ and $n_0=1$.
- Base case ($n=1$): $T(1) = q \leq c(1)^2$.
- Recursive case ($n>1$):
  - By strong induction, assume $T(k)\leq c\cdot k^2$ for $n > k \geq 1$.
  - $T(n) = 4\,T(n/2) + n$
  - $\leq 4\,c\cdot(n/2)^2+n$
  - $= c\,n^2 + n$
  - $= O(n^2)$. ←This is not correct! You need to show $T(n) \leq c\,n^2$!

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

- [Assume $T(1)=q$ where q is a constant.]
- Correct solution: Show that, for $n \geq n_0$, $T(n) \leq c_1\ n^2 - c_2\ n$.
- Set $c_1 = q+1$ and $c_2=1$ and $n_0=1$.
- Base case ($n=1$): $T(1) = q \leq (q+1)\ (1)^2 - (1)(1)$.
- Recursive case ($n>1$):
  - By strong induction, assume $T(k) \leq c_1 \cdot k^2 - c_2 \cdot k$ for $n > k \geq 1$.
  - $T(n) = 4\ T(n/2) + n = 4\ (c_1\ (n/2)^2 - c_2\ (n/2)\ ) + n = c_1\ n^2 - 2\ c_2\ n + n$
    $= c_1\ n^2 - c_2\ n + (1 - c_2)\ n$
  - Since $(1 - c_2) = 0$, $T(n) \leq c_1\ n^2 - c_2\ n$.

Summary for substitution method

- Guess the time complexity and verify that it is correct by induction.

- Sometimes, the verification is a bit tricky.

Telescoping method

Telescoping method

- Example: T(n) = 2 T(n/2) + n   (Recurrence for merge sort)
- This implies: $\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1.$
- By telescoping, we have:

$$\text{Log n}\left\{\begin{array}{l}
\frac{T(n)}{n} = \cancel{\frac{T(n/2)}{n/2}} + 1 \\
\cancel{\frac{T(n/2)}{n/2}} = \cancel{\frac{T(n/4)}{n/4}} + 1 \\
\cancel{\frac{T(n/4)}{n/4}} = \cancel{\frac{T(n/8)}{n/8}} + 1 \\
\qquad \ldots \\
\cancel{\frac{T(2)}{2}} = \frac{T(1)}{1} + 1
\end{array}\right. \Rightarrow \quad \frac{T(n)}{n} = \frac{T(1)}{1} + \log n$$

Hence, T(n) = O(n log n).

Recursion tree

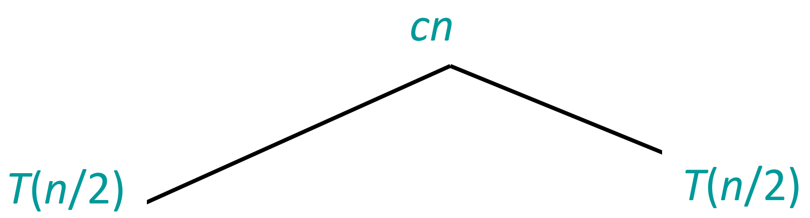
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$$T(n)$$

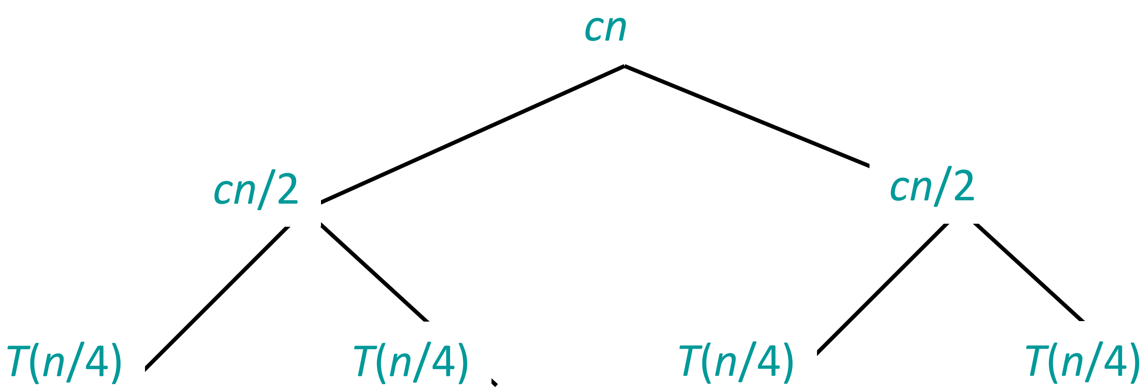
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



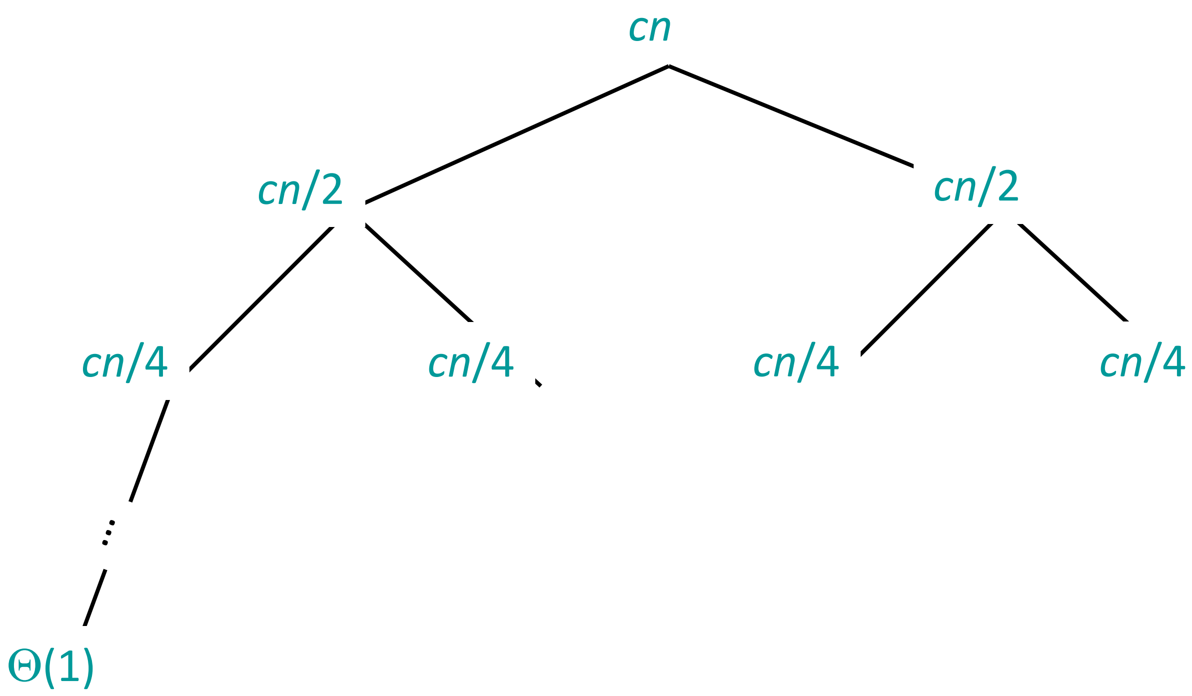
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



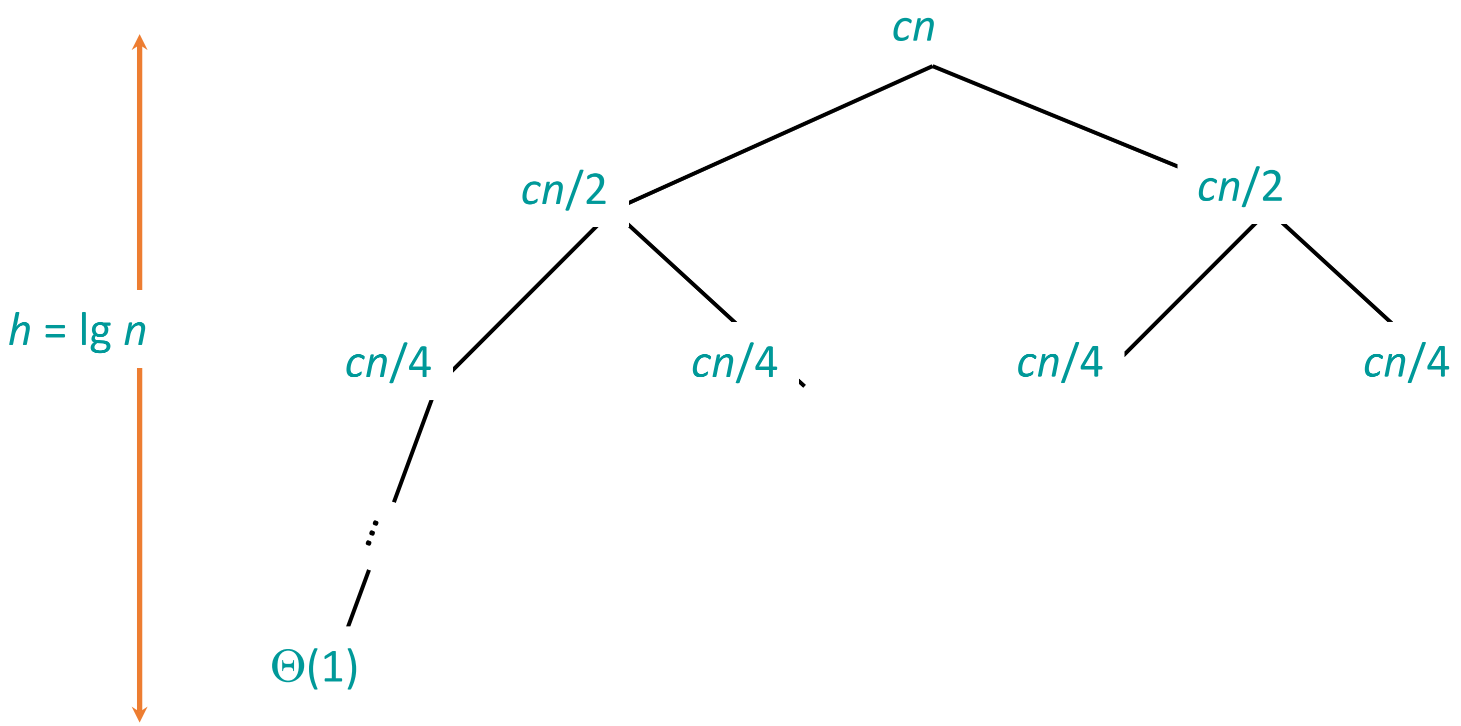
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



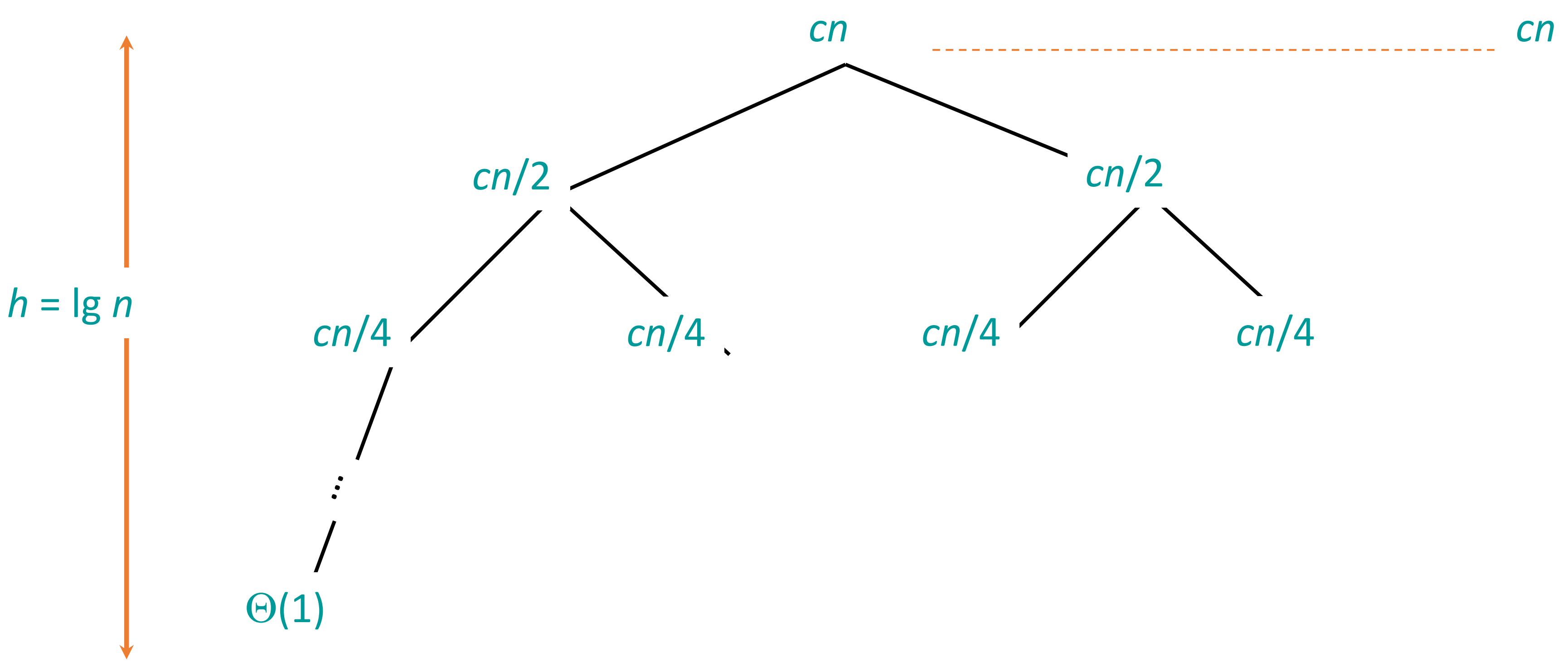
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



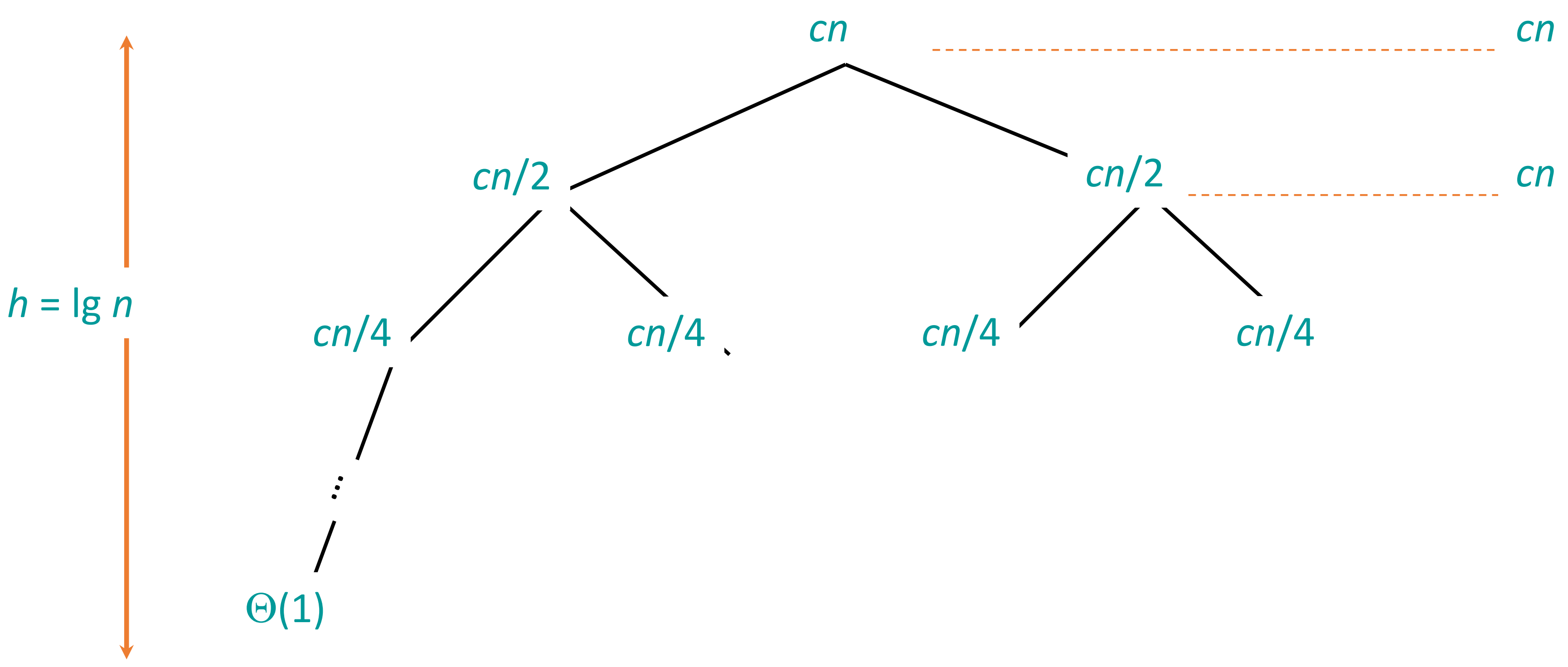
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



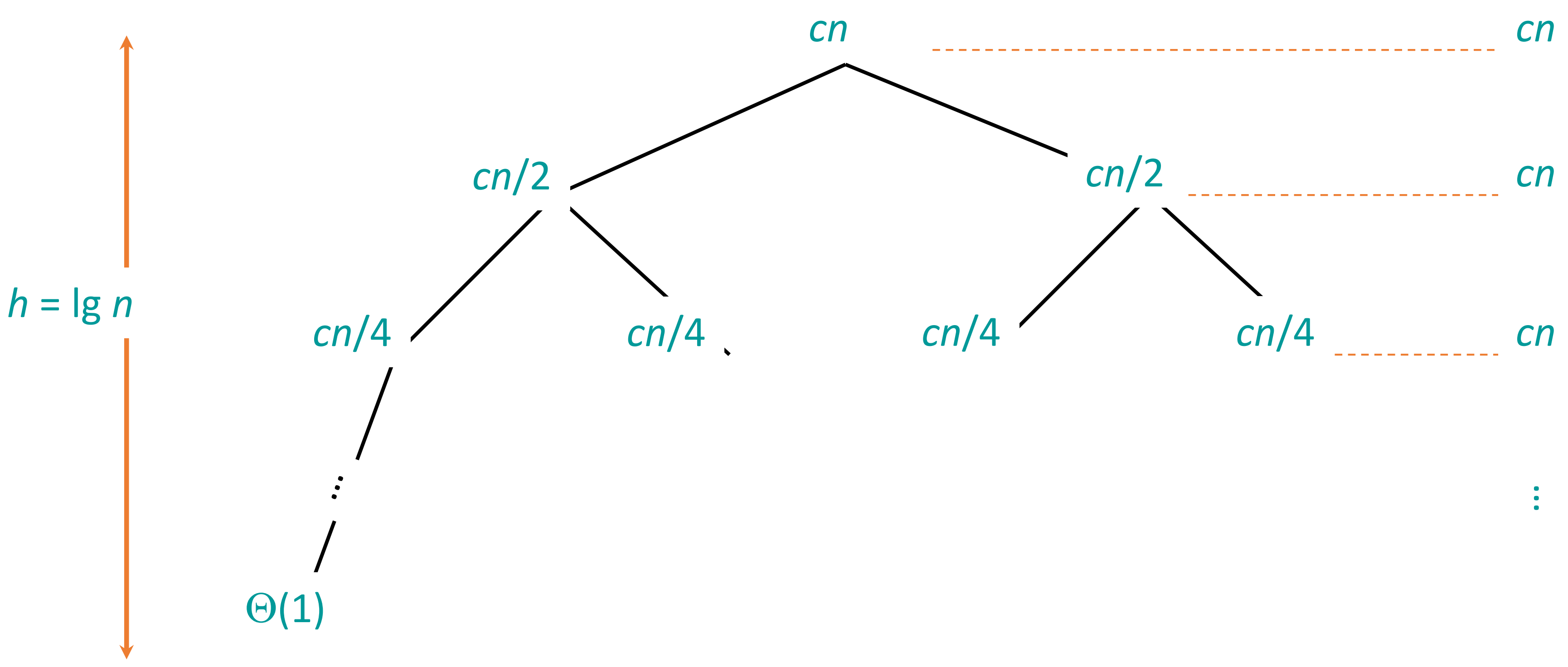
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



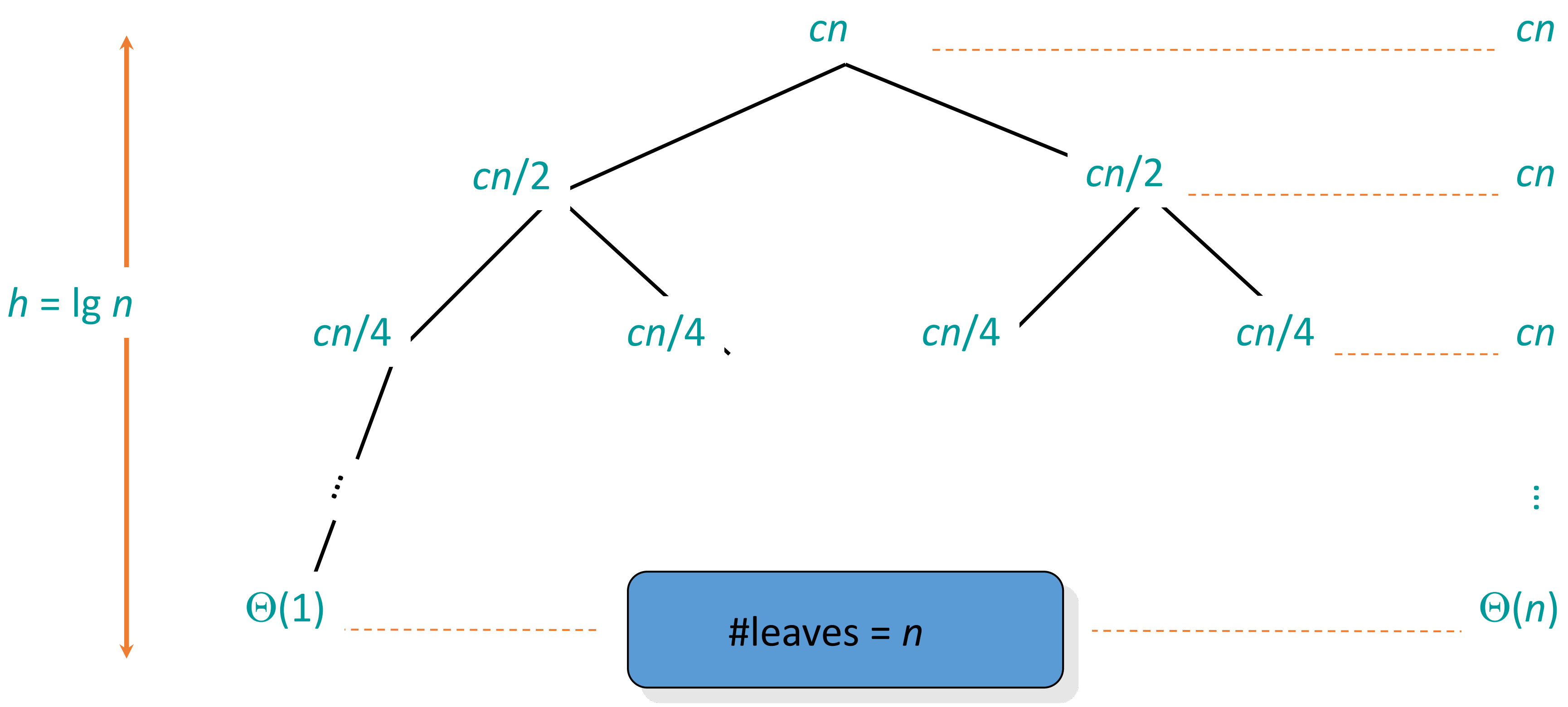
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



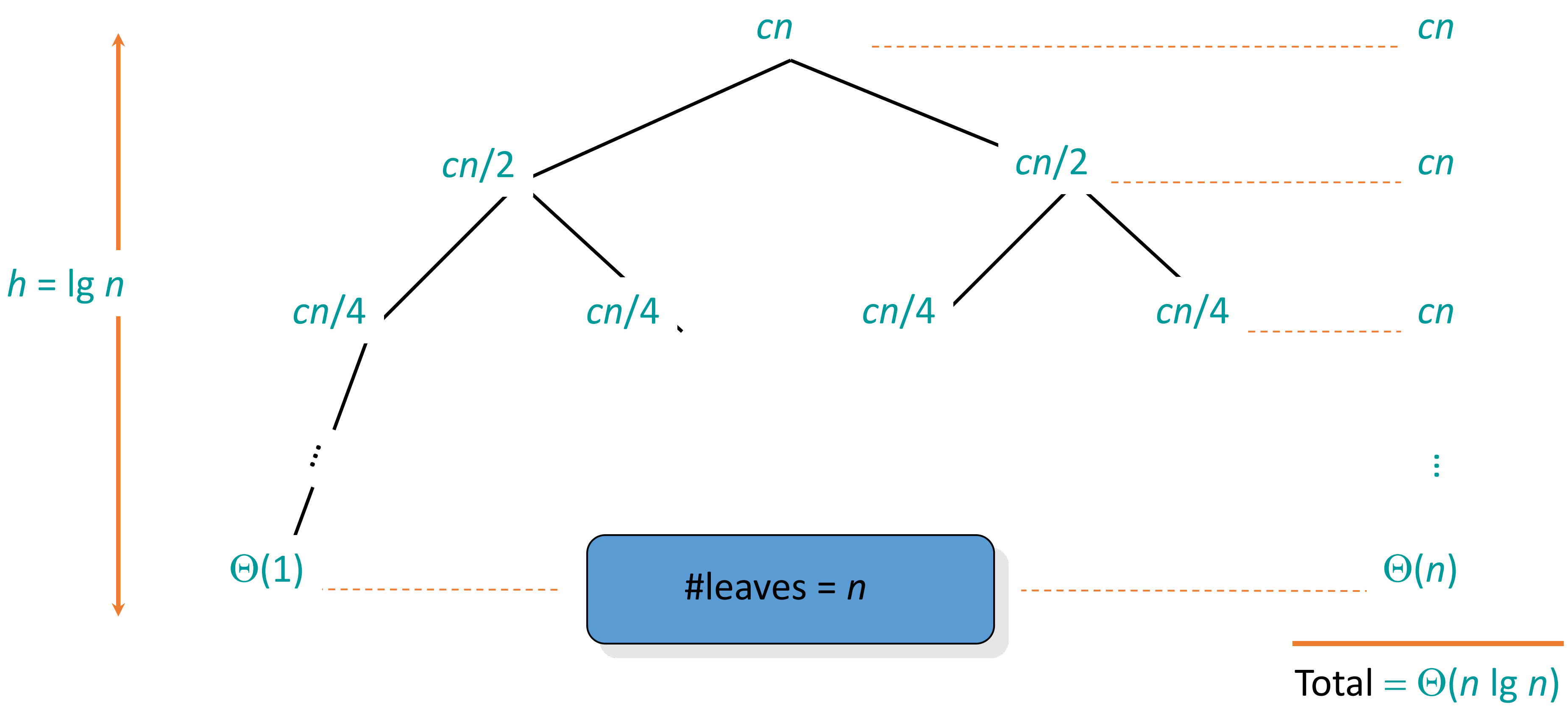
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Master method

The master method

- The master method applies to recurrences of the form
  - $T(n) = a\,T(n/b) + f\,(n)$,

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

Three common cases

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

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

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

Three common cases

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

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

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

2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

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

Three common cases (cont.)

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

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor),

   and $f(n)$ satisfies the ***regularity condition*** that $a f(n/b) \le c f(n)$ for some constant $c < 1$.

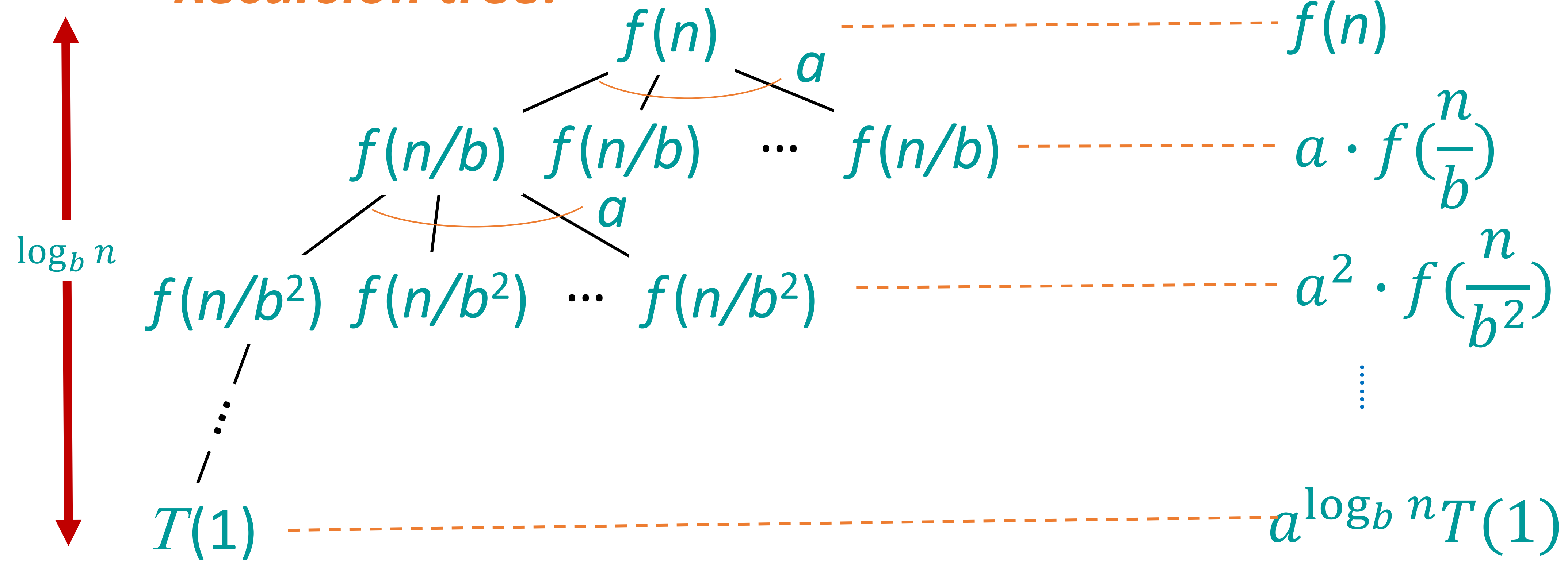
   Solution: $T(n) = \Theta(f(n))$.

The regularity condition guarantees the sum of subproblems is smaller than f(n)

Idea of master theorem

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

Recursion tree:



Note: $a^{\log_b n} = n^{\log_b a}$.

Case 1: $f(n) = O(n^{\log_b a - \varepsilon})$

- Let $\delta = \log_b a - \varepsilon$. $f(n) = O(n^{\log_b a - \varepsilon}) \rightarrow f(n) \leq cn^{\delta}$ for some constant c.

Recursion tree:

$f(n)$

a

$f(n/b)$ $f(n/b)$ … $f(n/b)$

a

$\log_b n$

$f(n/b^2)$ $f(n/b^2)$ … $f(n/b^2)$

⋮

$T(1)$

$f(n) \leq cn^{\delta}$

$a \cdot f\left(\frac{n}{b}\right) \leq ca\left(\frac{n}{b}\right)^{\delta} = cn^{\delta}\left(\frac{a}{b^{\delta}}\right)$

$a^2 f(\frac{n}{b^2}) \leq ca^2\left(\frac{n}{b^2}\right)^{\delta} = cn^{\delta}\left(\frac{a}{b^{\delta}}\right)^2$

⋮

$a^{\log_b n} T(1)$

Case 1: $f(n) = O(n^{\log_b a - \varepsilon})$

- $T(n) \leq cn^{\delta} + cn^{\delta}\left(\frac{a}{b^{\delta}}\right) + cn^{\delta}\left(\frac{a}{b^{\delta}}\right)^2 + \cdots + cn^{\delta}\left(\frac{a}{b^{\delta}}\right)^{\log_b n}$
- $\delta = \log_b a - \varepsilon \rightarrow b^{\delta} = ab^{-\varepsilon} \rightarrow \frac{a}{b^{\delta}} = b^{\varepsilon}$
- Hence,
  - $T(n) \leq cn^{\delta}\left(1 + b^{\varepsilon} + b^{2\varepsilon} + \cdots + b^{\log_b n \cdot \varepsilon}\right)$

  $\rightarrow \mathrm{T(n)} \leq cn^{\delta} b^{(1+\log_b n)\varepsilon} = cb^{\varepsilon} n^{\delta+\varepsilon}$

  $\rightarrow \mathrm{T(n)} \leq cb^{\varepsilon} n^{\log_b a} = O\left(n^{\log_b a}\right)$.

Summary: Master Theorem

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

Case 1:

$$f(n) = O(n^{\log_b a - \epsilon})$$
$$T(n) = \Theta(n^{\log_b a})$$

← If ε=0, it is case 2.

Case 2:

$$f(n) = \Theta(n^{\log_b a} \log^k n)$$
$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

Case 3:

$$f(n) = \Omega(n^{\log_b a + \epsilon})$$
$$af(n/b) \leq cf(n), c < 1$$
$$T(n) = \Theta(f(n))$$

Examples

Ex. $T(n) = 4T(n/2) + n$
$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n.$
CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.
$\therefore\ T(n) = \Theta(n^2)$.

Ex. $T(n) = 4T(n/2) + n^2$
$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2.$
CASE 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.
$\therefore\ T(n) = \Theta(n^2 \lg n)$.

Examples

Ex. $T(n) = 4T(n/2) + n^3$
$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$
Case 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$
and $4(n/2)^3 \leq cn^3$ (reg. cond.) for $c = 1/2$.
$\therefore\ T(n) = \Theta(n^3)$.

Examples

Ex. $T(n) = 4T(n/2) + n^2/\lg n$
$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2/\lg n.$

$n^2/\lg n \notin O(n^{2-\varepsilon})$ → Not case 1
- *Reason:* for every constant $\varepsilon > 0$, we have $n^\varepsilon = \omega(\lg n)$.

$n^2/\lg n \notin \Theta(n^2 \log^k n)$ for any k≥0 → Not case 2
$n^2/\lg n \notin \Omega(n^{2+\varepsilon})$ → Not case 3
Master method does not apply.

Acknowledgement

- The slides are modified from
  - the slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
  - the slides from Prof. Leong Hon Wai
  - the slides from Prof. Lee Wee Sun