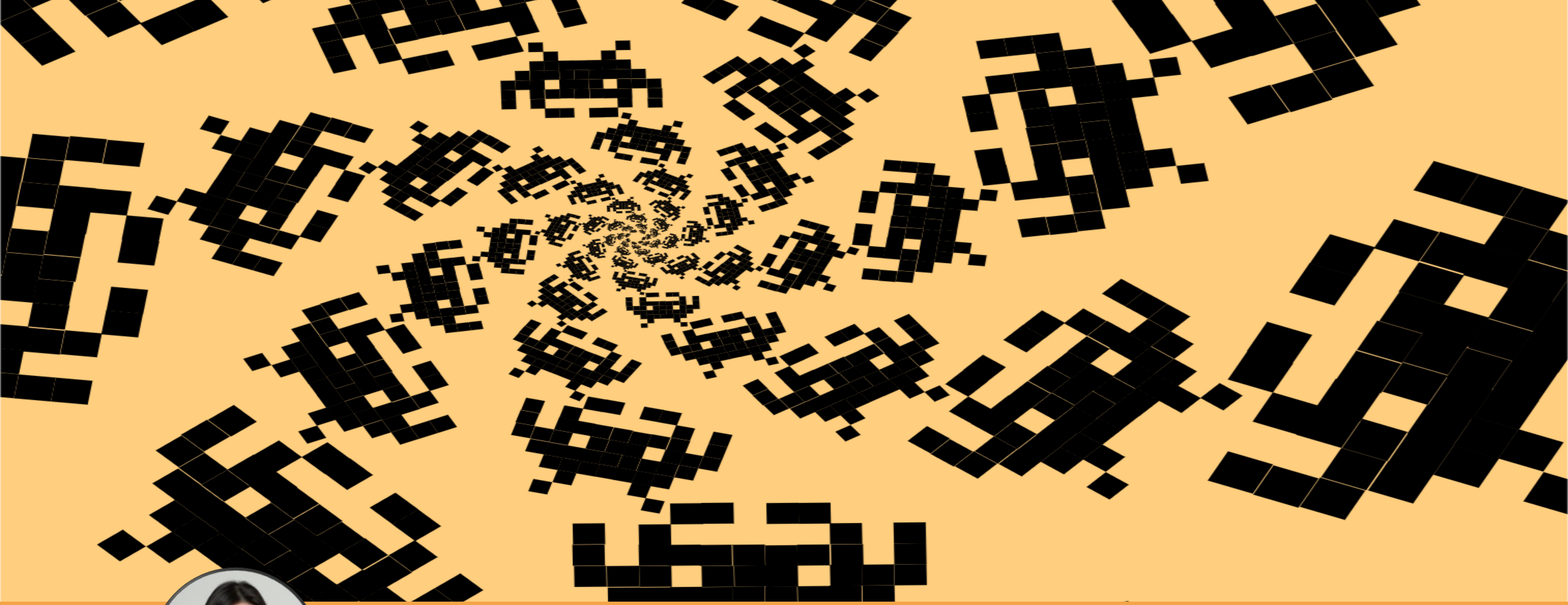


Algorithm Design and Analysis
Divide and Conquer (2)

http://ada.miulab.tw slido: #ADA2022

Yun-Nung (Vivian) Chen

國立臺灣大學
National Taiwan University

Outline

- Recurrence (遞迴)
- Divide-and-Conquer
- D&C #1: Tower of Hanoi (河內塔)
- D&C #2: Merge Sort
- D&C #3: Bitonic Champion
- D&C #4: Maximum Subarray
- Solving Recurrences
  - Substitution Method
  - Recursion-Tree Method
  - Master Method
- D&C #5: Matrix Multiplication
- D&C #6: Selection Problem
- D&C #7: Closest Pair of Points Problem

Divide-and-Conquer 首部曲

Divide-and-Conquer
之神乎奇技

What is Divide-and-Conquer?

- Solve a problem recursively
- Apply three steps at each level of the recursion

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem (比較小的同樣問題)
2. **Conquer** the subproblems by solving them recursively
   If the subproblem sizes are *small enough*
   - then solve the subproblems base case
   - else recursively solve itself recursive case
3. **Combine** the solutions to the subproblems into the solution for the original problem

Solving Recurrences

Textbook Chapter 4.3 – The substitution method for solving recurrences

Textbook Chapter 4.4 – The recursion-tree method for solving recurrences

Textbook Chapter 4.5 – The master method for solving recurrences

D&C Algorithm Time Complexity

- $T(n)$: running time for input size n
- $D(n)$: time of **Divide** for input size n
- $C(n)$: time of **Combine** for input size n
- a: number of subproblems
- n/b: size of each subproblem

$$T(n) = \begin{cases} O(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Solving Recurrences

1. **Substitution Method** (取代法)
    - Guess a bound and then prove by induction
2. **Recursion-Tree Method** (遞迴樹法)
    - Expand the recurrence into a tree and sum up the cost
3. **Master Method** (套公式大法/大師法)
    - Apply Master Theorem to a specific form of recurrences

- Useful simplification tricks
    - Ignore floors, ceilings, boundary conditions (proof in Ch. 4.6)
    - Assume base cases are constant (for small *n*)

Substitution Method

Textbook Chapter 4.3 – The substitution method for solving recurrences

Review

- Time Complexity for Merge Sort
- Theorem

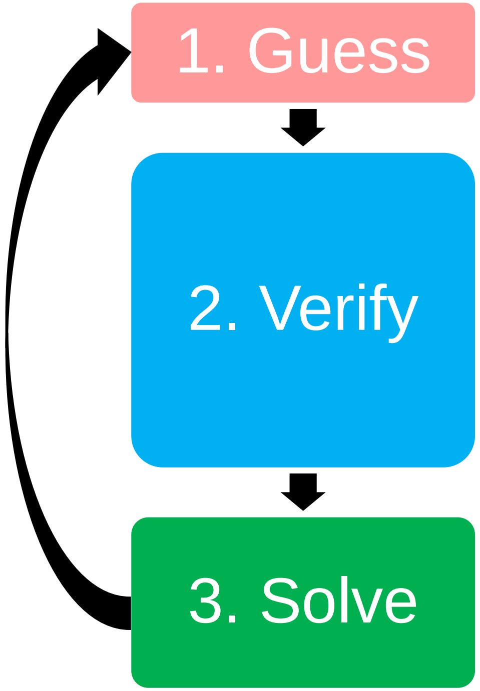
$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases} \Rightarrow T(n) = O(n \log n)$$

- Proof
  - There exists positive constant a, b s.t. $T(n) \leq \begin{cases} a & \text{if } n = 1 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$
  - Use induction to prove $T(n) \leq b \cdot n \log n + a \cdot n$
    - n = 1, trivial
    - n > 1, $T(n)$

$$\begin{aligned} T(n) &\leq 2T(n/2) + bn \\ &\leq 2[b \cdot \frac{n}{2} \log \frac{n}{2} + a \cdot \frac{n}{2}] + b \cdot n \\ &= b \cdot n \log n - b \cdot n + a \cdot n + b \cdot n \\ &= b \cdot n \log n + a \cdot n \end{aligned}$$

Substitution Method (取代法)
guess a bound and then prove by induction

Substitution Method (取代法)



- Guess the form of the solution
- Verify by mathematical induction (數學歸納法)
  - Prove it works for $n = 1$
  - Prove that if it works for $n = m$, then it works for $n = m + 1$

→ It can work for all positive integer n

- Solve constants to show that the solution works
- Prove O and Ω separately

Substitution Method Example

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

- Proof
  - $T(n) = O(n^3)$
    There exists positive constants n_0, c s.t. for all $n \geq n_0$, $T(n) \leq cn^3$

    Guess
  - Use induction to find the constants n_0, c
    - n = 1, trivial
    - n > 1,

$$\begin{aligned} T(n) &\leq 4T(n/2) + bn \\ &\leq 4c(n/2)^3 + bn \\ &= cn^3/2 + bn \\ &= cn^3 - (cn^3/2 - bn) \\ &\leq cn^3 \end{aligned}$$

Inductive hypothesis

$cn^3/2 - bn \geq 0$

e.g. $c \geq 2b, n \geq 1$

Verify

  - $T(n) \leq cn^3$ holds when $c = 2b, n_0 = 1$

Solve

Substitution Method Example

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

Tighter upper bound?

- Proof
  - $T(n) = O(n^2)$
    There exists positive constants n_0, c s.t. for all $n \geq n_0$, $T(n) \leq cn^2$
  - Use induction to find the constants n_0, c
    - n = 1, trivial
    - n > 1, $T(n) \leq 4T(n/2) + bn$

Inductive hypothesis

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

証不出來…
猜錯了？還是推導錯了？

沒猜錯 推導也沒錯
這是取代法的小盲點

Substitution Method Example

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

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

- Proof
  - $T(n) = O(n^2)$
    There exists positive constants n_0, c_1, c_2 s.t. for all $n \geq n_0, T(n) \leq c_1 n^2 - c_2 n$
  - Use induction to find the constants n_0, c_1, c_2
    - n = 1, $T(1) \leq c_1 - c_2$ holds for $c_1 \geq c_2 + 1$
    - n > 1, $T(n) \leq 4T(n/2) + bn$

Inductive hypothesis

$$\begin{aligned} T(n) &\leq 4T(n/2) + bn \\ &\leq 4[c_1(n/2)^2 - c_2(n/2)] + bn \\ &= c_1 n^2 - 2c_2 n + bn \\ &= c_1 n^2 - c_2 n - (c_2 n - bn) \\ &\leq c_1 n^2 - c_2 n \end{aligned}$$

$c_2 n - bn \geq 0$

e.g. $c_2 \geq b, n \geq 0$

- $T(n) \leq c_1 n^2 - c_2 n$ holds when $c_1 = b + 1, c_2 = b, n_0 = 0$

Guess

Verify

Solve

Useful Tricks

- Guess based on seen recurrences
- Use the recursion-tree method
- From loose bound to tight bound
- Strengthen the inductive hypothesis by subtracting a low-order term
- Change variables
  - E.g., $T(n) = 2T(\sqrt{n}) + \log n$

1. Change variable: $k = \log n, n = 2^k \rightarrow T(2^k) = 2T(2^{k/2}) + k$
2. Change variable again: $S(k) = T(2^k) \rightarrow S(k) = 2S(k/2) + k$
3. Solve recurrence $S(k) = \Theta(k \log k) \rightarrow T(2^k) = \Theta(k \log k) \rightarrow T(n) = \Theta(\log n \log \log n)$

Recursion-Tree Method

Textbook Chapter 4.4 – The recursion-tree method for solving recurrences

Review

- Time Complexity for Merge Sort
- Theorem

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases} \Rightarrow T(n) = O(n \log n)$$

- Proof

Recursion-Tree Method (遞迴樹法)
Expand the recurrence into a tree and sum up the cost

$$\begin{aligned} T(n) &\leq 2T(\frac{n}{2}) + cn \\ &\leq 2[2T(\frac{n}{4}) + c\frac{n}{2}] + cn = 4T(\frac{n}{4}) + 2cn \\ &\leq 4[2T(\frac{n}{8}) + c\frac{n}{4}] + 2cn = 8T(\frac{n}{8}) + 3cn \\ &\vdots \\ &\leq 2^k T(\frac{n}{2^k}) + kcn \end{aligned}$$

1st expansion

2nd expansion

kth expansion

The expansion stops when $2^k = n$

$$\begin{aligned} T(n) &\leq nT(1) + cn \log_2 n \\ &= O(n) + O(n \log n) \\ &= O(n \log n) \end{aligned}$$

Recursion-Tree Method (遞迴樹法)

1. Expand
2. Sumup
3. Verify

- Expand a recurrence into a tree
- Sum up the cost of all nodes as a good guess
- Verify the guess as in the substitution method

- Advantages
  - Promote intuition
  - Generate good guesses for the substitution method

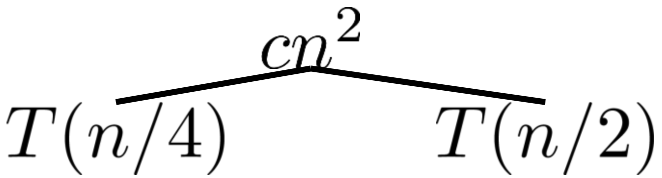
Recursion-Tree Example

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

$$T(n)$$

Recursion-Tree Example

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

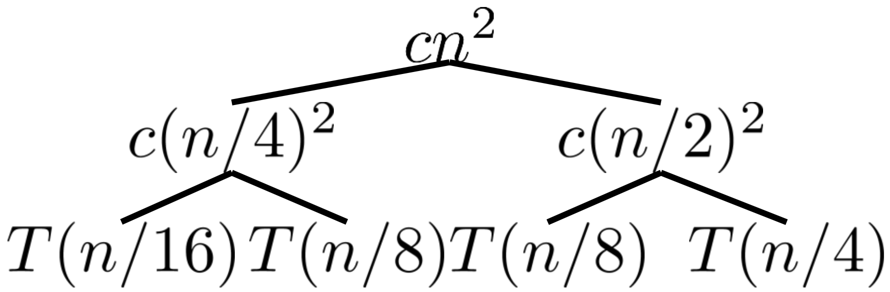


cn^2

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

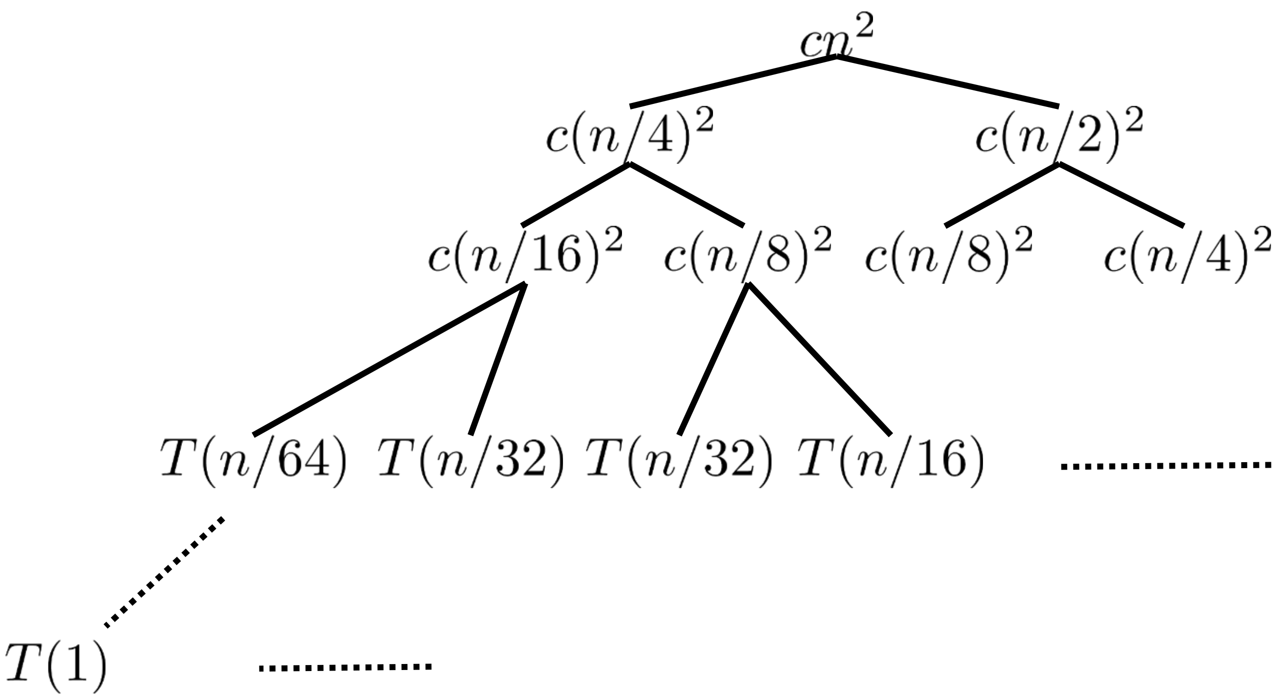
Recursion-Tree Example

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



Recursion-Tree Example

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



cn^2

$\frac{5}{16}cn^2$

$(\frac{5}{16})^2cn^2$

$(\frac{5}{16})^3cn^2$

⋮

+

$$T(n) \leq (1 + \frac{5}{16} + (\frac{5}{16})^2 + (\frac{5}{16})^3 + \cdots)cn^2 = \frac{1}{1-\frac{5}{16}}cn^2 = \frac{16}{11}cn^2 = O(n^2)$$

Master Theorem

Textbook Chapter 4.4 – The recursion-tree method for solving recurrences

Master Theorem

The proof is in Ch. 4.6

divide a problem of size n into a subproblems, each of size $\frac{n}{b}$ is solved in time $T\left(\frac{n}{b}\right)$ recursively

---

Let $T(n)$ be a positive function satisfying the following recurrence relation

$$\boxed{T(n) = \begin{cases} O(1) & \text{if } n \le 1 \\ a \cdot T(\frac{n}{b}) + f(n) & \text{if } n > 1, \end{cases}}$$

Should follow this format

where $a \ge 1$ and $b > 1$ are constants.

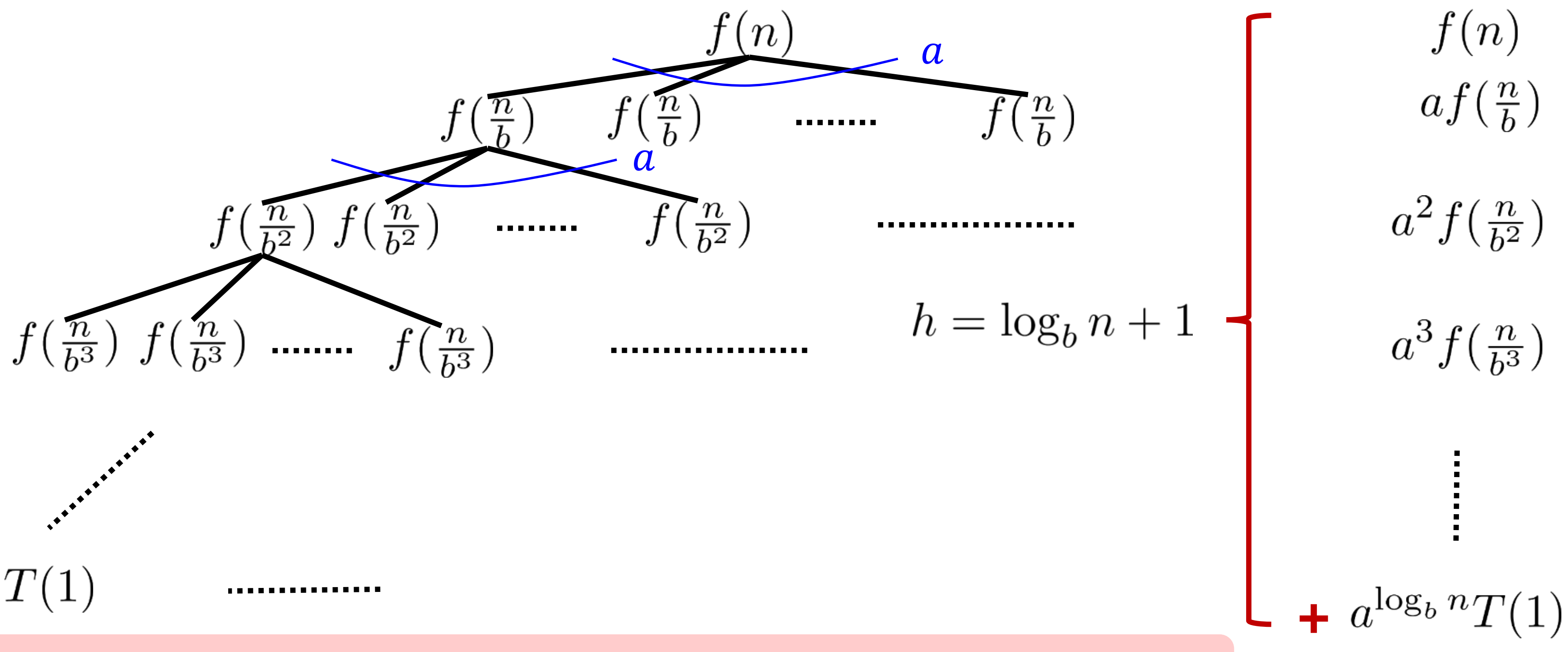
- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \le c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

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

Recursion-Tree for Master Theorem

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



$$T(n) = f(n) + af(\frac{n}{b}) + a^2 f(\frac{n}{b^2}) + a^3 f(\frac{n}{b^3}) + \cdots + a^{\log_b n}T(1)$$

$$a^{\log_b n} = n^{\log_n a \log_b n} = n^{\log_b a} \quad \log_n a = \frac{\log_b a}{\log_b n}$$

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

Three Cases

- $T(n) = aT(\frac{n}{b}) + f(n)$
  - $a \geq 1$, the number of subproblems
  - $b > 1$, the factor by which the subproblem size decreases
  - $f(n)$ = work to divide/combine subproblems

$$T(n) = f(n) + af(\frac{n}{b}) + a^2 f(\frac{n}{b^2}) + a^3 f(\frac{n}{b^3}) + \cdots + n^{\log_b a} T(1)$$

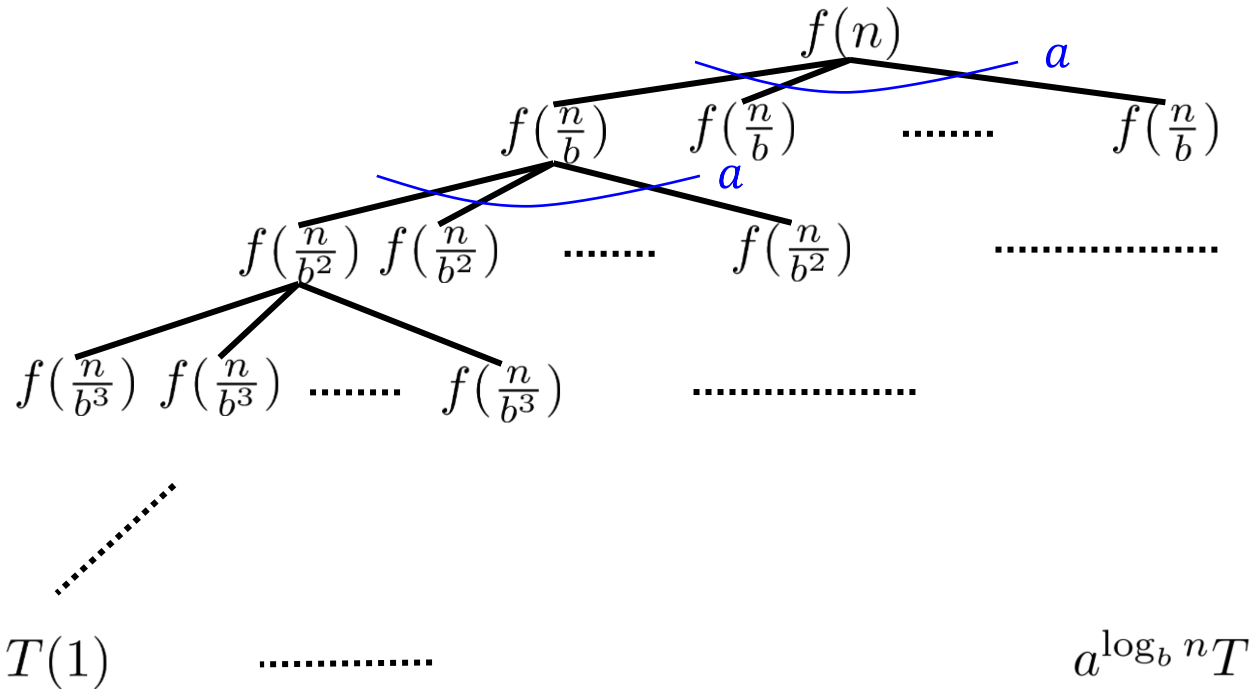
- Compare $f(n)$ with $n^{\log_b a}$

1. Case 1: $f(n)$ grows polynomially slower than $n^{\log_b a}$
2. Case 2: $f(n)$ and $n^{\log_b a}$ grow at similar rates
3. Case 3: $f(n)$ grows polynomially faster than $n^{\log_b a}$

Case 1:

Total cost dominated by the leaves

$$T(n) = 9T(\frac{n}{3}) + n, T(1) = 1$$



$$f(n) = n$$

$$af(\frac{n}{b}) = \frac{9}{3}n$$

$$a^2f(\frac{n}{b^2}) = (\frac{9}{3})^2n$$

$$a^3f(\frac{n}{b^3}) = (\frac{9}{3})^3n$$

$$a^{\log_b n}T(1) = 9^{\log_3 n} = (\frac{9}{3})^{\log_3 n}n$$

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

Case 1:

Total cost dominated by the leaves

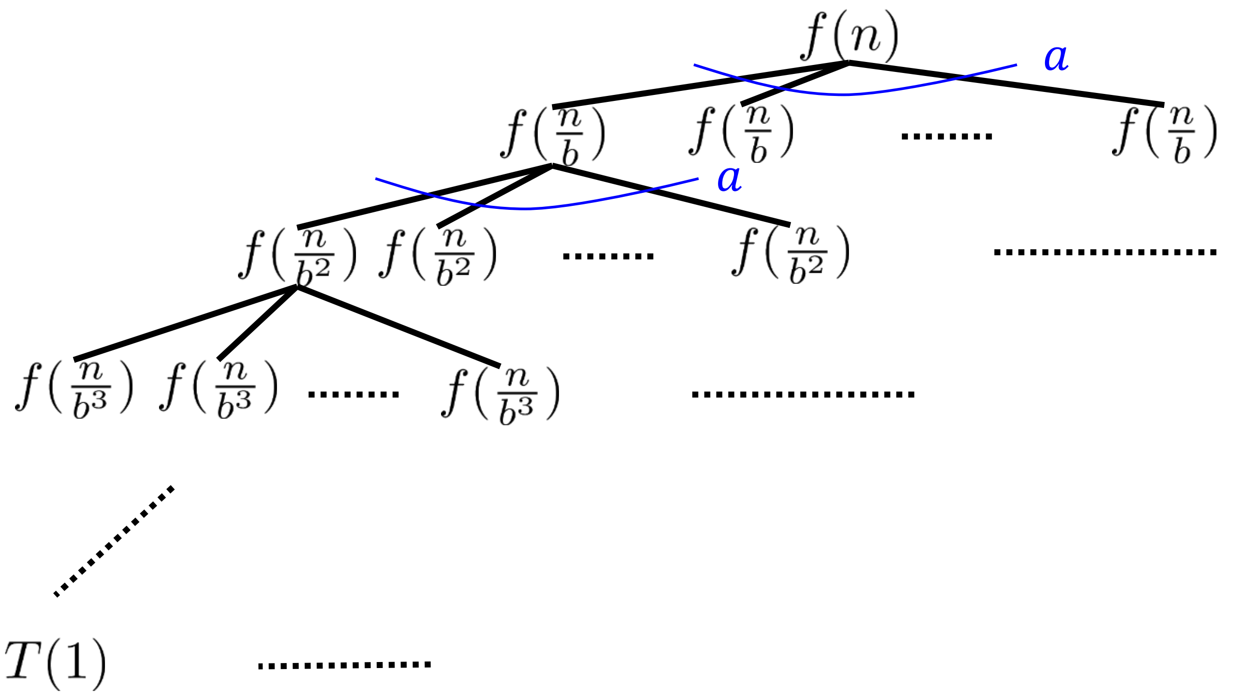
$$T(n) = 9T(\tfrac{n}{3}) + n, T(1) = 1$$

$$\begin{aligned}
T(n) &= (1 + \frac{9}{3} + (\frac{9}{3})^2 + \cdots + (\frac{9}{3})^{\log_3 n})n \\
&= \frac{(\frac{9}{3})^{1+\log_3 n} - 1}{3 - 1} n \\
&= \frac{3n}{2} \cdot \frac{9^{\log_3 n}}{3^{\log_3 n}} - \frac{1}{2} n \\
&= \frac{3n}{2} \cdot \frac{n^{\log_3 9}}{n} - \frac{1}{2} n \\
&= \Theta(n^{\log_3 9}) = \Theta(n^2)
\end{aligned}$$

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

Case 2:
Total cost evenly distributed among levels

$$T(n) = T(\frac{2n}{3}) + 1, T(1) = 1$$



$$f(n) = 1$$

$$af(\frac{n}{b}) = 1$$

$$a^2 f(\frac{n}{b^2}) = 1$$

$$a^3 f(\frac{n}{b^3}) = 1$$

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

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

Case 2:

Total cost evenly distributed among levels

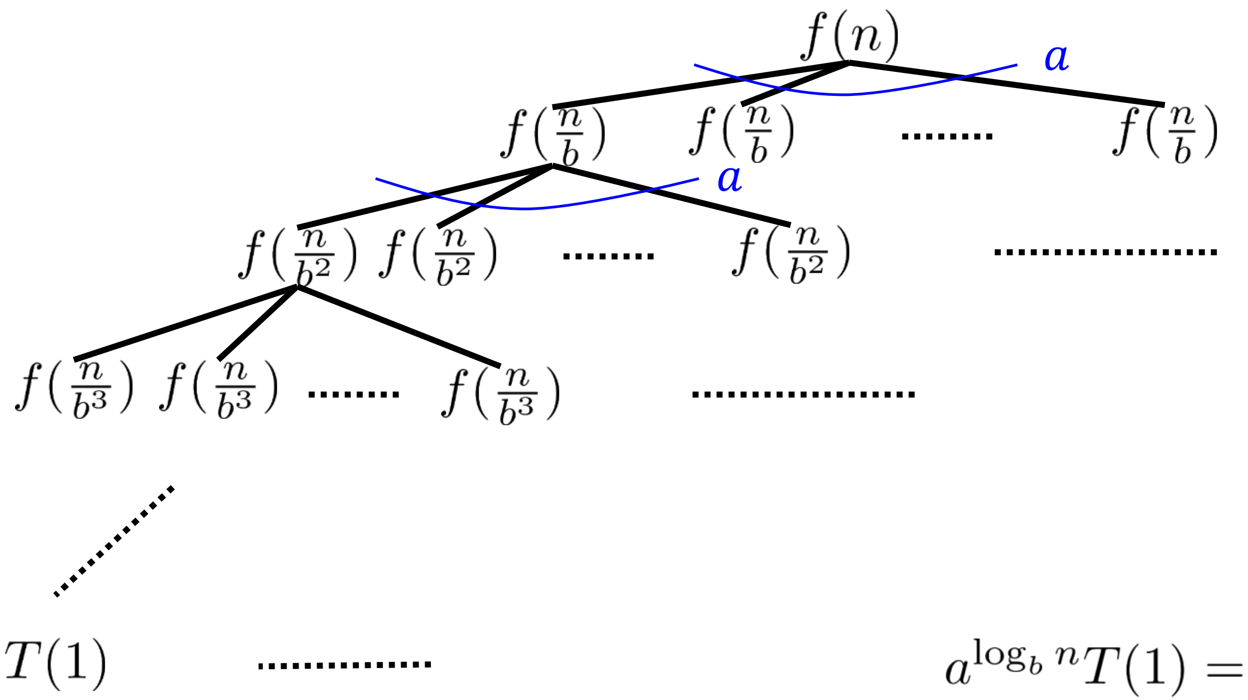
$$T(n) = T(\frac{2n}{3}) + 1, T(1) = 1$$

$$\begin{aligned} T(n) &= 1 + 1 + 1 + \cdots + 1 \\ &= \log_{\frac{3}{2}} n + 1 \\ &= \Theta(\log n) \end{aligned}$$

- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Case 3:
Total cost dominated by root cost

$$T(n) = 3T(\tfrac{n}{4}) + n^5, T(1) = 1$$



$$f(n) = n^5$$

$$af(\tfrac{n}{b}) = 3(\tfrac{n}{4})^5$$

$$a^2 f(\tfrac{n}{b^2}) = 3^2(\tfrac{n}{4^2})^5$$

$$a^3 f(\tfrac{n}{b^3}) = 3^3(\tfrac{n}{4^3})^5$$

$$\vdots$$

$$a^{\log_b n} T(1) = 3^{\log_4 n} = 3^{\log_4 n}(\tfrac{n}{4^{\log_4 n}})^5$$

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

Case 3:

Total cost dominated by root cost

$$T(n) = 3T(\frac{n}{4}) + n^5, T(1) = 1$$

$$T(n) = (1 + \frac{3}{4^5} + (\frac{3}{4^5})^2 + \cdots + (\frac{3}{4^5})^{\log_4 n})n^5$$

$$T(n) > n^5$$

$$T(n) \leq \frac{1}{1-\frac{3}{4^5}}n^5$$

$$T(n) = \Theta(n^5)$$

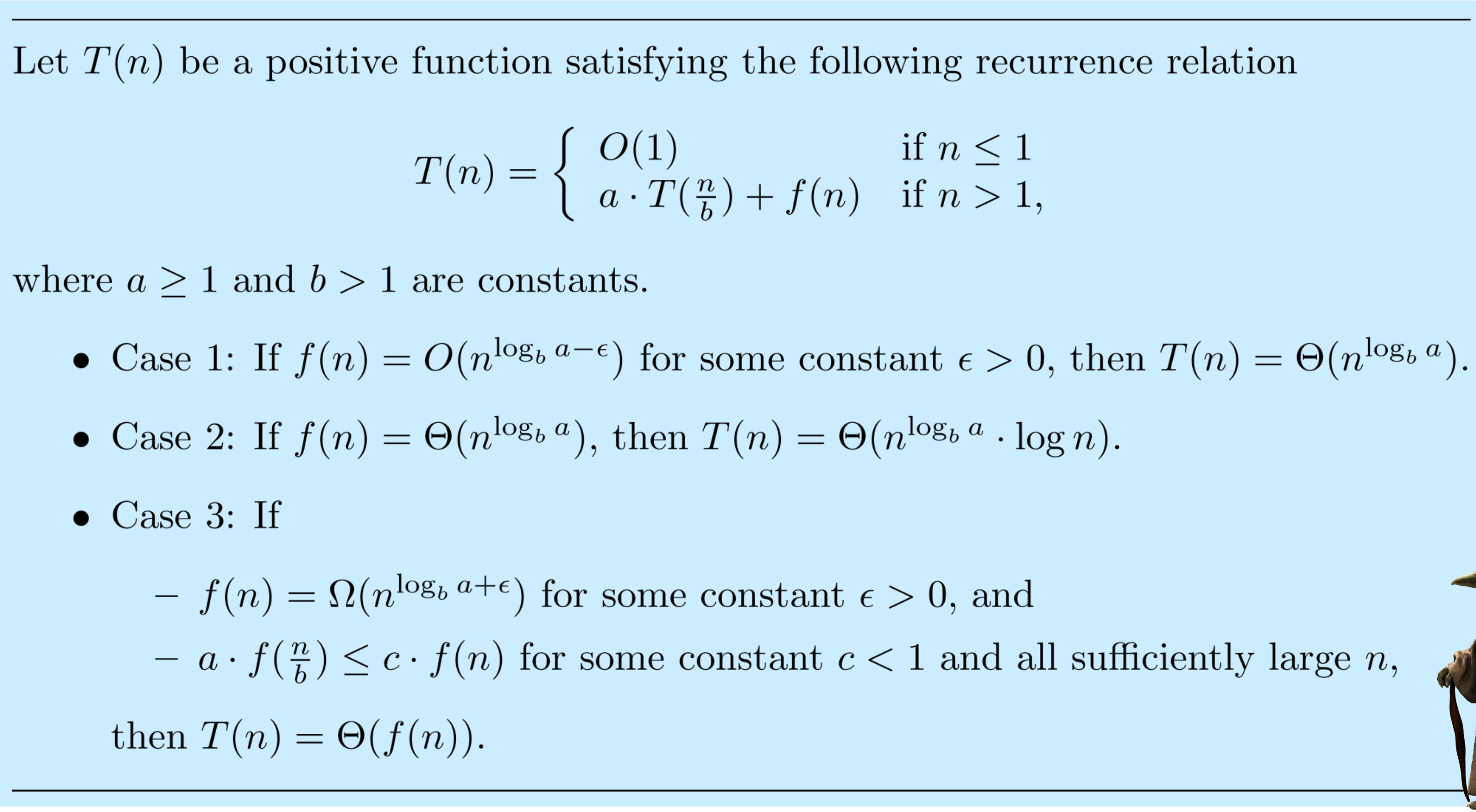
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and
  - $\boxed{a \cdot f(\frac{n}{b}) \leq c \cdot f(n)}$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

Master Theorem

The proof is in Ch. 4.6

divide a problem of size n into a subproblems, each of size $\frac{n}{b}$ is solved in time $T\left(\frac{n}{b}\right)$ recursively



Let $T(n)$ be a positive function satisfying the following recurrence relation

$$T(n) = \begin{cases} O(1) & \text{if } n \le 1 \\ a \cdot T(\frac{n}{b}) + f(n) & \text{if } n > 1, \end{cases}$$

where $a \ge 1$ and $b > 1$ are constants.

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \le c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

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

Examples

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

- Case 1: If $T(n) = 9 \cdot T(n/3) + n$, then $T(n) = \Theta(n^2)$.

  Observe that $n = O(n^2) = O(n^{\log_3 9})$.

- Case 2: If $T(n) = T(2n/3) + 1$, then $T(n) = \Theta(\log n)$.

  Observe that $1 = \Theta(n^0) = \Theta(n^{\log_{3/2} 1})$.

- Case 3: If $T(n) = 3 \cdot T(n/4) + n^5$, then $T(n) = \Theta(n^5)$.
  - $n^5 = \Omega(n^{\log_4 3+\epsilon})$ with $\epsilon = 0.00001$.
  - $3(\frac{n}{4})^5 \leq cn^5$ with $c = 0.99999$.

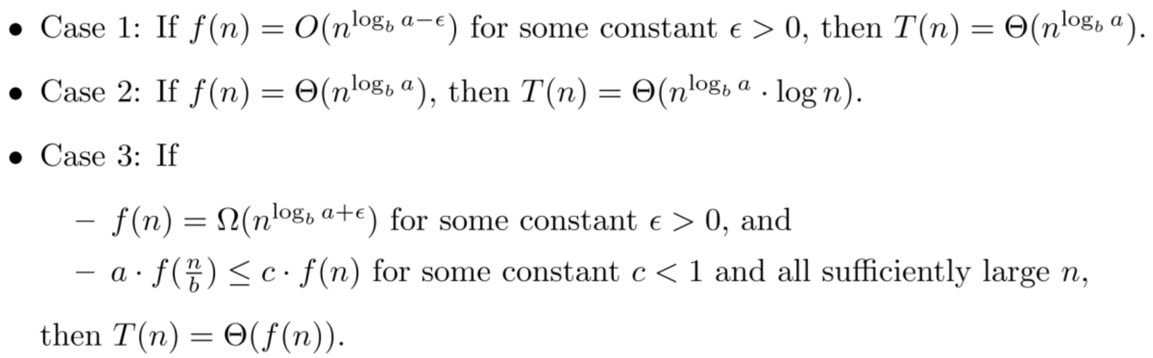
Floors and Ceilings

- Master theorem can be extended to recurrences with floors and ceilings
- The proof is in the Ch. 4.6

$$T(n) = aT(\lceil \frac{n}{b} \rceil) + f(n)$$

$$T(n) = aT(\lfloor \frac{n}{b} \rfloor) + f(n)$$

Theorem 1



- Case 1: If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \le c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \ge 2 \end{cases} \Rightarrow T(n) = O(n \log n)$$

- Case 2

$$f(n) = \Theta(n) = \Theta(n^1) = \Theta(n^{\log_2 2}) = \Theta(n^{\log_b a})$$

$$T(n) = \Theta(f(n) \log n) = O(n \log n)$$

Theorem 2

- Case 1: If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \le c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

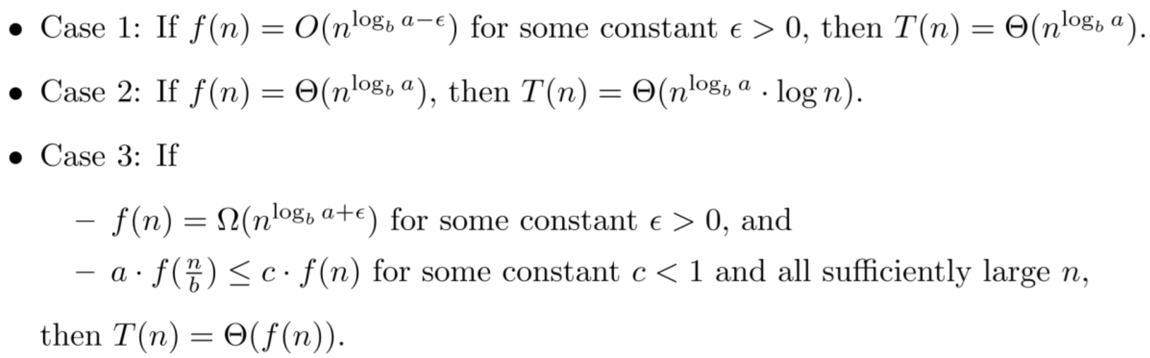
$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(1) & \text{if } n \geq 2 \end{cases} \Rightarrow T(n) = O(n)$$

- Case 1

$$f(n) = O(1) = O(n) = O(n^{\log_2 2}) = O(n^{\log_b a})$$

$$T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$$

Theorem 3



- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(n/2) + O(1) & \text{if } n \geq 2 \end{cases} \quad ➡ \quad T(n) = O(\log n)$$

- Case 2

$$f(n) = \Theta(1) = \Theta(n^0) = \Theta(n^{\log_2 1}) = \Theta(n^{\log_b a})$$

$$T(n) = \Theta(f(n) \log n) = O(\log n)$$

To Be Continue…

Question?

Important announcement will be sent to @ntu.edu.tw mailbox & post to the course website

Course Website: http://ada.miulab.tw
Email: ada-ta@csie.ntu.edu.tw