Lec 02 渐近符号、递归及解法

Asymptotic Notation

- O notation

$f(n) = O(g(n))$ means there are some suitable constants $c > 0$, $n_0 > 0$
such that $0 \le f(n) \le c \cdot g(n)$ for all $n \ge n_0$

Example: $2n^2 = O(n^3)$ (or $2n^2 \in O(n^3)$) ☆
the equation is not symmetric here!

another way to think about what it really means is that
$f(n)$ is in the set of functions that are like $g(n)$, i.e.
$O(g(n)) = \{ f(n) : \exists c, n_0 > 0,\ 0 \le f(n) \le c \cdot g(n), \text{for all } n \ge n_0 \}$

- Macro convention
(宏)

a set in a formula represents an anonymous function in that set

Example: $f(n) = \underbrace{n^3}_{\text{"basically"}} + \underbrace{O(n^2)}_{\text{"error bound"}}$

means there is a function $h(n)$ which is in $O(n^2)$,
such that $f(n) = n^3 + h(n)$

Example: $n^2 + O(n) = O(n^2)$ (or $n^2 + O(n) \subset O(n^2)$)
the equation is not symmetric!

$\underbrace{\text{for any}}_{\text{means}}$ $f(n) \in O(n)$, $\underbrace{\text{there is}}_{\text{"}\exists\text{"}}$ an $h(n) \in O(n^2)$, such that $n^2 + f(n) = h(n)$

- Ω notation (lower bounds)

$\Omega(g(n)) = \{ f(n) : \exists c, n_0 > 0,\ 0 \le c\, g(n) \le f(n), \text{for all } n \ge n_0 \}$

Example: $\sqrt{n} = \Omega(\lg n)$

Analogies: O is "$\le$", Ω is "$\ge$", Θ is "$=$"

- $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Example: $n^2 + O(n) = \Theta(n^2)$

- o & ω notations

o is "$<$" (strictly), ω is "$>$".

def: for every constant c, there exists a constant n_0, ...

$\forall c > 0, \exists n_0 > 0, \ldots$

Example: $2n^2 = o(n^3)$

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

Solving Recurrences (3 main methods)

- substitution method

1. guess the form of the solution
2. verify whether the recurrence satifies this bound by induction (归纳法)
3. solve for constants

$\underbrace{O(1)+O(1)+\cdots+O(1)}_{\text{无穷个}} \neq O(1)$

Example:

$T(n) = 4T(\frac{n}{2}) + n \quad , T(1) = \Theta(1)$

Guess $T(n) = O(n^3)$

induction procedure

Assume $T(k) \leq c \cdot k^3$. for $k \leq n$

$T(n) = 4T(\frac{n}{2}) + n \leq 4 \cdot c \cdot (\frac{n}{2})^3 + n = \frac{1}{2} \cdot c \cdot n^3 + n$

$= \underbrace{c \cdot n^3}_{\text{desired}} - \underbrace{(\frac{1}{2} c \cdot n^3 - n)}_{\text{residual}}$

$\leq cn^3$, if residual part is non-negative e.g. $c \geq 1, n \geq 1$

$= O(n^3)$

and Base case:

$T(1) = \Theta(1) \leq c$, if c is chosen sufficiently large

tight bound

Try $T(n) = O(n^2)$

Assume $T(k) \leq c \cdot k^2$ for $k \leq n$

$T(n) = 4T(\frac{n}{2}) + n$

$\leq 4 \cdot c \cdot (\frac{n}{2})^2 + n$

$= c \cdot n^2 + n \quad \underbrace{= O(n^2)}_{\text{true, but useless}}$

$= c \cdot n^2 - (-n)$

$\not\leq c \cdot n^2$

"strengthen the induction hypothesis"

Assume $T(k) \leq c_1 \cdot k^2 - c_2 k$

$T(n) = 4T(\frac{n}{2}) + n$

$= 4[c_1 \cdot (\frac{n}{2})^2 - c_2 \cdot \frac{n}{2}] + n$

$= c_1 \cdot n^2 + (1 - 2c_2) \cdot n$

$= \underbrace{c_1 n^2 - c_2 \cdot n}_{\text{desired}} - \underbrace{(-1 + c_2) n}_{\text{residual}}$, want non-negative

$\leq c_1 \cdot n^2 - c_2 \cdot n$, if $c_2 \geq 1$

and Base case,

$$\begin{cases} T(1) \leq C_1 \cdot 1^2 - C_2 \cdot 1 \\ T(1) = \Theta(1) \end{cases} \Rightarrow \text{so we need to choose } C_1 \text{ to be sufficiently larger than } C_2$$

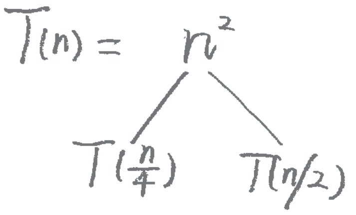
- recursion-tree method (not that rigorous)

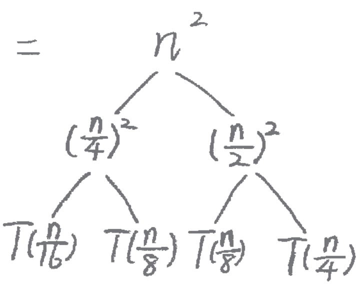
"technically, what you should do is to find out what the answer is with recursion-tree method, then prove that it is actually right with substitution method"

Example:

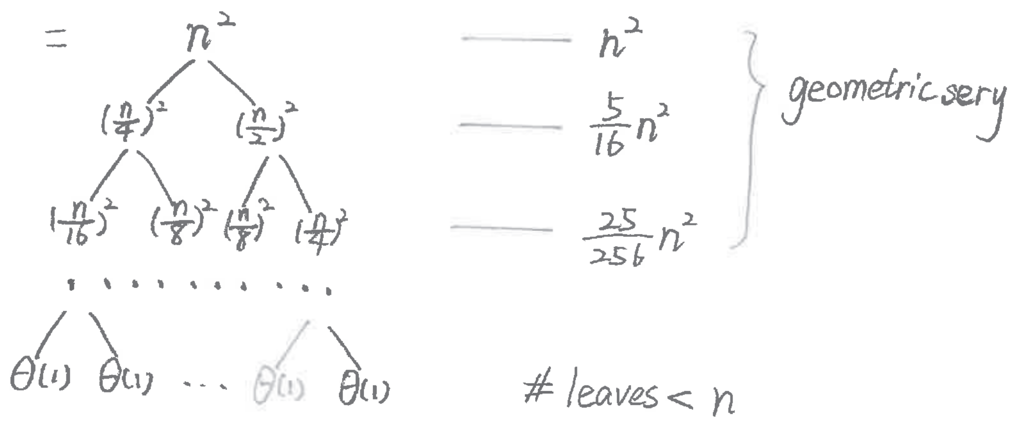
$$T(n) = T(\tfrac{n}{4}) + T(\tfrac{n}{2}) + n^2$$

draw a picture (用树的形式展开递归)
"expand"





= . . .



leaves < n

total (level by level)

$$\leq \left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \cdots + \left(\frac{5}{16}\right)^k + \cdots\right) \cdot n^2$$

it doesn't go on infinitely, but let's assume it goes on infinitely, that would be an upper bound that goes on forever

$$= \lim_{n \to \infty} \frac{1 \times \left[1 - \left(\frac{5}{16}\right)^n\right]}{1 - \frac{5}{16}} \cdot n^2$$

$$= \frac{16}{11} n^2$$

$$< 2n^2 = O(n^2)$$

- master method

it is pretty restrictive, it only applies to a particular family of recurrences of the form $T(n) = aT(n/b) + f(n)$ ← f(n)是非递归的代价

\#subproblems = a   every sub-problems you recurse on should be of the same size $\frac{n}{b}$

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

to make sure the subproblems are getting smaller

for large enough n, ($\exists n_0$) $f(n)$ is positive

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

△ case 1 $\quad f(n) = O(n^{\log_b a - \varepsilon})$, for some $\varepsilon > 0$ ($\exists \varepsilon > 0$)

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

△ case 2 $\quad f(n) = \Theta(n^{\log_b a} \cdot \log_2^k n)$

$$\Rightarrow T(n) = \Theta(n^{\log_b a} \cdot \log_2^{k+1} n)$$

△ case 3 $\quad f(n) = \Omega(n^{\log_b a + \varepsilon})$, for some $\varepsilon > 0$ ($\exists \varepsilon > 0$)

& $af(n/b) \leq (1-\varepsilon') \cdot f(n)$, for some $\varepsilon' > 0$ ($\exists \varepsilon' > 0$)

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

Example:

$T(n) = 4T(\frac{n}{2}) + n \quad (a=4, b=2, f(n)=n)$

$n^{\log_b a} = n^2 > n = f(n)$, in case 1

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

Example:

$T(n) = 4T(\frac{n}{2}) + n^2$

in case 2, so $T(n) = \Theta(n^2 \log_2 n)$

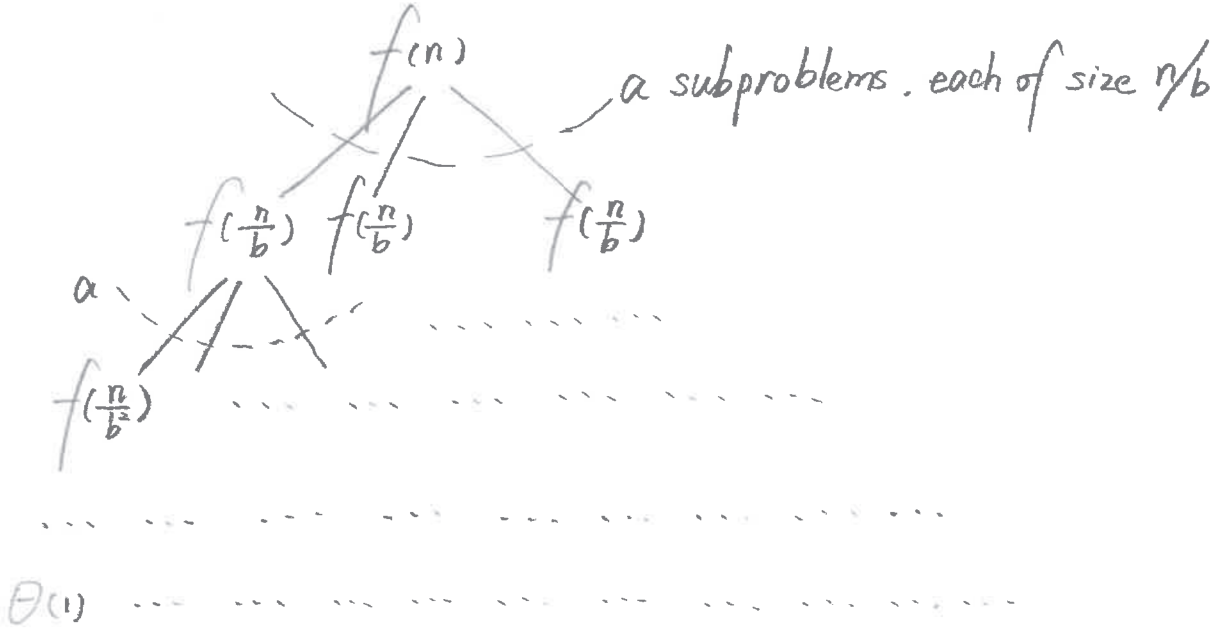
Example:

$T(n) = 4T(\frac{n}{2}) + n^3$

in case 3, so $T(n) = \Theta(n^3)$

proof sketch / intuition

key words: recursion tree, level by level



height of this tree is $h = \log_b n$

number of leaves is $a^h = a^{\log_b n} = n^{\log_b a}$

"case 3" is that costs decrease geometrically as we go down the tree "dominated by $f(n)$"

"case 1" is that costs increase geometrically as we go down the tree "dominated by $\Theta(n^{\log_b a})$"

1st-level cost: $f(n)$

2nd-level cost: $af(\frac{n}{b})$

3rd-level cost: $a^2 f(\frac{n}{b^2})$

......

"case 2" is that "the top is roughly/asymptotically equal to the bottom", the total cost is "one-level cost × height of the tree", i.g.

$f(n) \cdot \log_b n \approx f(n) \cdot \log_2 n$

$\underline{f(n) = \Theta(n^{\log_b a} \cdot \log_2^k n)}$ $\Theta(n^{\log_b a} \cdot \log_2^{k+1} n)$