Lec2 Asymptotic notation

Asymptotic notation

- **Big-O** $f(n) = \mathcal{O}(g(n))$
  - There are suitable constants $c > 0, n_0 > 0$, such that $0 \leq f(n) \leq c * g(n)$ for all $n \geq n_0$
  - e.g. $2n^2 = \mathcal{O}(n^3)$ or $2n^2 \in \mathcal{O}(n^3)$ 等于号：属于某个集合
  - roughly corresponds to less than or equal to $\leq$
  - equal sign asymmetric 正向成立，反向不成立
  - on right-hand side, error term 误差项

    e.g. $f(n) = n^3 + \mathcal{O}(n^2)$ means there is a function $h(n) \in \mathcal{O}(n^2)$ such that $f(n) = n^3 + h(n)$

    in other ways, some lower terms bounded by some constant times n^2
  - on left-hand side

    e.g. $n^2 + \mathcal{O}(n) = \mathcal{O}(n^2)$ means for any $f(n) \in \mathcal{O}(n)$, there is an $h(n) \in \mathcal{O}(n^2)$ such that $n^2 + f(n) = h(n)$

    a chain of equal sign relations, the first one is/bounded by the last one
- **Big-Omega** $f(n) = \Omega(g(n))$
  - There exist constants $c > 0, n_0 > 0$ such that $0 \leq c * g(n) \leq f(n)$ for all $n \geq n_0$
  - e.g. $\sqrt{n} = \Omega(\log n)$ 当n无穷大，root n is at least log n
  - roughly correspond to greater than or equal to $\geq$
- Capital-theta $f(n) = \theta(g(n))$
  - $\theta(g(n)) = \mathcal{O}(g(n)) \cap \Omega(g(n))$
  - 和等于很相近，但又有区别

Strict Notations

- small-o o less than $<$
- small-omega ω greater than $>$

Difference with big notation: instead of saying exist, for all constant c, n_0 exists

e.g. $2n^2 = o(n^3)$, $n^2/2 = \theta(n^2) \neq o(n^2)$

Solving recurrence

Three methods to solve recurrence: substitution method(替代法)

Substitution method

Steps:

1. guess the form of the solution, not need constants 不需要完全猜出答案来，猜出形式即可，常数项不需要
2. verify by induction 数学归纳
3. solve the constants 得到常数系数

Ex.

$$T(n) = 4T(n/2) + n$$
$$T(1) = \theta(1)$$

Steps:

1. Guess 什么函数 double argument, output 增长4倍 -> $\theta(n^2)$

   思路一：Guess $T(n) = \mathcal{O}(n^3)$

   Assume $T(k) \leq k^3$ for $k < n$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4c * (n/2)^3 + n = \frac{1}{2}c * n^3 + n \\ &= c * n^3 - (\frac{1}{2}c * n^3 - n) \\ &\leq c * n^3 (desired) \end{aligned}$$

   等式成立，需要满足 $\frac{1}{2}c * n^3 - n \geq 0$

   当 $c = 2$ 时，不等式成立 (因为 $n \geq 1$) 所以Upper boundn^3成立

   思路二： Guess $T(n) = \mathcal{O}(n^2)$

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

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

   等式成立，需要满足 $-n \geq 0$

   思路三：改进归纳假设 Guess $T(n) = \mathcal{O}(n^2)$

   Assume $T(k) \leq c_1 * k^2 - c_2 * k$ for $k < n$ 加入对低阶项的考虑 （从$c * n^2$ 分出一些抵消n）

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4(c_1 * (n/2)^2 - c_2 * n/2) + n = c_1 * n^2 + (1 - 2c_2) * n \\ &= c_1 * n^2 - c_2 * n - (c_2 - 1) * n \\ &\leq c_1 * n^2 - c_2 * n (desired) \end{aligned}$$

   等式成立，需要residual $(c_2 - 1) * n \geq 0$

   当$c_2 \geq 1$, 推理正确

   c_1的取值也有限制，因为base case: $T(1) \leq c_1 * 1^2 - c_2$ c_1 at least greater than c_2

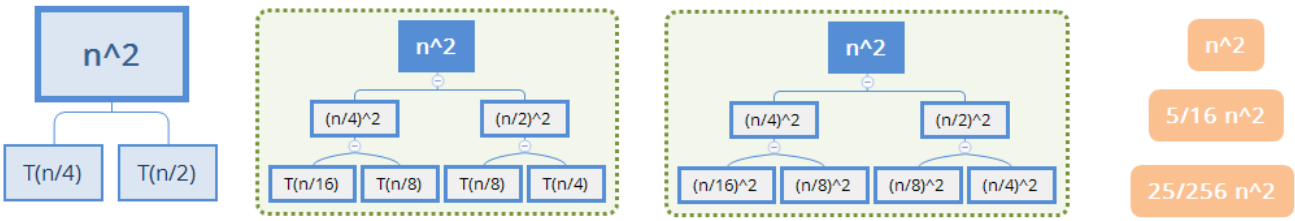
Recursion tree method

More intuition than substitution

Usually find out answers with recursion tree method, and prove it using substitution method

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

Recursion tree解法：expand成一棵树，



How many leave nodes in the tree ? Recursing at different speed in different subtrees.

Upper bound: $\leq n$ (初始递归大小是n, 每次递归成1/4+1/2, 减少1/4,最后至多n个)

geometric series求和

Total (level-by-level) $\leq (1 + \frac{5}{16} + \frac{25}{256} + \ldots + \frac{5^k}{16^k})n^2 \leq 2n^2 \leq \mathcal{O}(n^2)$

Master method 主定理法

Disadvantage: The master method is restrictive, can only applied to a particular family of recurrences $T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$, $f(n)$ is asymptotically positive 渐进趋正, 不参与递归.

渐进趋正 = $f(n) > 0$ at least when $n > n_0$ / $f(n)$ is positive when n is large enough

Theorem:

Compare $f(n)$ with $n^{log_b a}$ (#leaves in recursion tree)

Case 1: $f(n) = o(n^{log_b a - \epsilon})$ for some $\epsilon > 0$ => $T(n) = \theta(n^{log_b a})$

Case 2: $f(n) = \theta(n^{log_b a} * lg(n^k))$ for $k \geq 0$ => $T(n) = \theta(n^{log_b a} * lg(n^{k+1}))$

Case 3: $f(n) = \Omega(n^{log_b a + \epsilon})$ for some $\epsilon > 0$ => $T(n) = \theta(f(n))$

Ex.

1. $T(n) = 4T(n/2) + n$
   $n^{log_b a} = n^{log_2 4} = n^2$ => $f(n) = n = o(n^2)$ => case 1
   $T(n) = \theta(n^2)$
2. $T(n) = 4T(n/2) + n^2$
   $f(n) = n^2 = \theta(n^2)$ => case 2, and $k = 0$
   $T(n) = \theta(n^2 lgn)$

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

   $f(n) = \Omega(n^3)$ => case 3

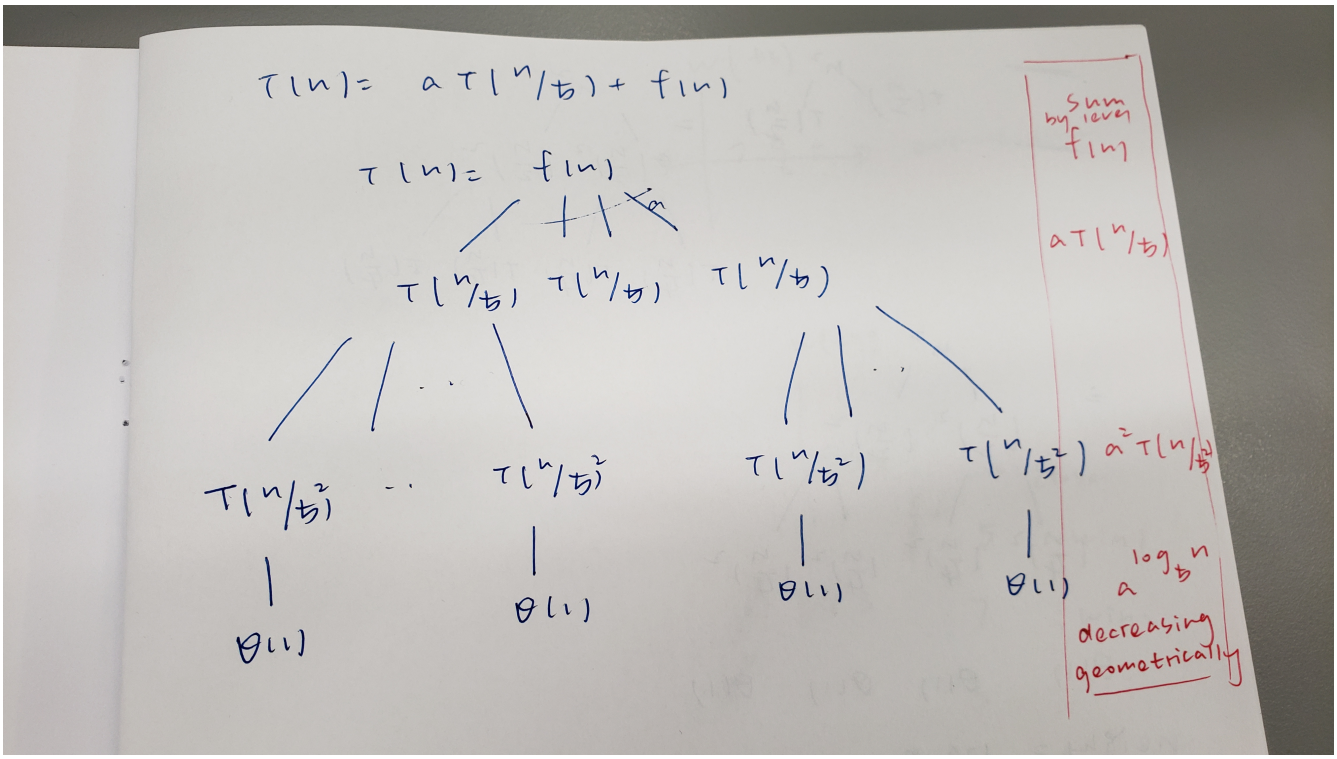
   $T(n) = \theta(n^3)$

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

   cannot be solved by master method, solve by recursion tree

Proof sketch/ Intuition 证明Master method:

以递归树形式展开



height = $\log_b n$

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

Sum at bottom level : $\theta(n^{log_b a})$

把每一层节点的值相加，得到一个数列。

Case 3 时 $f(n) = \Omega(n^{log_b a})$: 这个数列以几何级数递减，那么首项会占据主导。可以得出 $T(n) = \theta(f(n))$

Case 1时 $f(n) = o(n^{\log_b a})$ 这个数列以几何级数递增，那么末项占主导。因此 $T(n) = \theta(n^{log_b a})$

Case 2: 每层之和基本相同