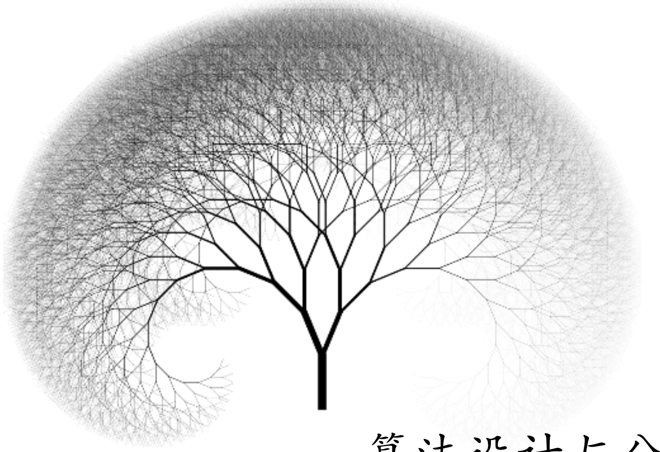


算法设计与分析

刘 安
苏州大学 计算机科学与技术学院
http://web.suda.edu.cn/anliu/

主定理

Master Theorem

分治算法运行时间的递归表示

- 将原问题分解为a个子问题递归求解，每个子问题的规模是原问题的$1/b$
- 分解问题和合并解的时间为n^c，其中n是原问题的规模
- $T(n) = aT(n/b) + n^c$，$a \geq 1, b \geq 2, c > 0$，$T(1) = O(1)$
- 二分搜索：$T(n) = T(n/2) + 1$
- 归并排序：$T(n) = 2T(n/2) + n$
- 整数乘法
  - 直接分治：$T(n) = 4T(n/2) + n$
  - Karatsuba算法：$T(n) = 3T(n/2) + n$
- 矩阵乘法
  - 直接分治：$T(n) = 8T(n/2) + n^2$
  - Strassen算法：$T(n) = 7T(n/2) + n^2$

主定理（简化形式）

- $T(n) = aT(n/b) + n^c$，$a \geq 1, b \geq 2, c > 0$，$T(1) = O(1)$
  - 如果$a < b^c$，那么$T(n) = O(n^c)$
  - 如果$a = b^c$，那么$T(n) = O(n^c \log n)$
  - 如果$a > b^c$，那么$T(n) = O(n^{log_b a})$

使用主定理求解递归关系

- $T(n) = aT(n/b) + n^c$，$a \geq 1, b \geq 2, c > 0$，$T(1) = O(1)$
  - 如果$a < b^c$，那么$T(n) = O(n^c)$
  - 如果$a = b^c$，那么$T(n) = O(n^c \log n)$
  - 如果$a > b^c$，那么$T(n) = O(n^{log_b a})$
- 二分搜索：$T(n) = T(n/2) + 1$
  - $a = 1, b = 2, c = 0$
  - $\Rightarrow a = b^c$
  - $\Rightarrow T(n) = O(\log n)$

使用主定理求解递归关系

- $T(n) = aT(n/b) + n^c$，$a \geq 1, b \geq 2, c > 0$，$T(1) = O(1)$
  - 如果$a < b^c$，那么$T(n) = O(n^c)$
  - 如果$a = b^c$，那么$T(n) = O(n^c \log n)$
  - 如果$a > b^c$，那么$T(n) = O(n^{log_b a})$
- 归并排序：$T(n) = 2T(n/2) + n$
  - $a = 2, b = 2, c = 1$
  - $\Rightarrow a = b^c$
  - $\Rightarrow T(n) = O(n \log n)$

使用主定理求解递归关系

- $T(n) = aT(n/b) + n^c$，$a \geq 1, b \geq 2, c > 0$，$T(1) = O(1)$
  - 如果$a < b^c$，那么$T(n) = O(n^c)$
  - 如果$a = b^c$，那么$T(n) = O(n^c \log n)$
  - 如果$a > b^c$，那么$T(n) = O(n^{log_b a})$
- 整数乘法（直接分治）：$T(n) = 4T(n/2) + n$
  - $a = 4, b = 2, c = 1$
  - $\Rightarrow a > b^c$
  - $\Rightarrow T(n) = O(n^{\log_2 4}) = O(n^2)$
- 整数乘法（Karatsuba算法）：$T(n) = 3T(n/2) + n$
  - $a = 3, b = 2, c = 1$
  - $\Rightarrow a > b^c$
  - $\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})$

使用主定理求解递归关系

- $T(n) = aT(n/b) + n^c$，$a \geq 1, b \geq 2, c > 0$，$T(1) = O(1)$
  - 如果$a < b^c$，那么$T(n) = O(n^c)$
  - 如果$a = b^c$，那么$T(n) = O(n^c \log n)$
  - 如果$a > b^c$，那么$T(n) = O(n^{log_b a})$
- 矩阵乘法（直接分治）：$T(n) = 8T(n/2) + n^2$
  - $a = 8, b = 2, c = 2$
  - $\Rightarrow a > b^c$
  - $\Rightarrow T(n) = O(n^{\log_2 8}) = O(n^3)$
- 矩阵乘法（Strassen算法）：$T(n) = 7T(n/2) + n^2$
  - $a = 7, b = 2, c = 2$
  - $\Rightarrow a > b^c$
  - $\Rightarrow T(n) = O(n^{\log_2 7}) = O(n^{2.81})$

主定理（简化形式）不适用情况

- 子问题数量不是常数
  - $T(n) = nT(n/2) + n^2$
- 子问题数量小于1
  - $T(n) = \frac{1}{2}T(n/2) + n^2$
- 分解问题和合并解的时间不是n^c
  - $T(n) = 2T(n/2) + n \log n$

主定理（一般形式）

- $T(n) = aT(n/b) + f(n)$，$a > 0, b > 1$，$T(1) = O(1)$
  - 如果$\exists \epsilon > 0$使得$f(n) = O(n^{\log_b a - \epsilon})$，那么$T(n) = \Theta(n^{log_b a})$
  - 如果$\exists k \geq 0$使得$f(n) = \Theta(n^{\log_b a} \log^k n)$，那么$T(n) = \Theta(n^{log_b a} \log^{k+1} n)$
  - 如果$\exists \epsilon > 0$使得$f(n) = \Omega(n^{\log_b a + \epsilon})$且对于某个常数$c < 1$和足够大的$n$有$af(n/b) \leq cf(n)$，那么$T(n) = \Theta(f(n))$
- 主要考虑函数$n^{\log_b a}$与函数$f(n)$的增长率关系
- 情况1：$n^{\log_b a}$比$f(n)$增长的更快
- 情况2：$n^{\log_b a}$和$f(n)$的增长率类似
- 情况3：$n^{\log_b a}$比$f(n)$增长的更慢

主定理（一般形式）

- $T(n) = aT(n/b) + f(n)$，$a > 0, b > 1$，$T(1) = O(1)$
  - 如果$\exists \epsilon > 0$使得$f(n) = O(n^{\log_b a - \epsilon})$，那么$T(n) = \Theta(n^{log_b a})$
  - 如果$\exists k \geq 0$使得$f(n) = \Theta(n^{\log_b a} \log^k n)$，那么$T(n) = \Theta(n^{log_b a} \log^{k+1} n)$
  - 如果$\exists \epsilon > 0$使得$f(n) = \Omega(n^{\log_b a + \epsilon})$且对于某个常数$c < 1$和足够大的$n$有$af(n/b) \leq cf(n)$，那么$T(n) = \Theta(f(n))$
- 主要考虑函数$n^{\log_b a}$与函数$f(n)$的增长率关系
- 情况1：$n^{\log_b a}$比$f(n)$增长的更快
  - 至少要快$\Theta(n^\epsilon)$倍
- $T(n) = 9T(n/3) + n$
  - $n^{\log_b a} = n^2$，$f(n) = n = O(n^{2-\epsilon})$，$\epsilon \leq 1$
  - $\Rightarrow T(n) = O(n^2)$

主定理（一般形式）

- $T(n) = aT(n/b) + f(n)$，$a > 0, b > 1$，$T(1) = O(1)$
  - 如果$\exists \epsilon > 0$使得$f(n) = O(n^{\log_b a - \epsilon})$，那么$T(n) = \Theta(n^{log_b a})$
  - 如果$\exists k \geq 0$使得$f(n) = \Theta(n^{\log_b a} \log^k n)$，那么$T(n) = \Theta(n^{log_b a} \log^{k+1} n)$
  - 如果$\exists \epsilon > 0$使得$f(n) = \Omega(n^{\log_b a + \epsilon})$且对于某个常数$c < 1$和足够大的$n$有$af(n/b) \leq cf(n)$，那么$T(n) = \Theta(f(n))$
- 主要考虑函数$n^{\log_b a}$与函数$f(n)$的增长率关系
- 情况2：$n^{\log_b a}$与$f(n)$的增长率类似
  - $f(n)$比$n^{\log_b a}$要快$\Theta(\log^k n)$倍，其中$k \geq 0$
- $T(n) = T(2n/3) + 1$
  - $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$，$f(n) = 1 = \Theta(n^{\log_b a} \log^0 n)$
  - $\Rightarrow T(n) = \Theta(\log n)$

主定理（一般形式）

- $T(n) = aT(n/b) + f(n)$，$a > 0,\ b > 1$，$T(1) = O(1)$
  - 如果$\exists \epsilon > 0$使得$f(n) = O(n^{\log_b a - \epsilon})$，那么$T(n) = \Theta(n^{log_b a})$
  - 如果$\exists k \geq 0$使得$f(n) = \Theta(n^{\log_b a} \log^k n)$，那么$T(n) = \Theta(n^{log_b a} \log^{k+1} n)$
  - 如果$\exists \epsilon > 0$使得$f(n) = \Omega(n^{\log_b a + \epsilon})$且对于某个常数$c < 1$和足够大的$n$有$af(n/b) \leq cf(n)$，那么$T(n) = \Theta(f(n))$
- 主要考虑函数$n^{\log_b a}$与函数$f(n)$的增长率关系
- 情况3：$n^{\log_b a}$比$f(n)$增长的更慢
  - $f(n)$比$n^{\log_b a}$增长的更快，至少要快$\Theta(n^\epsilon)$倍，且$af(n/b) \leq cf(n)$
- $T(n) = 3T(n/4) + n \log n$
  - $n^{\log_b a} = n^{\log_4 3} = n^{0.793}$，$f(n) = n \log n = \Omega(n^{\log_4 3 + \epsilon})$，$\epsilon \leq 0.207$
  - $af(n/b) = 3(n/4)\log(n/4) \leq (3/4) n \log n = cf(n)$，$c = 3/4$
  - $\Rightarrow T(n) = \Theta(n \log n)$

使用主定理求解递归关系

- $T(n) = 8T(n/2) + \Theta(1)$
  - $n^{\log_b a} = n^{\log_2 8} = n^3$，$f(n) = \Theta(1)$
  - $f(n) = \Theta(1) = O(n^{3-\epsilon})$，对于$\forall \epsilon < 3$成立
  - $\Rightarrow T(n) = \Theta(n^3)$
- $T(n) = 7T(n/2) + \Theta(n^2)$
  - $n^{\log_b a} = n^{\log_2 7} = n^{2.81}$，$f(n) = n^2$
  - $f(n) = \Theta(n^2) = O(n^{2.81-\epsilon})$，对于$\forall \epsilon < 0.8$成立
  - $\Rightarrow T(n) = \Theta(n^{2.81})$

使用主定理求解递归关系

- $T(n) = 2T(n/2) + \Theta(n)$
  - $n^{\log_b a} = n^{\log_2 2} = n$，$f(n) = n$
  - $f(n) = n = \Theta(n^{\log_b a} \log^0 n)$
  - $\Rightarrow T(n) = \Theta(n \log n)$
- $T(n) = 2T(n/2) + n \log n$
  - $n^{\log_b a} = n^{\log_2 2} = n$，$f(n) = n \log n$
  - 虽然$f(n)$比$n^{\log_b a}$增长的更快，但注意到$f(n) = n \log n = \Theta(n^{\log_b a} \log^1 n)$
  - 根据情况2，有$T(n) = \Theta(n \log^2 n)$
  - $f(n)$虽然比$n^{\log_b a}$增长的更快，但只是快了$\Theta(\log n)$倍，而不是$\Theta(n^\epsilon)$倍
    - 所以不能使用情况3

主定理（一般形式）不适用情况

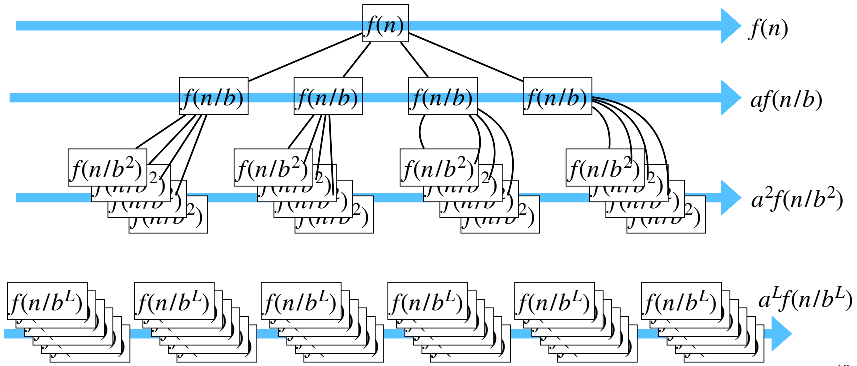
- $n^{\log_b a}$与$f(n)$的增长率不可比
- $n^{\log_b a}$比$f(n)$增长的更快，但没有快$\Theta(n^\epsilon)$倍
- $f(n)$比$n^{\log_b a}$增长的更快，但没有快$\Theta(n^\epsilon)$倍
- $T(n) = 2T(n/2) + n/\log n$
  - $n^{\log_b a} = n$，$f(n) = n/\log n$
  - $n^{\log_b a}$比$f(n)$增长的更快，但也只是快了$\Theta(\log n)$倍，所以不能使用情况1
  - $f(n) = n/\log n = \Theta(n^{\log_b a} \log^{-1} n)$，能否使用情况2?
    - 不能，因为这里$k = -1$，而情况2要求$k \geq 0$

递归树

Recursion Tree

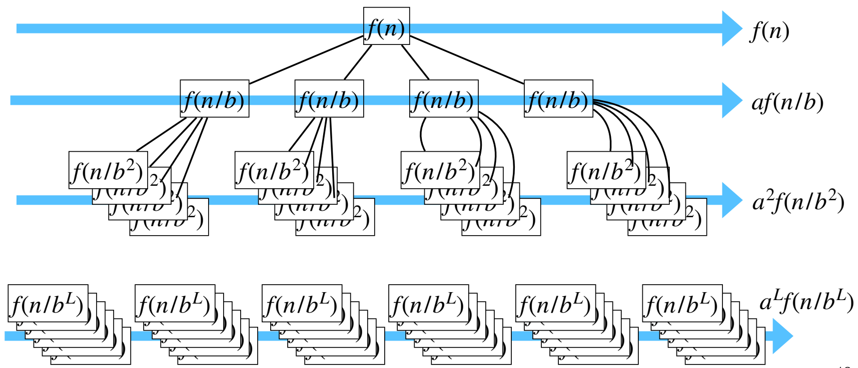
递归树

- 有根树，根代表原问题，根以外的每个节点代表一个子问题
- 节点的值表示解决该问题所花费的除递归调用之外的时间
- 原问题的运行时间等于该树所有节点的值的和
- $T(n) = aT(n/b) + f(n)$



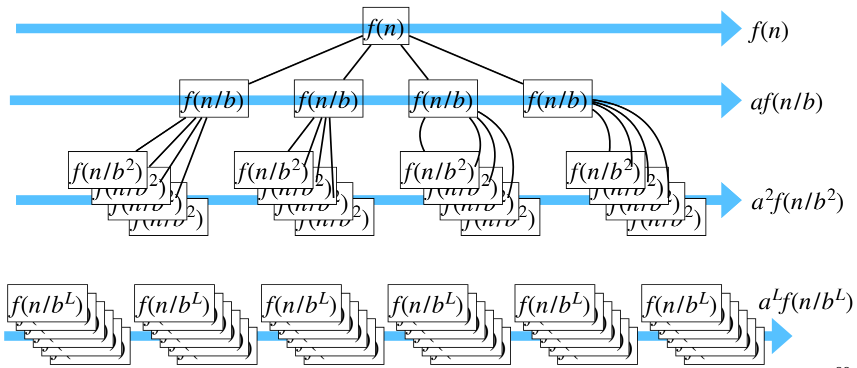
递归树

- 叶子节点是递归的基本情况：可以假设当$n \leq n_0 = 1$时，$T(n) = 1$
- 假设n是b的整数次幂：$n/b^L = 1 \Rightarrow L = \log_b n$
- $T(n) = \sum_{i=0}^{L} a^i f(n/b^i)$



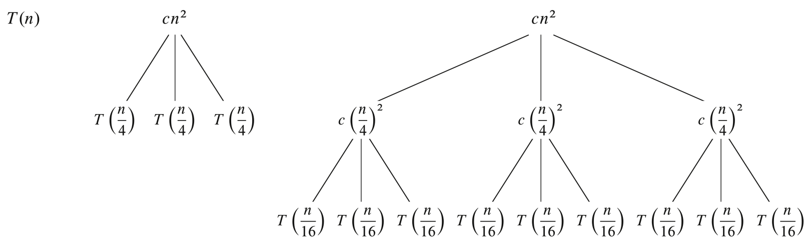
有关递归树的简单结论

- 假设每一层节点值之和是上一层节点值之和的r倍（即构成几何级数）
  - 如果$r < 1$，那么$T(n) = O(f(n))$
  - 如果$r = 1$，那么$T(n) = O(f(n)\log n)$
  - 如果$r > 1$，那么$T(n) = O(a^L) = O(a^{\log_b n}) = O(n^{\log_b a})$



使用递归树求解递归关系

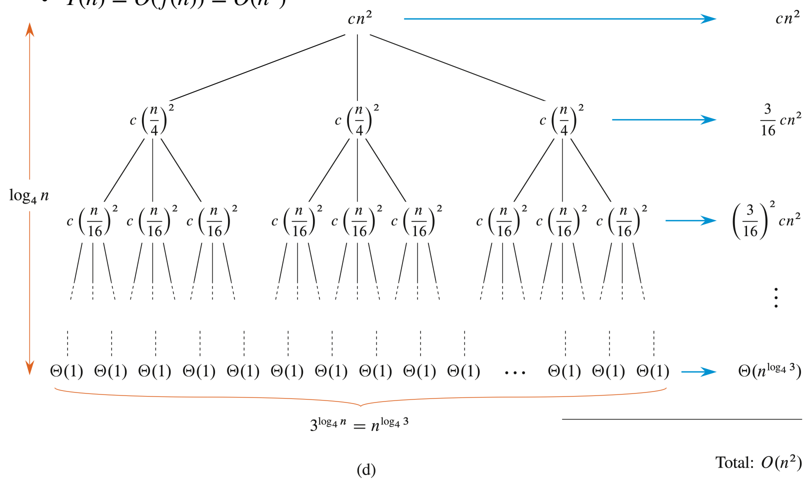
- $T(n) = 3T(n/4) + \Theta(n^2)$



(a) (b) (c)

使用递归树求解递归关系

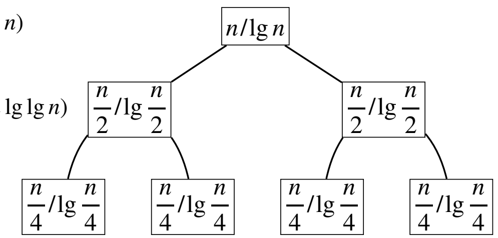
- $T(n) = 3T(n/4) + \Theta(n^2)$
- $T(n) = O(f(n)) = O(n^2)$



(d)

使用递归树求解递归关系

- 有的递归树各层节点值之和并不构成几何级数
- $T(n) = 2T(n/2) + n/\lg n$
- 第i层的和等于$n/\lg \dfrac{n}{2^i} = n/(\lg n - i) \quad \Rightarrow \lg n - i \geq 1 \Rightarrow i \leq \lg n - 1$
- $T(n) = \sum_{i=0}^{L} n/(\lg n - i) = \sum_{i=0}^{\lg n - 1} n/(\lg n - i) = \sum_{j=1}^{\lg n} n/j$
- 调和级数$H_n = \sum_{i=1}^{n} \dfrac{1}{i} = \Theta(\log n)$
- $T(n) = \sum_{j=1}^{\lg n} n/j = nH_{\lg n} = \Theta(n \lg\lg n)$



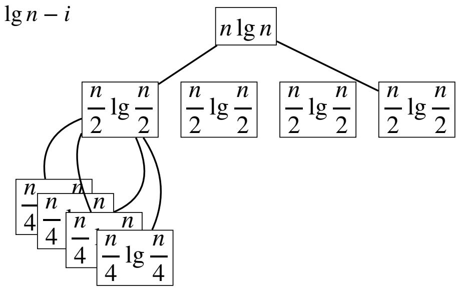
使用递归树求解递归关系

- 有的递归树各层节点值之和并不构成几何级数
- $T(n) = 4T(n/2) + n\lg n$
- 第i层有4^i个节点，每个节点的值等于$(n/2^i)\lg(n/2^i) = (n/2^i)(\lg n - i)$
- $T(n) = \sum_{i=0}^{\lg n - 1} n2^i(\lg n - i)$

$$= \sum_{j=1}^{\lg n} n2^{\lg n - j}j \qquad \text{令} j = \lg n - i$$

$$= \sum_{j=1}^{\lg n} \frac{n^2 j}{2^j}$$

$$= n^2 \sum_{j=1}^{\lg n} \frac{j}{2^j} = \Theta(n^2)$$

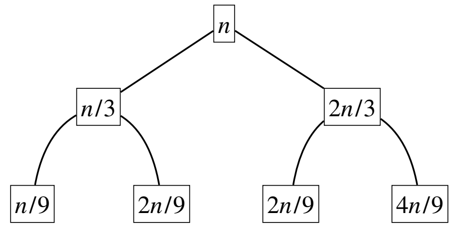


使用递归树求解递归关系

- $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$
- 第0层有1个节点，每个节点的值等于n
- 第1层有$n^{1/2}$个节点，每个节点的值等于$n^{1/2}$
- 第2层有$n^{3/4}$个节点，每个节点的值等于$n^{1/4}$
- 第i层有$n^{\frac{2^i-1}{2^i}}$个节点，每个节点的值等于$n^{\frac{1}{2^i}}$
- $T(n) = \sum_{i=0}^{L} n$
  - $n^{\frac{1}{2^L}} = 1$?
  - $n^{\frac{1}{2^L}} = 2$
  - $\Rightarrow L = \lg\lg n$
- $T(n) = \Theta(n \lg\lg n)$

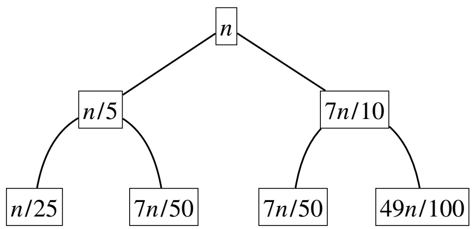
使用递归树求解递归关系

- $T(n) = T(n/3) + T(2n/3) + n$
- 如果第i层没有缺失节点，那么所有节点值之和为n
- 叶子结点的深度L
  - $\log_3 n \le L \le \log_{\frac{3}{2}} n$
- $T(n) \le \sum_{i=0}^{\log_{\frac{3}{2}} n} n = n \log_{\frac{3}{2}} n$
- $T(n) \ge \sum_{i=0}^{\log_3 n} n = n \log_3 n$
- $T(n) = \Theta(n \log n)$



使用递归树求解递归关系

- $T(n) = T(n/5) + T(7n/10) + n$
- 第0层所有节点值之和等于n
- 第1层所有节点值之和等于$9n/10$
- 第2层所有节点值之和等于$81n/100$
- $\cdots$
- $T(n) = \Theta(n)$



处理递归关系的一些技巧

域转换 (domain transformation)

- 归并排序的运行时间：$T(n) = 2T(n/2) + O(n)$
  - 更准确的表达：$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n)$
  - 如何处理向上取整和向下取整
- 最坏情况分析：关注运行时间的上界
  - $T(n) \le 2T(\lceil n/2 \rceil) + n \le 2T(n/2 + 1) + n$
- 域转换：假设$S(n) = T(n + \alpha)$，选择恰当的α使得$S(n) \le 2S(n/2) + O(n)$

$$\begin{aligned} S(n) &= T(n + \alpha) \\ &\le 2T(n/2 + \alpha/2 + 1) + n + \alpha \\ &= 2S(n/2 - \alpha/2 + 1) + n + \alpha \end{aligned} \qquad \Rightarrow S(n) = O(n \log n)$$

令$\alpha = 2$即可满足要求

- $T(n) = S(n-2) = O((n-2)\log(n-2)) = O(n \log n)$