算法设计与分析

LECTURE 2

Lecture 2

数学基础

Outline

- 函数的渐近增长率
- 蛮力算法的逐步改进
- 分治递归求解

如何比较两个算法？

- 算法分析
  - 关键操作计数作为代价
  - 较大的输入规模
  - 重要成分
    - 考虑 f(n) 的主导成分
    - 常数系数可忽略

- 函数渐近增长率
  - O, Ω , Θ, ω, o

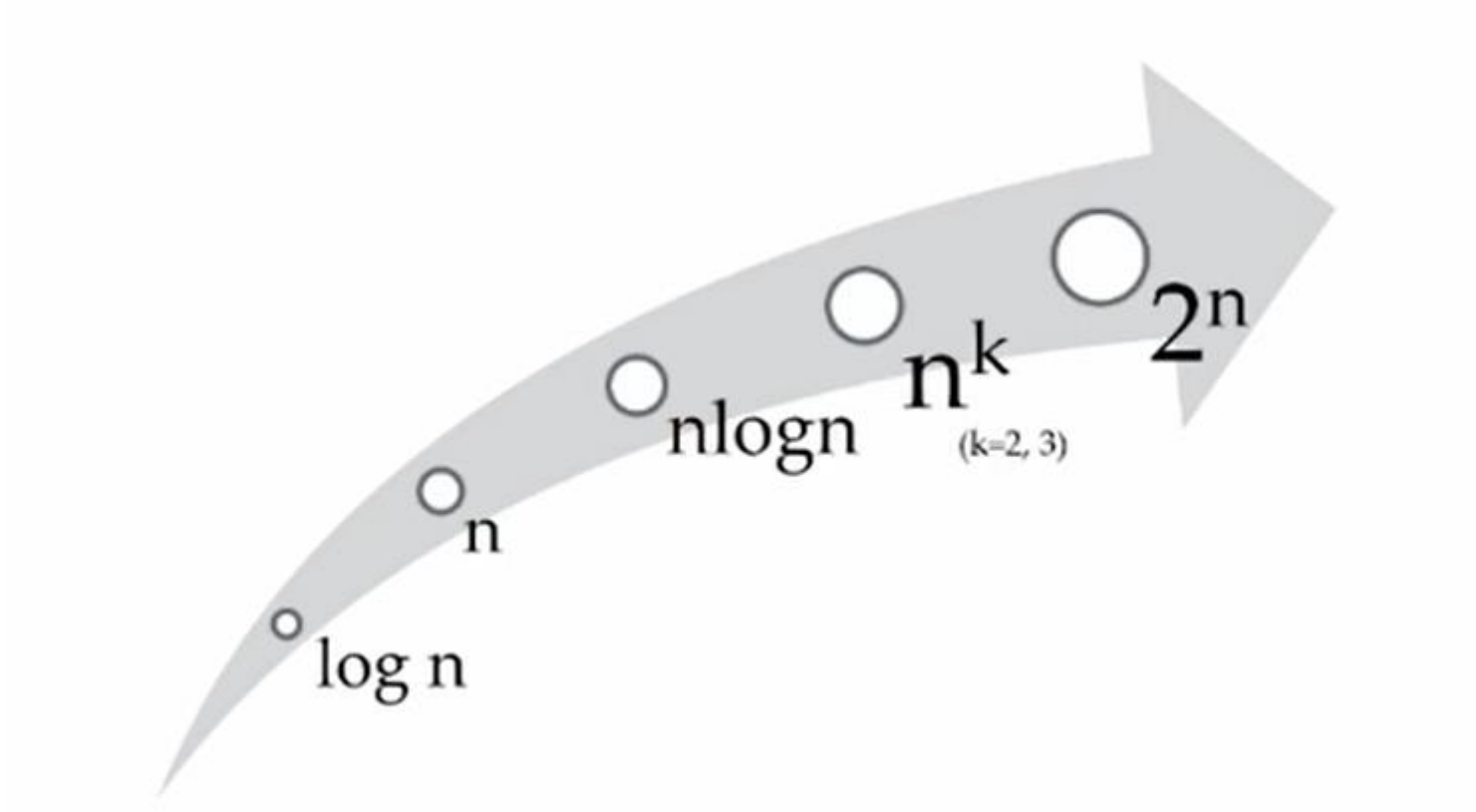
Big Oh

- $f(n) \in O(g(n))$
  - 对于足够大的输入规模 n，$g(n)$ 是 $f(n)$ 的上界

- 定义：
  - 存在常数 $c > 0$ 和 $n_0 > 0$，满足 $0 \le f(n) \le cg(n)$ 对所有 均 $n \ge n_0$ 成立

- $f(n) \in O\big(g(n)\big)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$

函数渐近增长率



函数渐近增长率

- $\log n$
  - $\log n \in O(n^{\alpha})$ for any $\alpha > 0$
- Power n^k
  - $n^k \in O(c^n)$ for any $c > 1$

Big Ω

- $f(n) \in \Omega(g(n))$

- 定义：
  - 存在常数 $c > 0$ 和 $n_0 > 0$，满足 $0 \le cg(n) \le f(n)$ 对所有 均 $n \ge n_0$ 成立

- $f(n) \in \Omega\big(g(n)\big)$ if $\lim\limits_{n \to \infty} \dfrac{f(n)}{g(n)} = c > 0$

Big Θ

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

- 定义：
  - 存在常数 $c_1 > 0$ 、$c_2 > 0$ 和 $n_0 > 0$ ，满足 $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ 对所有 均 $n \ge n_0$ 成立

- $f(n) \in \Theta\big(g(n)\big)$ if $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = c,\ (0 < c < \infty)$

Example

algorithm		1	2	3	4
Run time in *ns*		$1.3n^3$	$10n^2$	$47n\log n$	$48n$
time for size	10^3	1.3*s*	10*ms*	0.4*ms*	0.05*ms*
	10^4	22*m*	1*s*	6*ms*	0.5*ms*
	10^5	15*d*	1.7*m*	78*ms*	5*ms*
	10^6	41*yrs*	2.8*hrs*	0.94*s*	48ms
	10^7	41*mill*	1.7*wks*	11*s*	0.48*s*
max Size in time	sec	920	10,000	1.0×10^6	2.1×10^7
	min	3,600	77,000	4.9×10^7	1.3×10^9
	hr	14,000	6.0×10^5	2.4×10^9	7.6×10^{10}
	day	41,000	2.9×10^6	5.0×10^{10}	1.8×10^{12}
time for 10 times size		×1000	×100	×10+	×10

on 400Mhz Pentium II, in C
from: Jon Bentley: *Programming Pearls*

Little Oh

- $f(n) \in o(g(n))$

- 定义：
  - 对任意常数 $c > 0$，均存在常数 $n_0 > 0$，满足 $0 \le f(n) < cg(n)$ 对所有 $n \ge n_0$ 均成立

- $f(n) \in o\big(g(n)\big)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Little ω

- $f(n) \in \omega(g(n))$

- 定义：
  - 对任意常数 $c > 0$，均存在常数 $n_0 > 0$，满足 $0 \leq cg(n) < f(n)$ 对所有 $n \geq n_0$ 均成立

- $f(n) \in \omega\big(g(n)\big)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

Lecture 2

数学基础

Outline

- 蛮力算法的逐步改进
  - 旋转数组
  - 最大子序列和

蛮力算法设计

旋转数组

- <时间，空间>
  - From $<O(n^2), O(1)>$
  - To $<O(n), O(n)>$
  - To $<O(n), O(1)>$

最大子序列和

- 时间
  - From $O(n^3)$
  - To $O(n^2)$
  - To $O(n\ \log n)$
  - To $O(n)$

189. 旋转数组

难度 中等　1136　收藏　分享　切换为英文　接收动态　反馈

53. 最大子序和

难度 简单　3761　收藏　分享　切换为英文　接收动态　反馈

给定一个整数数组 `nums` ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

示例 1:

```
输入: nums = [-2,1,-3,4,-1,2,1,-5,4]
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6 。
```

```
向右旋转 1 步: [7,1,2,3,4,5,6]
向右旋转 2 步: [6,7,1,2,3,4,5]
向右旋转 3 步: [5,6,7,1,2,3,4]
```

旋转数组

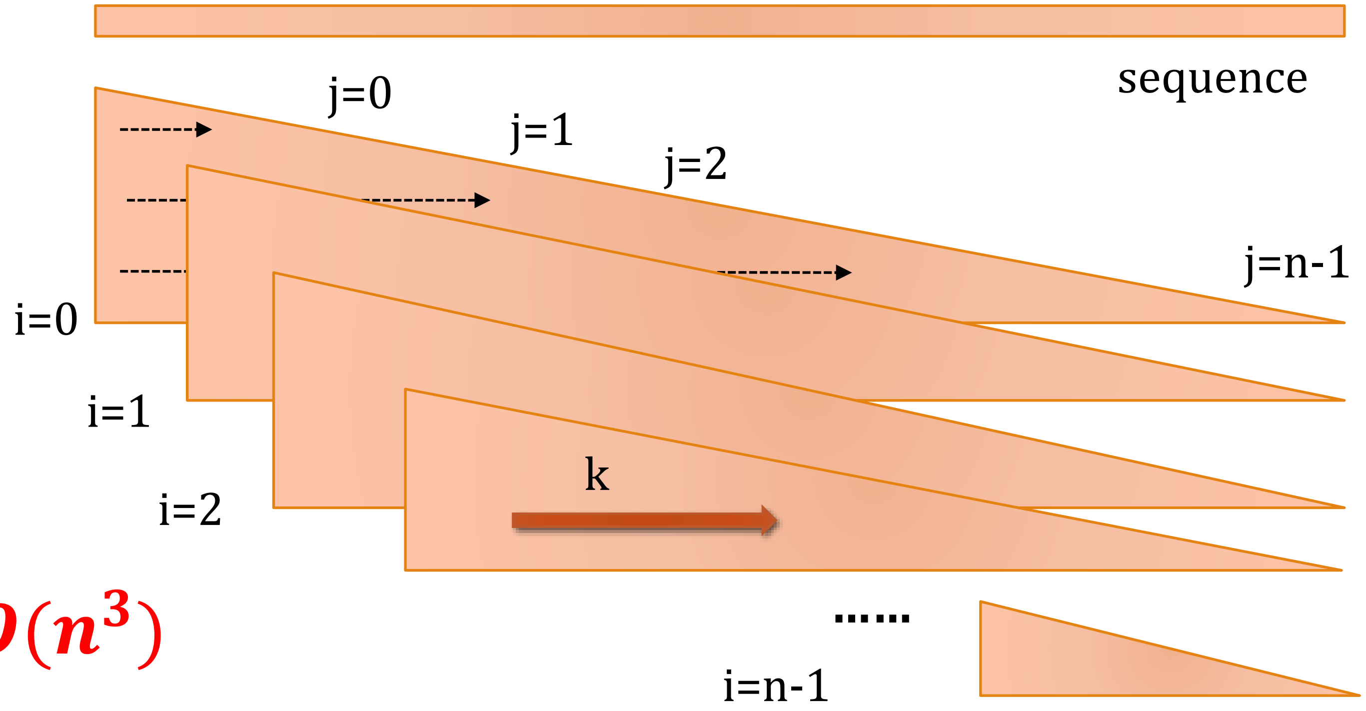
- 示例：1，2，3，4 | 5，6，7 => 5，6，7，1，2，3，4
- 蛮力 Brute force

	时间	空间
BF 1	$O(n^2)$	$O(1)$
BF 2	$O(n)$	$O(n)$
?	$O(n)$	$O(1)$

最大子序列和

- 输入：nums = [-2,1,-3,4,-1,2,1,-5,4]
- 输出：6；连续子数组 [4,-1,2,1] 的和最大，为 6 。

```
A brute-force algorithm:
MaxSum = 0;
 for (i = 0; i < N; i++)
  for (j = i; j < N; j++)
  {
   ThisSum = 0;
   for (k = i; k <= j; k++)
   ThisSum += A[k];
   if (ThisSum > MaxSum)
    MaxSum = ThisSum;
  }
 return MaxSum;
```



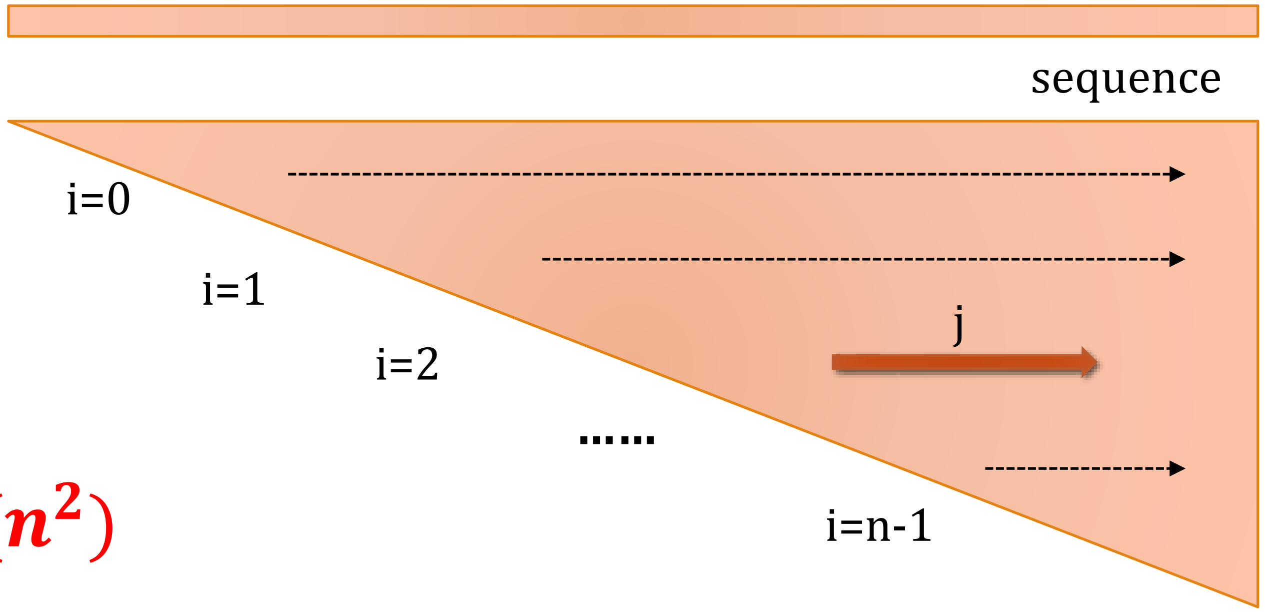
$O(n^3)$

最大子序列和

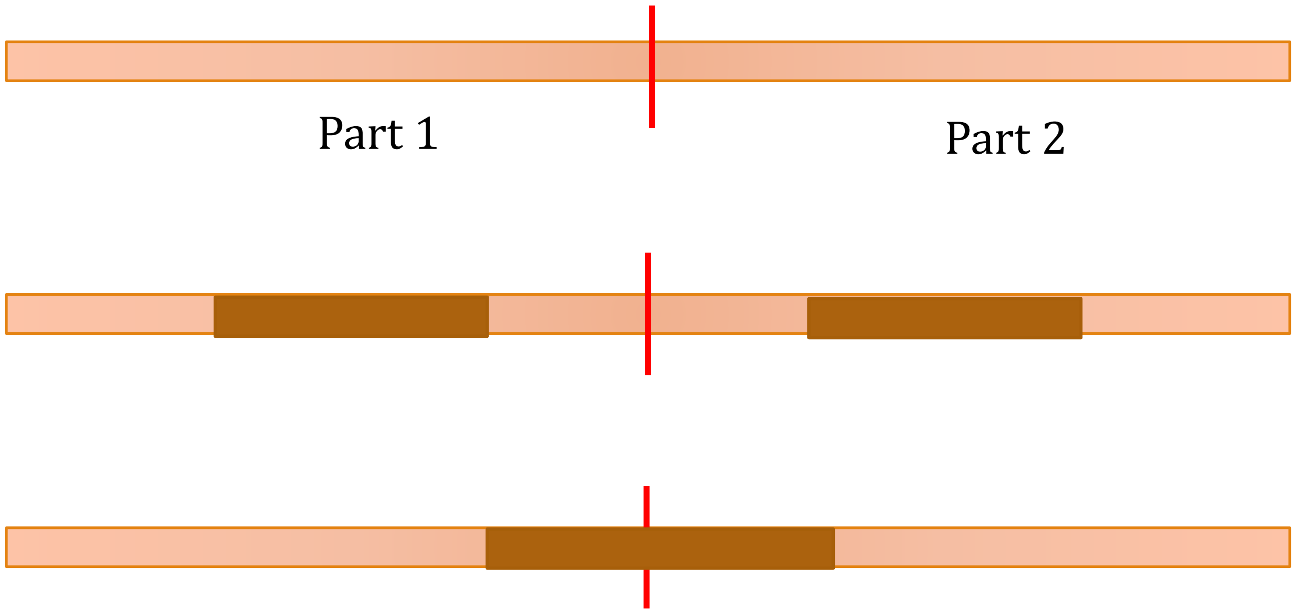
- 输入： nums = [-2,1,-3,4,-1,2,1,-5,4]
- 输出：6；连续子数组 [4,-1,2,1] 的和最大，为 6 。

```
An improved algorithm
MaxSum = 0;
 for (i = 0; i < N; i++)
 {
   ThisSum = 0;
   for (j = i; j < N; j++)
   {
     ThisSum += A[j];
     if (ThisSum > MaxSum)
       MaxSum = ThisSum;
   }
 }
 return MaxSum;
```

$O(n^2)$



最大子序列和



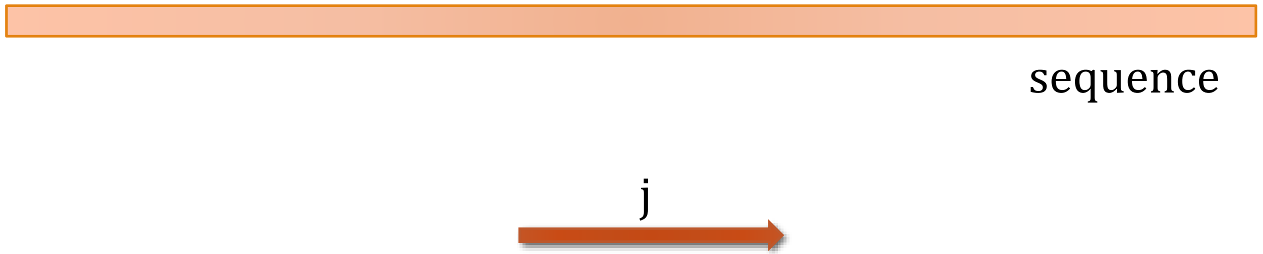
or

$O(n\log n)$

最大子序列和

- 输入： nums = [-2,1,-3,4,-1,2,1,-5,4]
- 输出：6；连续子数组 [4,-1,2,1] 的和最大，为 6 。

```
ThisSum = MaxSum = 0;
 for (j = 0; j < N; j++)
 {
  ThisSum += A[j];
  if (ThisSum > MaxSum)
    MaxSum = ThisSum;
  else if (ThisSum < 0)
    ThisSum = 0;
 }
 return MaxSum;
```



$O(n)$

Lecture 2

数学基础

Outline

- 递归算法
  - 分治策略
  - 递归方程
- 分治递归求解
  - Master 定理

递归算法设计

- 阶乘函数 Fac(n) 计算 n!
  - if n=1 then return 1 else return Fac(n-1)*n

M(1)=0 and M(n)=M(n-1)+1 for n>0
(关键操作：*)

- 汉诺塔问题
  - if n=1 then move d(1) to peg 3 else
  - Hanoi(n-1, peg1, peg2); move d(n) to peg3; Hanoi(n-1, peg2, peg3)

- 如何用递归方程描述递归算法？

M(1)=1 and M(n)=2M(n-1)+1 for n>1
(关键操作：move)

递归算法设计

- 计算一个数的比特数
  - 输入：一个正的十进制数 n
  - 输出：n 的二进制数表示的位数

```
def BitCounting(n):
    if n==1:
        return 1
    else:
        return BitCounting(n/2) + 1
```

T(1)=0 and T(n)=T(n/2)+1 for n>1

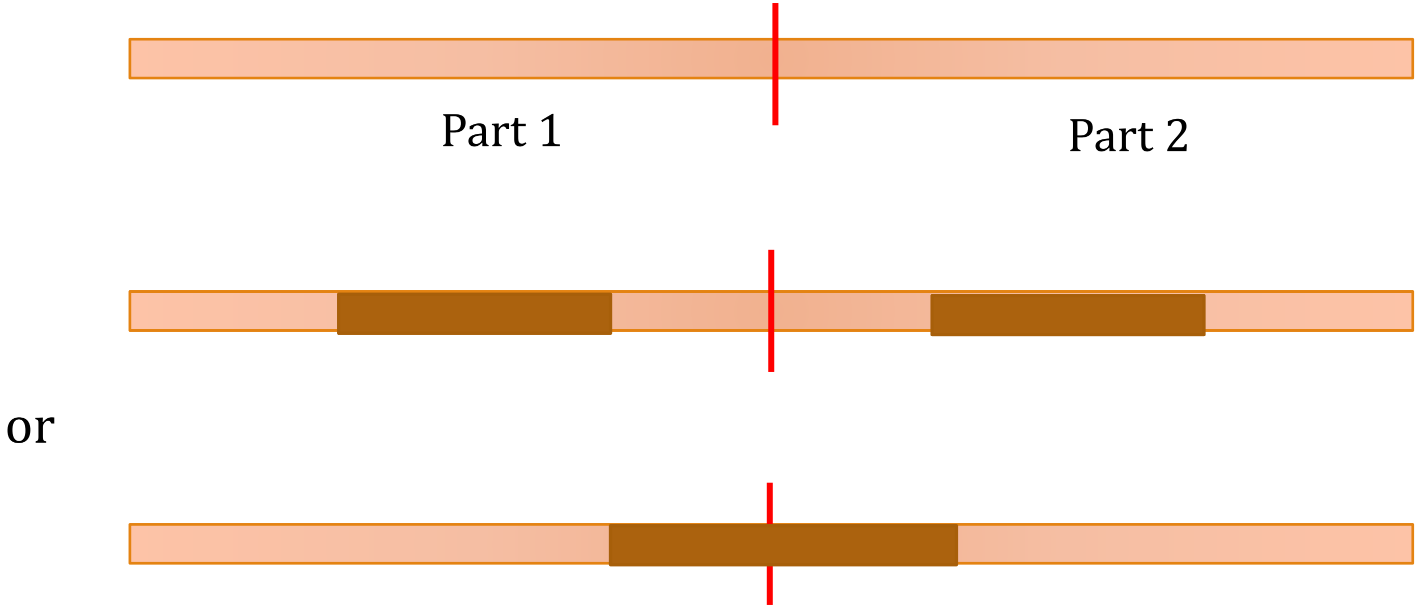
分治策略

- Divide
  - 大问题分成若干个小问题
- Conquer
  - 通过递归，解决小问题
- Combine
  - 组合小问题的结果，解决原始问题

分治策略

- 蛮力递归
  - 问题规模：通常线性减少
    - n, n-1, n-2,...
- 分治递归
  - 问题规模：通常指数减少
    - n, n/2, n/4, n/8,...

最大子序列和-分治策略



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

递归算法分析

- 用递归方程刻画递归算法
- 解递归方程

- 例子：Bit counting

$$T(n)=\begin{cases}0 & n=1\\ T\left(\left\lfloor\frac{n}{2}\right\rfloor\right)+1 & n>1\end{cases}$$

递归算法分析—展开

- 递归方程：$T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + 1$

- 为了简化，假设 $n = 2^k$

- $T(n) = T\left(\frac{n}{2}\right) + 1 = T\left(\frac{n}{4}\right) + 1 + 1 = T\left(\frac{n}{8}\right) + 1 + 1 + 1 = \cdots$
  $T(n) = T\left(\frac{n}{2^k}\right) + \log n = \log n$ (T(1)=0)

递归算法分析—Guess and Prove

- 递归方程：$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$
- Guess
  - $T(n) \in O(n)$?
    - $T(n) \le cn$
  - $T(n) \in O(n^2)$?
    - $T(n) \le cn^2$
  - $T(n) \in O(n\log n)$?
    - $T(n) \le cn\log n$

$T(n) = 2T\left(\frac{n}{2}\right) + n \le 2c\left(\frac{n}{2}\right) + n = (c+1)n$

Fail!

$T(n) = 2T\left(\frac{n}{2}\right) + n \le 2c\left(\frac{n}{2}\right)\log\frac{n}{2} + n$
$= cn\log n - cn\log 2 + n = cn\log n - cn + n$
$\le cn\log n \quad for\ c \ge 1$

- Prove

分治递归

分治策略

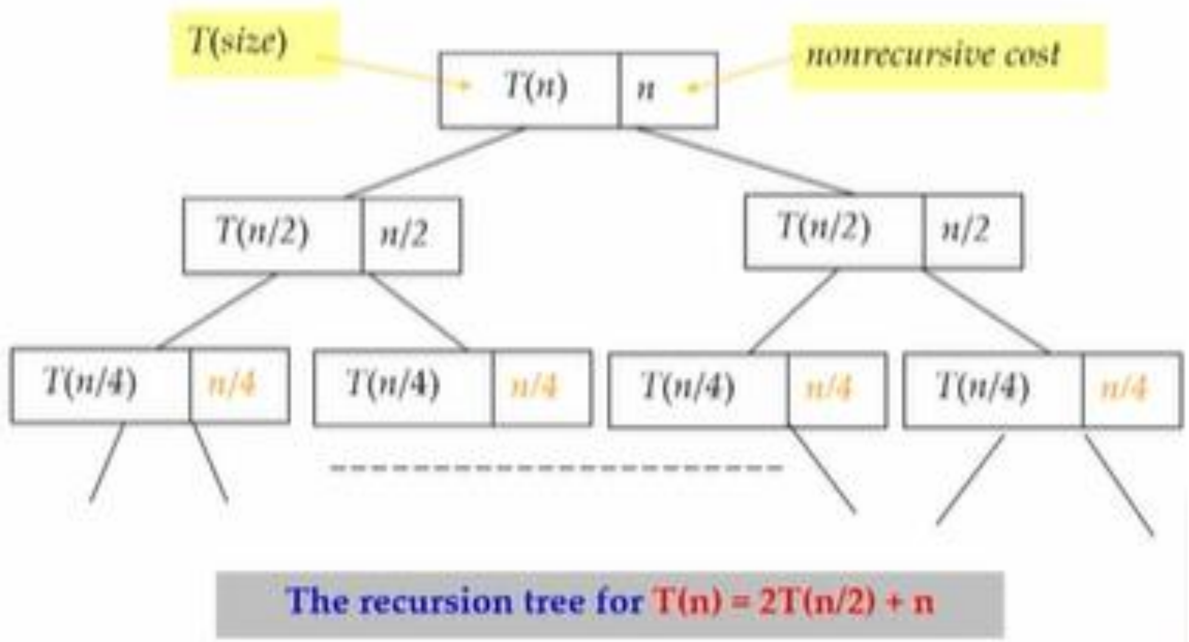
- Divide
- Solve
- Combine

分治递归方程

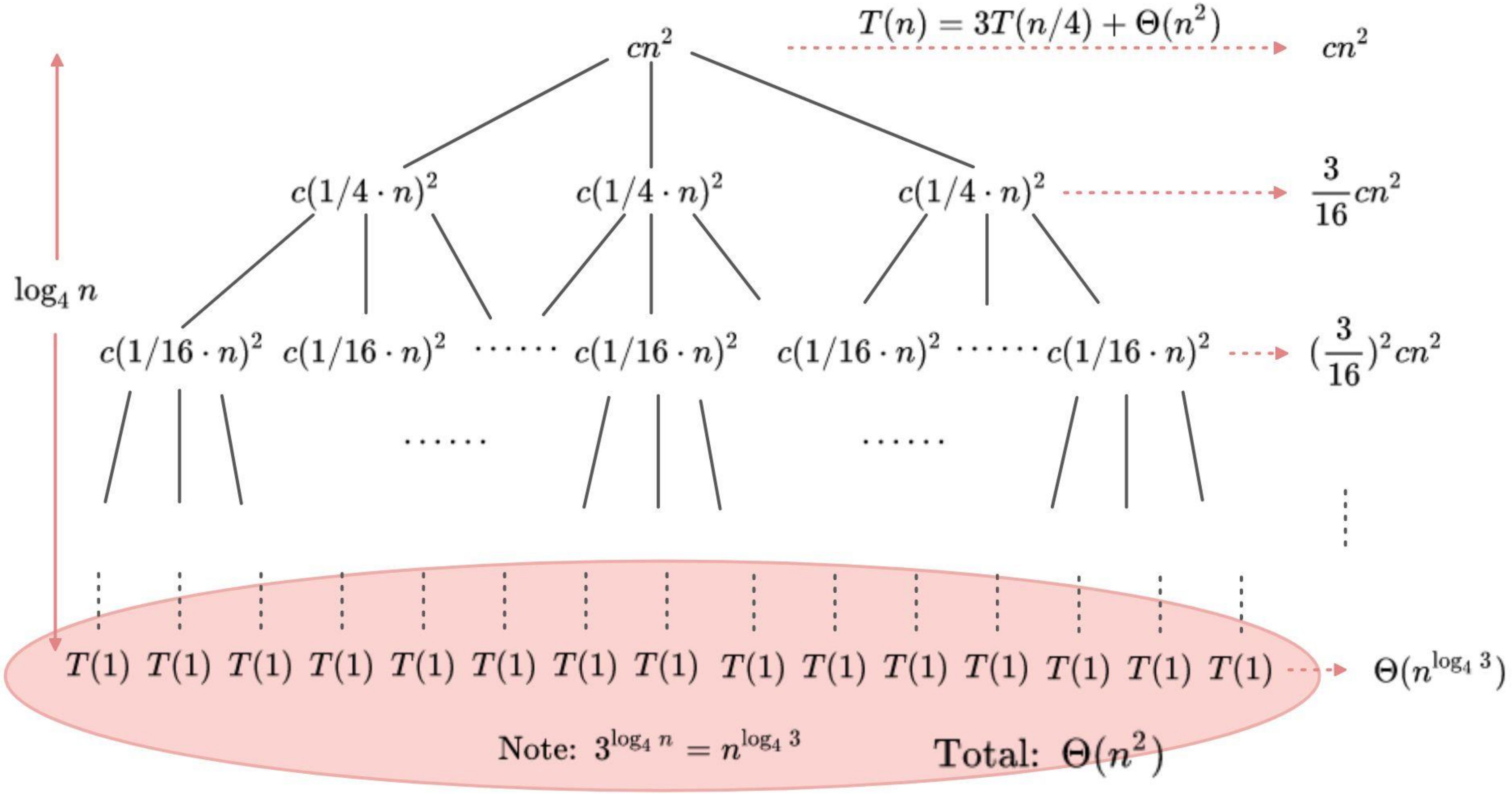
$$T(n) = \underbrace{a}_{\text{划分成}a\text{个子问题}} \cdot \underbrace{T(\frac{n}{b})}_{\text{划分后的子问题规模为原来的}1/b} + \underbrace{f(n)}_{\text{子问题划分与合并的代价}}$$

递归树

- 非叶子节点
  - 非递归代价
  - 递归代价
- 叶子节点
  - base case



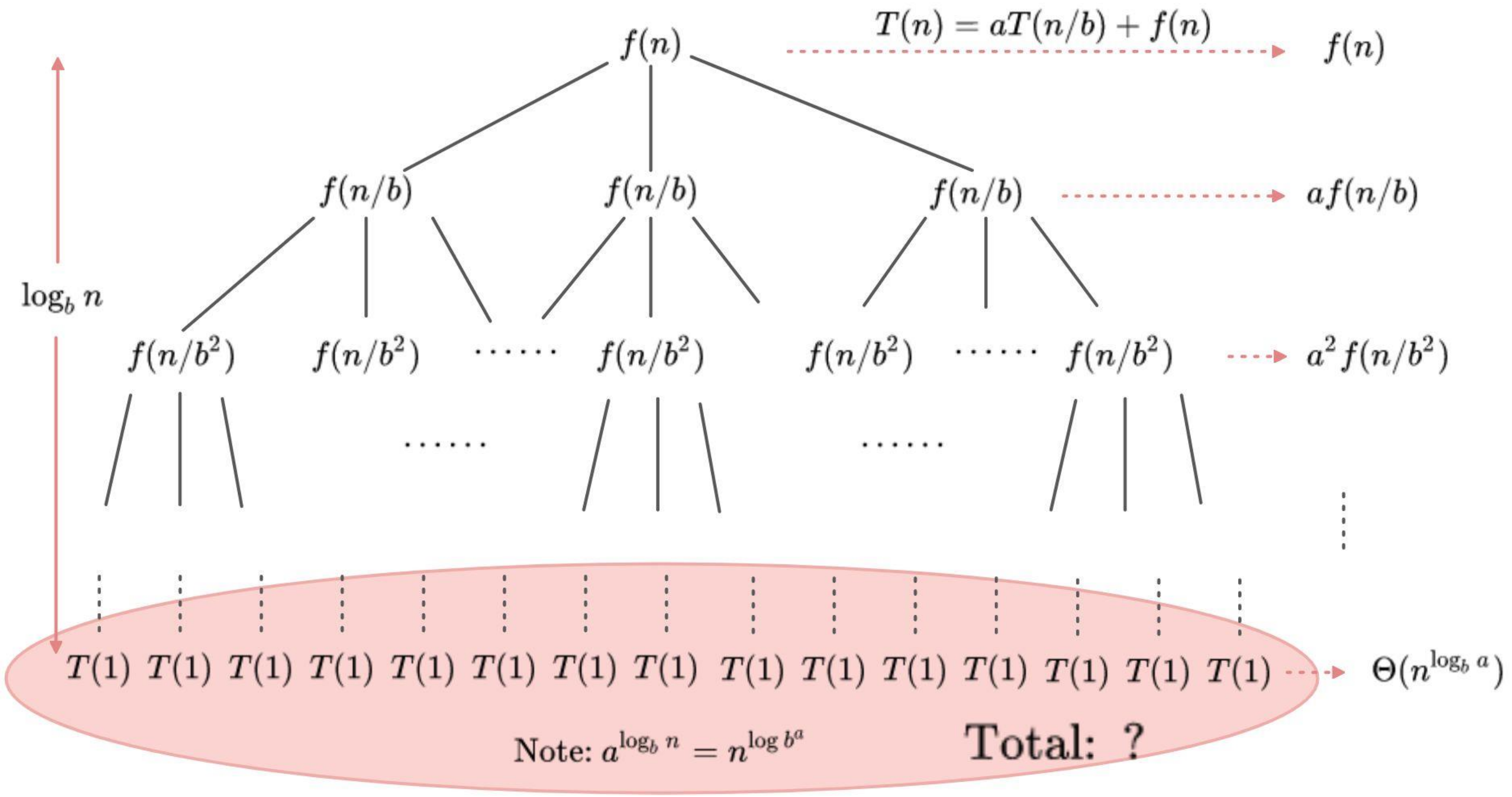
递归树



$T(n) = 3T(n/4) + \Theta(n^2)$
cn^2
$c(1/4 \cdot n)^2$
$c(1/4 \cdot n)^2$
$c(1/4 \cdot n)^2$
$\frac{3}{16}cn^2$
$\log_4 n$
$c(1/16 \cdot n)^2$
$c(1/16 \cdot n)^2$
$\cdots\cdots$
$c(1/16 \cdot n)^2$
$c(1/16 \cdot n)^2$
$\cdots\cdots$
$c(1/16 \cdot n)^2$
$(\frac{3}{16})^2 cn^2$
$T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$
$\Theta(n^{\log_4 3})$
Note: $3^{\log_4 n} = n^{\log_4 3}$
Total: $\Theta(n^2)$

分治递归求解

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- 递归树
  - 递归树的高度$\log_b n$
  - 底层叶节点个数
    $a^{\log_b n} = n^{\log_b a}$



Master 定理

- 基于递归树对所有代价求和，可得到递归方程的解。
  - 逐层求和（sum of row-sums）
  - $f(n), af\left(\frac{n}{b}\right), a^2 f\left(\frac{n}{b^2}\right), \dots \dots, \mathrm{n}^{\log_b a}$

- The 0-th row-sum
  - f(n)，根节点的非递归代价
- The $\log_b n$-th row-sum
  - $n^{\log_b a}$，假设基础情况的代价为1

Master 定理

- 如果 Row-sums 的序列为等比序列
- $f(n), af\left(\frac{n}{b}\right), a^2 f\left(\frac{n}{b^2}\right), \dots \dots, \mathrm{n}^{\log_b a}$

- 公比大于1的等比级数： $T(n) \in \Theta\left(n^{\log_b a}\right)$
- 公比等于1的等比级数：$T(n) \in \Theta(f(n)\log n)$
- 公比小于1的等比级数：$T(n) \in \Theta(f(n))$

Master 定理

- Case 1: $f(n) \in O(n^{E-\varepsilon}), (\varepsilon > 0)$
  - $T(n) \in \Theta(n^E)$

$$E = \log_b a$$

- Case 2: $f(n) \in \Theta(n^E)$
  - $T(n) \in \Theta(f(n)\log n)$

- Case 3: $f(n) \in \Omega(n^{E+\varepsilon}), (\varepsilon > 0)$, 且存在常数 $c < 1$，使得对所有充分大的 $n, af\left(\frac{n}{b}\right) \leq cf(n)$
  - $T(n) \in \Theta(f(n))$

Master 定理

- Example 1： $T(n) = 9T\left(\frac{n}{3}\right) + n$
  - $a = 9, b = 3, E = 2, f(n) = n \in O(n^{E-1})$
  - Case 1: $T(n) \in \Theta(n^2)$
- Example 2: $T(n) = T\left(\frac{2n}{3}\right) + 1$
  - $a = 1, b = \frac{3}{2}, E = 0, f(n) = 1 \in \Theta(n^E)$
  - Case 2: $T(n) \in \Theta(\log n)$
- Example 3: $T(n) = 3T\left(\frac{n}{4}\right) + n\log n$
  - $a = 3, b = 4, E = \log_4 3, f(n) = n\log n \in \Omega(n^{E+\varepsilon})$
  - $af\left(\frac{n}{b}\right) = \frac{3}{4}\mathrm{n}\log\frac{n}{4} = \frac{3}{4}n\log n - \frac{3}{2}n \le \frac{3}{4}n\log n = cf(n),\ c < 1$
  - Case 3: $T(n) \in \Theta(n\log n)$

Thanks