

计算机问题求解 – 论题2-5

- 分治法与递归

2014年04月01日

阅读检查

Occasionally, we shall see recurrences that are not equalities but rather inequalities, such as $T(n) \leq 2T(n/2) + \Theta(n)$. Because such a recurrence states only an upper bound on $T(n)$, we will couch its solution using O -notation rather than Θ -notation. Similarly, if the inequality were reversed to $T(n) \geq 2T(n/2) + \Theta(n)$, then because the recurrence gives only a lower bound on $T(n)$, we would use Ω -notation in its solution.

Mergesort Revisited

MERGE-SORT (A, p, r)

1 **if** $p < r$

2 $q = \lfloor (p + r) / 2 \rfloor$

3 **MERGE-SORT** (A, p, q)

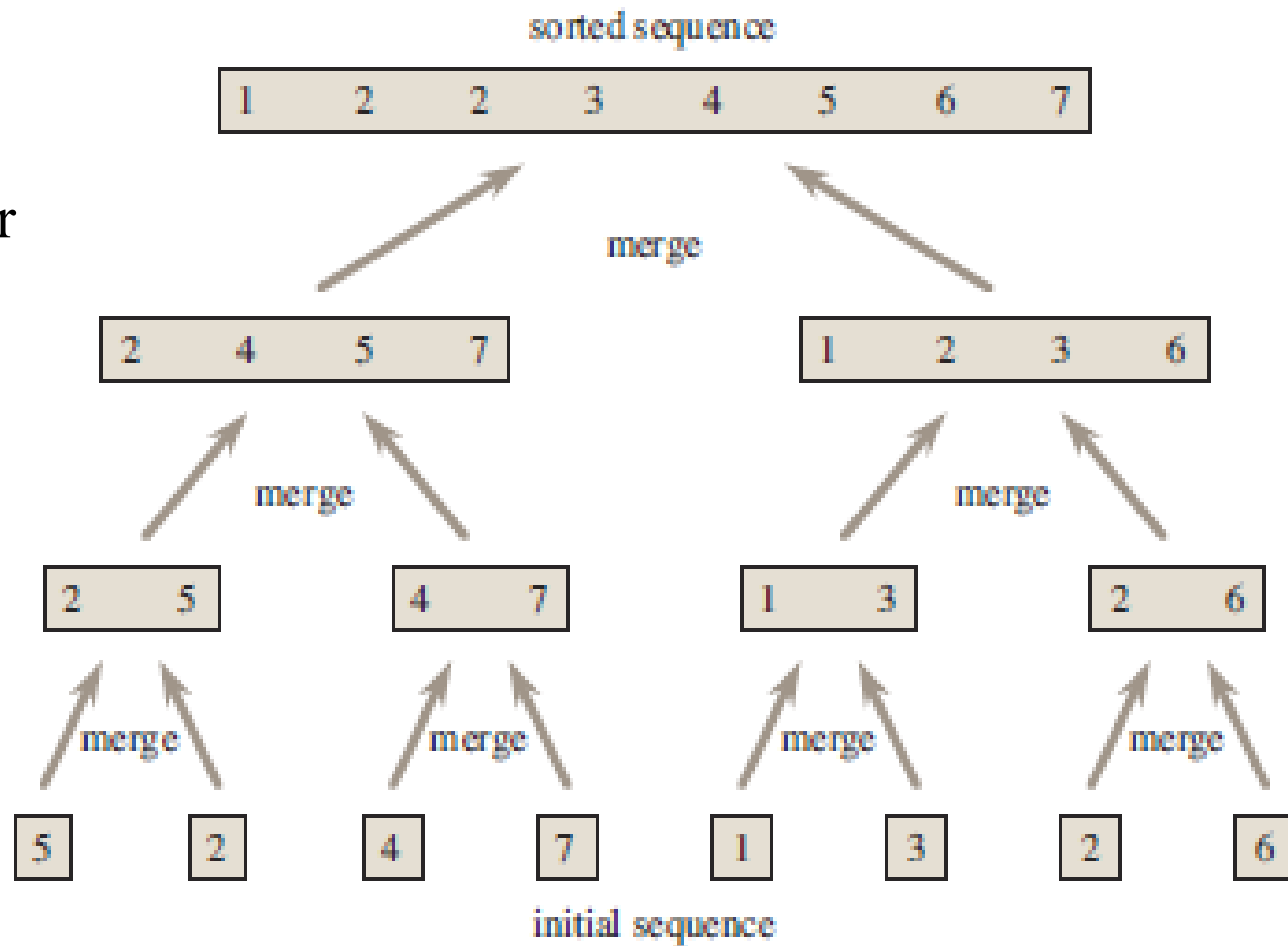
4 **MERGE-SORT** ($A, q + 1, r$)

5 **MERGE** (A, p, q, r)

问题1:

这个算法究竟是如何“排”序的？

↑
conquer
↓



问题2:

人的思维“分而治之”
如何变为算法策略的呢?

问题3: 如何考虑分治算法的代价?

递归代价与非递归代价

导出递归式

MERGE-SORT(A, p, r)

1 if $p < r$

2 $q = \lfloor (p + r)/2 \rfloor$

3 MERGE-SORT(A, p, q)

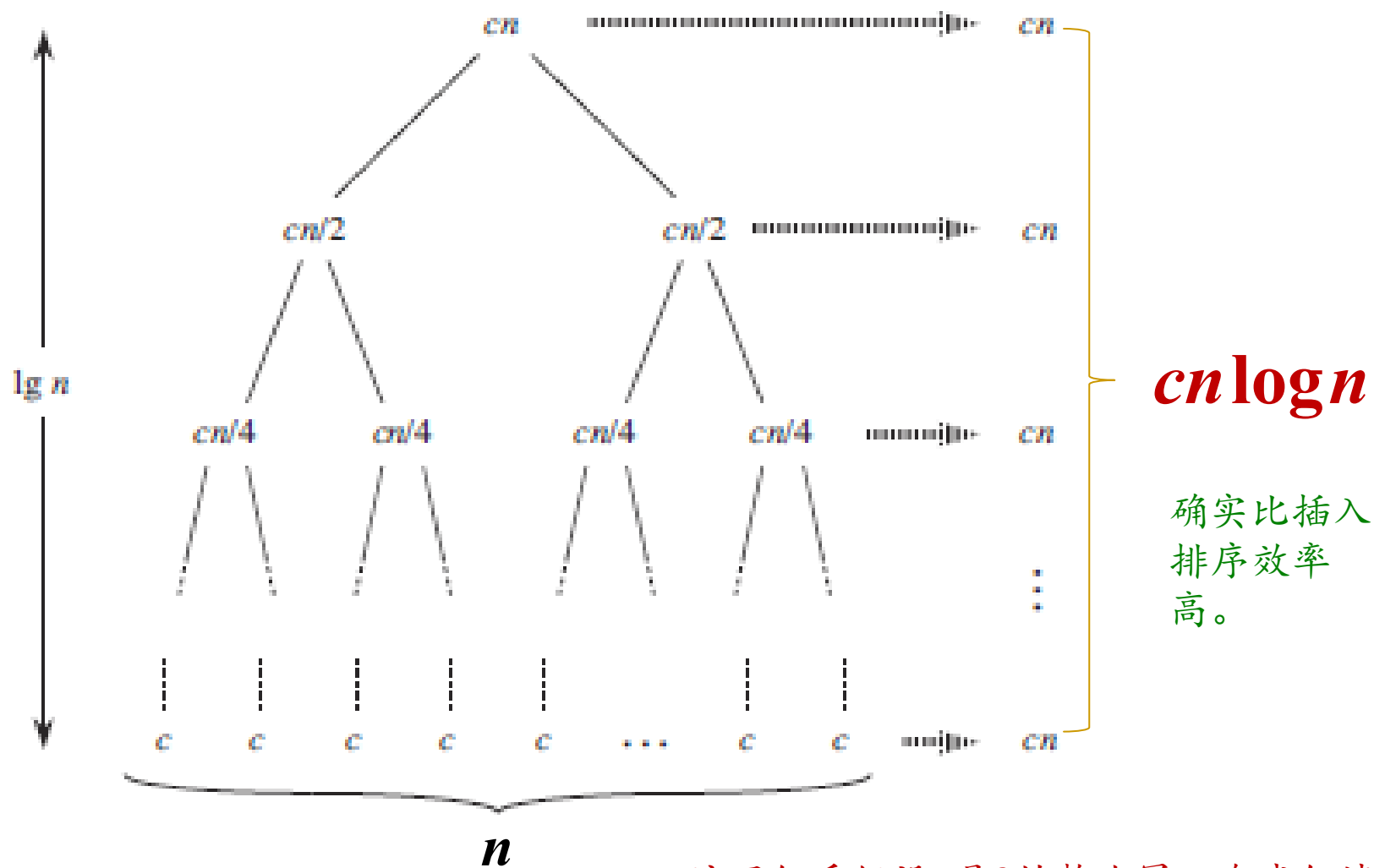
4 MERGE-SORT($A, q + 1, r$)

5 MERGE(A, p, q, r)

两次递归，理想情况下每次问题规模是原来的一半。

非递归开销

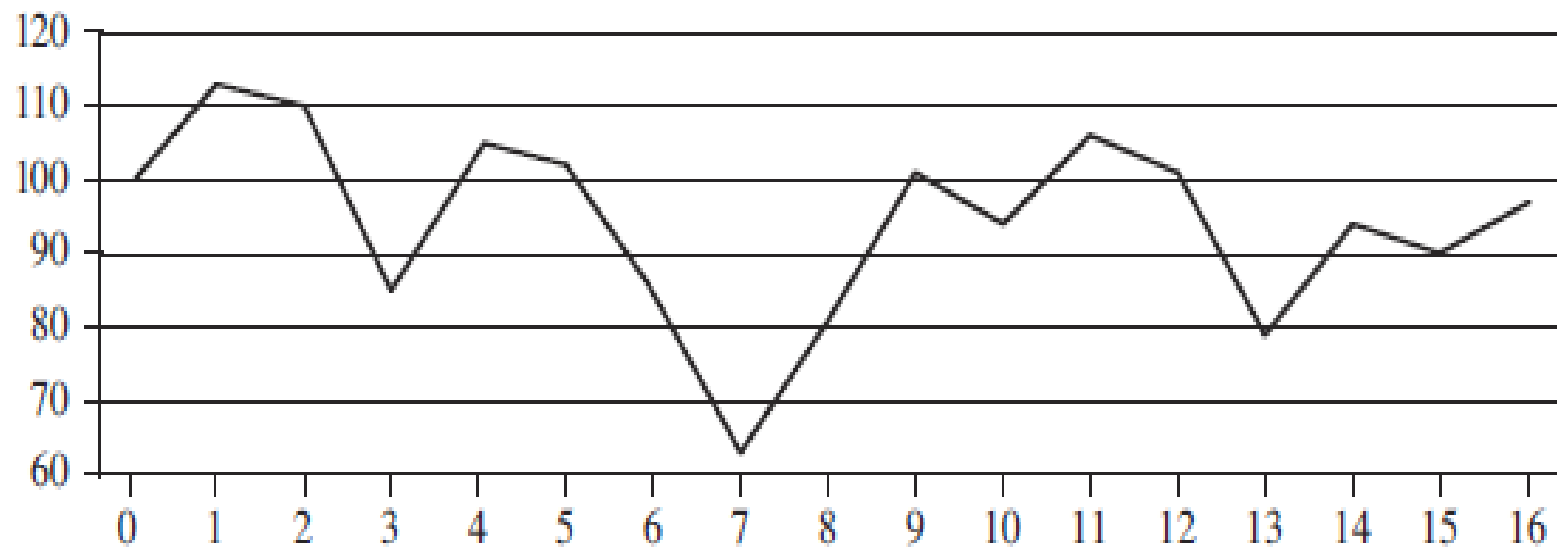
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



这里似乎假设 n 是2的整次幂，在我们涉及的大多数情况下，这不影响结果。

问题4:

书上的投资回报问题是怎样被转化为最大子数组问题的？



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

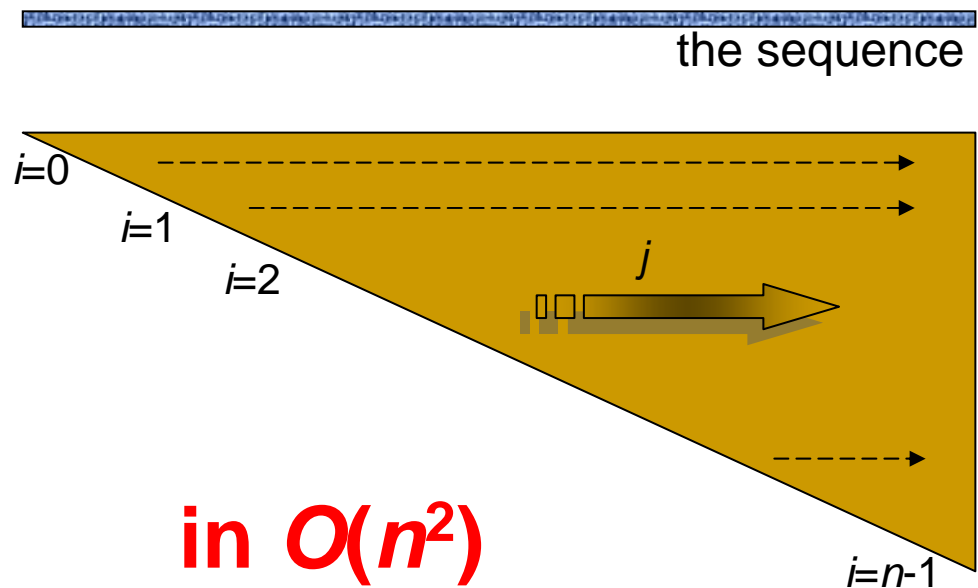
Maximum subarray

简单的遍历所有可能的子序列

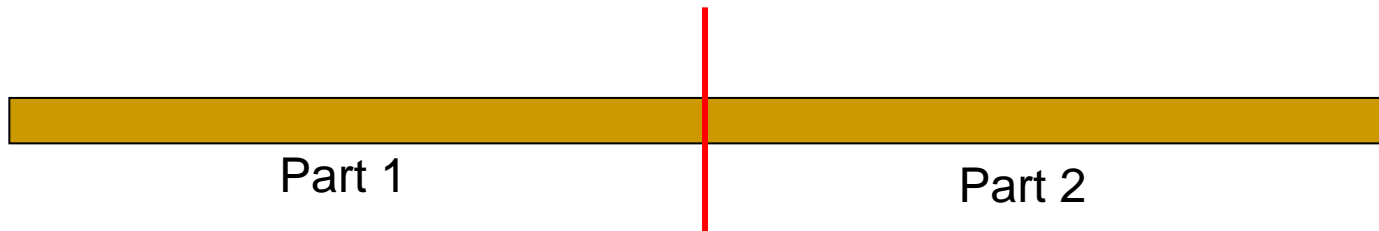
下面的过程遍历的顺序为:

$(0,0), (0,1), \dots, (0,n-1); (1,1), (1,2), \dots, (1,n-1), \dots$
 $(n-2,n-2), (n-2, n-1), (n-1,n-1)$

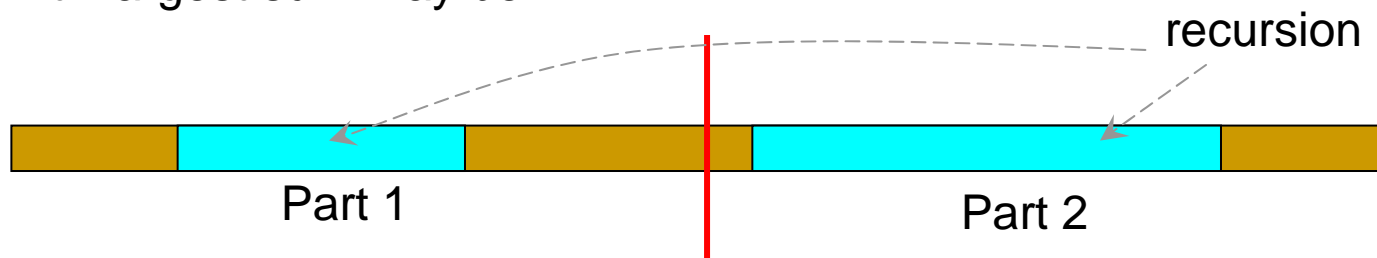
```
MaxSum = 0;
for (i = 0; i < N; i++)
{
    ThisSum = 0;
    for (j = i; j < N; j++)
    {
        ThisSum += A[j];
        if (ThisSum > MaxSum)
            MaxSum = ThisSum;
    }
}
return MaxSum;
```



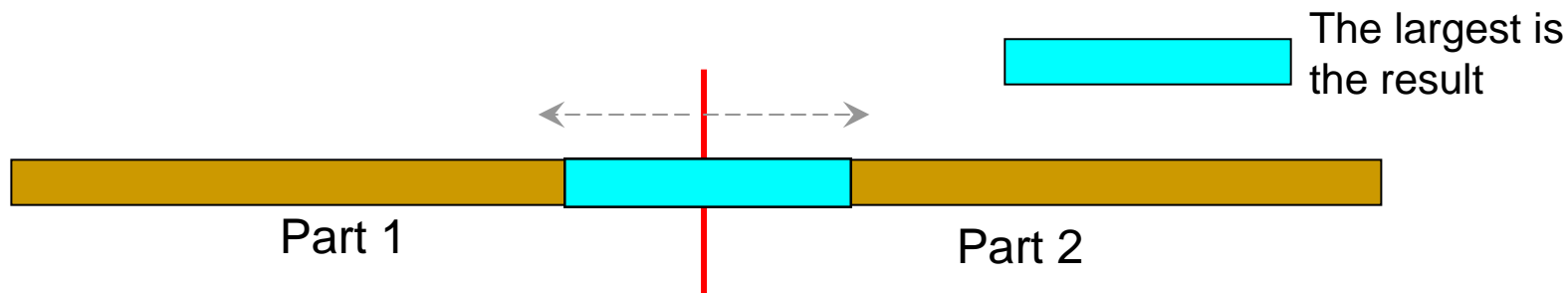
用分治法解最大子数组问题



the sub with largest sum may be in:



or:



问题5：跨中点的部分如何计算？

FIND-MAXIMUM-SUBARRAY(*A*, *low*, *high*)

```
1  if high == low
2      return (low, high, A[low])           // base case: only one element
3  else mid =  $\lfloor (\textit{low} + \textit{high}) / 2 \rfloor$ 
4      (left-low, left-high, left-sum) =
          FIND-MAXIMUM-SUBARRAY(A, low, mid)
5      (right-low, right-high, right-sum) =
          FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6      (cross-low, cross-high, cross-sum) =
          FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7      if left-sum  $\geq$  right-sum and left-sum  $\geq$  cross-sum
8          return (left-low, left-high, left-sum)
9      elseif right-sum  $\geq$  left-sum and right-sum  $\geq$  cross-sum
10         return (right-low, right-high, right-sum)
11     else return (cross-low, cross-high, cross-sum)
```

非递归代价：常量

递归，理想状况下
问题规模是原来的一
半。

非递归代
价：线性

非递归代价：
常量

$O(n \log n)$

矩阵乘法：似乎非得 $\Omega(n^3)$

If $A = (a_{ij})$ and $B = (b_{ij})$ are square $n \times n$ matrices, then in the product $C = A \cdot B$, we define the entry c_{ij} , for $i, j = 1, 2, \dots, n$, by

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

SQUARE-MATRIX-MULTIPLY(A, B)

```
1   $n = A.rows$ 
2  let  $C$  be a new  $n \times n$  matrix
3  for  $i = 1$  to  $n$ 
4      for  $j = 1$  to  $n$ 
5           $c_{ij} = 0$ 
6          for  $k = 1$  to  $n$ 
7               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return  $C$ 
```

Suppose that we partition each of A , B , and C into four $n/2 \times n/2$ matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Equation (4.10) corresponds to the four equations

$$\left. \begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \end{aligned} \right\}$$

1个 n 阶方阵相乘的问题
可以分解为8个 $n/2$ 阶方
阵相乘的子问题。

仍然是立方复杂度

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

问题6:

决定上面的递归代价
比较大的原因是什么?

问题7:

你能否描述Strassen
方法的基本思想?

复杂的组合为了减少一次乘法

$$P_1 = A_{11} \cdot S_1$$

$$P_2 = S_2 \cdot B_{22}$$

$$P_3 = S_3 \cdot B_{11}$$

$$P_4 = A_{22} \cdot S_4$$

$$P_5 = S_5 \cdot S_6$$

$$P_6 = S_7 \cdot S_8$$

$$P_7 = S_9 \cdot S_{10}$$



$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

诸 S_i 只需通过加减法计算

这个算法曾经引起轰动

Strassen's algorithm runs in $O(n^{2.81})$ time, which is asymptotically better than the simple SQUARE-MATRIX-MULTIPLY procedure.

Strassen's method is not at all obvious. (This might be the biggest understatement in this book.)

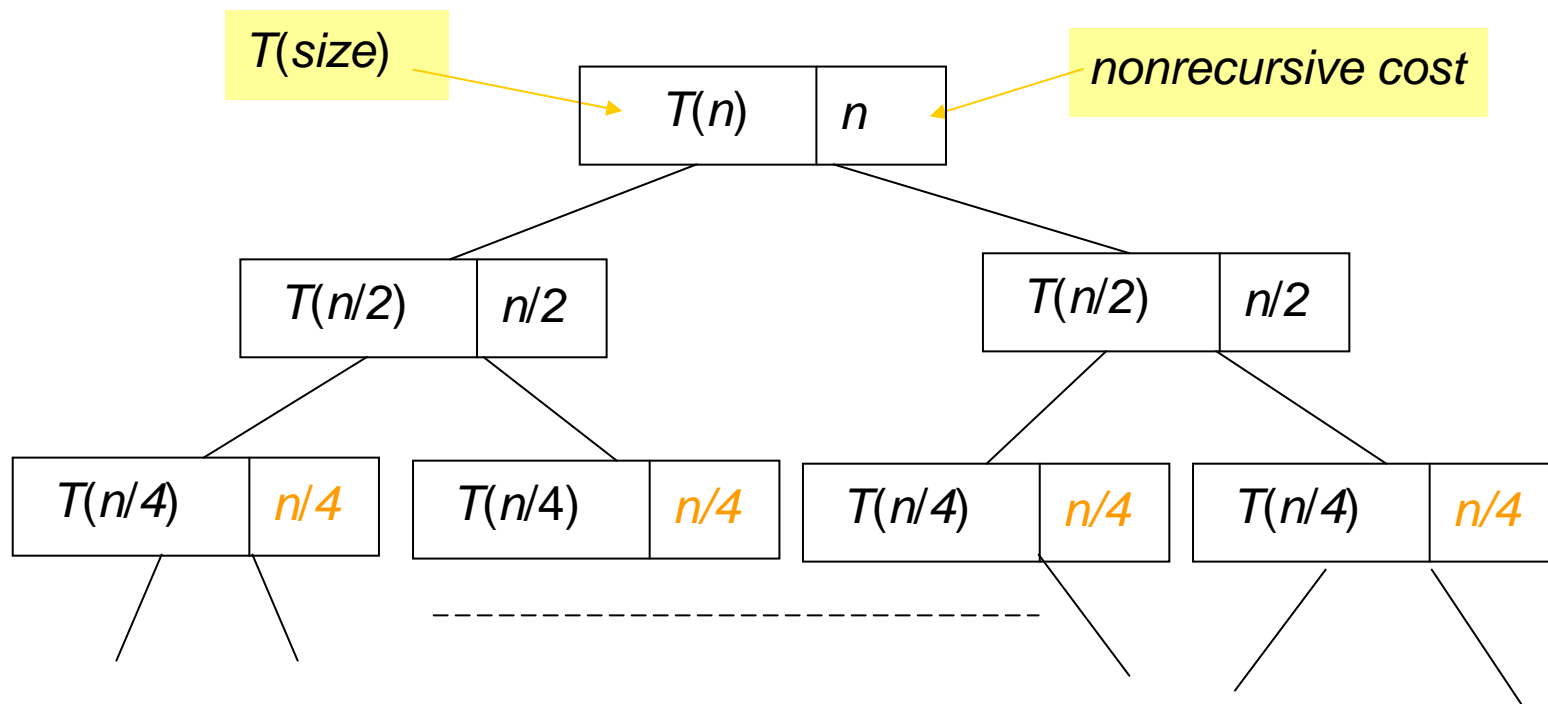
问题8:
你对于这个结果是否有感性认识?

问题9:

为什么降低子问题个数会导致复杂度的阶下降?

递归树

对应于 $T(n)=T(n/2)+T(n/2)+n$ 的递归树



所有节点的非递归开销的和就是算法的开销

树的“高度”和“胖度”决定了节点的多寡

在非递归开销不变的情况下，降低“高度”、缩小“胖度”

递归算法的复杂度分析和算法优化

- 递归算法(分治算法)的复杂度递推公式
 - $T(n) = \sum T(n_i) + f(n)$. 其中:
 - $\sum T(n_i)$ 是分治后诸 n_i 的处理代价和累和
 - $f(n)$ 是划分（**divide**）的代价和合并（**combine**）的代价
 - $T(n)=aT(n/b) + f(n)$, 当诸 n_i 一样大时:
 - a : 分成了几块?
 - 子问题有多少!
 - b :每块的大小
- 如何解上述隐式公式?
 - 代入法、递归树方法、**master**定理

- 问题：分治法的优化主要着眼点应该在哪里？

$$T(n) = \sum T(n_i) + f(n)$$

代入法解隐式递推公式

The *substitution method* for solving recurrences comprises two steps:

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show that the solution works.

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

我们猜测解是： $T(n) = O(n \lg n) \iff T(n) \leq cn \lg n$, c 是某个大于0的常数

将猜测代入：

$$\begin{aligned} T(n) &\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n, \end{aligned}$$

$c \geq 1$ 即可

证明: $T(n) = O(n \lg n)$

- 回忆一下:
- Giving $g:N \rightarrow R^+$, then $O(g)$ is the set of $f:N \rightarrow R^+$, such that for some $c \in R^+$ and some $n_0 \in N$, $f(n) \leq cg(n)$ for all $n \geq n_0$.

必须找到合适的c和n0

证明:

$T(n) \leq cn \lg n$, for all $n \geq n_0$
 c, n_0 是大于0的某个常数

■ 奠基:

□ $n=1$?

$$T(1) \leq c1 \lg 1 = 0$$

□ 选择 $n_0=2$ 进行奠基

■ $T(2)=2T(1)+2=4$

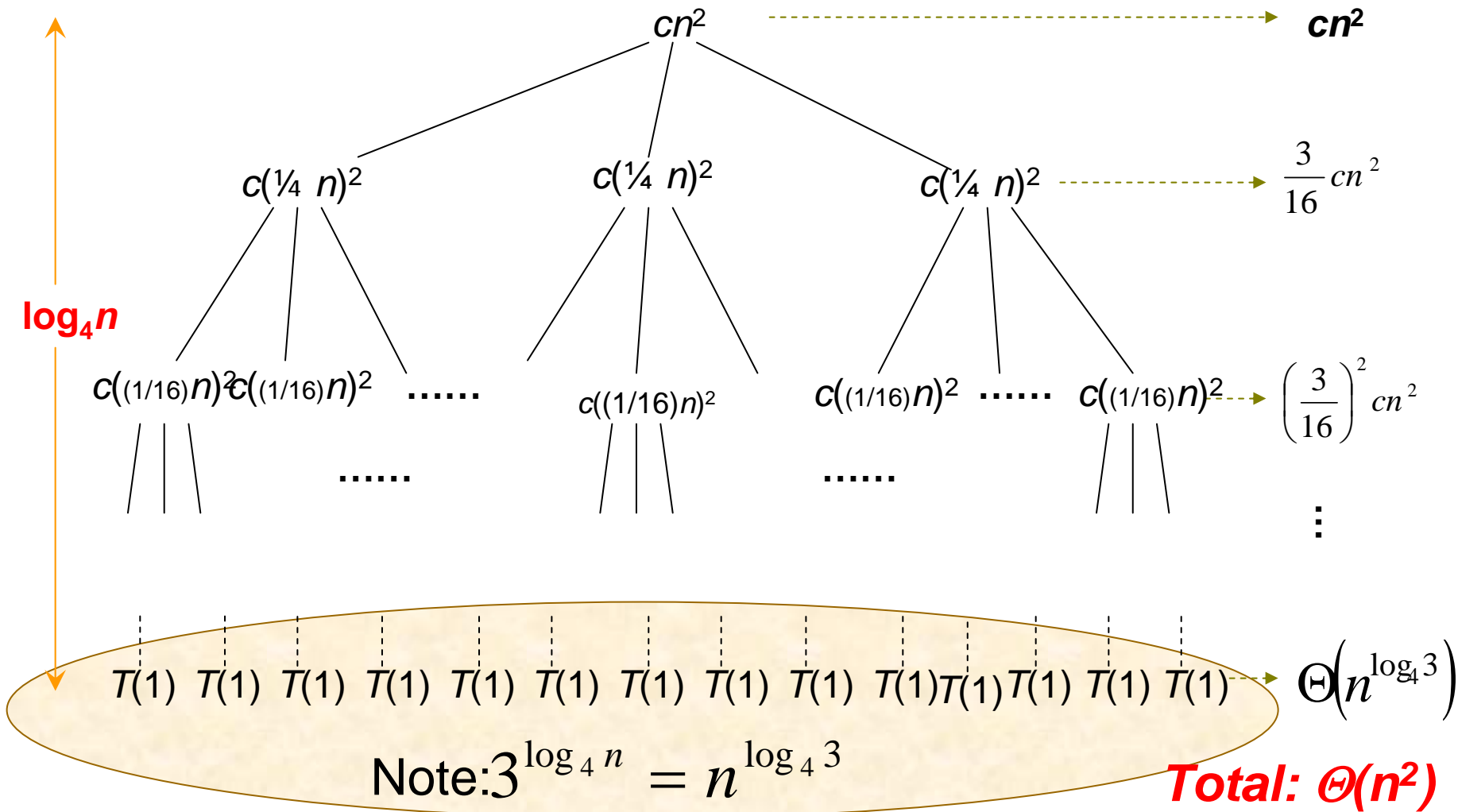
■ 当选定 $c=2$ 时:

□ $cn \lg n = 2 * 2 \lg 2 = 4$

□ 归纳: 略

递归树方法可以直观地
帮助我们猜测结果

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



对应于分治法的递归树方程

- divide-and-conquer:

$$T(n) = aT(n/b) + f(n)$$

- After k th expansion:

$$T(n) = a^k T\left(\frac{n}{b^k}\right) + \sum_{i=1}^{k-1} a^{i-1} f\left(\frac{n}{b^{i-1}}\right)$$

Solving the Divide-and-Conquer

- $T(n)=aT(n/b)+f(n)$ 作为一种典型的分治算法递归式:
 - Let base-cases occur at depth $D(\text{leaf})$, then $n/b^D=1$, that is $D=\lg(n)/\lg(b)$
 - Let the number of leaves of the tree be L , then $L=a^D$, that is $L=a^{(\lg(n)/\lg(b))}$.
 - By a little algebra: $L=n^{\lg(a)/\lg(b)}$, $\lg(a)/\lg(b)=\log_b a$

$$L = b^{\frac{\lg n}{\lg b}} = 2^{\lg a \frac{\lg n}{\lg b}} = 2^{\frac{\lg n}{\lg b} \lg a} = 2^{\lg n \frac{\lg a}{\lg b}} = n^{\log_b a}$$

因此，这个值决定了树的“胖”度。

Divide-and-Conquer: the Solution

- The recursion tree has depth $D = \lg(n) / \lg(c)$, so there are about that many row-sums.
- The 0th row-sum is $f(n)$, the nonrecursive cost of the root.
- The D th row-sum is $n^{\log_b a}$, assuming base cases cost 1, or $\Theta(n^{\log_b a})$ in any event.
- The solution of divide-and-conquer equation is the non-recursive costs of all nodes in the tree, which is the sum of the row-sums.

Solution by Row-sums

- [Little Master Theorem] Row-sums decide the solution of the equation for divide-and-conquer:
 - Increasing geometric series: $T(n) \in \Theta(n^E)$
 - Constant: $T(n) \in \Theta(f(n) \log n)$
 - Decreasing geometric series: $T(n) \in \Theta(f(n))$

This can be generalized to get a result not using explicitly row-sums.

Master Theorem

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

对简单的分治法解矩阵相乘, $\log_b a = 3$ 。

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$. ■

问题10:
在比较两个函数大小时,
Polynomially larger / smaller
是什么意思?

Using Master Theorem

Example 1 $T(n) = 9T\left(\frac{n}{3}\right) + n$

$$a = 9, b = 3, \log_b a = 2, f(n) = n = O(n^{\log_b a - 1}),$$

case 1 applies, $T(n) = \Theta(n^2)$

Example 2 $T(n) = T\left(\frac{2n}{3}\right) + 1$

$$a = 1, b = \frac{3}{2}, \log_b a = 0, f(n) = 1 = \Theta(n^0),$$

case 2 applies, $T(n) = \Theta(\lg n)$

Using Master Theorem

$$T(n) = 3T(n/4) + n \lg n ,$$

we have $a = 3$, $b = 4$, $f(n) = n \lg n$, and $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$. Since $f(n) = \Omega(n^{\log_4 3 + \epsilon})$, where $\epsilon \approx 0.2$, case 3 applies if we can show that the regularity condition holds for $f(n)$. For sufficiently large n , we have that $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$. Consequently, by case 3, the solution to the recurrence is $T(n) = \Theta(n \lg n)$.

Master定理的条件有空隙

- $T(n)=2T(n/2)+n\lg n$
 - $a=2, b=2, \log_b a=1, f(n)=n\lg n$
 - We have $f(n)=\Omega(n^1)$, but no $\varepsilon>0$ satisfies $f(n)=\Omega(n^{1+\varepsilon})$, since $\lg n$ grows slower than n^ε for any small positive ε .
 - So, case 3 doesn't apply.
 - However, neither case 2 applies.

问题11:

在这几个用分治法的例子中, 算法复杂度的阶均比用“brute-force”的方法减低了, 究竟是什么原因呢?

课外作业

- TC p75-: ex.4.1-5;
- TC p.87-: 4.3-3, 4.3-7;
- TC p.92-: 4.4-2, 4.4-8;
- TC p.96-: 4.5-4;
- TC p.107-: 4.1, 4.3, 4.4