计算机问题求解 — 论题2–5

– 分治法与递归

2016年03月24日

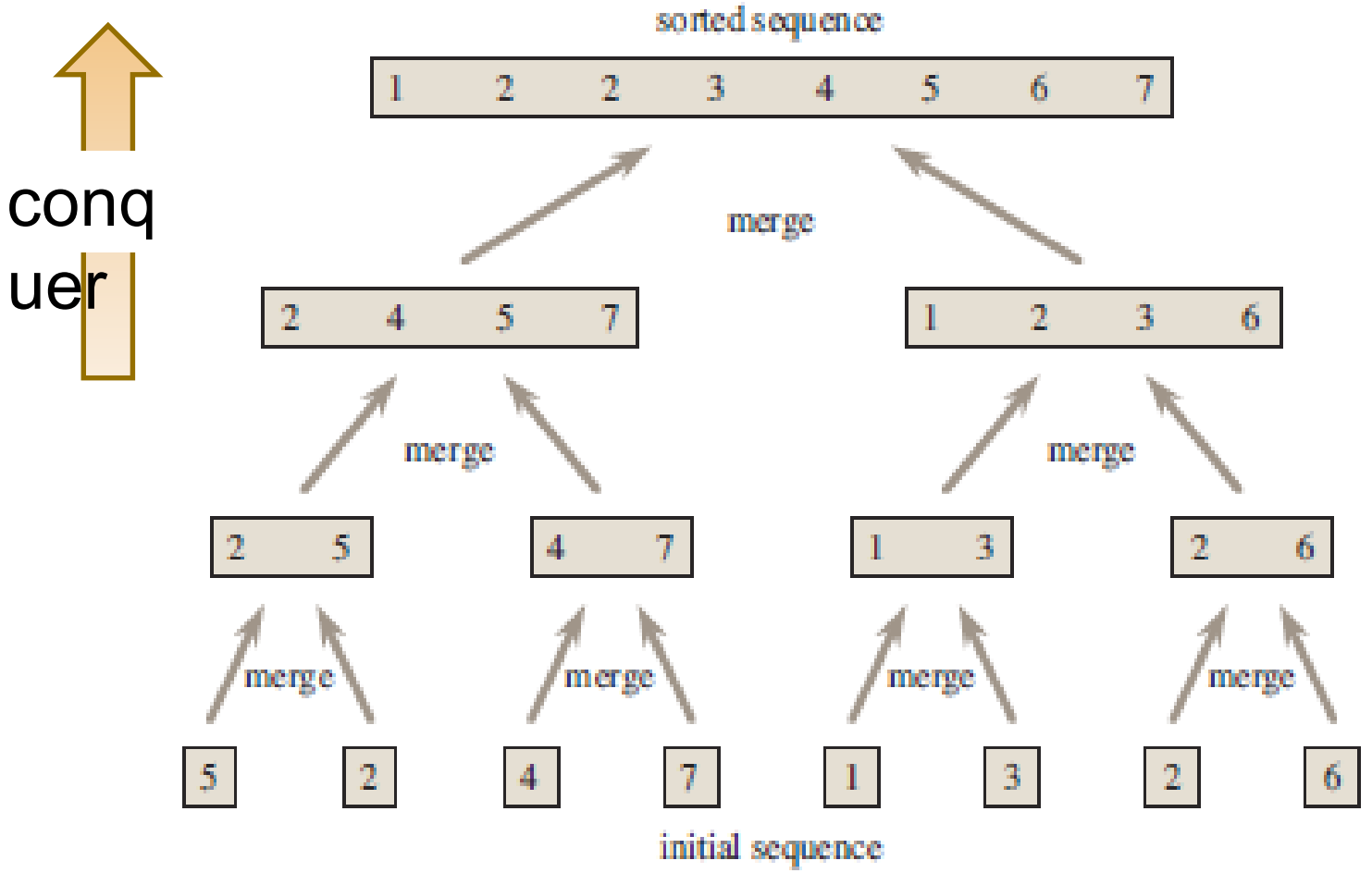
Mergesort Revisited

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
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?

conq
uer


sorted sequence
1 2 2 3 4 5 6 7
merge
2 4 5 7
1 2 3 6
merge
merge
2 5
4 7
1 3
2 6
merge
merge
merge
merge
5
2
4
7
1
3
2
6
initial sequence

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

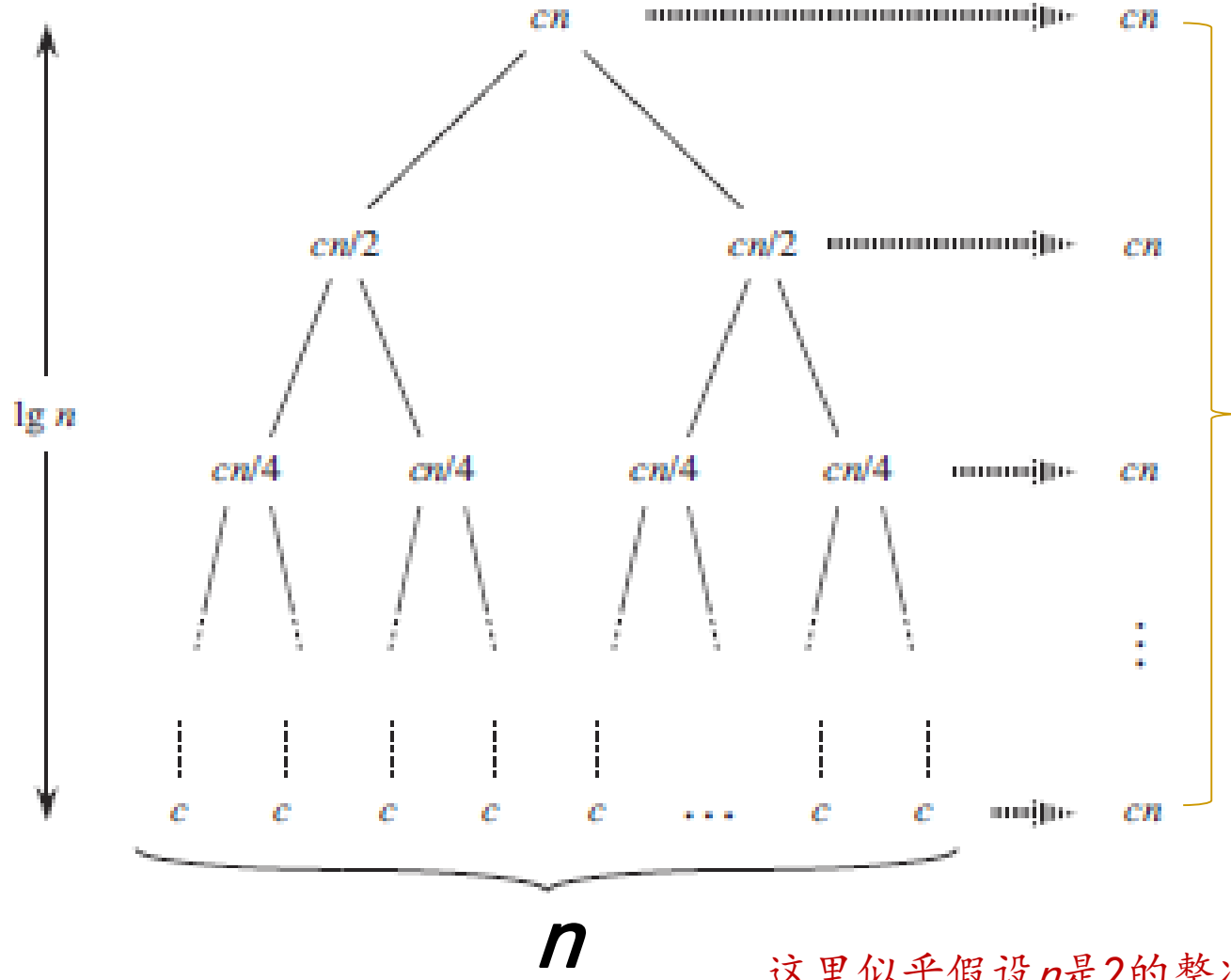
导出递归式

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
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

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



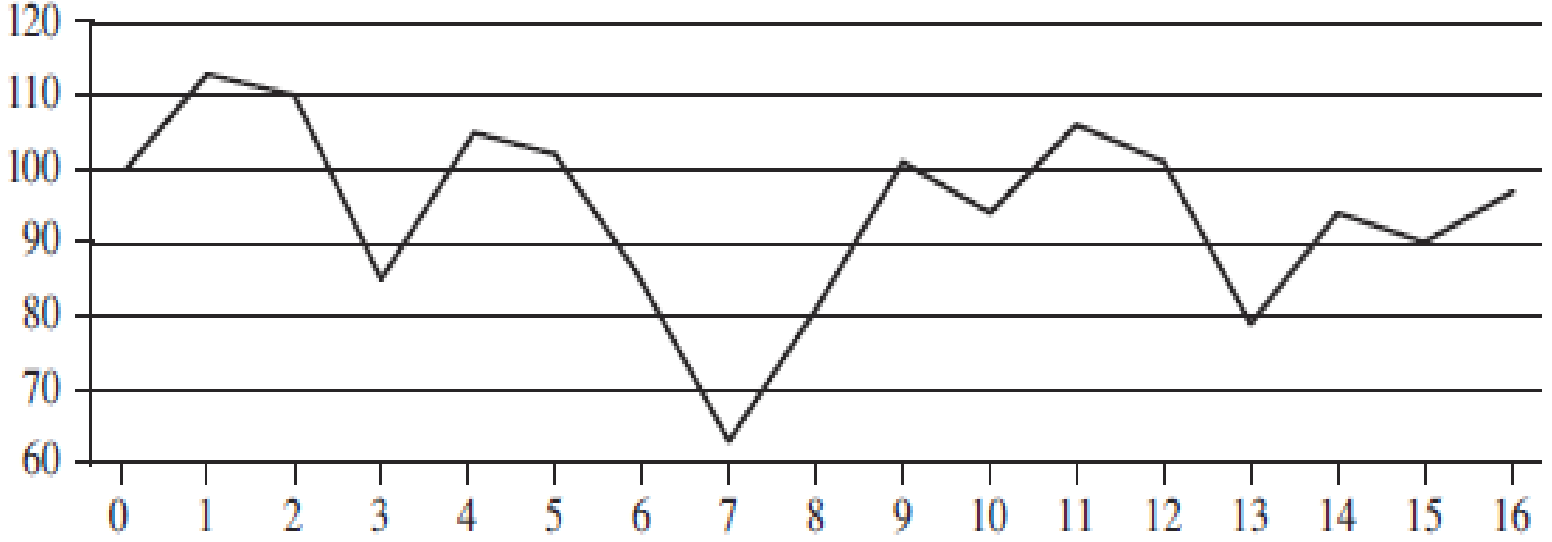
$cn\log n$

确实比插入
排序效率高。

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

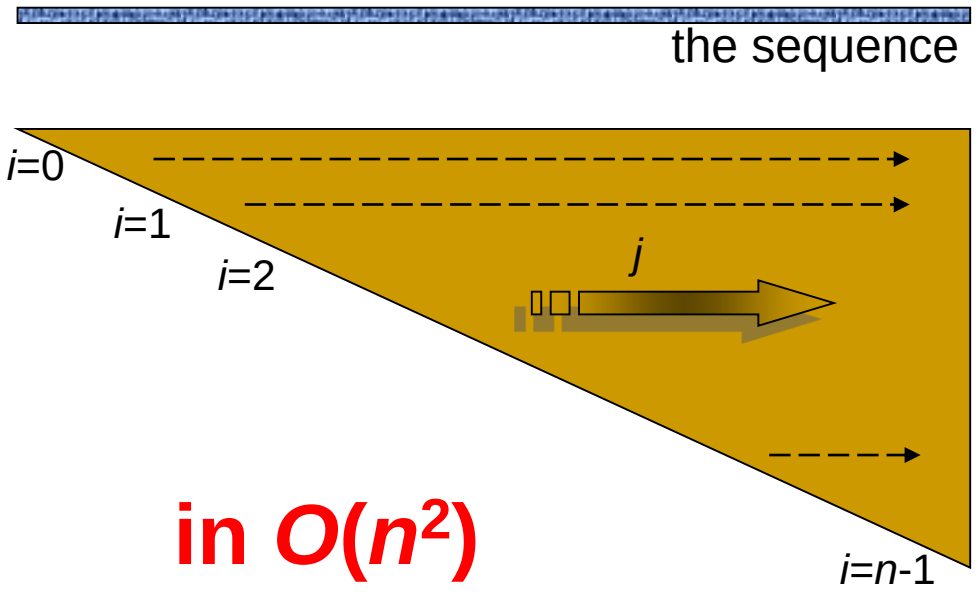
简单的遍历所有可能的子序列

下面的过程遍历的顺序为：

(0,0), (0,1), …, (0,*n*-1); (1,1), (1,2), …, (1,*n*-1), ……

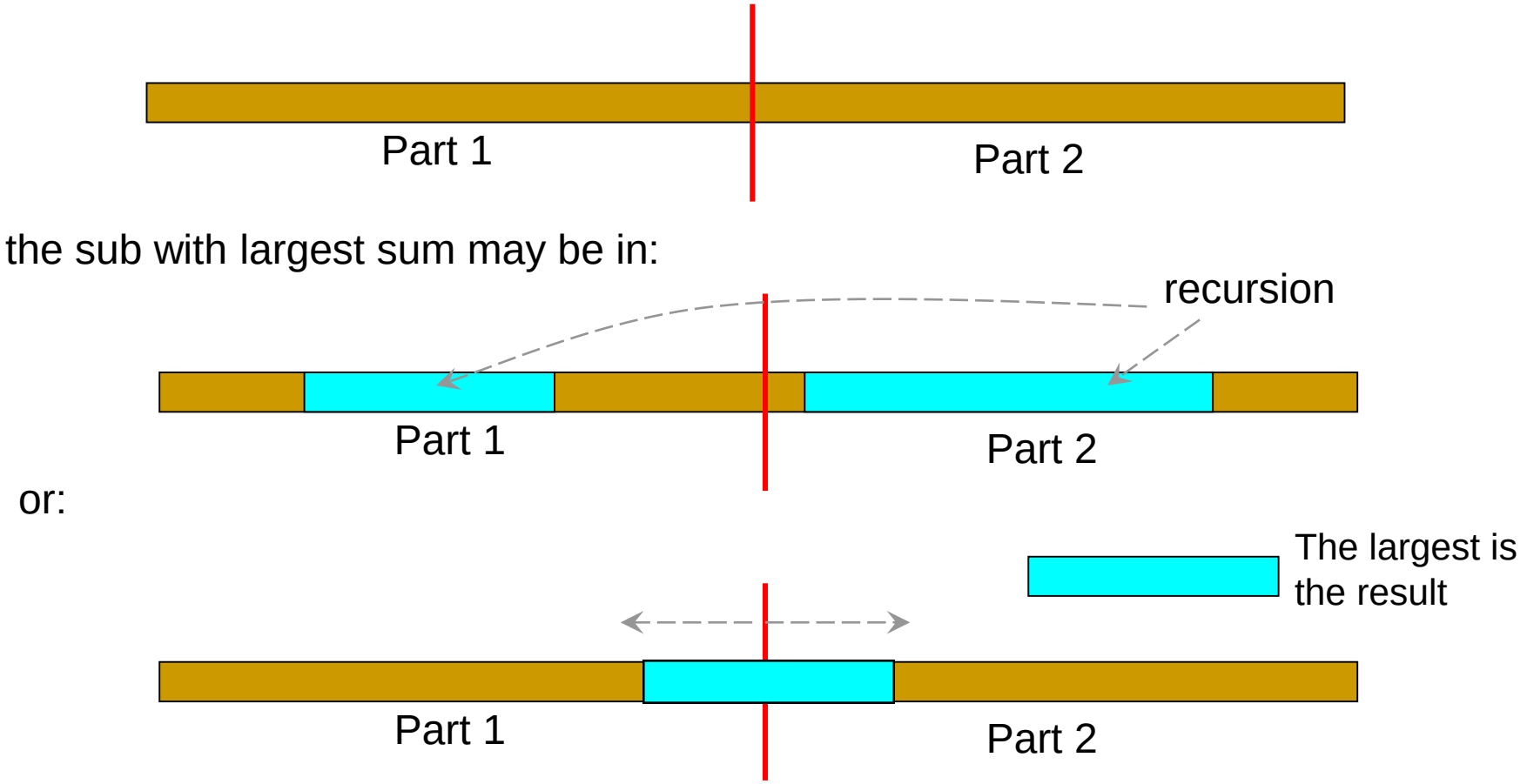
(*n*-2,*n*-2), (*n*-2, *n*-1), (*n*-1,*n*-1)

```
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;
```



in $O(n^2)$

用分治法解最大子数组问题



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 = ⌊(low + high)/2⌋
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 ≥ right-sum and left-sum ≥ cross-sum
8            return (left-low, left-high, left-sum)
9       elseif right-sum ≥ left-sum and right-sum ≥ 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 $i, j = 1, 2, \ldots, n$, 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 × 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 · b_kj
8  return C
```

问题：你能就以下这句话，说清楚Ω,Θ,O和o的区别吗？

You might at first think that any matrix multiplication algorithm must take $\Omega(n^3)$ time, since the natural definition of matrix multiplication requires that many multiplications. You would be incorrect, however: we have a way to multiply matrices in $o(n^3)$ time. In this section, we shall see Strassen's remarkable recursive algorithm for multiplying $n \times n$ matrices. It runs in $\Theta(n^{\lg 7})$ time, which we shall show in Section 4.5. Since $\lg 7$ lies between 2.80 and 2.81, Strassen's algorithm runs in $O(n^{2.81})$ time, which is asymptotically better than the simple SQUARE-MATRIX-MULTIPLY procedure.

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

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

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

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 \\
P_2 &= S_2 \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 \\
P_5 &= S_5 \cdot S_6 \\
P_6 &= S_7 \cdot S_8 \\
P_7 &= S_9 \cdot S_{10}
\end{aligned}
$$

诸S_i只需通过加减法计算

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

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：

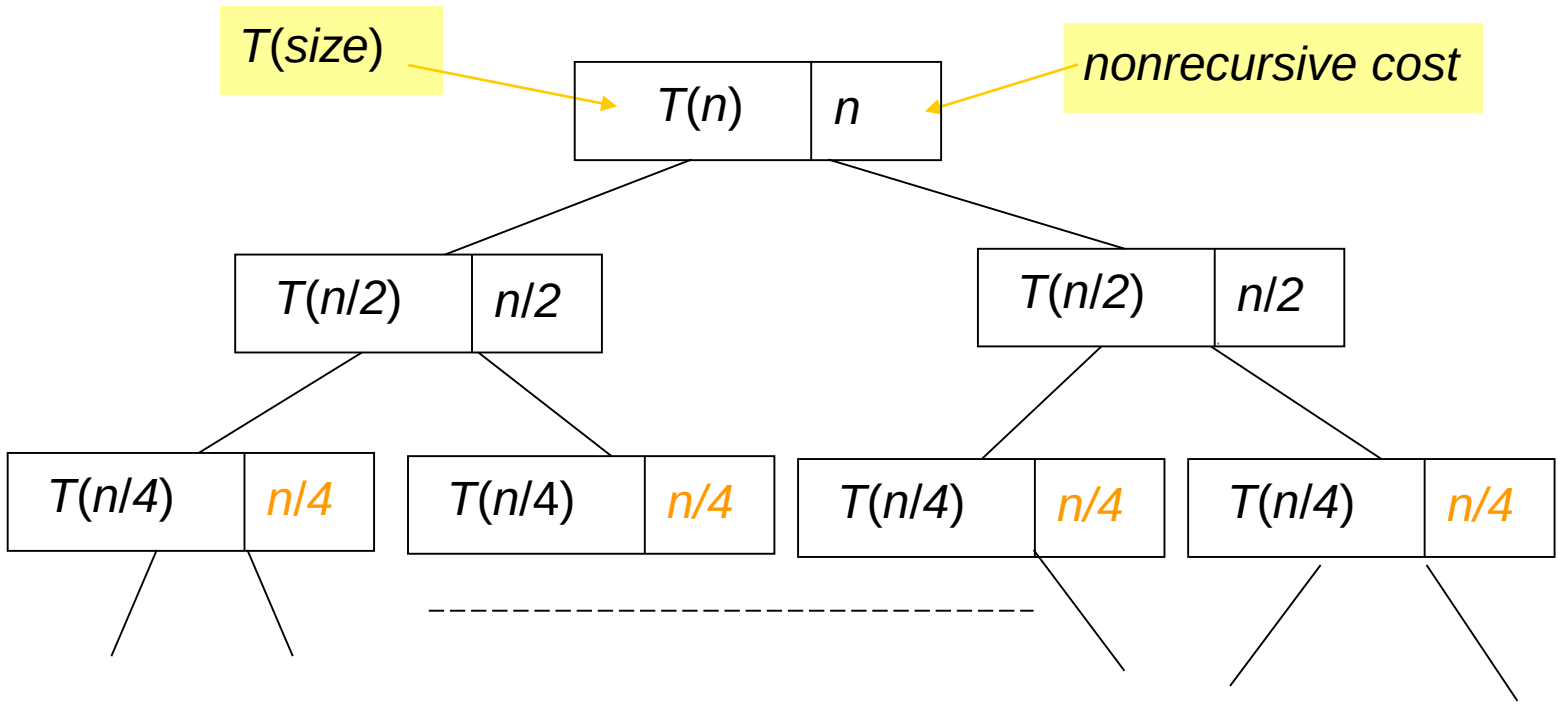
你对于这个结果是否有感性认识？

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”

- 问题： 分治法的优化主要着眼点应该在哪里？

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

分治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ni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ni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

代入法解(隐式)递归公式

问题：什么叫解递归公式？(在算法分析范畴内)

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

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：$T(n) = O(nlgn)$ $\Longleftrightarrow$ $T(n) \le cnlgn, c$是某个大于0的常数

将猜测代入：

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

$c \ge 1$即可

证明：$T(n) = O(nlgn)$

- 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：

$$T(n) \leq cnlgn, for\ all\ n \geq n0$$
$$c, n0\text{是大于}0\text{的某个常数}$$

- 奠基：
  - n=1?

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

  - 选择n0=2进行奠基
    - T(2)=2T(1)+2=4
    - 当选定c=2时：
      - cnlgn=2*2lg2=4
  - 归纳：略

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

但是，如果我们猜测解是T(n)=O(n)，会发生什么？得到什么结论？

将猜测代入：T(n)<=2cn/2+n =cn+n

但是，如果我们猜测解是T(n)=O(n^2)，会发生什么？得到什么结论？

将猜测代入：T(n)<=2c$(n/2)^2$+n =c/2n^2+n<=n^2,只要C>=2即可

问题：T(n)的紧致界就是nlgn吗？我们还缺什么？

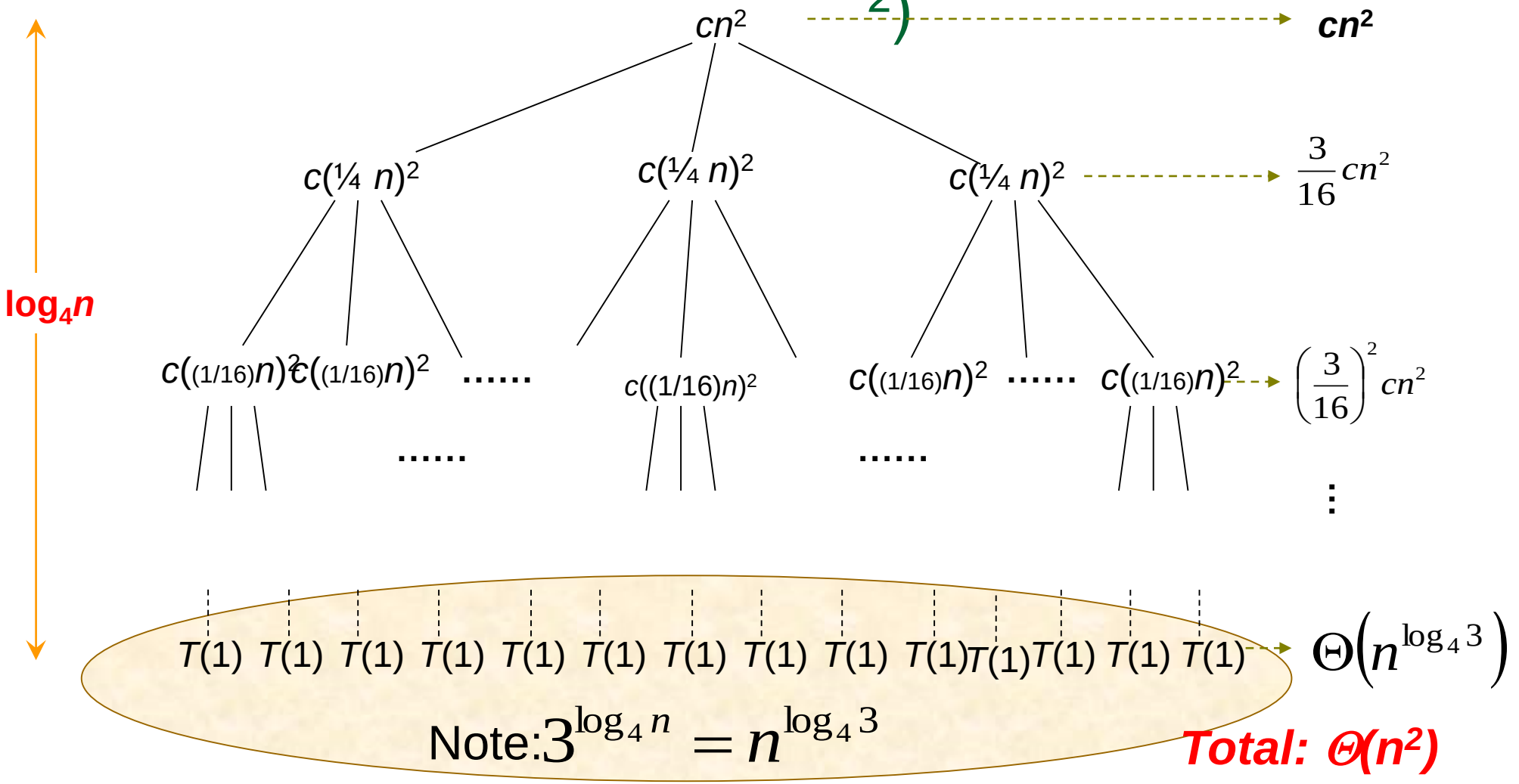
$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

- 我们证明了结论T(n)=O(nlgn)

猜一个递归式的解，容易吗？

递归树方法可以直观地
帮助我们猜测结果

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



Note: $3^{\log_4 n} = n^{\log_4 3}$

Total: $\Theta(n^2)$

分治法一般递归式的递归树分析

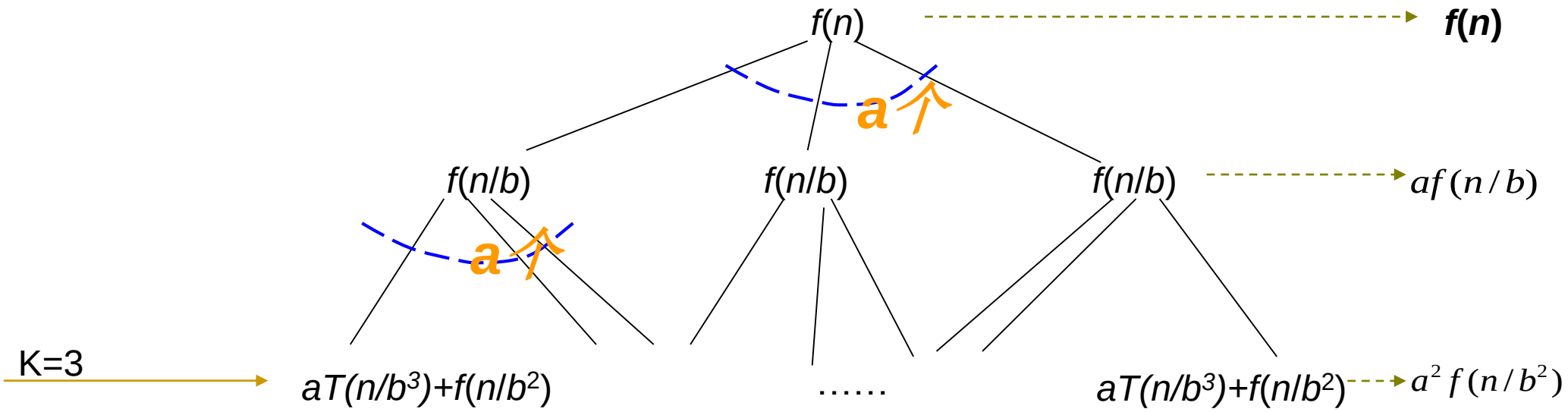
- 一般递归式：

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

  - 递归树扩展到第k层时：

$$T(n) = a^k T\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)$$

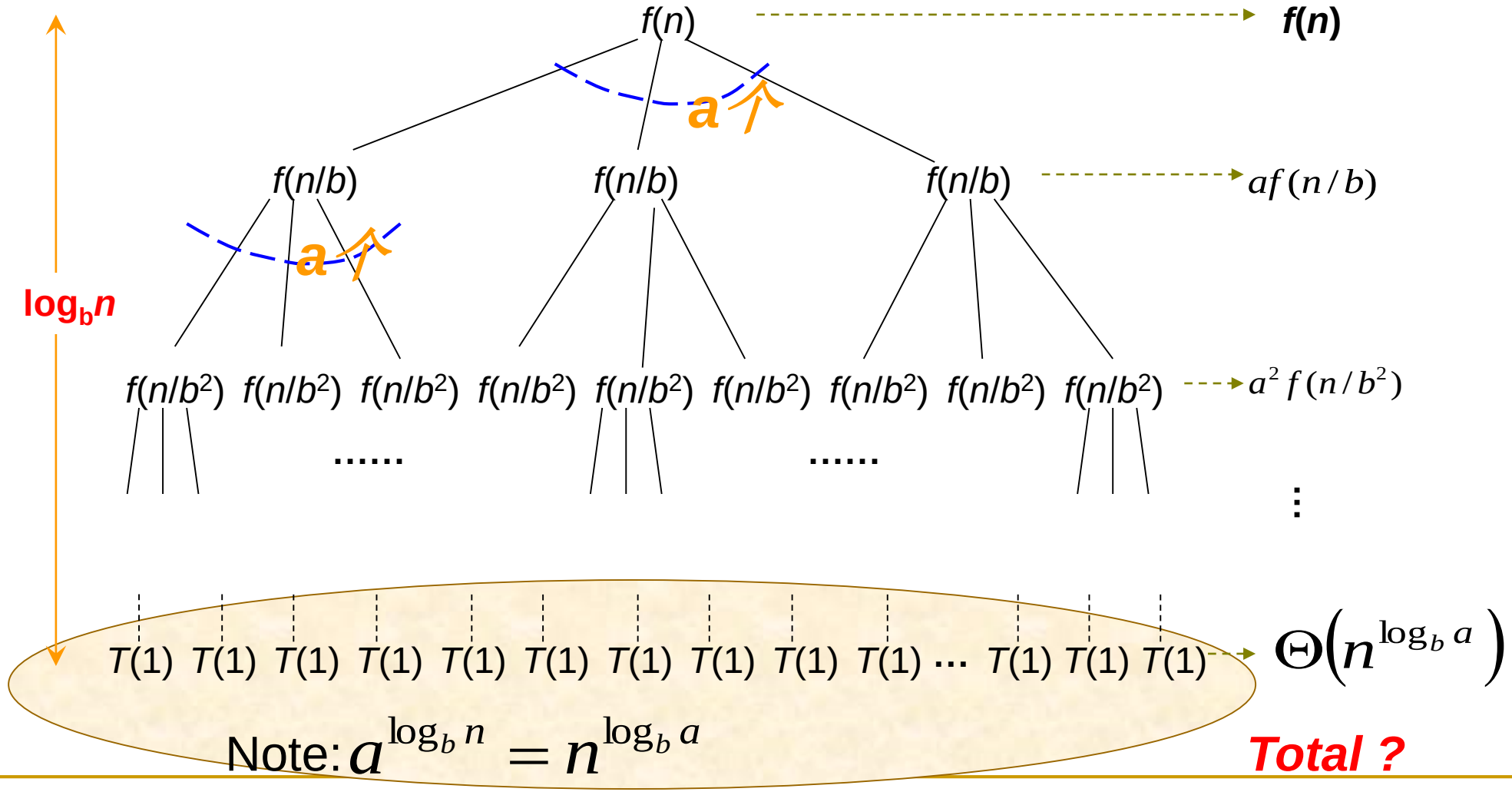
$$T(n)=aT(n/b)+f(n)$$



$$T(n) = a^k T\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)$$

当递归树扩张结束时：

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



$$\text{Total: } \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

单纯从左式看，我们可以怎样思考得到它的简单形式？

问题：当我们将上式和如下递归表达式放在一起时，你能看出上式两个部分分别代表了分治算法中的什么开销吗？

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

提示：a代表了分治为几块；b代表每块的大小；f(n)代表了分解和合并的开销；其中还有一个隐含的T(1)代表了求解小问的开销

令 $g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$ Total: $\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$

直观上，我们可以“相信”这样一些结论：

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $g(n) = O(n^{\log_b a})$. then $T(n) = \Theta(n^{\log_b a})$.

2. If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \lg n)$. then $T(n) = \Theta(n^{\log_b a} \lg n)$

3. If $af(n/b) \leq cf(n)$ for some constant $c < 1$ and for all sufficiently large n, then $g(n) = \Theta(f(n))$

   If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, 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

$$Example1 \quad T(n) = 9T\left(\frac{n}{3}\right) + n$$

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

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

$$Example2 \quad T(n) = T\left(\frac{2n}{3}\right) + 1$$

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

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

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 that n^ε for any small positive ε.
  - So, case 3 doesn't apply.
  - However, neither case 2 applies.

Open Topics:

- 请证明master theorem
- 设计一个算法，用三次实数乘法完成复数a+bi和c+di的相乘。算法需接受a,b,c,d四个参数，生产实部ac-bd和虚部ad+bc

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