



武汉大学

WUHAN UNIVERSITY



第3章 递归

林 海

Lin.hai@whu.edu.cn



3.1 递归

例1 阶乘函数

阶乘函数可递归地定义为：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

边界条件

递归方程

边界条件与递归方程是递归函数的二个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。



3.1 递归

例1 阶乘函数

阶乘函数可递归地定义为：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

边界条件

递归方程

Factorial(n)

 If n==0

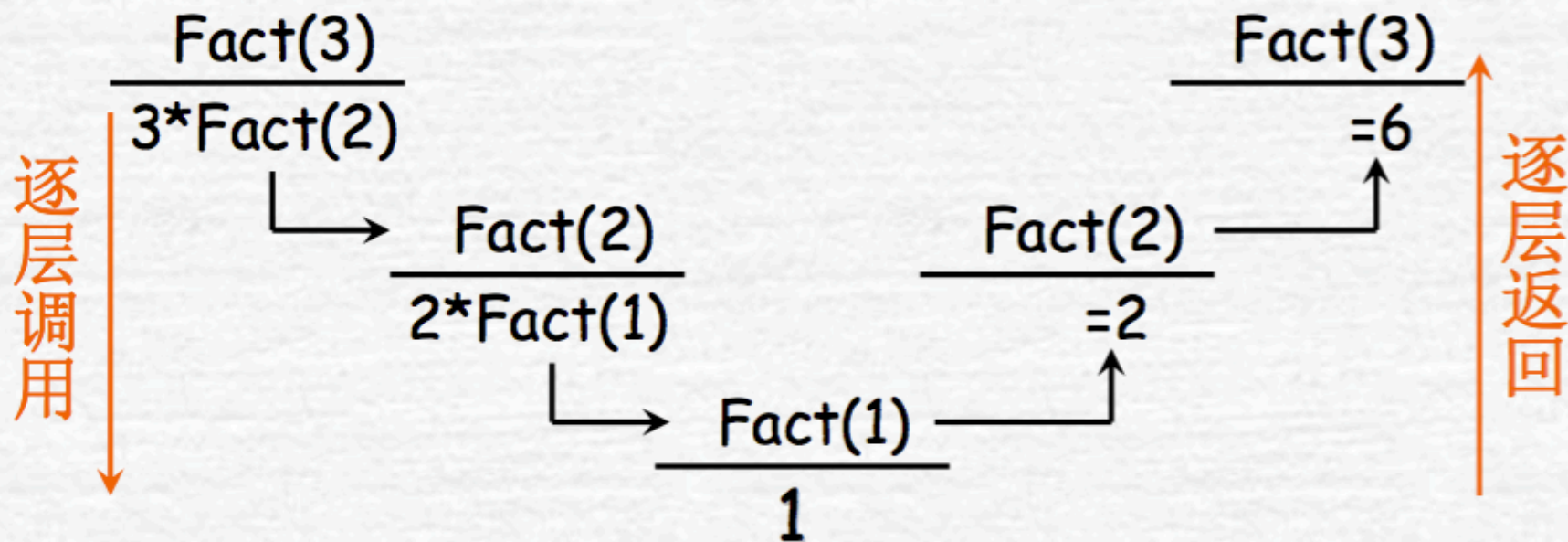
 return 1

 Return n*Factorial(n-1)



3.1 递归

- n 阶乘的定义:
$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n > 0 \end{cases}$$
- 以求 $3!$ 为例的计算过程





3.1 递归

例2 Fibonacci数列

无穷数列1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ..., 被称为Fibonacci数列。它可以递归地定义为：

$$F(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

边界条件

递归方程

第n个Fibonacci数可递归地计算如下：

Fibonacci(n)

if (n <= 1) return 1;

return Fibonacci(n-1)+Fibonacci(n-2);



3.1 递归

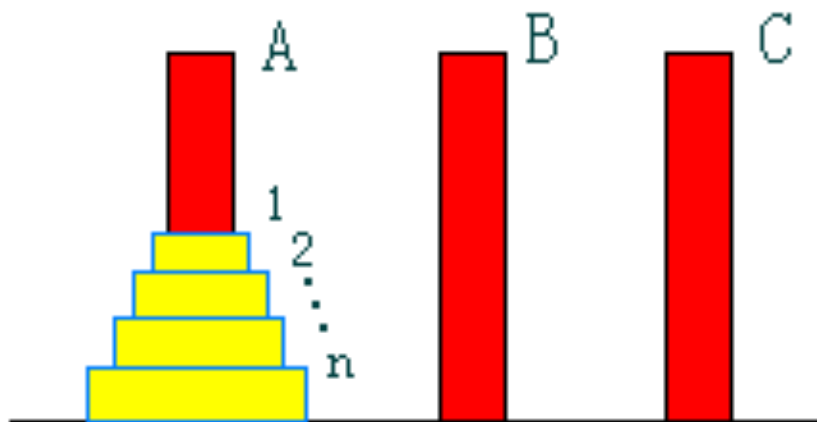
例 Hanoi塔问题

设a,b,c是3个塔座。开始时，在塔座a上有一叠共n个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1,2,...,n,现要求将塔座a上的这一叠圆盘移到塔座b上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：

规则1：每次只能移动1个圆盘；

规则2：任何时刻都不允许将较大的圆盘压在较小的圆盘之上；

规则3：在满足移动规则1和2的前提下，可将圆盘移至a,b,c中任一塔座上。

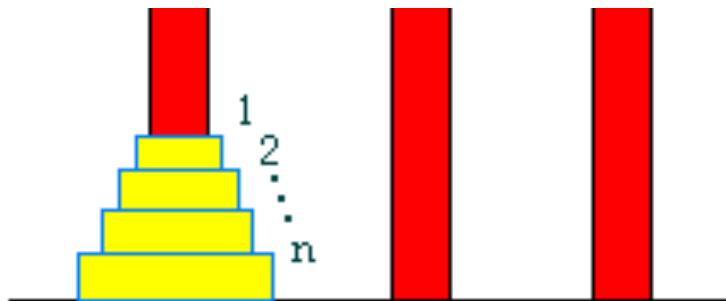




3.1 递归

例6 Hanoi塔问题

```
public static void hanoi(int n, int a, int b, int c)
{
    if (n > 0)
    {
        hanoi(n-1, a, c, b);
        move(a,b);
        hanoi(n-1, c, b, a);
    }
}
```





递归小结

优点：结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。

缺点：递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。



3.2 二分搜索的递归

输入：非降序排列的数组 $A[1..n]$
和元素 x

输出：如果 $x=A[j]$, $1 \leq j \leq n$, 则输出 j , 否则输出0.

1. $low \leftarrow 1; high \leftarrow n; j \leftarrow 0$
2. **while**($low \leq high$) **and** ($j=0$)
3. $mid \leftarrow \lfloor (low+high)/2 \rfloor$
4. **if** $x=A[mid]$ **then** $j \leftarrow mid$
5. **else if** $x < A[mid]$ **then**
 $high \leftarrow mid-1$
6. **else** $low \leftarrow mid+1$
7. **end while**
8. **return** j

```
binarySearch(a, x, right, left)
{ while (left <= right) {
    middle = (left + right)/2;
    if (x == a[middle]) return
middle;
    if (x > a[middle]) return
binarySearch(a, x, right, middle+1);
    else return binarySearch(a, x,
middle+1, left);
}
return -1; // 未找到x
}
```



3.2 二分搜索（递归）

■ 算法复杂度

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

$$\Rightarrow T(n) = T(n/2) + 1 = T(n/2^2) + 1 + 1$$

$$= T(n/2^{\log n}) + \log n = 1 + \log n = \Theta(\log n)$$



3.2 选择排序的递归（归纳）

算法 1.4 SELECTIONSORT

输入： n 个元素的数组 $A[1 \cdots n]$ 。

输出：按非降序排列的数组 $A[1 \cdots n]$ 。

```
1. for  $i \leftarrow 1$  to  $n - 1$ 
2.    $k \leftarrow i$ 
3.   for  $j \leftarrow i + 1$  to  $n$     {查找第  $i$  小的元素}
4.     if  $A[j] < A[k]$  then  $k \leftarrow j$ 
5.   end for
6.   if  $k \neq i$  then 交换  $A[i]$  与  $A[k]$ 
7. end for
```

算法 5.1 SELECTIONSORTREC

输入： n 个元素的数组 $A[1 \cdots n]$ 。

输出：按非降序排列的数组 $A[1 \cdots n]$ 。

1. $sort(1)$

过程 $sort(i)$ {对 $A[i \cdots n]$ 排序}

```
1. if  $i < n$  then
2.    $k \leftarrow i$ 
3.   for  $j \leftarrow i + 1$  to  $n$ 
4.     if  $A[j] < A[k]$  then  $k \leftarrow j$ 
5.   end for
6.   if  $k \neq i$  then 互换  $A[i]$  和  $A[k]$ 
7.    $sort(i + 1)$ 
8. end if
```



3.2 选择排序的递归

复杂度函数：

$$C(n) = \begin{cases} 0 & \text{若 } n = 1 \\ C(n-1) + (n-1) & \text{若 } n \geq 2 \end{cases}$$

这个递推式的解是

$$C(n) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$$

由于运行时间和元素的比较次数成线性关系，所以说它是 $\Theta(n^2)$ 的。



3.2 插入排序的递归

算法 1.5 INSERTIONSORT

输入： n 个元素的数组 $A[1 \cdots n]$ 。

输出：按非降序排列的数组 $A[1 \cdots n]$ 。

```
1. for  $i \leftarrow 2$  to  $n$ 
2.    $x \leftarrow A[i]$ 
3.    $j \leftarrow i - 1$ 
4.   while  $(j > 0)$  and  $(A[j] > x)$ 
5.      $A[j + 1] \leftarrow A[j]$ 
6.      $j \leftarrow j - 1$ 
7.   end while
8.    $A[j + 1] \leftarrow x$ 
9. end for
```

算法 5.2 INSERTIONSORTREC

输入： n 个元素的数组 $A[1 \cdots n]$ 。

输出：按非降序排列的数组 $A[1 \cdots n]$ 。

1. $sort(n)$

过程 $sort(i)$ {对 $A[1 \cdots i]$ 排序}

```
1. if  $i > 1$  then
2.    $x \leftarrow A[i]$ 
3.    $sort(i - 1)$ 
4.    $j \leftarrow i - 1$ 
5.   while  $j > 0$  and  $A[j] > x$ 
6.      $A[j + 1] \leftarrow A[j]$ 
7.      $j \leftarrow j - 1$ 
8.   end while
9.    $A[j + 1] \leftarrow x$ 
10. end if
```



3.2 插入排序的递归

复杂度（平均）：

$$C(n) = \begin{cases} 0 & \text{if } n = 1 \\ C(n-1) + \frac{n}{2} - \frac{1}{n} + \frac{1}{2} & \text{if } n > 1 \end{cases}$$

$$\sum_{i=2}^n \left(\frac{i}{2} - \frac{1}{i} + \frac{1}{2} \right) = \frac{n(n+1)}{4} - \frac{1}{2} - \sum_{i=2}^n \frac{1}{i} + \frac{n-1}{2} = \frac{n^2}{4} + \frac{3n}{4} - \sum_{i=1}^n \frac{1}{i}$$



3.6 生成排序

排列问题

设计一个递归算法生成 n 个元素 $\{r_1, r_2, \dots, r_n\}$ 的全排列。

设 $R = \{r_1, r_2, \dots, r_n\}$ 是要进行排列的 n 个元素， $R_i = R - \{r_i\}$ 。

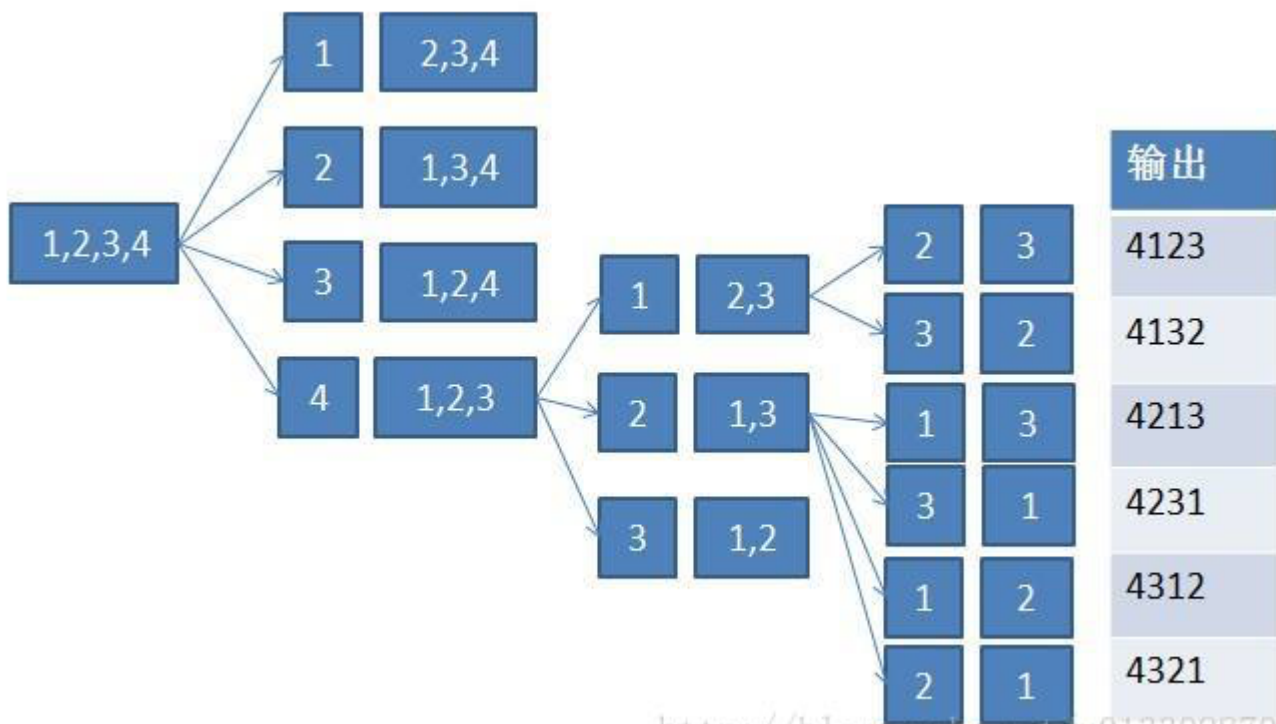
集合 X 中元素的全排列记为 $\text{perm}(X)$ 。

$(r_i)\text{perm}(X)$ 表示在全排列 $\text{perm}(X)$ 的每一个排列前加上前缀得到的排列。 R 的全排列可归纳定义如下：

当 $n=1$ 时， $\text{perm}(R)=(r)$ ，其中 r 是集合 R 中唯一的元素；
当 $n>1$ 时， $\text{perm}(R)$ 由 $(r_1)\text{perm}(R_1)$ ， $(r_2)\text{perm}(R_2)$ ， \dots ， $(r_n)\text{perm}(R_n)$ 构成。



排列问题



<http://blog.csdn.net/u013309870>



3.6 生成排序

排列问题

当 $n=1$ 时, $\text{perm}(R)=(r)$, 其中 r 是集合 R 中唯一的元素;
当 $n>1$ 时, $\text{perm}(R)$ 由 $(r_1)\text{perm}(R_1)$, $(r_2)\text{perm}(R_2)$, ..., $(r_n)\text{perm}(R_n)$ 构成。

```
Perm(list[], k, m)
    if(k == m)
        for(i = 0; i <= m; i++)
            printf("%d ", list[i]);
    else
        for(i = k; i <= m; i++)
            swap(&list[k], &list[i]);
            Perm(list, k + 1, m);
            swap(&list[k], &list[i]);
```



3.6 生成排序

复杂度（迭代次数）：

$$f(n) = \begin{cases} 0 & \text{若 } n = 1 \\ nf(n-1) + n & \text{若 } n \geq 2 \end{cases}$$

求解得：

$$f(n) = n! h(n) = n! \sum_{j=1}^{n-1} \frac{1}{j!} < 2n!$$

所以，算法的运行时间由输出语句来决定，也就是 $\Theta(nn!)$ 。



5.7 整数划分

整数划分问题

将正整数 n 表示成一系列正整数之和： $n=n_1+n_2+\dots+n_k$ ，
其中 $n_1\geq n_2\geq\dots\geq n_k\geq 1$ ， $k\geq 1$ 。

正整数 n 的这种表示称为正整数 n 的划分。求正整数 n 的不同划分个数。

例如正整数6有如下11种不同的划分：

6；

5+1；

4+2, 4+1+1；

3+3, 3+2+1, 3+1+1+1；

2+2+2, 2+2+1+1, 2+1+1+1+1；

1+1+1+1+1+1。



5.7 整数划分

整数划分问题

前面的几个例子中，问题本身都具有比较明显的递归关系，因而容易用递归函数直接求解。

在本例中，如果设 $p(n)$ 为正整数 n 的划分数，则难以找到递归关系，因此考虑增加一个自变量：将最大加数 n_1 不大于 m 的划分个数记作 $q(n, m)$ 。可以建立 $q(n, m)$ 的如下递归关系。

(1) $q(n, 1) = 1, n > 1$

(2) $q(n, m) = q(n, n), m > n$

(3) $q(n, n) = 1 + q(n, n-1);$

正整数 n 的划分由 $n_1 = n$ 的划分和 $n_1 \leq n-1$ 的划分组成。

(4) $q(n, m) = q(n, m-1) + q(n-m, m), n > m > 1;$

正整数 n 的最大加数 n_1 不大于 m 的划分由 $n_1 = m$ 的划分和 $n_1 \leq m-1$ 的划分组成。



5.7 整数划分

例 整数划分问题

前面的几个例子中，问题本身都具有比较明显的递归关系，因而容易用递归函数直接求解。

在本例中，如果设 $p(n)$ 为正整数 n 的划分数，则难以找到递归关系，因此考虑增加一个自变量：将最大加数 n_1 不大于 m 的划分个数记作 $q(n, m)$ 。可以建立 $q(n, m)$ 的如下递归关系。

$$q(n, m) = \begin{cases} 1 & n = 1, m = 1 \\ q(n, n) & n < m \\ 1 + q(n, n - 1) & n = m \\ q(n, m - 1) + q(n - m, m) & n > m > 1 \end{cases}$$

正整数 n 的划分数 $p(n) = q(n, n)$ 。



复杂度的递归方法求解

■ 代入法

- 先猜测解的形式，确定它的某个界的存在
- 再用归纳法去证明
- 这种方法只适用于解的形式容易猜的情况。
需要靠经验

■ 递归树方法

- 通过画递归树来求解

■ 主方法

- 对 $T(n) = aT(n/b) + f(n)$ 形式的求解



代入法

确定下面递归式的上界： $T(n) = 2T(\lfloor n/2 \rfloor) + n$

1. 猜测： $n \lg n$
2. 归纳：假设 $m < n$ 时成立，证明 n 成立

另 $m = n/2$ ，通过定义求得 n 成立

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

边界条件：

$$\begin{array}{l} \text{由 } T(1)=1 \end{array} \quad \rightarrow \quad \begin{array}{l} T(2)=4 \\ T(3)=5 \end{array}$$



代入法

考虑如下递归式：

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

猜测解为 $T(n) = O(n)$

得到

$$T(n) \leq c \lfloor n/2 \rfloor + c \lceil n/2 \rceil + 1 = cn + 1$$

与定义不符

新的猜测为 $T(n) \leq cn - d$ ， d 是大于等于 0 的一个常数。

$$\begin{aligned} T(n) &\leq (c \lfloor n/2 \rfloor - d) + (c \lceil n/2 \rceil - d) + 1 \\ &= cn - 2d + 1 \leq cn - d \end{aligned}$$



代入法

递归式： $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$

证明上界和下界类似，在此就只证明下界。

猜测 $T(n) \geq c(n + d) \lg(n + d)$

$$\begin{aligned} T(n) &\geq c(\lfloor \frac{n}{2} \rfloor + d) \lg(\lfloor \frac{n}{2} \rfloor + d) + c(\lceil \frac{n}{2} \rceil + d) \lg(\lceil \frac{n}{2} \rceil + d) + dn \\ &\geq c(\frac{n}{2} - 1 + d) \lg(\frac{n}{2} - 1 + d) + c(\frac{n}{2} + d) \lg(\frac{n}{2} + d) + dn \\ &\geq 2c(\frac{n}{2} - 1 + d) \lg(\frac{n}{2} - 1 + d) + dn \\ &= c(n + d + d - 2) \lg(n + 2d - 2) - c(2d - 2) + (d - c)n \end{aligned}$$

要使 $c(n + d + d - 2) \lg(n + 2d - 2) - c(2d - 2) + (d - c)n \geq c(n + d) \lg(n + d)$ ，只需 $d \geq 2, 0 < c \leq d$ 即可。



递归树方法

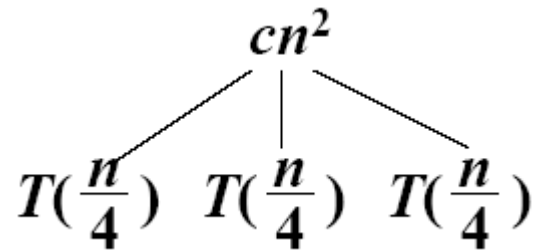
- Solve $T(n) = 3T(n/4) + \Theta(n^2)$, we have

$$T(n)$$



递归树方法

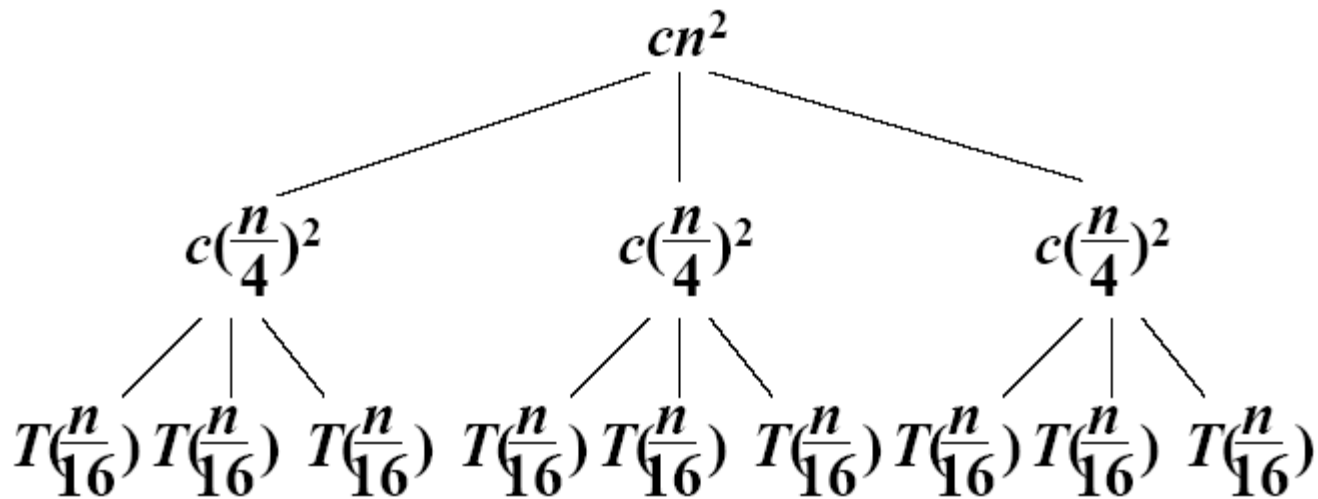
- Solve $T(n) = 3T(n/4) + \Theta(n^2)$, we have





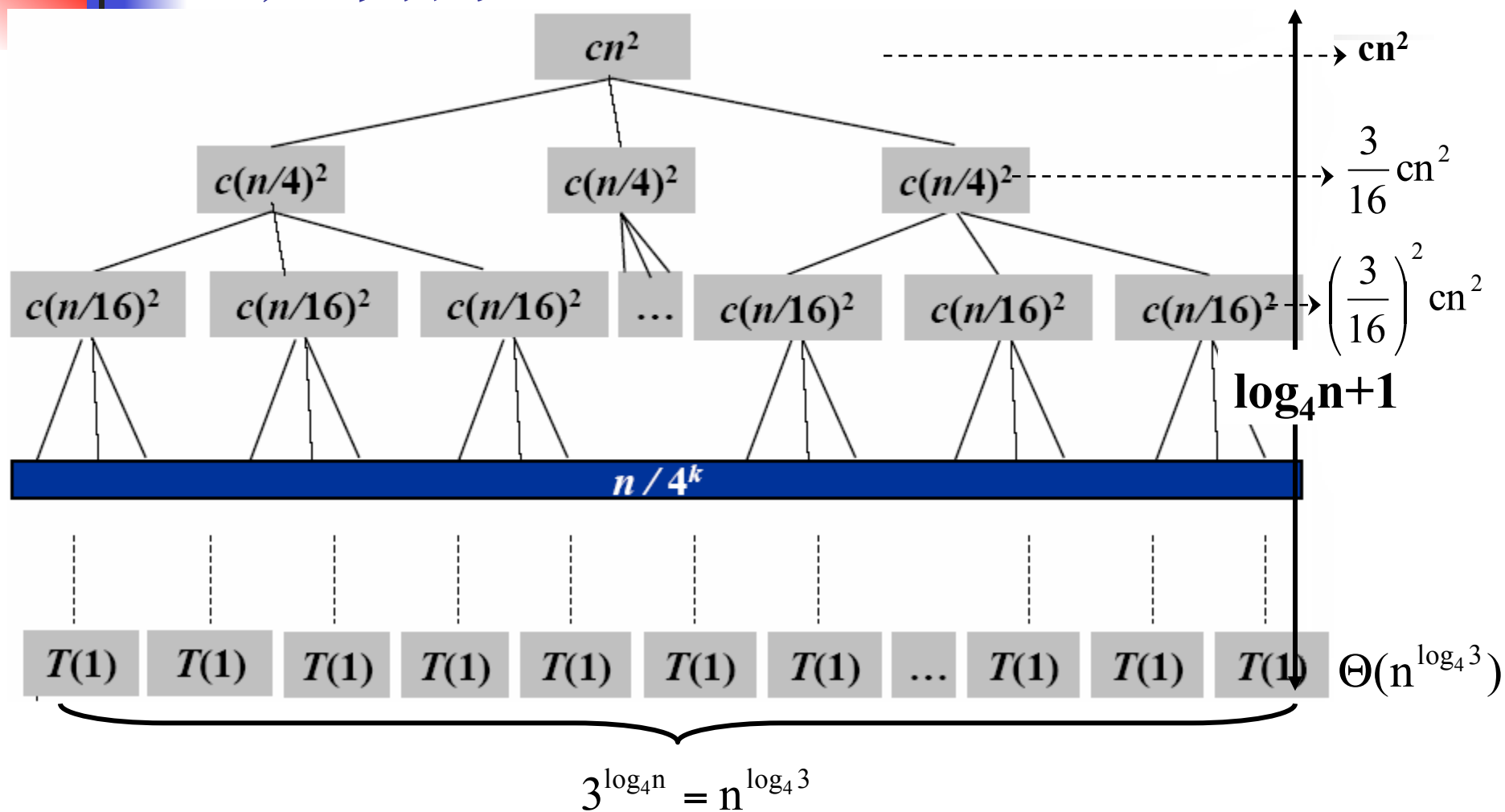
递归树方法

- Solve $T(n) = 3T(n/4) + \Theta(n^2)$, we have





递归树方法



- The fully expanded tree has $\lg_4 n + 1$ levels, i.e., it has height $\lg_4 n$



递归树方法

- 对整棵树求和

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) \quad (\text{根据公式(A.5)}) \end{aligned}$$



递归树方法

- 对整棵树求和

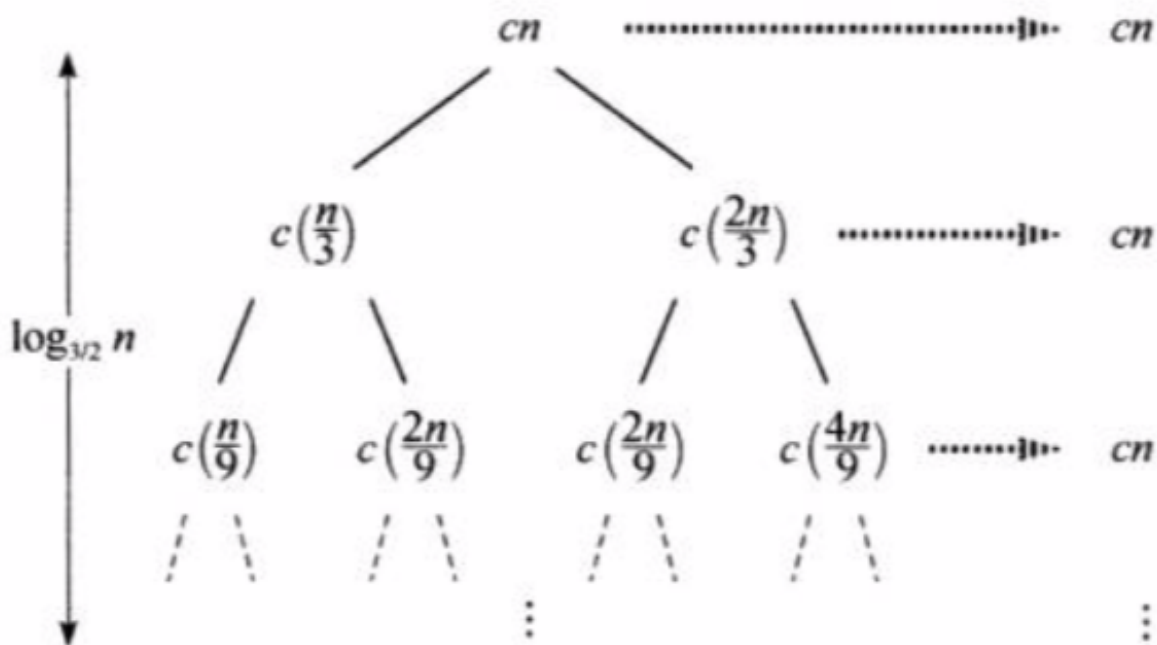
$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i c n^2 + \Theta(n^{\log_4 3}) < \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i c n^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1}{1 - (3/16)} c n^2 + \Theta(n^{\log_4 3}) \\ &= \frac{16}{13} c n^2 + \Theta(n^{\log_4 3}) \\ &= O(n^2) \end{aligned}$$



递归树方法

如下递归式的递归树：

$$T(n) = T(n/3) + T(2n/3) + O(n)$$





递归树方法

如下递归式的递归树：

$$T(n) = T(n/3) + T(2n/3) + O(n)$$

The shortest path from the root to a leaf in the recursion tree is $n \rightarrow (1/3)n \rightarrow (1/3)^2 n \rightarrow \dots \rightarrow 1$. Since $(1/3)^k n = 1$ when $k = \log_3 n$, the height of the part of the tree in which every node has two children is $\log_3 n$. Since the values at each of these levels of the tree add up to cn , the solution to the recurrence is at least $cn \log_3 n = \Omega(n \lg n)$.

从根到叶的

最长简单路径是 $n \rightarrow (2/3)n \rightarrow (2/3)^2 n \rightarrow \dots \rightarrow 1$ 。

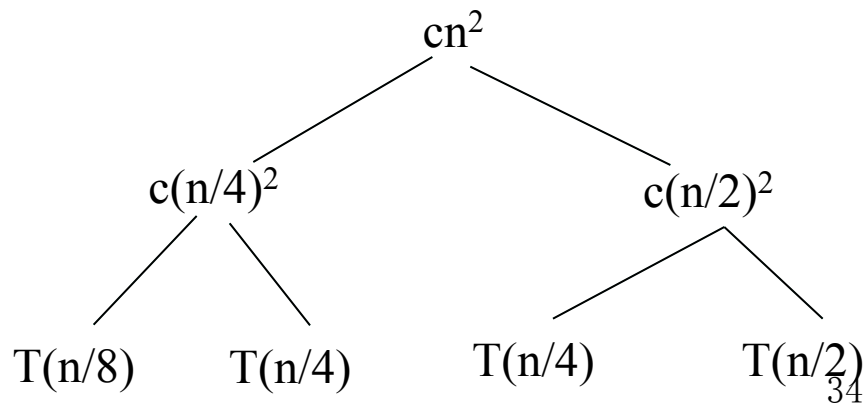
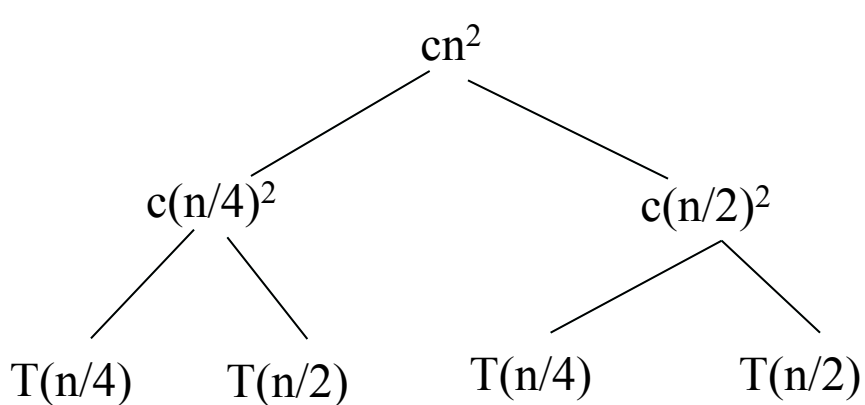
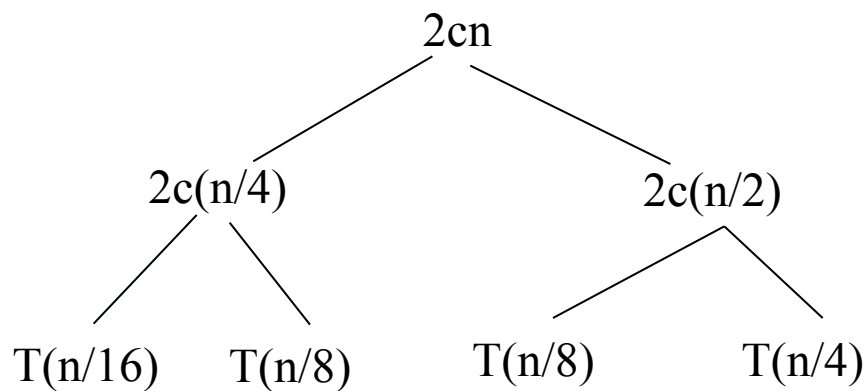
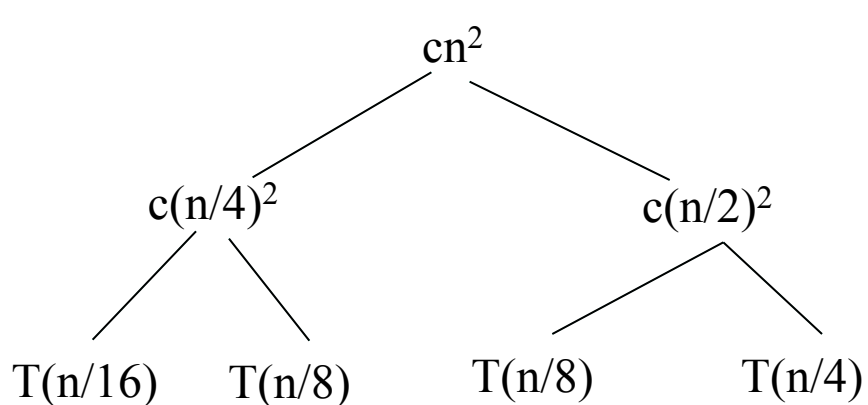
由于当 $k = \log_{3/2} n$ 时， $(2/3)^k n = 1$ ，因此树高为 $\log_{3/2} n$ 。

即 $O(cn \log_{3/2} n) = O(n \lg n)$



递归树方法

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

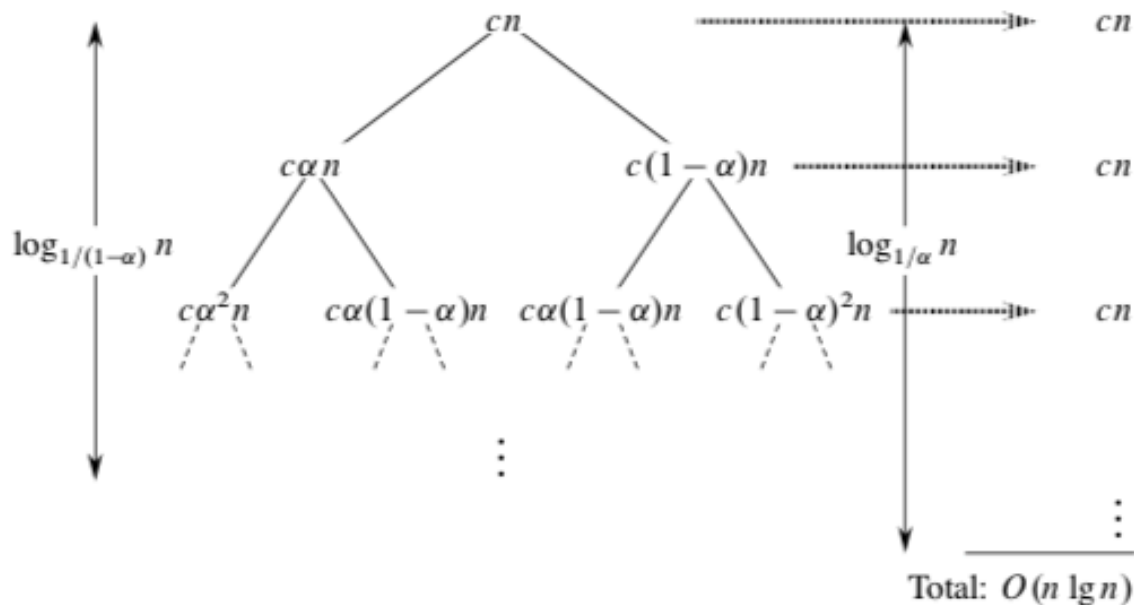




递归树方法

对递归式 $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$, 利用递归树给出一个渐近紧确解, 其中 $0 < \alpha < 1$ 和 $c > 0$ 是常数。

Without loss of generality, let $\alpha \geq 1-\alpha$, so that $0 < 1-\alpha \leq 1/2$ and $1/2 \leq \alpha < 1$.



The recursion tree is full for $\log_{1/(1-\alpha)} n$ levels, each contributing cn , so we guess $\Omega(n \log_{1/(1-\alpha)} n) = \Omega(n \lg n)$. It has $\log_{1/\alpha} n$ levels, each contributing $\leq cn$, so we guess $O(n \log_{1/\alpha} n) = O(n \lg n)$.



求解递归式—主方法

定理 4.1 (主定理) 令 $a \geq 1$ 和 $b > 1$ 是常数, $f(n)$ 是一个函数, $T(n)$ 是定义在非负整数上的递归式:

$$T(n) = aT(n/b) + n^x$$

$$T(n) = \begin{cases} \Theta(n^x) & \text{if } a < b^x \\ \Theta(n^x \log n) & \text{if } a = b^x \\ \Theta(n^{\log_b a}) & \text{if } a > b^x \end{cases}$$



求解递归式—主方法

对下列递归式，使用主方法求出渐近紧确界

a. $T(n) = 2T(n/4) + 1$

b. $T(n) = 2T(n/4) + \sqrt{n}$

c. $T(n) = 2T(n/4) + n$

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

a. $\Theta(n^{\log_4 2})$

b. $\Theta(n^{1/2 \log n})$

c. $\Theta(n)$

d. $\Theta(n^2)$