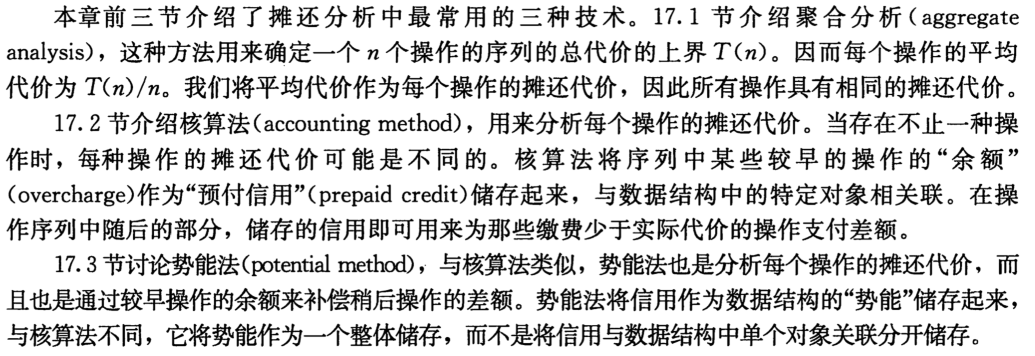
**平摊分析**是一种分析方法，证明在一系列操作中，每个操作的平均代价很小，即使在这个系列中一个操作的代价比较大。**平摊分析保证在最坏情况下每个操作的平均性能。**三种常用的平摊分析的方法：聚集法、记账法、势能法。



堆通常是一个可以被看做一棵完全二叉树的数组对象。

二项堆是指满足以下性质的二项树的集合：

1.每棵二项树都满足最小堆性质，即结点关键字大于等于其父结点的值

2.不能有两棵或以上的二项树有相同度数（包括度数为0）。换句话说，具有度数k的二项树有0个或1个。

红黑树是许多“平衡”搜索树中的一种，可以保证在最坏情况下基本动态集合操作的时间复杂度为。

**贪心算法：**

通过一系列的选择来得到问题的一个解，每次的选择都是在当前状态下的某种意义的最好选择，并希望通过每次的贪心选择导致最终结果是问题的一个最优解。

**两个性质**：

贪心选择性质：是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到最优子结构性质

**这是与动态规划算法的主要区别。**

**动态规划法中，每一步所作的选择往往依赖于相关子问题的解，**因而只有在解出相关子问题后才能作出选择。贪心算法中，凭着当前的状态作出当前看来是最好的选择，**即局部最优，**然后再去求解给出这一选择之后所产生的相应的子问题。

动态规划与分治法相似，都是通过组合子问题的解来求解原问题。

分治法将问题**划分为互不相交的子问题，递归地求解子问题**，再将它们的解组合起来，求出原问题的解，与之相反，动态规划应用于子问题重叠的情况，即不同的子问题具有公共的子问题。在这种情况下，分治法会做许多不必要的工作，它会反复地求解那些公共子问题。而动态规划算法对每个子问题只求解一次，将其解保存在一个表格中，从而无需每次求解一个子子问题时都重新计算，避免了这种不必要的计算工作。

**DP的两种方法：**

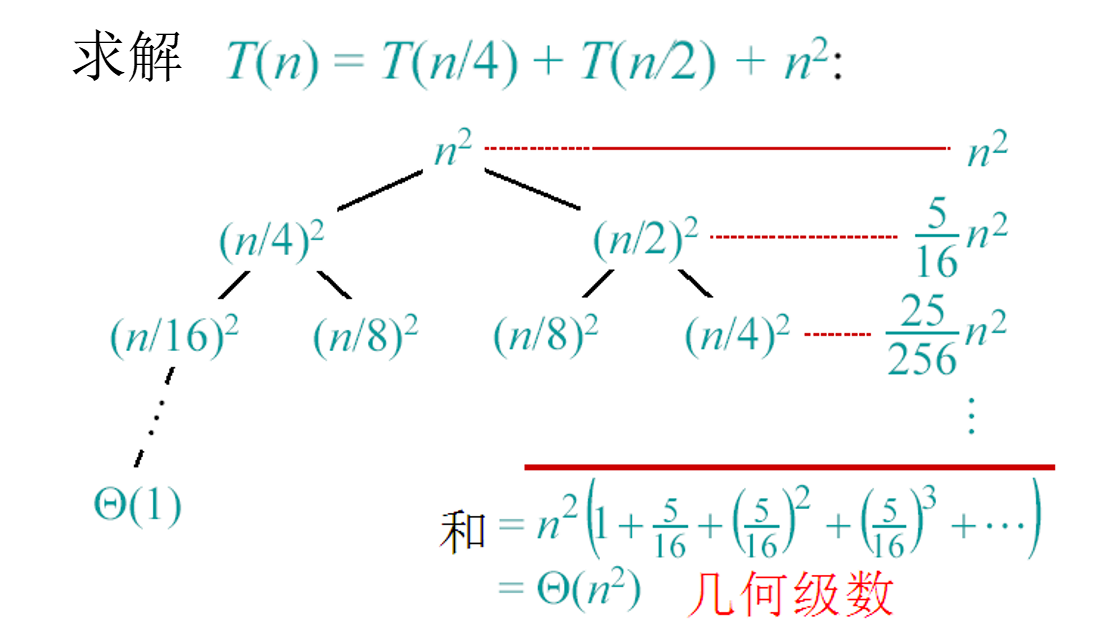
**从下向上迭代方法**:从递归的分而治之算法开始；找到子问题之间的依赖关系（计算子问题需要什么样的解）；按照正确的顺序解决子问题

**从上到下的递归方法 (记忆)：**从递归的分而治之算法开始； 保持原始算法从上到下的方法 ；将子问题的解保存在表中 (可能需要很多存储空间）； 只有当子问题的解在表中没有的时候才进行递归 如果所有的子问题都必须计算至少一次，从下向上的方法更好，因为减少了递归和维护表的负担。 如果很多子问题不需要计算，从上到下的DP方法更好，因为只有用到的项被计算

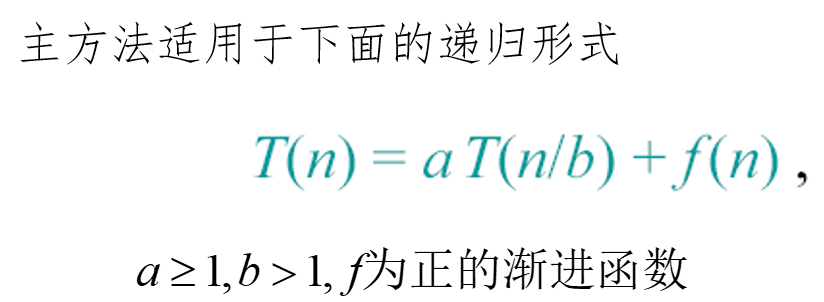
**求解递归式三种方法**

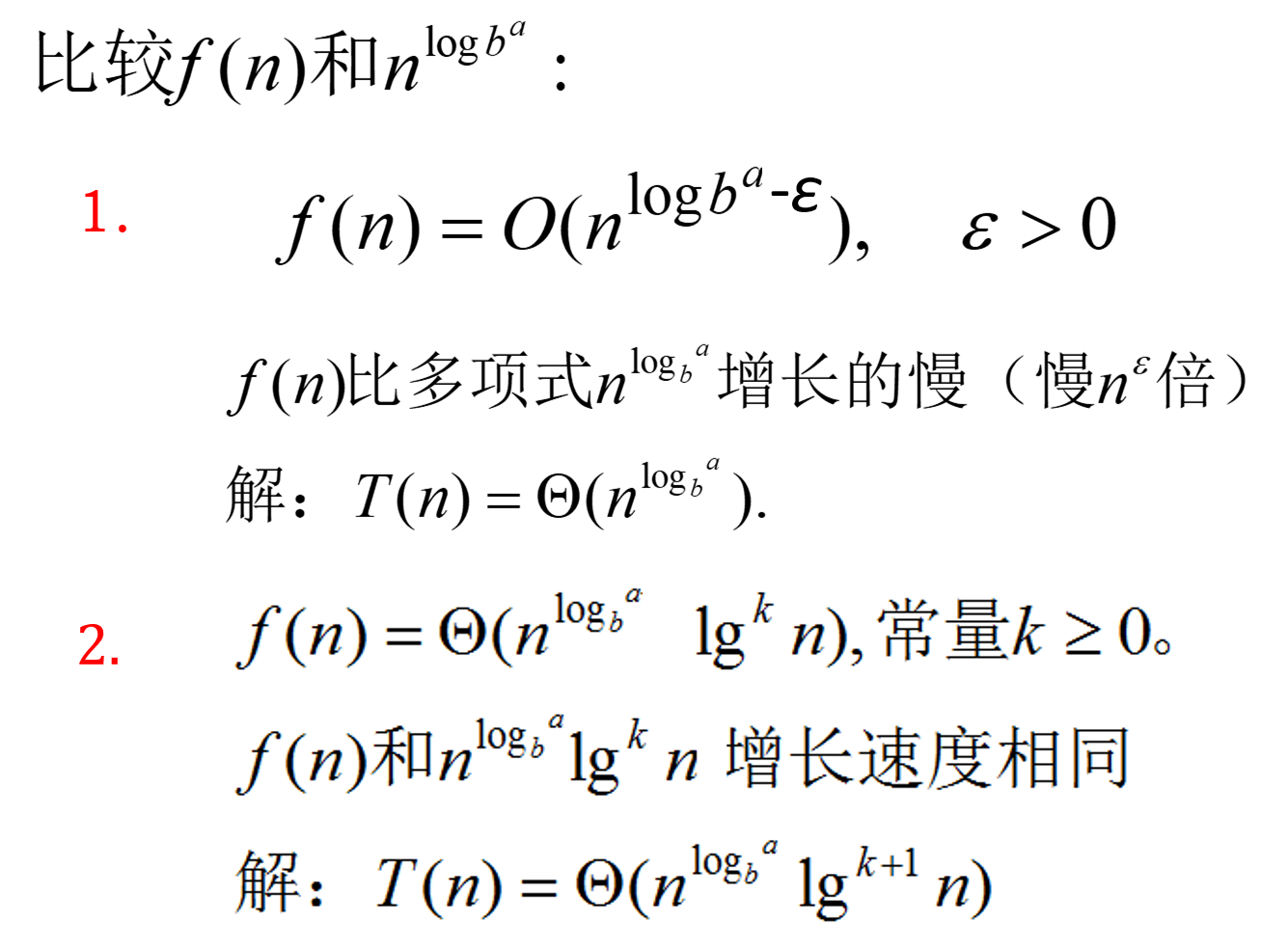
1.猜测法

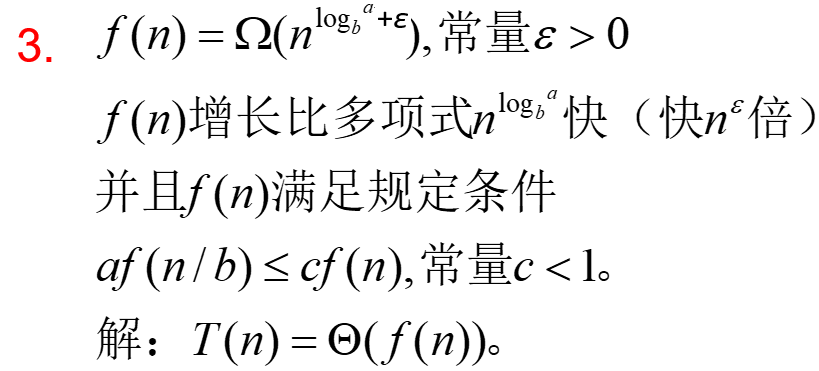
2.递归树

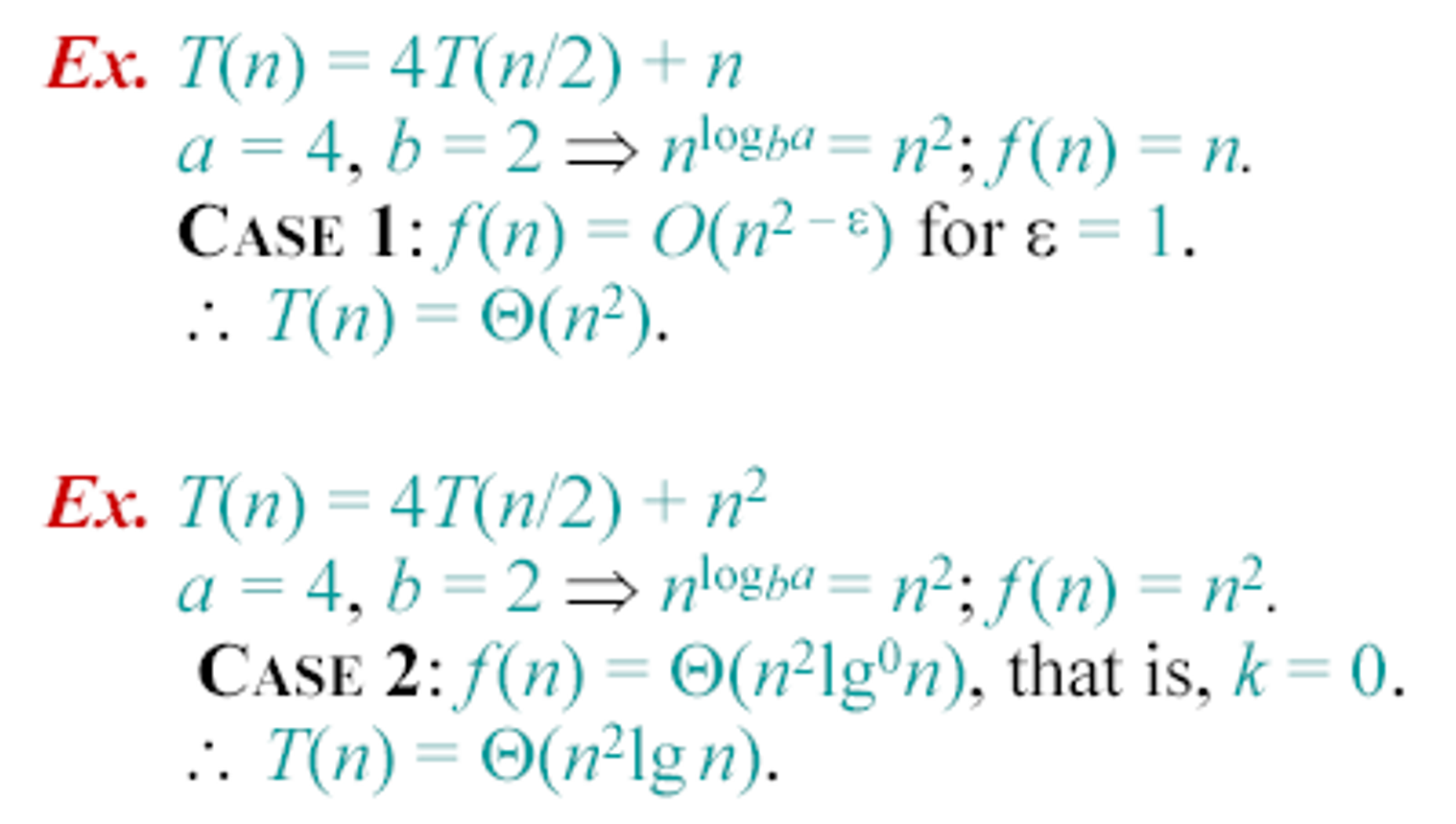


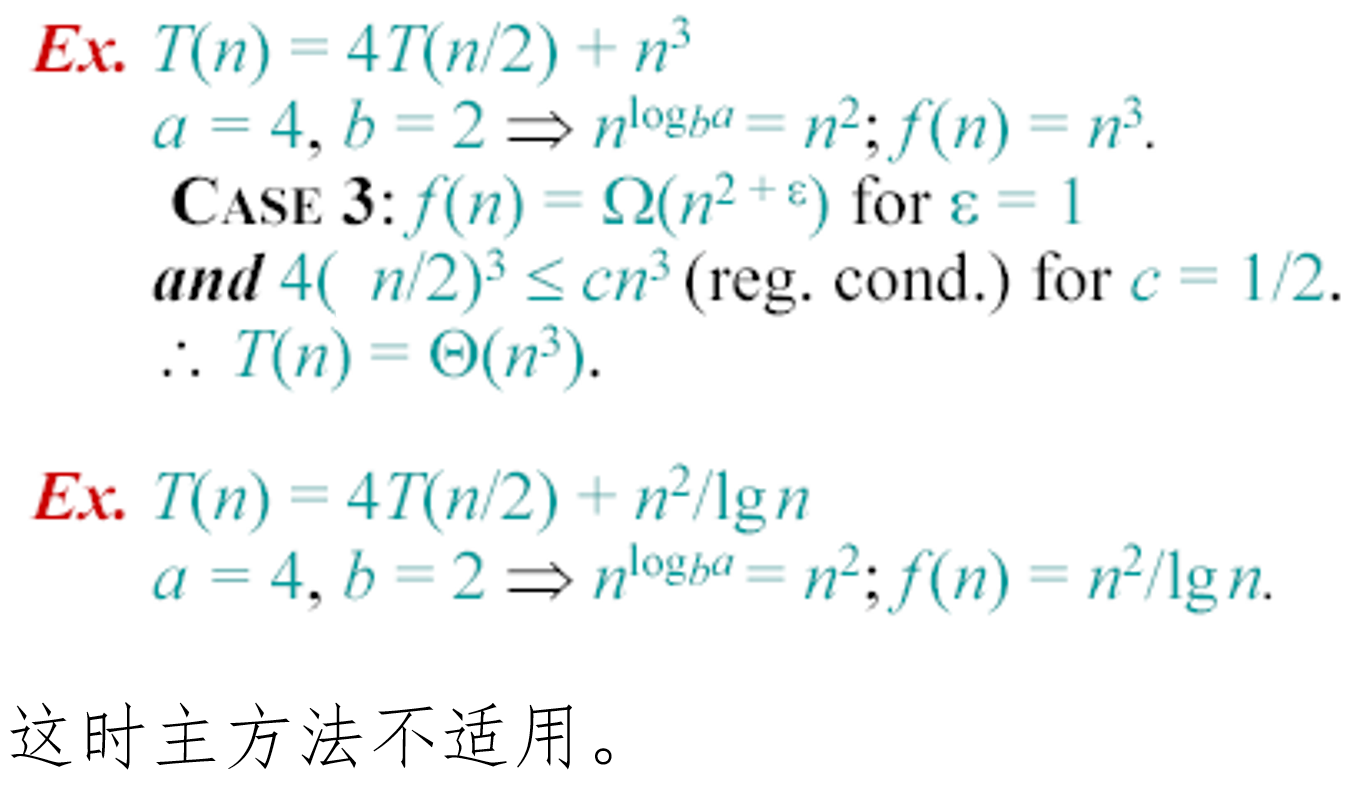
3.主方法











递归树和猜测法的解如果不会求，都可以通过主方法计算出来，或者用主方法验证结果。

顶点覆盖问题是 NP完全的。

**P、NP、NP-hard、NPC(10分)**

如果该问题存在一个多项式时间*O*(*nk*)求解算法，其中*k*为一常数。

因此我们可以定义**复杂类P为多项式时间可解**的具体判定问题的集合。

P类语言 ={*L* : *L* is accepted by a polynomial-time algorithm}

复杂类NP指的是一类**能够被多项式时间验证算法**所验证语言的集合。

多项式时间归约提供一种证明一个问题至少与另一个问题一样难的形式化方法。

NPC表示所有NP完全语言构成的语言类。NPC是NP中最难的问题。

**NPC问题的证明方法：**

**引理** 如果*L*是一个语言，对某个语言*L*'∈NPC ，使得*L*' ≤ P *L* ，则*L*是NP难的。而且，如果有*L*∈NP ，则*L*∈NPC。

证明：因为*L*'是NP完全的，对任意的*L*"∈NP ，我们有*L*"≤P *L*'，按照假设有*L* '≤P*L*，这样按照传递性，我们有*L*" ≤ P *L* ，这表明*L*是NP难的。如果*L*∈NP ，则由NP完全的定义，可知*L*∈ NPC，故所证成立。

证明一个语言属于NP完全的证明方法：首先证明*L*∈NP;最后证明是NP难的，

**其证明过程如下：**

a)首先选择一个已知属于NP完全的语言*L*';

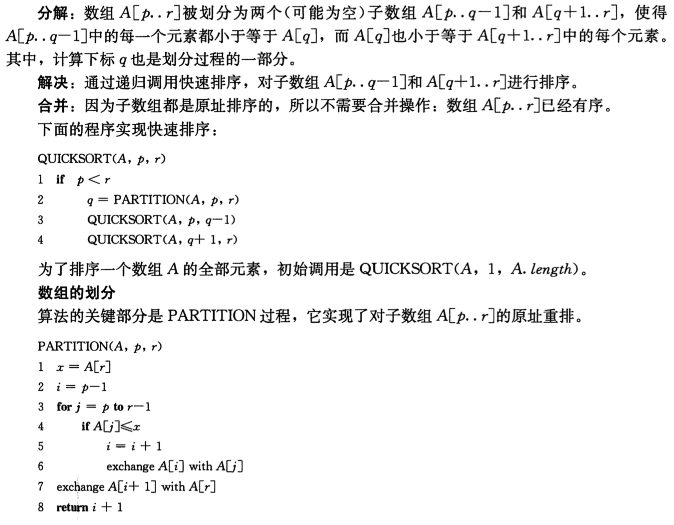
b)设计一个计算函数*f*的算法，能将*L*'的每个实例*x*∈{0, 1}\*映射到*L*的实例*f*(*x*) ;

c)证明上述转换函数*f*满足，对任意的*x*∈{0, 1}\*， *x*∈*L*'当且仅当有*f* (*x*)∈*L* ；

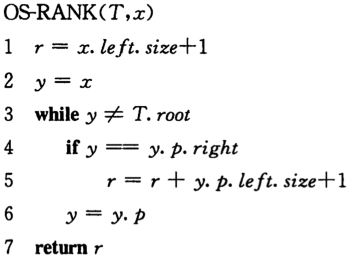
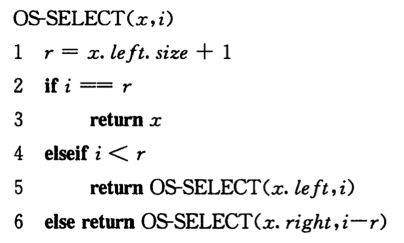
d)证明计算函数*f*的算法是多项式时间算法。

**分治法的基本思想，分治法的步骤，并以快排为例，描述对应的步骤的操作内容。**

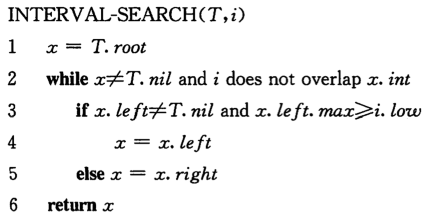
分治法(divide-and-conquer)的基本思想是将原问题可以分解为若干子问题分别进行求解，通过适当综合子问题的解，可以得到原问题的解。在许多情况下，各子问题的求解方法与原问题的求解方法类似，因而可以采用递归技术来实现。（5分）



**动态顺序统计:**

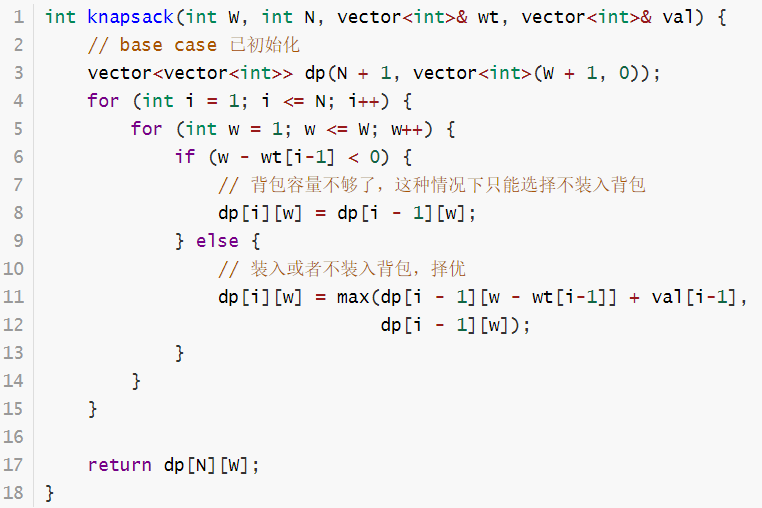


**区间树:**



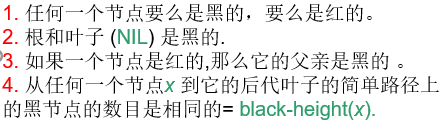
设计算法求解背包问题：设有个物品，其重量分别是,所有物品的重量之和大于等于背所能放置的重量。设计算法从中找出若干物品放入背包之中，使其重量之和正好为。解答：是已经放入的重量和，是实际已经放入物品的数目，表示即将放入的第个物品

**回溯法**



**红黑树**

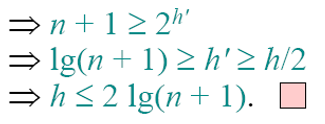
**性质:**



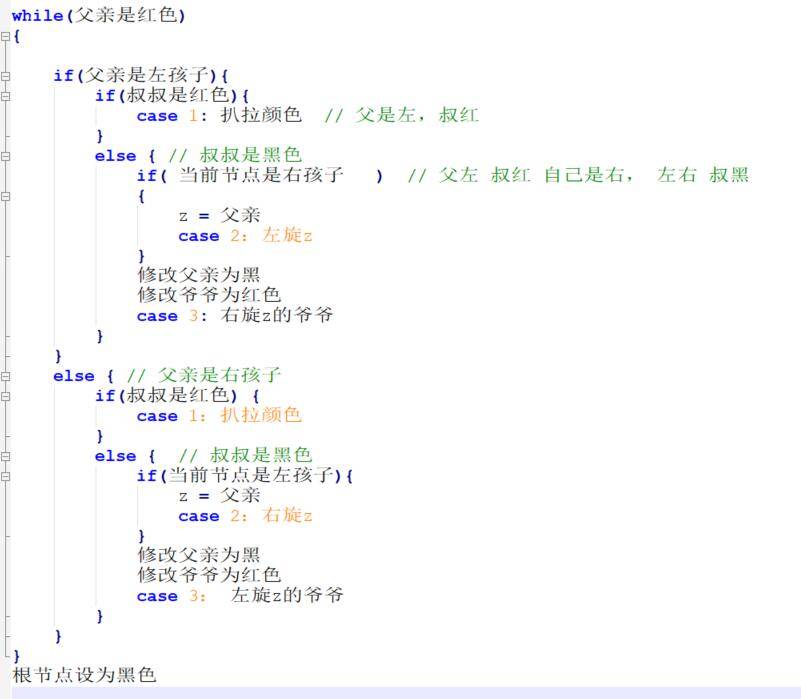
**高度:一棵有 n keys个键的红黑树的高度**



**证明：将红节点合并进它们的父节点，合并之后树的高度为h’，原树高度为h由于每条路径上最多只有一半叶子是红色的，我们可以得到 h′≥ h/2, 每个树的叶子个数是 n + 1**

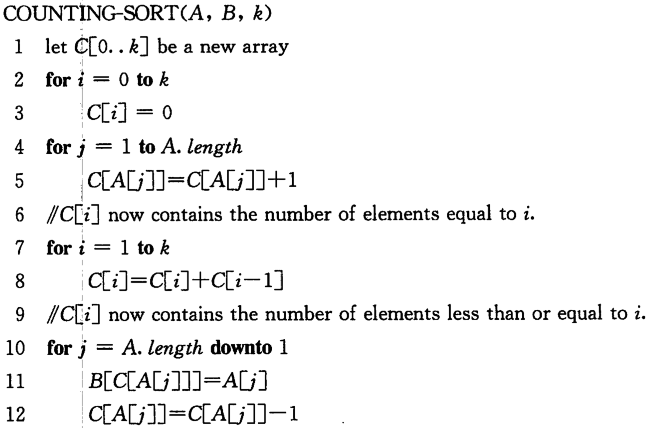


**插入:**



**线性时间排序**

**计数排序:**



假设: n个输入元素中的每一个是介于0~k之间的整数(k为整数），当k=O(n)时，计数排序的运行时间为O(n).

**基本思想：**(1)对每个元素x, 确定出比x小的元素个数，(2)由此而确定其在数组中的位置。由此可知，算法实现需要3轮：即(1)先计数，(2)再确定各元素的位置，(3)最后再依次按确定的位置将元素放到指定的位置。

**给出归并排序的算法描述。证明归并排序算法的时间函数为。**

**算法描述**：归并排序主要由分解、求解和合并三步操作组成。

分解:将规模为个数的排序划分成个数的排序

求解:分别求解个规模为个数的排序

合并:将两个规模为个数的已经排好序的子序列按照大小关系合并为一个序列

**证明**：时间性能：

1、分解：不费时间

2、求解：2T(n/2)

3、合并：O(n)

综上： 归并排序的复杂度为；

**分析：**先猜测,

然后证明T(n)<= c nlogn (c是某个常数）

先假设：对n/2成立，即T(n/2)<=cn/2 log(n/2),

则T(n)=2T（n/2）+ n

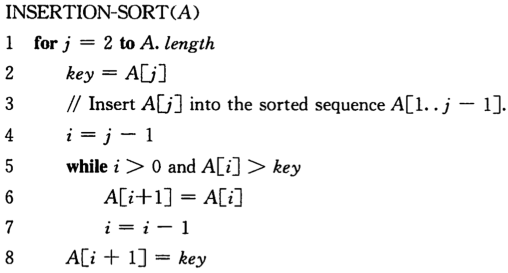
<=2 (cn/2 log(n/2))+n

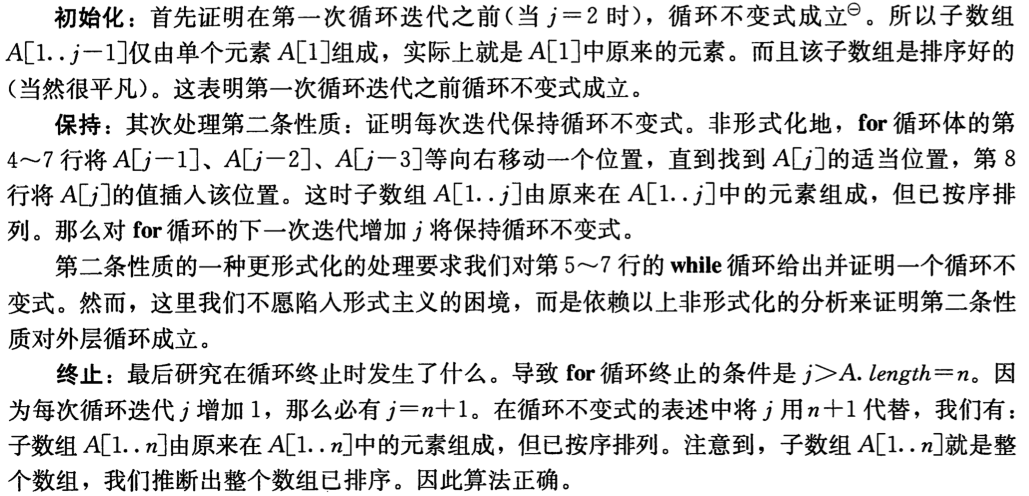
<=cn log(n/2)+n=cnlogn-cnlog2+n<=cnlogn

因此 T(n)=O(nlogn);

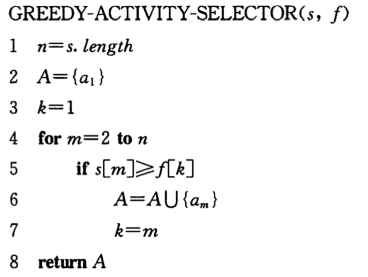
证明完毕

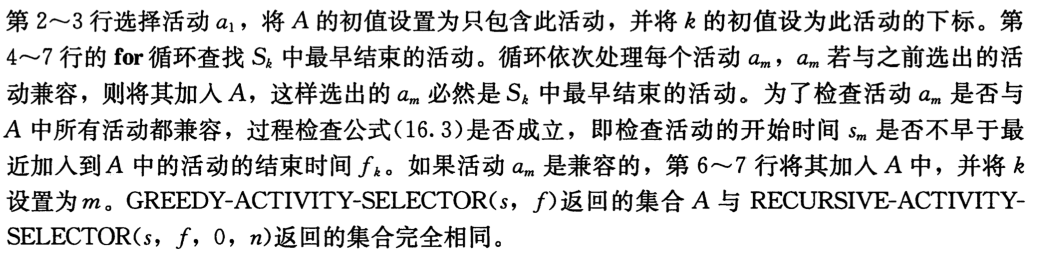
**插入排序:**





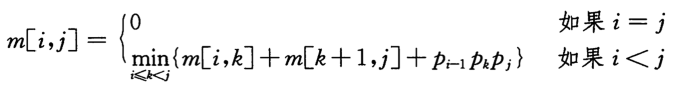
**贪心算法：**

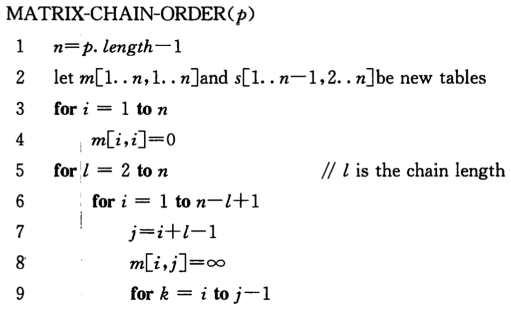


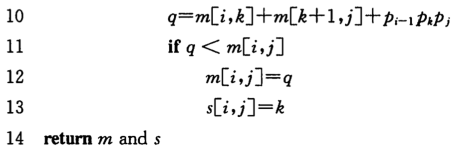


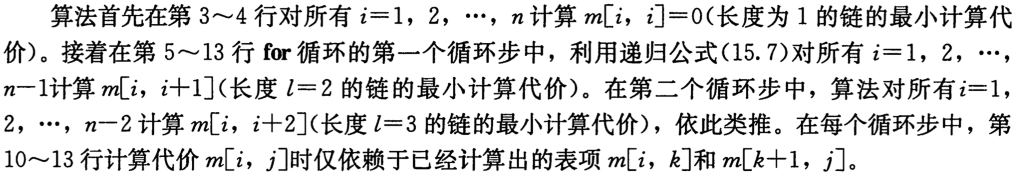
**动态规划:**

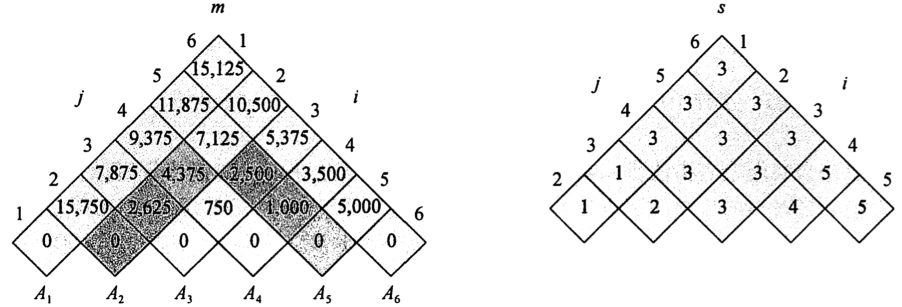
**1.矩阵链相乘:**



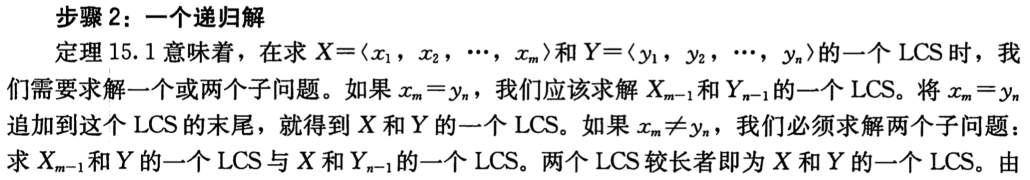


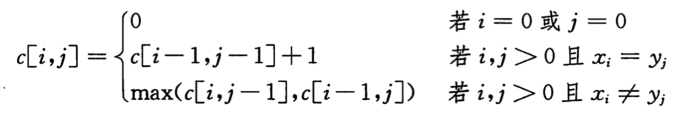


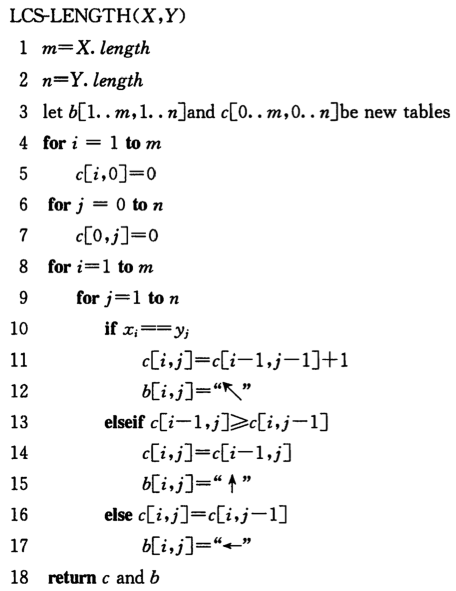


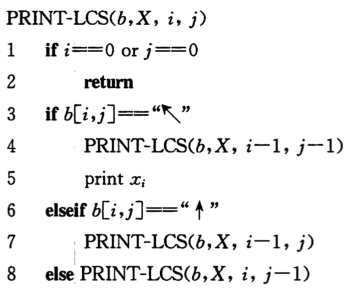


**2.最长公共子序列:**

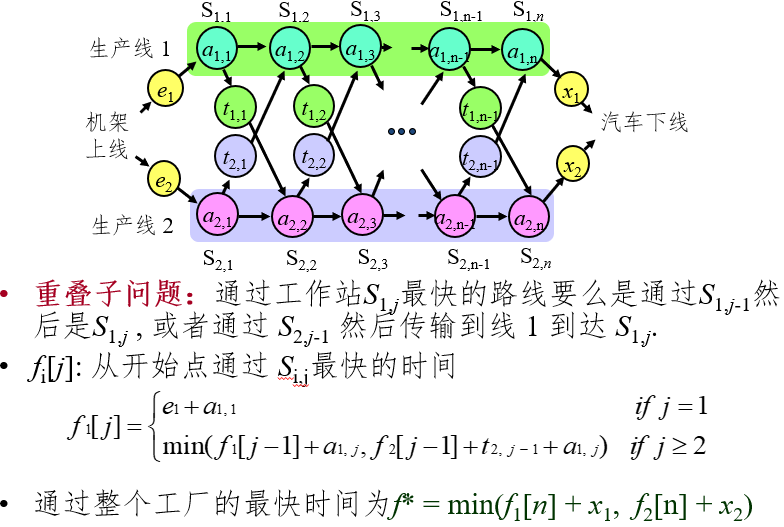


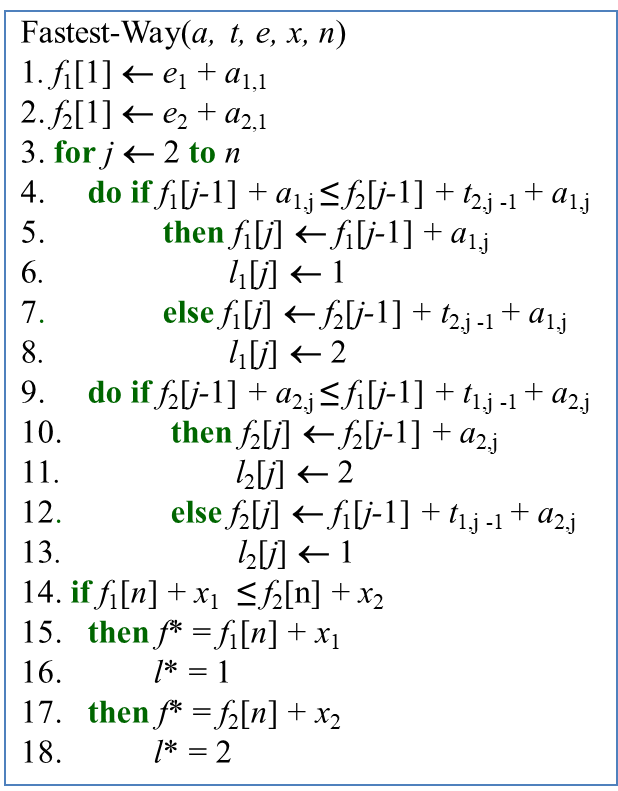






**3.调度问题:**



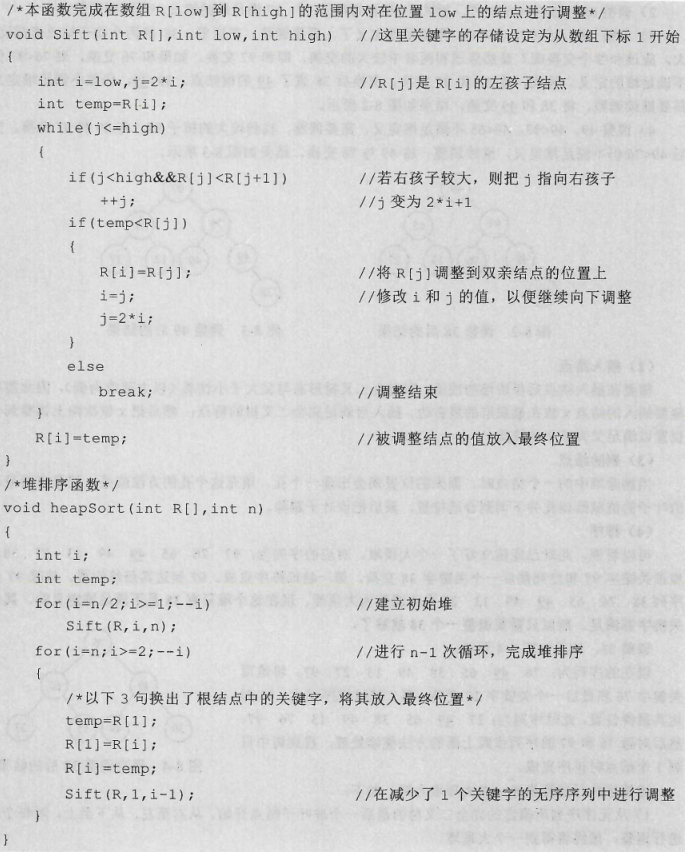
****

**扩展:**

**平衡二叉树的判定**



**堆排序:**



**判定有向图为有向树：**

