**《算法分析与设计》 – 期末知识点大纲**

**题型：20分选择题，20分判断题，20分填空题，25分解答题，15分算法题**

记诵重点是填空题大概率考绪论部分的算法相关概念，其他部分理解就没问题。

**——————————————————————————————————**

绪论

**算法（Algorithm）:** 对特定问题求解步骤的一种描述，是指令的有限序列。

从宏观层面看：算法是解决一个精确定义的计算问题的工具。

从技术层面看：算法是一个计算过程，它接受一些输入，并产生一些输出；

从抽象层次看：算法是将一个输入转化为输出的计算步骤序列；

**算法的五大特性：**

⑴ 输入：一个算法有零个或多个输入。

⑵ 输出：一个算法有一个或多个输出。

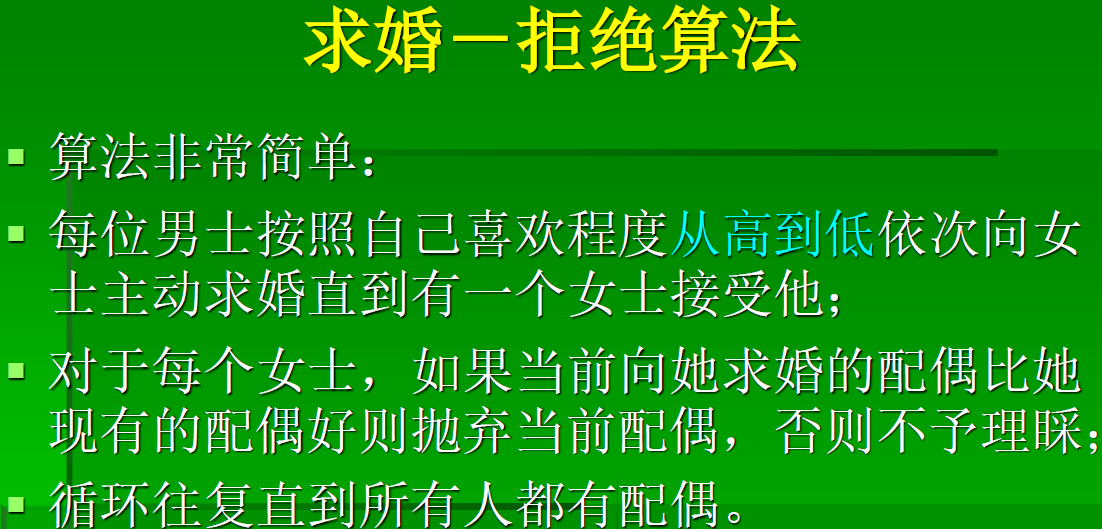
⑶ 有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。

⑷ 确定性：算法中的每一条指令无歧义，有确切的含义，对于相同的输入只能得到相同的输出。

⑸ 可行性：算法描述的操作都是基本的，可以通过执行有限次来实现，算法的实现问题。

我们强调的是算法的不变性：必须能够一步一步地照着执行。

简单的算法示例：求婚-拒绝算法（稳定婚姻匹配问题）



**算法设计的一般结构化过程：**

理解问题，预测所有可能的输入，在精确解和近似解间做选择，确定适当的数据结构，算法设计技术，描述算法，跟踪算法，分析算法的效率，根据算法编写代码

**基本的大致步骤是 理解问题及题目所给输入->根据输入取舍结果的精度->确定数据结构和算法大致思路->设计算法大致思路、描述跟踪算法步骤->分析效率->实际编写**

**实际编写是算法设计步骤的结果**

**算法的描述方法：**

自然语言描述、程序设计语言、流程图、伪代码

自然语言描述：有吸引力、容易理解，但是自然语言固有不严密性，产生歧义和错误

用于粗线条描述算法思想。

程序设计语言：算法的具体实现，步骤的执行序列，能直接被计算机执行，但抽象性差，对语言要求较高。

流程图：几何图形描述，流程直观、但缺少严密性灵活性，不方便

伪代码（Pseudocode）：自然语言与类编程语言组成的混合结构。

优点：表达能力强，抽象性强，容易理解

一种伪代码形式：

1. r = m % n;

2. 循环直到 r 等于0

2.1 m = n;

2.2 n = r;

2.3 r = m % n;

3. 输出 n ;

**注意：伪代码并没有通用的表示形式规范**

算法与计算机的关系：不可分割，算法是计算机的灵魂

算法是问题驱动

重要的问题类型：查找、排序、图、组合、几何问题

时间复杂度

**算法分析（Algorithm Analysis）：分析**时间复杂度（Time Complexity）和空间复杂度（Space Complexity）

用于设计算法（设计低复杂度的算法）和选择算法（选择复杂度最低的算法）

对于单一操作的算法,运行时间 = 操作时间 X 操作次数

操作时间取决于计算机，而操作次数取决于算法

不同计算机的速度不同，指令集结构、缓存技术等技术细节不同，这些不同导致一个算法在计算机具体实现的运行时间（操作时间）不同。

**时间复杂度估算：**一个算法所耗费的时间, 主要取决于算法中指令重复执行的次数，即语句的频度相关。

一个算法中所有语句的频度之和构成了该算法的运行时间。

算法分析理论中使用渐进方法，先把运行时间化成简单的形式，然后比较它们的阶。

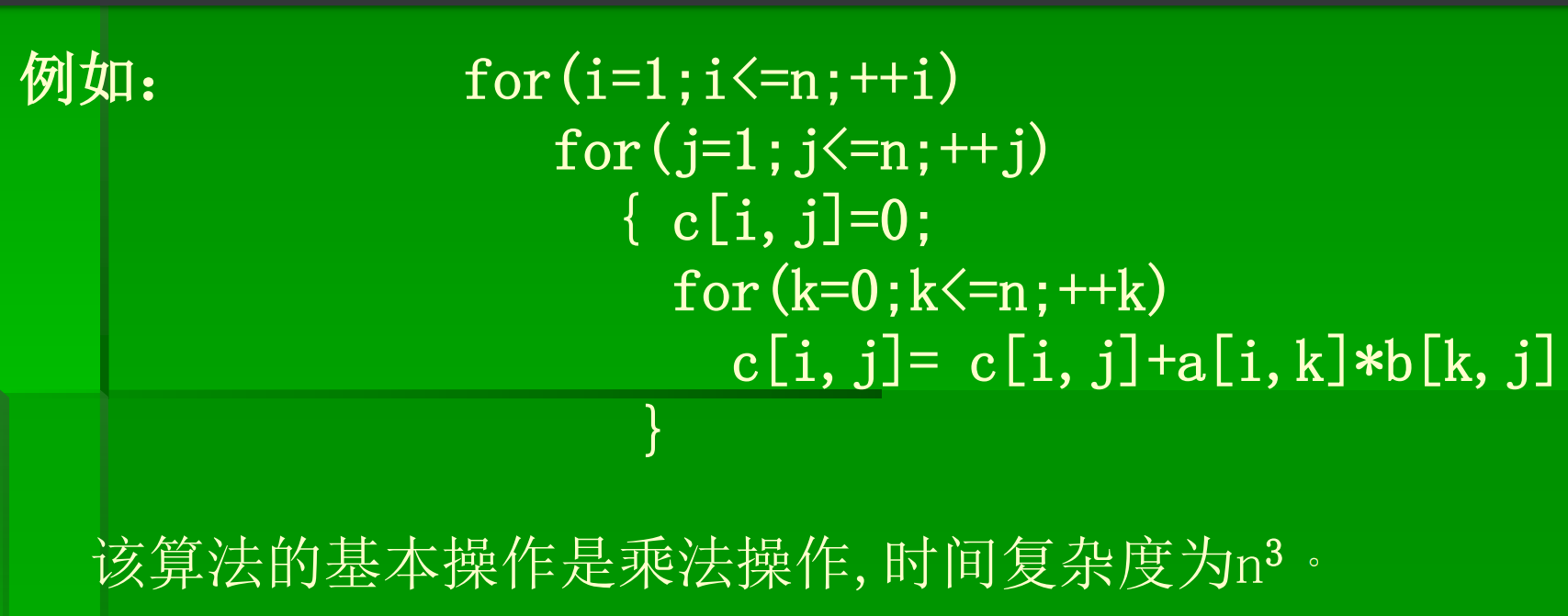
渐进：考虑算法在输入规模趋向无穷时的效率分析。

渐进分析是我们的大思维：**忽略机器、编程或编译器的影响**，只观察在输入规模趋向无穷时算法效率的表现。

**简化的方法是：只保留最大项，忽略系数**

**原操作：**,从算法中选取一种对于所研究的问题来说是基本(或者说是主要) 的原操作,以该基本操作在算法中重复执行的次数作为算法运行时间的衡量准则。

一般是**最深层次循环体内的语句中的基本语句**。



**T(n)表示：**记号T (n) = O (f (n))来表示当n充分大时，T (n)不超过 f (n)的常数倍。

**大O符号:**若存在两个正的常数c和n0，对于任意n≥n0，都有 T(n)≤c×f(n)，则称T(n)=O(f(n))

**算法数量级：**(n为问题的规模,c为一常量)：

Ο(1)常数级 Ο(logn)对数级 Ο(n)线性级 Ο(nc)多项式级 Ο(cn)指数级 Ο(n!)阶乘级

T(n) = O(f (n)) 如果f (n)为多项式（如n，n log n，n 2），称此算法为多项式时间算法；

2 n ，n!这样的算法称为指数时间算法。 从指数算法到多项式算法是一种巨大的改进。

一般来说如果选用了阶乘级的算法，则当问题规模等于或者大于10时就要认真考虑算法的适用性问题。原则上一个算法的时间复杂度, 最好不要采用指数级和阶乘级的算法, 而应尽可能选用多项式级或线性级等时间复杂度级别较小的算法。

**大Ω符号:**若存在两个正的常数c和n0，对于任意n≥n0，都有T(n)≥c×g(n)，则称T(n)=Ω(g(n))

**Θ符号:**若存在三个正的常数c1、c2和n0，对于任意n≥n0 都有c1×f(n)≥T(n)≥c2×f(n)，则称T(n)=Θ(f(n))

**非递归算法分析的一般步骤：**

1. 决定用哪个（或哪些）参数作为算法问题规模的度量
2. 找出算法中的基本语句
3. 检查基本语句的执行次数是否只依赖于问题规模
4. 建立基本语句执行次数的表达式
5. 用渐进符号表示这个表达式

建立一个代表算法运行时间的表达式， 然后用渐进符号表示这个求和表达式。

**后验分析（Posteriori）：**也称算法的实验分析，它是一种事后计算的方法，通常需要将算法转换为对 应的程序并上机运行。

一般步骤：

1. 明确实验目的

2. 决定度量算法效率的方法，为实验准备算法的程序实现 －计数法 －计时法

3. 决定输入样本，生成实验数据

4. 对输入样本运行算法对应的程序，记录得到的实验数据

5. 分析得到的实验数据

数据分析可以使用表格法或者散点图回归方法

蛮力算法

把问题的所有情况或所有过程交给计算机去一一尝试，也即是检查搜索空间中的每一个解直至找到问题的解。 常常直接基于问题的描述。

蛮力策略的应用很广，具体表现形式各异，如：选择排序、冒泡排序、插入排序、顺序查找、朴素的字符串匹配等，都是蛮力策略的具体应用。

基本技术：扫描技术

关键：依次处理所有元素

基本的扫描技术：遍历 （1）集合 （2）线性表 （3）树 （4）图

**基于以下原因，蛮力法也是一种重要的算法设计技术：**

（1）理论上可以解决可计算领域的各种问题。

（2）蛮力法经常用来解决一些较小规模的问题。

（3）对于一些重要的问题蛮力法可以产生一些合理的算法，具备一些实用价值，而且不受问题规模的限制。

（4）蛮力法可以作为某类问题时间性能的底限，来衡量同样问题的更高效算法。

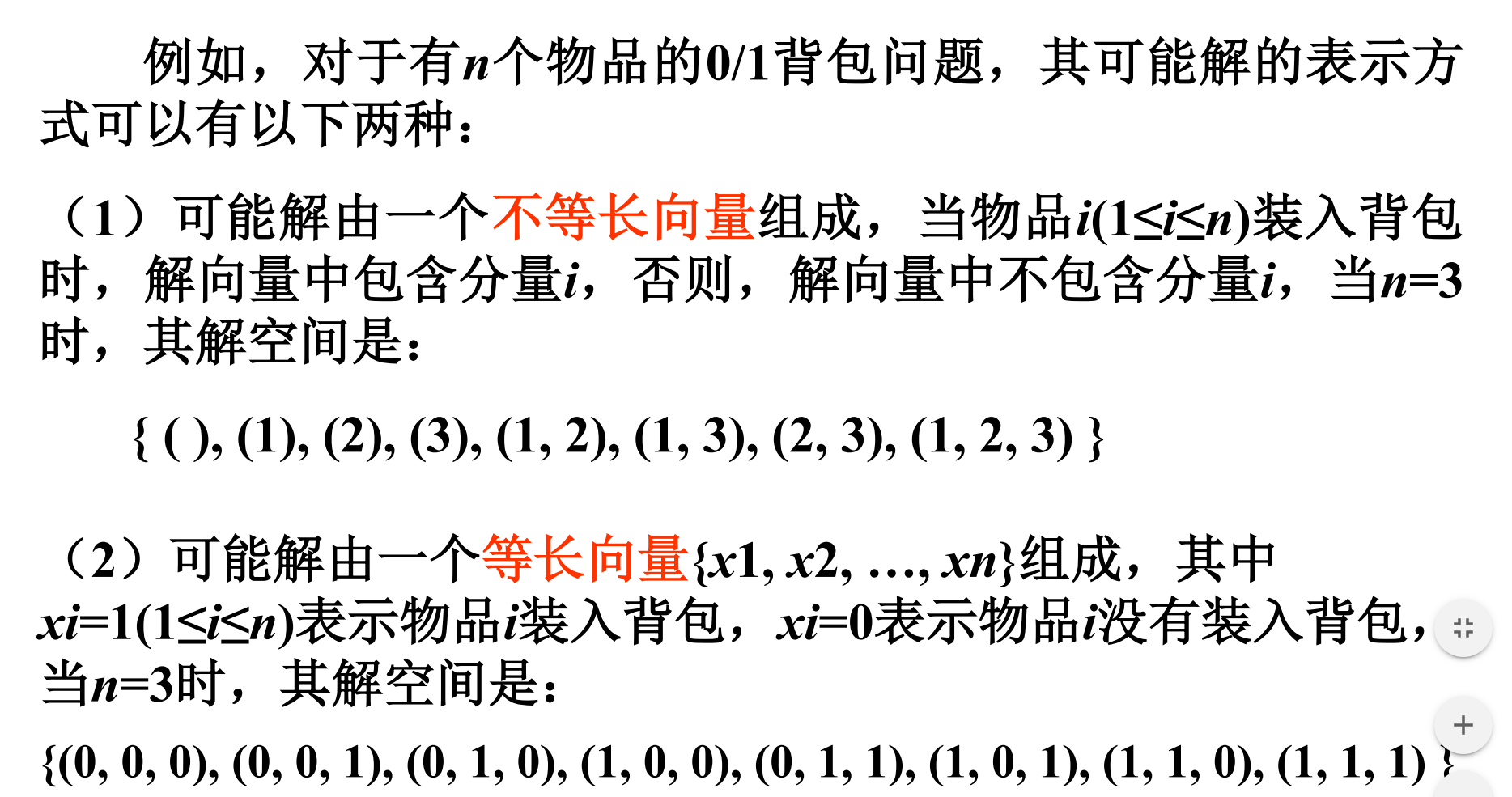
问题的解空间

所有问题都可以表述为搜索问题。

搜索的空间就是解的空间，而搜索就是在解的空间找出需要的一个。

可能解构成了问题的解空间

解空间的大小与解的表达方式有关，可能解的表示方式和它相应的解释隐含了解空间及其大小。



可以将搜索空间组织成一棵树，**即解空间树（Solution Space Trees,也称状态空间树）**。

完整解在树的叶子上。沿着树的每一个分枝经过一系列的步骤来构造完整解。

从树的根结点到叶子结点的路径就构成了解空间的一个可能解。

对于大部分问题来说，其解空间的规模为输入规模的**指数函数**甚至更高。

**实际问题难以求解的原因：**

搜索空间中可能解的数目太多；

问题如此复杂以至于为得到任何解答，不得不采用问题的简化模型；

实际问题往往随时间而变；

实际问题往往存在很多约束条件

回溯法

**智能穷举的核心思想**：

穷举（在最坏情况下要对整个解空间进行数级搜索）

智能（利用各种途径减少搜索量）

深度优先策略遍历、判断该结点是否包含问题的（最优）解、不包含则剪枝

**剪枝函数（Pruning Function）**：

（1）用约束条件剪去得不到可行解的子树；

（2）用评估函数剪去得不到最优解的子树。

问题的解空间树是虚拟的，并不需要在算法运行时构造一棵真正的树结构，只需要 存储从根结点到当前结点的路径

分支限界法

**过程:**

确定一个合理的限界函数及界[down, up]

按照广度优先策略遍历问题的解空间树，估算孩子结点的 评估函数的可能取值

如果某孩子结点的评估函数可能取得的值超出评估函数的界，则将其丢弃；

否则，将其加入待处理结点表中。

依次从表中选取使评估函数的值取得极值的结点成为当前扩展结点

重复上述过程，直到找到最优解。

分支限界法的一般过程

1．根据限界函数确定目标函数的界[down, up]；

2．将待处理结点表PT初始化为空；

3．对根结点的每个孩子结点x执行下列操作

3.1 估算结点x的目标函数值value;

3.2 若(value>=down)，则将结点x加入表PT中；

4．循环直到某个叶子结点的目标函数值在表PT中最大

4.1 i=表PT中值最大的结点；

4.2 对结点i的每个孩子结点x执行下列操作

4.2.1 估算结点x的目标函数值value;

4.2.2 若(value>=down)，则将结点x加入表PT中；

4.2.3 若(结点x是叶子结点且结点x的value值在表PT中最大)， 则将结点x对应的解输出，算法结束；

4.2.4 若(结点x是叶子结点但结点x的value值在表PT中不是最大)， 则令down=value，并且将表PT中所有小于value的结点删除；

分治法

**步骤：“分治合”**

分：将要求解的较大规模的问题分割成k个更小规模的子问题。

治：对这k个子问题分别求解。如果子问题的规模仍然不够小，则再划分为k个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止。

合：将求出的小规模的问题的解合并为一个更大规模的问 题的解，自底向上逐步求出原来问题的解。

反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。

这自然导致递归过程的产生。

**递归（Recursion）：**子程序（或函数）直 接调用自己或通过一系列调用语句间接调用自己

递归有两个基本要素：

⑴ 边界条件：确定递归到何时终止；

⑵ 递归模式：大问题是如何分解为小问题的。

分治是一种思想，递归是一种手段

**启发式规则：**

1. 平衡子问题：最好使子问题的规模大致相同。也就是将一个问题划分成大小相等的k个子问题（通常k＝2），这种使子问题规模大致相等的做法是出自一种平衡（Balancing）子问题的思想，它几乎总是比子问题规模不等的做法要好。
2. 独立子问题：各子问题之间相互独立，这涉及到分治法的效率，如果各子问题不是独立的，则分治法需要重复地解公共的子问题。

**划分策略：**

**黑盒划分策略(**根据问题的规模**)，分治排序**

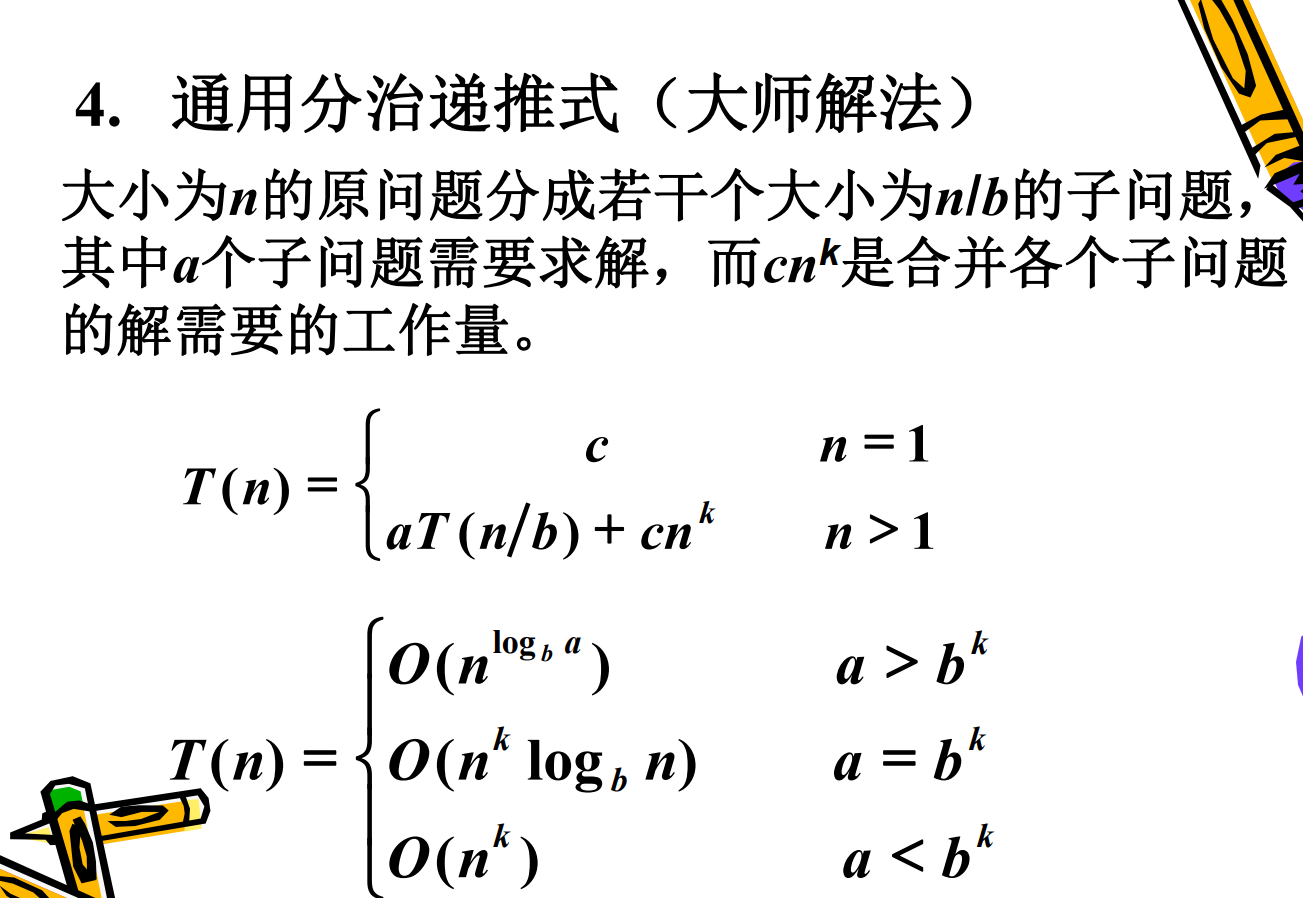
**白盒划分策略(**根据划分对象的特定属性值（也称之为参照值）**)，快速排序**

**排序的归并排序、快速排序**

复杂度计算关键：根据递归过程建立递推关系式，然后 求解这个递推关系式。

或者猜测技术:对递推关系式估计一个上限，然后（用数学 归纳法）证明它正确。

1. 扩展递归技术、通用分治递归式：



减治法

策略：

（1）原问题的解只存在于其中一个较小规模的子问题中；

（2）原问题的解与其中一个较小规模的解之间存在某种对应关系。

**所以不需要合并，只需要切割问题**

应用减治法处理问题的效率是很高的，一般是O(log2 n)数量级。

贪心法

策略：

一步一步地构建问题的最优解决方案，其中每一步只需考虑眼前的最佳选择（局部判断规则），即通过局部最优到达全局最优。

**并非对所有问题都能得到整体最优解，但对许多问题它能产生整体最优解。**

在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

区间调度、区间划分

获得最优解的条件：

**最优子结构性质：**一个问题的最优解包含其子问题的最优解

**贪心选择性质：**整体最优解可以通过 一系列局部最优的选择，即贪心选择来得到。

Prim：使用最近顶点策略

Kruskal：最短边策略

动态规划法

策略：将复杂的问题分解为小的简单的问题，解决这些小问题，再合并成为大问题的解。

重点是建立动态规划的局部最优解的递归方程。

将待求解问题**分解成若干个相互重叠的子问题**

每个子问题对应决策过程的**一个阶段**

一般来说，子问题的重叠关系表现在对给定问题求解的递推关系（也就是动态规划函数）中，将**子问题的解求解一次并填入表中**，当需要再**次求解此子问题时，可以通过查表获得该子问题的解**而不用再次求解，从而避免了大量重复计算。

要求：

能够分解为相互重叠的若干子问题；

满足最优性原理（也称**最优子结构性质**）：该问题的最优解中也包含着其子问题的最优解。

证明使用反证法。