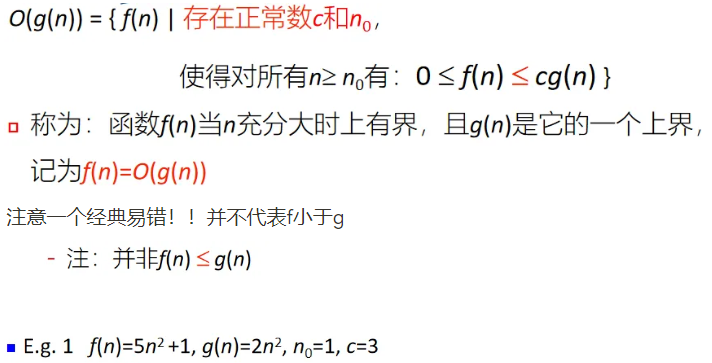
[计算机保研/考研面试题——数据结构与算法篇\_计算机保研面试专业课问题-CSDN博客](https://blog.csdn.net/m0_53140426/article/details/140249150)

[2025计算机保研面试题目整理（按题目类型） - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/1905643606880027762)

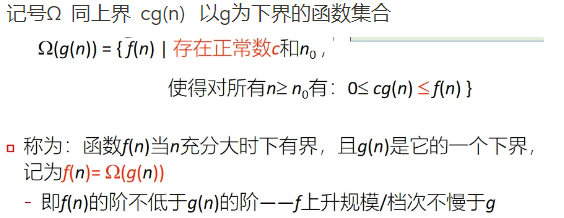
五种渐进界：

**渐进上界：O f（n）<=cg（n） f（n）=O（g(n)） 理解为一种记号就行了**

事实上O（g(n)）是**一个函数集合**！ 找一个c 和n0就行

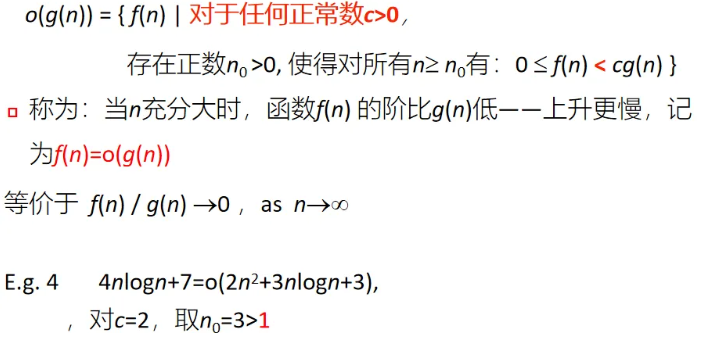


**渐进下界：记号Ω 同上界 cg(n) 以g为下界的函数集合**

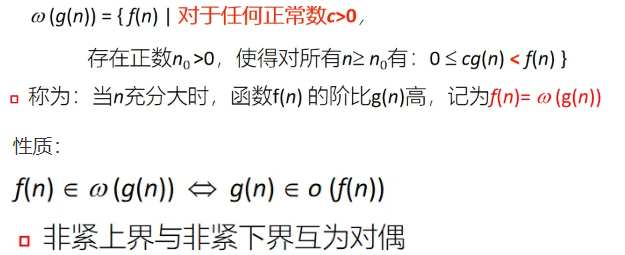


**非紧上界： 记号o（不是紧的就用小的）** **对任何常数c>0！**也就是说怎么都比你大

注意等价 不取等

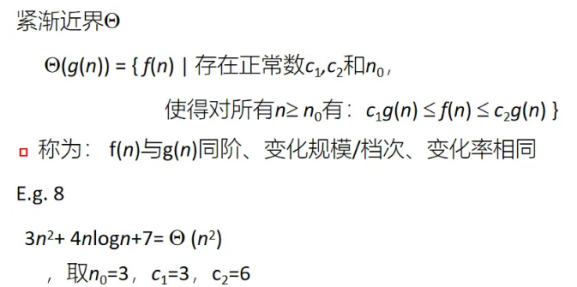


**非紧下界：同样取小 小Ω=w**



**紧渐近界：**

**强调同阶** 特殊的符号 =O（g（n））∩Ω（g（n）） 理解 大于等于



渐进界的一些性质：传递（五种符号都有） 自反 对称（注意） 互对称

递归与分治：

**分治是一种策略，递归是具体的一个实现**

**分治：相同子任务分解 子任务并行求解 子任务结果合成 子任务要独立**

基于递归：斐波那契 汉诺塔 排列问题

递归逻辑强 容易用**数学归纳法证明**正确性 而且易于推导复杂性方程

缺点是计算慢，占有空间多 容易爆（尝试转为非递归）

分治法的四个条件：

1.问题规模缩小到一定程度可以容易地解决

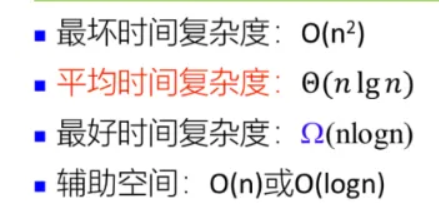
2.分解出的各个子问题相互独立（**即不包含公共子问题**，如果不独立，分治时会重复求解，效率不高）

3.可分解为若干规模相同的子问题（平衡子问题）

4.子问题的解合并得到原问题的解（如果不具备这个特征，可考虑贪心or动态规划）

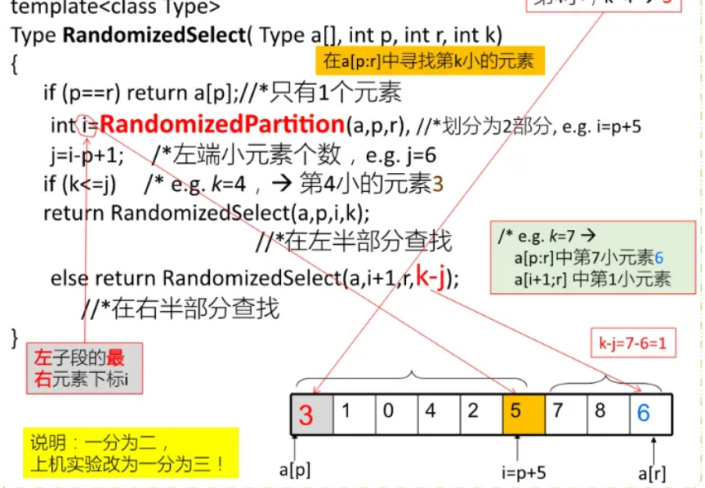
典型代表：二分搜索

快排：

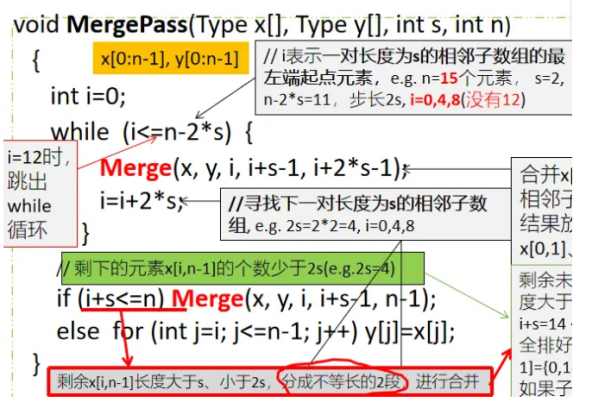
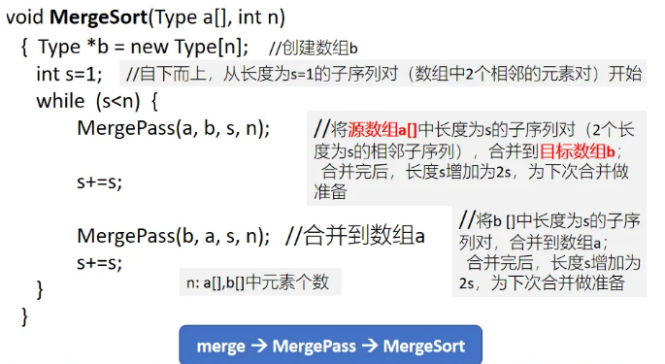
（改进：取随机数作为bench）

**线性时间选择问题的随机选择算法：**求未排序序列中第k小元素！ 理解 这里分两部分

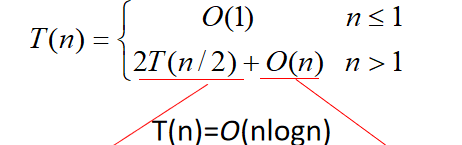
出口：l==r 只有一个元素，它是第几小都行 用快排的那个函数



合并排序：



**最坏时间复杂度O(nlogn)平均时间复杂度O(nlogn)**辅助空间：O(n)



**vector：是动态数组，存储在堆内存中**

减治法：

只需解决其中1个子问题，就可得到原问题解

理解：减治法的核心思想是将原始问题转化为一个或多个规模更小的同类问题，然后逐步解决这些更小的问题，最终得到原始问题的解。在某些情况下，我们只需要解决其中一个子问题，就可以得到原问题的解。（子问题解决，大问题本质是子问题，所以大问题解决，然后大大问题……）

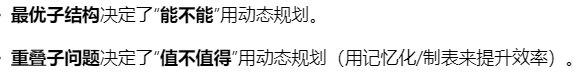
**核心：通过某种方式减少问题的一个或多个维度的规模，但不一定分解成独立的子问题**。

动态规划——**分解得到的子问题往往不是独立的**（不同于分治法）

**记录已计算的子问题从而避免重复计算  
动态规划问题能够成立的两个核心性质——最优子结构，重叠子问题（重叠子问题保证子问题只计算一次，使得效率更高）**

最优子结构是指：一个问题的最优解包含其子问题的最优解。

换而言之，可以通过子问题的最优解来构造出原问题的最优解



**矩阵连乘：**计算次数最少

for(int r=2;r<=n;r++){

    //自下而上，从规模为r=2的子问题开始，依次计算规模r=2,3,…,n-1的多个子问题的最优解

    for(int i=1;i<=n-r+1;i++){

      //求解长为r的子问题最优解（1为最左端，n-r+1为最右端），选取不同的断点，看最优解

      int j=i+r-1;

      m[i][j] =0+ m[i+1][j]+ p[i-1]\*p[i]\*p[j];//先是断点在最左端

      s[i][j]=i;//断点位置

      for(int k=i+1;k<j;k++){

        //依次考察后续断点位置k=i+1,i+2,…,j-1

        int t = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];

        //记录最优解及断点位置

        if (t < m[i][j]) { m[i][j] = t; s[i][j] = k;

      }

**最优子结构的证明：反证法 逆否命题**

**假设由问题最优解导出的子问题的解不是最优，说明在这个假设下，可构造出比原问题最优解更好的解，使矛盾**

最长子序列：

如果我的X和Y最后一项相同，那么最长公共子序列的长度可以从两个字符串都去掉最后一个字符后的最长公共子序列长度加 1 得到；（Zk中k代表子序列长度）

如果X和Y最后一项不同，则最长公共子序列应该是去掉X的最后一个字符或者去掉Y的最后一个字符后的两个子问题的最大值。

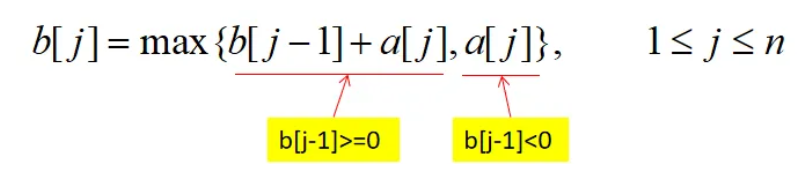
情况1：若xm=yn，则zk=xm=yn，且Z(k-1)={z1, z2,…,zk-1 }是X(m-1)= {x1,x2,…,xm-1}和Y(n-1)={y1,y2,…,yn-1}的最长公共子序列

情况2：若xm≠yn且zk≠xm，则Z(k)是X(m-1)和Y(n)的的最长公共子序列 zk可能等于、或不等于yn

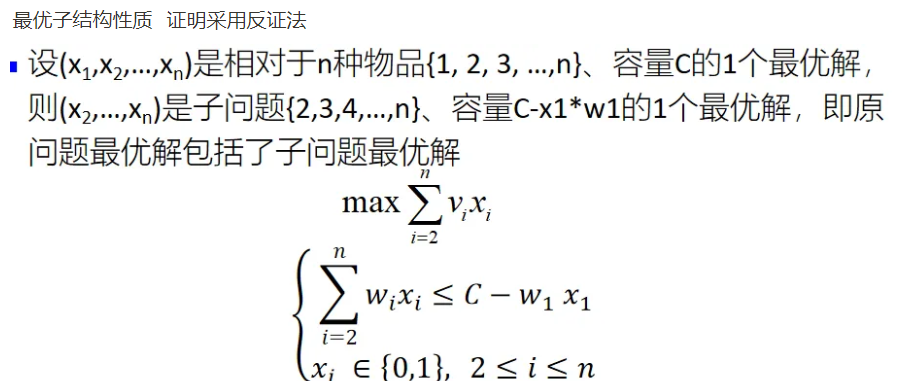
情况3：若xm≠yn且zk≠yn，则Z(k)是X(m)和Y(n-1)的最长公共子序列 zk可能等于、 或不等于xm

由此可见，2个序列的最长公共子序列包含了这2个序列的前缀（i.e. 子问题）的最长公共子序列。 结论：最长公共子序列问题具有最优子结构性质

最大子段和：

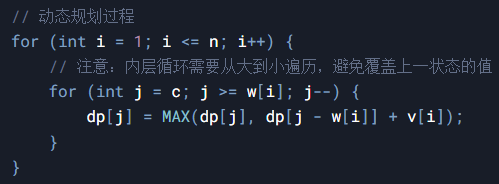


01背包：



改进：

可以发现0-1背包的状态转移方程 dp[i][j] = max{dp[i-1][j-w[i]]+v[i]，dp[i-1][j]}的特点，当前状态仅依赖前一状态的剩余体积与当前物品体积v[i]的关系。根据这个特点，我们可以**将dp降到一维即dp[j] = max{dp[j]，dp[j-w[i]]+v[i]}**。从这个方程中我们可以发现，有两个dp[j]，但是要区分开。等号左边的dp[j]是当前i的状态，右边中括号内的dp[j]是第i-1状态下的值。

（外层循环顺序倒不影响，一维的话内层顺序就必须固定了！）

贪心：

**贪心并不从整体最优考虑，只是在某种意义上、基于当前状况的局部最优选择**

**局部贪心不能保证全局最优！**

**贪心选择性质是指：一个问题的全局最优解可以通过一系列局部最优选择（即贪心选择）来得到**

**贪心选择性质的证明！！两步！**

1.贪心选择在前：证明的第一步是说明，做贪心选择是安全的，即存在一个全局最优解包含了我们所做的这个贪心选择。（就像活动选择问题中，总存在一个最优解包含了结束最早的活动）。

2.最优子结构：证明在做了贪心选择后，剩下的子问题仍然是一个最优子结构问题。即，原问题的最优解 = 贪心选择 + 剩余子问题的最优解。

虽然贪心算法从原理上不能保证对所有问题都得到整体最优解，但贪心算法对一些问题能产生全局最优解，如单源最短路经问题，最小生成树问题等，这些问题具有**贪心选择性质**

活动安排问题：

贪心：哪个活动结束的早且与已安排活动相容，哪个先

要先对活动进行排序（依据结束时间的早晚 非递减序列）

背包问题，最优装载问题，哈夫曼编码

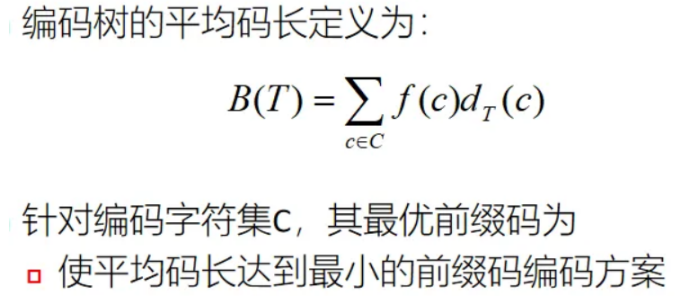
出现频率高的字符用较短的编码，出现频率较低的字符用较长的编码 目标：缩短文件中全部字符编码的总码长

对字母表中的每个字符规定一个0、1串作为其代码，要求任一字符的代码都不是其它字符代码的前缀 算法——经典的树的构造的过程

**特点：最先合并的具有最小频率的2个字符**

定长码也是前缀码，但不是最优前缀码（注意下前缀码的定义）

平均码长！ f（c）：字符的出现频率



算法用最小堆实现优先队列Q。初始化优先队列需要O(n)计算时间，最小堆的removeMin和put运算均需O(logn)时间，n-1次的合并总共需要O(nlogn)计算时间

**因此，关于n个字符的哈夫曼算法的计算时间为O(nlogn)**

**贪心的经典之单源最短路径：dijkstra**

维护两个数组，A（确定最短的点）和B（没确定的）

1. 选定一个点，这个点满足两个条件：1.未被选过，2.距离最短

2. 对于这个点的所有邻近点去尝试松弛

两重循环，第一重循环n-1次使最后所有点都进数组，第二重找最小并更新

Prim和kruskal算法：  
prim：算法从一个根顶点开始，逐步“生长”出一棵树。在每一步，总是选择连接当前树与树外顶点的那条最小权重的边，并将该边和其连接的树外顶点加入到树中。（根节点任选）

初始化一个最小堆（优先队列），用于存放所有连接 S 与 V-S（未在树中的顶点）的边（(u, v, weight)，其中 u 在 S 中，v 不在 S 中）。

Kruskal：将所有的边按权重从小到大排序，然后依此选择边。如果加入当前边不会与已选择的边形成环，就将其加入生成树中

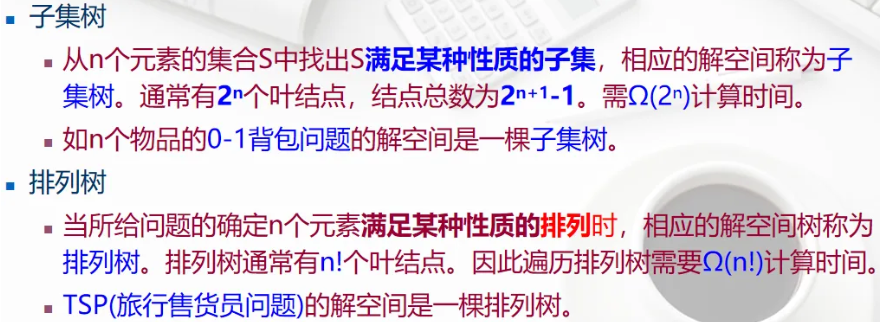
回溯：

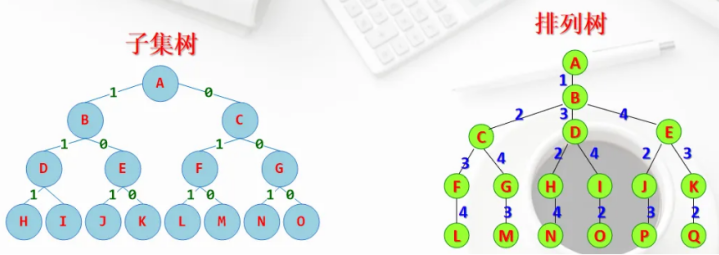
推销员问题：每个城市一遍，最后回到住地的路线，使总的路程最短。

n-1个点的排列问题，有(n-1)!条可选路线 时间复杂度n！

0-1背包问题

子集树与排列树：





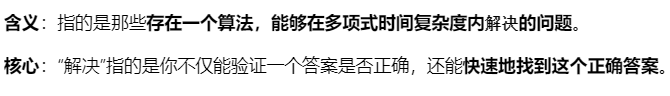
批处理作业处理调度，n皇后问题 图着色问题

保研过：

NP问题，P问题

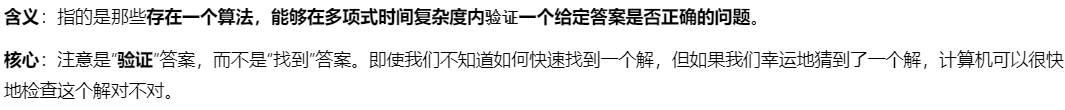
P代表：Polynomial Time，即多项式时间

P问题：



NP问题：

多项式时间内无法解决，却可以验证



Bellman-ford：

