算法是一系列解决问题的清晰指令，也就是说，能够对符合一定规范的输入，在有限时间内获得所要求的输出。 简单的说，就是解决问题的一种方法或过程。

算法－特征：（1）确定性（2）多样性（3）有穷性 （4）输出

所需资源越少，该算法越好，计算机最重要的资源是**时间和空间**

把**基本操作（最重要的操作）次数**作为算法**运行时间的度量单位。**

- **$T(n)\approx c_{op}C(n)$**

基本操作的执行时间　基本操作次数

算法**输入规模 n** 为时间效率的参数

算法的时间效率和空间效率都用**输入规模的函数**进行度量

O(g(n)) 是增长次数小于等于 g(n)(以及其常数倍，n 趋向于无穷大)的函数集合。

符号O

成立条件：对于所有足够大的 n， t(n) 的上界由 g(n)的常数倍数所确定。即，存在大于 0 的常数 c 和非负的整数 n_0，使得：对于所有的 $n\geq n_0$ 来说， $t(n)\leq c\,g(n)$

Ω(g(n))代表增长次数大于等于 g(n)(以及其常数倍,n 趋向于无穷大)的函数集合

符号Ω

成立条件：对于所有足够大的 n， t(n)的下界由 g(n)的常数倍所确定，

即，存在大于 0 的常数 c 和非负的整数 n_0，使得：

对于所有的 $n\geq n_0$ 来说， $t(n)\geq c\,g(n)$

Θ(g(n))是增长次数等于 g(n) (以及其常数倍,n 趋向于无穷大)的函数集合。

符号Θ

成立条件：对于所有足够大的 n， t(n) 的上界和下界都由 g(n)的常数倍数所确定，

即，存在大于 0 的常数 c_1，c_2 和非负的整数 n_0，使得：

对于所有的 $n\geq n_0$ 来说， $c_2g(n)\leq t(n)\leq c_1g(n)$

算法的整体效率是由具有较大的增长次数的部分所决定的，即它的效率较差的部分。

1	
log *n*	
n	
n* log *n	
n^2	
n^3	
2^n	
n!	

动态规划算法的基本要素

（1） 最优子结构性质（2）重叠子问题性质

矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为最优子结构性质

递归算法求解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。这种性质称为**子问题的重叠性质**

动态规划算法，对每一个子问题只解一次，而后将其解保存在一个**表格**中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果
通常不同的子问题个数随问题的大小呈多项式增长
动态规划算法的步骤
(1)找出最优解的性质，并刻划其结构特征。
(2)递归地定义最优值。
(3)以自底向上的方式计算出最优值。
(4)根据计算最优值时得到的信息，构造最优解。
基本思想也是将待求解问题分解成若干个子问题，能够保存已解决的子问题的答案， 而在需要时再找出已求得的答案，就可 以避免大量重复计算，从而得到多项式 时间算法
贪心法只根据当前已有的信息就做出选择，而且一旦做出了选择，不管将来有什么结果，这个选择都不会改变。 换言之，贪心法并不是从整体最优考虑，它所做出的选择只是在某种意义上的局部最优,这种局部最优选择并不总能获得整体最优解（Optimal Solution），但通常能获得近似最优解
贪心法求解的问题的特征：
（1）最优子结构性质
当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质，也称此问题满足最优性原理。
（2）贪心选择性质
所谓贪心选择性质是指问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来得到。
动态规划法通常以自底向上的方式求解各个子问题，而贪心法则通常以自顶向下的方式做出一系列的贪心选择。

图着色贪心策略：选择一种颜色，以任意顶点作为开始顶点，依次考察图中的未被着色的每个顶点，如果一个顶点可以用颜色 1 着色，换言之，该顶点的邻接点都还未被着色，则用颜色 1 为该顶点着色。 当没有顶点能以这种颜色着色时，选择颜色 2 和一个未被着色的顶点作为开始顶点，用第二种颜色为尽可能多的顶点着色，如果还有未着色的顶点，则选取颜色 3 并为尽可能多的顶点着色，依此类推。
最小生成树问题至少有两种合理的贪心策略：
（1） **最近顶点策略：**任选一个顶点，并以此建立起生成树，每一步的贪心选择是简单地把不在生成树中的最近顶点添加到生成树中。
Prim 算法应用贪心策略，使生成树以一种自然的方式生长，即从任意顶点开始，每一步为这棵树添加一个分枝，直到生成树中包含全部顶点。
（2） **最短边策略：**设 $G=(V，E)$是一个无向连通网，令 $T=(U，TE)$是 G 的最小生成树。最短边策略从 $TE=\{\}$开始，每一次贪心选择都是在边集 E 中选取最短边(u, v)，如果边(u, v)加入集合 TE 中不产生回路，则将边(u, v)加入边集 TE 中，并将它在集合 E 中删去。
Kruskal 算法应用贪心策略，它使生成树以一种随意的方式生长，先让森林中的树木随意生长，每生长一次就将两棵树合并，到最后合并成一棵树
所有可能的**解向量**构成了问题的**解空间**。
问题的解空间一般用**解空间树（也称状态空间树）**的方式组织。
回溯法从根结点出发，按照深度优先策略遍历解空间树，搜索满足约束条件的解。
在用回溯法求解问题时，常常遇到两种典型的解空间树：
（1）**子集树（Subset Trees）：**当所给问题是从 n 个元素的集合中找出满足某种性质的子集

时，相应的解空间树称为子集树。在子集树中，$|S1|=|S2|=\ldots=|Sn|=c$，即每个结点有相同数目的子树，通常情况下 $c=2$，所以，子集树中共有 2^n 个叶子结点，因此，**遍历子集树需要Ω(2^n)时间。**

（2）**排列树（Permutation Trees）**：当所给问题是确定 n 个元素满足某种性质的排列时，相应的解空间树称为排列树。在排列树中，通常情况下，$|S_1|=n$，$|S_2|=n-1$，…，$|S_n|=1$，所以，排列树中共有 $n!$个叶子结点，因此，**遍历排列树需要Ω($n!$)时间。**

回溯法实际也是穷举,最坏情况下的时间代价肯定是指数阶.

分支限界法首先确定一个合理的限界函数，并根据限界函数确定目标函数的界[*down, up*]

按照广度优先策略遍历问题的解空间树，在分支结点上，依次搜索该结点的所有孩子结点，分别估算这些孩子结点的目标函数的可能取值。如果某孩子结点的目标函数可能取得的值超出目标函数的界，则将其丢弃，因为从这个结点生成的解不会比目前已经得到的解更好；否则，将其加入待处理结点表（以下简称表 PT）中。依次从表 PT 中选取使目标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。

分治法总体思想

将一个难以解决的大问题分割为 k 个子问题, 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。 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

分治法的设计思想是

1) 将一个难以直接解决的大问题，分割成一些规模较小的子问 ;这些子问题互相独立且与原问题相同; 2)递归地解子问题; 3) 将各个子问题的解合并得到原问题的解.

直接或间接地调用自身的算法称为**递归算法**

合并排序

基本思想 将待排序元素分成大小大致相同的 2 个子集合分别对 2 个子集合进行排序 最终将排好序的子集合合并成为所要求的排 好序的集合。

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

快速排序

最坏时间复杂度：O(n2) 平均时间复杂度：O(nlogn) 辅助空间：O(n)或 O(logn)

快速排序算法运行时间与划分是否对称有关