一、简答题：

1. **分别给出复杂度为O(nlogn)、O(n)、O(n2)、O(n3)的例子。**
2. O(nlogn)：快排、堆排序、归并排序
3. O(n)：桶排序、基数排序
4. O(n2)：冒泡排序、插入排序，选择排序
5. O(n3)：传统Diff算法、两个n阶矩阵的乘法运算
6. **举例说明贪心算法、动态规划、回溯、广搜等的算法复杂度（例子+算法框架，不用源码）。**
7. 贪心算法

 示例： 有n个物体，第i个物体的重量为wi（wi为正整数）。选择尽量多的物体，使得总重量不超过C。

由于只关心选择的物品的最大数量，所以装重的物体没有装轻的物体划算。这里只需对n个物体按重量递增排序，依次选择每个物体直到装不下为止。

int ans=0,sum=0; //ans-选择的物品数 sum-当前物品总装量

sort(w,w+n); //按物品重量递增排序

for(i=0;i<n;i++)

{

if(sum+w[i]<C) //如果能装载第i件物品，装载之

{

sum+=w[i];

ans++;

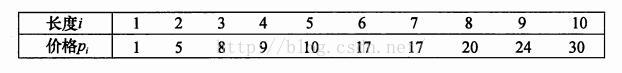
}

}

有上述示例显示，该贪心算法的算法复杂度为O（n）

1. 动态规划

示例：某公司购买钢条，并将其切割成若干段，将之出售。已知不同长度钢条的售价如下：

给定一个长度为n的钢条，该钢条经过有效切割，最多能卖出多少钱？

本题的局部最优解：长度为i的钢条最多可卖多少钱

最后解为：长度为n的钢条可卖多少钱。通过分析可以知道长度为n的钢条的最大收益：

（1）不切割，一整条出售的价格pn；

（2）将钢条分割成长度i j两段的最大收益，ri+rj；

for(int i=1;i<=n;i++)//从开始慢慢的往长度n推导

{

x=0;//价值

for(int j=1;j<=10;j++)//进行不同的切割

{

if(i<j)

break;

else

x=max(x,res[i-j]+m[j]);//进行不同的切割后，将切割后的价值进行比较，保留最大价值

}

res[i]=x;

}

有上述示例显示，该动态规划算法的算法复杂度为O（n^2）

1. 回溯

示例：在n\*n的棋盘上放置n个皇后，使他们互不攻击，皇后可以攻击与之处在同一行或同一列或同一条斜线上的其他皇后

void find(int a,int n)

{

for(int i=1;i<=n;i++)//遍历第i行的全部数

{

bool m=true;

for(int j=1;j<a;j++)//第i行前面的几行

if(res[j]==i||res[j]+a-j==i||res[j]-a+j==i)//判断两个皇后之间会不会互相相吃

{

m=false;

break;

}

if(m)

{

res[a]=i;//记录第几行应该放在那里

find(a+1,n);

}

}

}

有上述示例显示，该回溯的算法复杂度为O（n^2）

1. 广搜

示例:有一个由数字 0、1 组成的方阵中，存在一任意形状的封闭区域，封闭区域由数字1 包围构成，每个节点只能走上下左右 4 个方向。现要求只把【最大封闭区域】内的空间填写成2 。

使用广搜，把封闭区域找到

void BFS(int a,int b)

{

if(num[a][b]==1||num[a][b]==-1)//当当前格子已经确定状态后，返回

return ;

if(a==0||b==0||a==n-1||b==n-1)//如果为边界，直接定义为a-1（就是不需填写内容）

num[a][b]=-1;

if(a!=0&&res[a-1][b]!=1)//向上

{

res[a][b]=1;

DFS(a-1,b);

if(num[a-1][b]==-1)//说明没有被包围，填写为-1

num[a][b]=-1;

}

if(a!=n-1&&res[a+1][b]!=1)//向下

{

res[a][b]=1;

DFS(a+1,b);

if(num[a+1][b]==-1)

num[a][b]=-1;

}

if(b!=0&&res[a][b-1]!=1)//向左

{

res[a][b]=1;

DFS(a,b-1);

if(num[a][b-1]==-1)

num[a][b]=-1;

}

if(b!=n-1&&res[a][b+1]!=1)//向右

{

res[a][b]=1;

DFS(a,b+1);

if(num[a][b+1]==-1)

num[a][b]=-1;

}

}

有上述示例显示，该广搜算法的算法复杂度为O（n^2）

1. **举例说明贪心算法的特点。**

贪心算法（又称贪婪算法），在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题他能产生整体最优解或者是整体最优解的近似解。

如第二题所示示例中：由于只关心选择的物品的最大数量，所以装重的物体没有装轻的物体划算。这里只需对n个物体按重量递增排序，依次选择每个物体直到装不下为止。这就体现了贪心算法中显示的在某种意义上的局部最优解。

1. **举例说明动态规划的特点。**

（1）最优子结构性质。如果问题的最优解所包含的子问题的解也是最优的，我们就称该问题具有最优子结构性质。最优子结构性质为动态规划算法解决问题提供了重要线索。

（2）无后效性。即子问题的解一旦确定，就不再改变，不受在这之后、包含它的更大的问题的求解决策影响。

（3）子问题重叠性质。子问题重叠性质是指在用递归算法自顶向下对问题进行求解时，每次产生的子问题并不总是新问题，有些子问题会被重复计算多次。动态规划算法正是利用了这种子问题的重叠性质，对每一个子问题只计算一次，然后将其计算结果保存在一个表格中，当再次需要计算已经计算过的子问题时，只是在表格中简单地查看一下结果，从而获得较高的效率。

如第二题所示示例中：对于为了求解规模为n的原问题，先求解形式完全一样，但规模更小的子问题。即当完成首次切割后，我们将两段钢条看成两个独立的钢条切割问题实例。我们通过组合两个相关子问题的最优解，并在所有可能的两段切割方案中选取组合收益最大者，构成原问题的最优解。我们称钢条切割问题满足最优子结构（optimal substructure）性质：问题的最优解由相关子问题的最优解组合而成，而这些子问题可以独立求解。

1. **给出子集树的算法框架和排列树的算法框架。**

（1）子集树：

void Backtrack(int t){

if(t>n)

Output(x);

else{

for(int i=0;i<=1;i++){

x[t]=i;

if(Constraint(t)&&Bound(t))

Backtrack(t+1);

}

}

}

（2）排列树：

void Backtrack(int t){

if(t>n)

Output(x);

else{

for(int i=t;i<=n;i++){

Swap(x[t],x[i]);

if(Constraint(t)&&Bound(t))

Backtrack(x[t],x[i]);

Swap(x[t],x[i]);

}

}

}

1. **说明深搜和广搜的特点。**

深搜：

1. 深度优先搜索法有递归以及非递归两种设计方法。一般的，当搜索深度较小、问题递归方式比较明显时，用递归方法设计好，它可以使得程序结构更简捷易懂。当数据量较大时，由于系统堆栈容量的限制，递归容易产生溢出，用非递归方法设计比较好。
2. 深度优先遍历图的思想是，从图中某顶点v出发：访问顶点v；依次从v的未被访问的邻接点出发，对图进行深度优先遍历；直至图中和v有路径相通的顶点都被访问；若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行深度优先遍历，直到图中所有顶点均被访问过为止。

广搜：

1. 在产生新的子结点时，深度越小的结点越先得到扩展，即先产生它的子结点。
2. 当结点到跟结点的费用（有的书称为耗散值）和结点的深度成正比时，特别是当每一结点到根结点的费用等于深度时，用广度优先法得到的解是最优解，但如果不成正比，则得到的解不一定是最优解。这一类问题要求出最优解，一种方法是使用后面要介绍的其他方法求解，另外一种方法是改进前面深度（或广度）优先搜索算法：找到一个目标后，不是立即退出，而是记录下目标结点的路径和费用，如果有多个目标结点，就加以比较，留下较优的结点。把所有可能的路径都搜索完后，才输出记录的最优路径。
3. 广度优先搜索算法，一般需要存储产生的所有结点，占的存储空间要比深度优先大得多，因此程序设计中，必须考虑溢出和节省内存空间得问题。
4. 比较深度优先和广度优先两种搜索法，广度优先搜索法一般无回溯操作，即入栈和出栈的操作，所以运行速度比深度优先搜索算法法要快些。
5. **有哪些概率算法，各自的特点是什么？**

概率算法大致分为四类：数值随机化算法、蒙特卡洛算法、拉斯维加斯算法和舍伍德算法。

数值随机化算法：常用于数值问题的求解，所获得的往往是近似解，而且近似解的精准度随计算时间的增加而不断提高。

蒙特卡洛算法：主要用于求解问题的准确解，对于许多问题来说，近似解毫无意义。用蒙特卡洛算法可以求得一个解，但是这个解未必是正确的。它求得正确解的概率随计算时间的增加而不断提高。所以它的缺点就是无法有效的判断所获得的的解是不是正确解。

拉斯维加斯算法：它不会得到不正确的解。就是一但使用它找到一个解，这个解就一定是正确解，但是有时它会找不到解。同时它找到正确解的概率随计算时间的增加而不断提高。而且用同一拉斯维加斯算法反复对该实例求解多次，可使求解失效的概率变小

舍伍德算法：它总能求得一个解，并且求得的解是正确的。当一个确定性算法在最坏情况和平均情况下差别较大时可在这个确定型算法中引入随机性将之改造成一个舍伍德算法。

1. **单纯性算法的数学原理是什么？**

单纯性算法基本原理及主要步骤是：

1. 首先设法找到一个（初始）基可行解，
2. 然后再根据最优性理论判断这个基可行解是否最优解。
3. 若是最优解，则输出结果，计算停止；若不是最优解，则设法由当前的基可行解产生一个目标值更优的新的基可行解
4. 再利用最优性理论对所得的新基可行解进行判断，看其是否最优解，这样就构成一个迭代算法。

求解线性规划问题的目的就是要找出最优解。  
　　可能出现下列情况之一：（1）存在着一个最优解；（2）存在着无穷多个最优解；（3）不存在最优解，这只在两种情况下发生，即没有可行解或各项约束条件不阻止目标函数的值无限增大（或向负的方向无限增大）。  
　　要缩小对最优解的搜索范围，就必须认识最优解的一般性质，最优解如果存在的话，则它必然处于可行区域的边界上。  
　　任何一项约束条件的边界[方程](http://www.baike.com/wiki/方程)是用“＝”号来替换该约束条件中的“≤”或“≥”号而得到的。每一个边界方程确定一个超平面。因此，可行区域的边界是由那些满足一个或同时满足几个[边界方程](javascript:linkredwin('边界方程');)（即处在作为边界的一个或几个超平面上）的可行解所组成，而且最优解必在其中。最优解不仅是在可行区域的边界上，而且也在这个区域的一个隅角上。一个可行解，如果不处在由另两个可行解连接起来的任何线段上，它就是一个角点可行解。如果连接两个角点可行解的线段处在可行区域的边界上，这两个角点可行解就称为相邻的角点可行解。角点可行解具有下列三个重要性质：（1）如果存在着一个最优解，那么它必定是角点可行解。如果存在有多个最优解，那么至少有两个最优解必定是相邻的角点可行解。（2）只存在有限个数的角点可行解。（3）如果一个角点可行解按目标[函数值](http://www.baike.com/wiki/函数值)来衡量时比其所有的相邻角点可行解更好一些，那它就比所有其他角点可行解都更好，也就是最优解。

数学原理：

设：A=（B , N）（B为一个基，即线性无关向量组R(A)=R(B)）

XT= (XB , XN)T （XB 为基变量，XN为非基变量）

C= (CB , CN) （CB 为基变量系数，CN为非基变量系数）

则有：Z= (CB , CN) (XB , XN) T= CB XB+CN XN

AX =（ B , N） (XB , XN) T = B XB+ N XN = b

因为B为基, 故有 XB +B-1N XN = B-1b，

解得可行解XB=B-1b-B-1NXN，代入目标函数Z，

Z= CB B-1b + (CN- CB B-1N ) XN

令非基变量XN = 0 ，则有 XT = (XB , XN) T =（ B-1b , 0) T

Z = CB B-1b

所以：

如果CN- CB B-1N 小于0，无论XN取任何大于0值，只会让Z变小，因此我们可以通过CN- CB B-1N 来判断Z取得是不是最大值。

如果存在一个CN- CB B-1N 大于0，则说明Z的值会随着XN增大而增大，说明Z有调整的余地。

定理一：若某个基本可行解所对应的检验向量CN- CB B-1N<=0,则这个基本可行解就是最优解。

定理二：若某个基本可行解所对应的检验向量CN- CB B-1N 存在一个检验数=0,则该问题有无数多个最优解。

定理三：若某个基本可行解所对应的检验向量CN- CB B-1N j大于0，且aij,都小于0，则无解。

1. 【选做题】

给一个你最喜欢的算法题和源码、测试数据和结果（课堂和实验未涉及的题，可以自己设计）；并说明原因。