1. 逻辑回归估计参数时的目标函数 (就是极大似然估计那部分)(三次)，

逻辑回归本质上就是先进行线性回归然后进行Logsitic函数进行非线性转换，至于所以用logistic而不用其它，是因为这种归一化的方法往往比较合理（人家都说自己叫logistic了嘛 呵呵），能够打压过大和过小的结果（往往是噪音），以保证主流的结果不至于被忽视

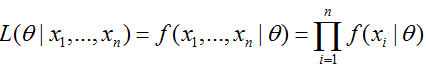
最大似然估计提供了一种给定观察数据来评估模型参数的方法，即：“模型已定，参数未知”

最大似然估计中采样需满足一个很重要的假设，就是所有的采样都是独立同分布的。下面我们具体描述一下最大似然估计：

首先，假设IMG_256为独立同分布的采样，θ为模型参数,f为我们所使用的模型，遵循我们上述的独立同分布假设。参数为θ的模型f产生上述采样可表示为

IMG_257

回到上面的“模型已定，参数未知”的说法，此时，我们已知的为IMG_258，未知为θ，故似然定义为:



　　在实际应用中常用的是两边取对数，得到公式如下：

IMG_260

　　其中IMG_261称为对数似然，而IMG_262称为平均对数似然。而我们平时所称的最大似然为最大的对数平均似然，即：

IMG_263

**逻辑回归的特征为什么先要进行离散化**：

在工业界，很少直接将连续值作为特征喂给逻辑回归模型，而是将连续特征离散化为一系列0、1特征交给逻辑回归模型，这样做的优势有以下几点：

1. 稀疏向量内积乘法运算速度快，计算结果方便存储，容易scalable（扩展）。

2. 离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄>30是1，否则0。如果特征没有离散化，一个异常数据“年龄300岁”会给模型造成很大的干扰。

3. 逻辑回归属于广义线性模型，表达能力受限；单变量离散化为N个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合。

4. 离散化后可以进行特征交叉，由M+N个变量变为M\*N个变量，进一步引入非线性，提升表达能力。

5. 特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问。

李沐少帅指出，模型是使用离散特征还是连续特征，其实是一个“海量离散特征+简单模型” 同 “少量连续特征+复杂模型”的权衡。既可以离散化用线性模型，也可以用连续特征加深度学习。就看是喜欢折腾特征还是折腾模型了。通常来说，前者容易，而且可以n个人一起并行做，有成功经验；后者目前看很赞，能走多远还须拭目以待。

大概的理解：

1）计算简单

2）简化模型

3）增强模型的泛化能力，不易受噪声的影响

1. 翻转二叉树 写代码

（3）逻辑回归估计参数时的目标函数 如果加上一个先验的服从高斯分布的假设，会是什么样  （其实就是在后面乘一个东西，取log后就变成加一个东西，实际就变成一个正则项）

先验概率分布乘以似然函数为后验概率分布，L2范数相当于加入了一个Gaussian先验。

（4）SVM在哪个地方引入的核函数

SVM在当数据不是线性的时候，或者在低维不可分的时候需要把它转化到更高维的线性可分，但是在高维空间的时候就会出现所谓的维数灾难，计算复杂量大，这时候就需要核函数，通过内积运算使得在低维空间到达与在高维空间一样的效果。这样节省计算量。

（5）如果用高斯核可以升到多少维  （无限维）

（6）什么是贝叶斯估计

极大似然估计和贝叶斯估计分别代表了频率派和贝叶斯派的观点。频率派认为，参数是客观存在的，只是未知而矣。因此，频率派最关心极大似然函数，只要参数求出来了，给定自变量X，Y也就固定了，极大似然估计如下所示:

IMG_256

D表示训练数据集，IMG_257是模型参数

相反的，贝叶斯派认为参数也是随机的，和一般随机变量没有本质区别，正是因为参数不能固定，当给定一个输入x后，我们不能用一个确定的y表示输出结果，必须用一个概率的方式表达出来，所以贝叶斯学派的预测值是一个期望值，如下所示：

IMG_258

其中x表示输入，y表示输出，D表示训练数据集，IMG_259是模型参数

该公式称为全贝叶斯预测。现在的问题是如何求IMG_260（后验概率），根据贝叶斯公式我们有：



可惜的是，上面的后验概率通常是很难计算的，因为要对所有的参数进行积分，不能找到一个典型的闭合解（解析解）。在这种情况下，我们采用了一种近似的方法求后验概率，这就是最大后验概率。

IMG_262

最大后验概率和极大似然估计很像，只是多了一项先验分布，它体现了贝叶斯认为参数也是随机变量的观点，在实际运算中通常通过超参数给出先验分布。

从以上可以看出，一方面，极大似然估计和最大后验概率都是参数的点估计。在频率学派中，参数固定了，预测值也就固定了。最大后验概率是贝叶斯学派的一种近似手段，因为完全贝叶斯估计不一定可行。另一方面，最大后验概率可以看作是对先验和MLE的一种折衷，如果数据量足够大，最大后验概率和最大似然估计趋向于一致，如果数据为0,最大后验仅由先验决定。

1. k折交叉验证 中k取值多少有什么关系 （知道bias和variance吗？估计是和这两个东西有关，  知乎上有个问题讨论了k值大小与bias和variance的关系）

首先 Error = Bias + Variance+Noise  
Error反映的是整个模型的准确度，Bias反映的是模型在样本上的输出与真实值之间的误差，即模型本身的精准度，Variance反映的是模型每一次输出结果与模型输出期望之间的误差，即模型的稳定性。Bias大，欠拟合，Variance大，过拟合

举一个例子，一次打靶实验，目标是为了打到10环，但是实际上只打到了7环，那么这里面的Error就是3。具体分析打到7环的原因，可能有两方面：一是瞄准出了问题，比如实际上射击瞄准的是9环而不是10环；二是枪本身的稳定性有问题，虽然瞄准的是9环，但是只打到了7环。那么在上面一次射击实验中，Bias就是1,反应的是模型期望与真实目标的差距，而在这次试验中，由于Variance所带来的误差就是2，即虽然瞄准的是9环，但由于本身模型缺乏稳定性，造成了实际结果与模型期望之间的差距。

在一个实际系统中，Bias与Variance往往是不能兼得的。如果要降低模型的Bias，就一定程度上会提高模型的Variance，反之亦然。造成这种现象的根本原因是，我们总是希望试图用有限训练样本去估计无限的真实数据。当我们更加相信这些数据的真实性，而忽视对模型的先验知识，就会尽量保证模型在训练样本上的准确度，这样可以减少模型的Bias。但是，这样学习到的模型，很可能会失去一定的泛化能力，从而造成过拟合，降低模型在真实数据上的表现，增加模型的不确定性。相反，如果更加相信我们对于模型的先验知识，在学习模型的过程中对模型增加更多的限制，就可以降低模型的variance，提高模型的稳定性，但也会使模型的Bias增大。Bias与Variance两者之间的trade-off是机器学习的基本主题之一，机会可以在各种机器模型中发现它的影子。

具体到K-fold Cross Validation的场景，其实是很好的理解的。首先看Variance的变化，还是举打靶的例子。假设我把抢瞄准在10环，虽然每一次射击都有偏差，但是这个偏差的方向是随机的，也就是有可能向上，也有可能向下。那么试验次数越多，应该上下的次数越接近，那么我们把所有射击的目标取一个平均值，也应该离中心更加接近。更加微观的分析，模型的预测值与期望产生较大偏差，在模型固定的情况下，原因还是出在数据上，比如说产生了某一些异常点。在最极端情况下，我们假设只有一个点是异常的，如果只训练一个模型，那么这个点会对整个模型带来影响，使得学习出的模型具有很大的variance。但是如果采用k-fold Cross Validation进行训练，只有1个模型会受到这个异常数据的影响，而其余k-1个模型都是正常的。在平均之后，这个异常数据的影响就大大减少了。相比之下，模型的bias是可以直接建模的，只需要保证模型在训练样本上训练误差最小就可以保证bias比较小，而要达到这个目的，就必须是用所有数据一起训练，才能达到模型的最优解。因此，k-fold Cross Validation的目标函数破坏了前面的情形，所以模型的Bias必然要会增大。

所以K折交叉验证，初始采样分割成K个子样本，一个单独的子样本被保留作为验证模型的数据，其他K-1个样本用来训练。交叉验证重复K次，每个[子样本](https://link.zhihu.com/?target=http://baike.baidu.com/view/1377649.htm" \t "https://www.zhihu.com/question/_blank)验证一次，平均K次的结果或者使用其它结合方式，最终得到一个单一估测。

当K值大的时候， 我们会有更少的Bias(偏差), 更多的Variance。  
当K值小的时候， 我们会有更多的Bias(偏差), 更少的Variance。

（8）过拟合了怎么办

（9）设计一个结构存取稀疏矩阵  （面试官最后告诉我了一个极度压缩的存法，相同行或列存偏差，我当时没听懂，还不懂装懂，最后还是没记住）

（10）平衡二叉树是什么

（11）系统设计题，给一个query，如何快速从10亿个query中找出和它最相似的 （面试官说可以对每个query找1000个最相似的，存起来，每天离线更新）

（12）编程题 给了一个链表，第1个结点标号为1，把链表中标号在M到N区间的部分反转  （见文档单链表的部分反转）

1. 编程题 在一个字符串中，找出最长的无重复字符的子串 （这个问题还是有难度的，我碰巧想到了用hash，但没想到完整解法，用hash的解法复杂度在O(N^2)，在网上查了下，好像还有更快解法）

例如，在"abcabcbb"中，其无重复字符的最长子字符串是"abc"，其长度为 3。

对于，"bbbbb"，其无重复字符的最长子字符串为"b"，长度为1。

O(n) 时间

#### 分析：遍历该字符串，每遍历一个字母时，利用map去找该字母最近一次出现是什么时候，中间这一段便是无重复字符的字符串。

代码：

1. **class** Solution {
2. **public**:
3. /\*\*
4. \* @param s: a string
5. \* @return: an integer
6. \*/
7. **int** lengthOfLongestSubstring(string s) {
8. // write your code here
9. **int** ret = 0;
10. map<**char**, **int**> m;
11. **int** start = 1;
12. **for** (**int** i = 1; i <= s.length(); i++)
13. {
14. **char** c = s[i - 1];
15. **if** (m[c] >= start)
16. {
17. start = m[c] + 1;
18. m[c] = i;
20. }
21. **else**
22. {
23. m[c] = i;
24. ret = max(ret, i - start + 1);
25. }
27. }
28. **return** ret;
29. }
30. };

（14）逻辑回归的值表示概率吗？（值越大可能性越高，但不能说是概率）

Logistic regression （逻辑回归）是当前业界比较常用的**[机器学习](http://lib.csdn.net/base/2" \o "机器学习知识库" \t "http://blog.csdn.net/suranxu007/article/details/_blank)**方法，用于估计某种事物的可能性。比如某用户购买某商品的可能性，某病人患有某种疾病的可能性，以及某广告被用户点击的可能性等。（注意这里是：“可能性”，而非数学上的“概率”，logisitc回归的结果并非数学定义中的概率值，不可以直接当做概率值来用。该结果往往用于和其他特征值加权求和，而非直接相乘）

**Logistic Regression的适用性**

1） 可用于概率预测，也可用于分类。

       并不是所有的机器学习方法都可以做可能性概率预测（比如SVM就不行，它只能得到1或者-1）。可能性预测的好处是结果又可比性：比如我们得到不同广告被点击的可能性后，就可以展现点击可能性最大的N个。这样以来，哪怕得到的可能性都很高，或者可能性都很低，我们都能取最优的topN。当用于分类问题时，仅需要设定一个阈值即可，可能性高于阈值是一类，低于阈值是另一类。

2） 仅能用于线性问题

       只有在feature和target是线性关系时，才能用Logistic Regression（不像SVM那样可以应对非线性问题）。这有两点指导意义，一方面当预先知道模型非线性时，果断不使用Logistic Regression； 另一方面，在使用Logistic Regression时注意选择和target呈线性关系的feature。

3）各feature之间不需要满足条件独立假设，但各个feature的贡献是独立计算的。

       逻辑回归不像朴素贝叶斯一样需要满足条件独立假设（因为它没有求后验概率）。但每个feature的贡献是独立计算的，即LR是不会自动帮你combine 不同的features产生新feature的 (时刻不能抱有这种幻想，那是决策树,LSA, pLSA, LDA或者你自己要干的事情)。举个例子，如果你需要TF\*IDF这样的feature，就必须明确的给出来，若仅仅分别给出两维 TF 和 IDF 是不够的，那样只会得到类似 a\*TF + b\*IDF 的结果，而不会有 c\*TF\*IDF 的效果。

（15）分类模型和回归模型的区别

**分类问题**和**回归问题**都要根据训练样本找到一个**实值函数g(x)**.  
**回归问题**是: 给定一个新的模式, 根据训练集推断它所对应的**输出y(实数)**是多少, 也就是使用**y=g(x)**来推断任一输入x所对应的输出值.  
**分类问题**是: 给定一个新的模式, 根据训练集推断它所对应的**类别(如: +1, -1)**, 也就是使用**y=sign(g(x))**来推断任一输入x所对应的类别.  
综上, 回归问题和分类问题的**本质一样**, 不同仅在于他们的输出的**取值范围不同**.  
分类问题中, 输出只允许取离散的几个值; 而在回归问题中, 输出可取任意实数. 分类一般针对**离散型结果**而言的, 回归是针对**连续型结果**的. 本质上是一样的.

## 1.回归问题的应用场景

回归问题通常是用来预测一个值，如预测房价、未来的天气情况等等，例如一个产品的实际价格为500元，通过回归分析预测值为499元，我们认为这是一个比较好的回归分析。一个比较常见的回归算法是线性回归算法（LR）。另外，回归分析用在神经网络上，其最上层是不需要加上softmax函数的，而是直接对前一层累加即可。回归是对真实值的一种逼近预测。

## 2.分类问题的应用场景

分类问题是用于将事物打上一个标签，通常结果为离散值。例如判断一幅图片上的动物是一只猫还是一只狗，分类通常是建立在回归之上，分类的最后一层通常要使用softmax函数进行判断其所属类别。分类并没有逼近的概念，最终正确结果只有一个，错误的就是错误的，不会有相近的概念。最常见的分类方法是逻辑回归，或者叫逻辑分类。

（16）分类模型可以做回归分析吗？反过来可以吗？（我回答是分类不可以做回归，回归倒是可以做分类，不知道对不对）

(17设计一个合理的电梯调度策略，调度两个电梯 ，考虑满足基本的接送需求，满足能耗最小，满足用户等待时间最短

（难到我了，我想的方法不好，面试告诉我了他的想法，类似于一个进程调度问题，每一时刻只可能有一个用户按按钮，把这条指令接收，判断当前电梯能否满足，能满足就执行，不能满足则放入一个队列里，实际情况还要细化）

(18)常用排序算法的时间和空间复杂度

**直接插入排序**所耗费的时间有很大差异。若文件初态为正序，则每个待插入的记录只需要比较一次就能够找到合适的位置插入，故算法的时间复杂度为O(n)，这时最好的情况。若初态为反序，则第i个待插入记录需要比较i+1次才能找到合适位置插入，故时间复杂度为O(n2)，这时最坏的情况。

直接插入排序的平均时间复杂度为O(n2)。

**二分插入排序**的比较次数与待排序记录的初始状态无关，仅依赖于记录的个数。当n较大时，比直接插入排序的最大比较次数少得多。但大于直接插入排序的最小比较次数。算法的移动次数与直接插入排序算法的相同，最坏的情况为n2/2，最好的情况为n，平均移动次数为O(n2)。

**希尔排序的**时间性能优于直接插入排序，原因如下：

　　（1）当文件初态基本有序时直接插入排序所需的比较和移动次数均较少。

　　（2）当n值较小时，n和n2的差别也较小，即直接插入排序的最好时间复杂度O(n)和最坏时间复杂度0(n2)差别不大。

　　（3）在希尔排序开始时增量较大，分组较多，每组的记录数目少，故各组内直接插入较快，后来增量di逐渐缩小，分组数逐渐减少，而各组的记录数目逐渐增多，但由于已经按di-1作为距离排过序，使文件较接近于有序状态，所以新的一趟排序过程也较快。

　　因此，希尔排序在效率上较直接插人排序有较大的改进。

　　希尔排序的平均时间复杂度为O(nlogn)。

**简单选择排序**是不稳定的排序。

　　时间复杂度：T(n)=O(n2)。

**堆排序**也是一种不稳定的排序算法。

　　堆排序优于简单选择排序的原因：

　　直接选择排序中，为了从R[1..n]中选出关键字最小的记录，必须进行n-1次比较，然后在R[2..n]中选出关键字最小的记录，又需要做n-2次比较。事实上，后面的n-2次比较中，有许多比较可能在前面的n-1次比较中已经做过，但由于前一趟排序时未保留这些比较结果，所以后一趟排序时又重复执行了这些比较操作。

　　堆排序可通过树形结构保存部分比较结果，可减少比较次数。

　　堆排序的最坏[时间复杂度](http://www.cnblogs.com/view/104946.htm" \t "http://www.cnblogs.com/liuling/p/_blank)为**O(nlogn)**。堆序的平均性能较接近于最坏性能。由于建初始堆所需的比较次数较多，所以堆排序不适宜于记录数较少的文件。

**冒泡排序**是一种稳定的排序方法。

•若文件初状为正序，则一趟起泡就可完成排序，排序码的比较次数为n-1，且没有记录移动，时间复杂度是O(n)

•若文件初态为逆序，则需要n-1趟起泡，每趟进行n-i次排序码的比较，且每次比较都移动三次，比较和移动次数均达到最大值∶O(n2)

•起泡排序平均时间复杂度为O(n2)

**快速排序**是不稳定的排序。

　　快速排序的时间复杂度为O(nlogn)。

　　当n较大时使用快排比较好，当序列基本有序时用快排反而不好。

**归并排序**是稳定的排序方法。

　　归并排序的时间复杂度为O(nlogn)。

　　速度仅次于快速排序，为稳定排序算法，一般用于对总体无序，但是各子项相对有序的数列。

**基数排序**是稳定的排序算法。

　　基数排序的时间复杂度为O(d(n+r)),d为位数，r为基数。

一、稳定性:

　   稳定：冒泡排序、插入排序、归并排序和基数排序

　　不稳定：选择排序、快速排序、希尔排序、堆排序

二、平均时间复杂度

　　O(n^2):直接插入排序，简单选择排序，冒泡排序。

　　在数据规模较小时（9W内），直接插入排序，简单选择排序差不多。当数据较大时，冒泡排序算法的时间代价最高。性能为O(n^2)的算法基本上是相邻元素进行比较，基本上都是稳定的。

　　O(nlogn):快速排序，归并排序，希尔排序，堆排序。

　　其中，快排是最好的， 其次是归并和希尔，堆排序在数据量很大时效果明显。

（19）问会什么机器学习算法，那讲下SVM的原理吧   （我把SVM的公式推导了一部分，面试官就着公式问了一些问题）

（20）你简历里说用过random forest 和 GBDT ，讲一下他们的区别

（21）数据库索引用什么建的 （b+树，并不知道原理）

（22）tcp关闭连接的过程， 为什么要四次？

（23）写程序 str2Int

（24）54张牌，分成6份，每份9张牌，大小王在一起的概率（我算错了，他告诉我是 8/53）

3\* C(18,2) / C(54,2)

（25）问了操作系统的内存管理， 分页和分段的区别

（26）银行家算法原理

（27）机器学习的大概过程是什么？

（28）特征选取怎么选？ 为什么信息增益可以用来选特征？

哪个项目是跟人合作的？（在实验室的项目，和导师合作）

有没有和导师意见不同的时候，怎么做的？

简历里有个项目是作为负责人，带领低年级学生做的，讲讲怎么带的？

低年级学生不听话怎么办？ （学院的项目，我讲到学院项目给钱给的不及时，有时候会自己掏钱请他们吃饭，当做是一种激励，面试官笑了。。。。）

有没有接触过机器学习的前沿，深度学习看过paper没有？（并没有）

（29）写程序  最短编辑距离

（30）如何从很多的query中找出一个query （我开始想到hash，后来经提示我想到了前缀树）

若允许有错误，可以再怎么解决（不知道，面试官提示了布隆过滤器）

（31）问我知道倒排索引不 ? (不知道)

想考察我学习能力，所以决定教教我倒排索引 ，简单画了个图，问了相关的问题  （答得不好）

（32）很多数中找前k大数

最有成就感的事？

和计算机专业的比有什么优势，什么劣势？

给你一个整数 m(m>2),写一个函数，输出所有小于m的素数。第二个程序：就要我就要我写堆排序中构建最小堆的程序，原来的堆是最小堆，插入一个值后如何保证 它还是最小堆。

如果输入关键词“货车追尾”，怎么样能识别出“货车”其实应该是“火车”。我说可以根据上下文语义，然后根据条件概率求最有可能的结果，然后他又问我如何根据上下文语义呢？我就噼里啪啦的讲的一大堆，可能比较乱，后来仔细一想其实很简单，就是多音字识别问题，和词义消歧问题差不多。  
 后来他问了我一些HMM和语言模型的联系

对于数组A[]中的每个A[i]求A[0~i-1]中比A[i]小且距离A[i]位置最近的数，如果没有则为-1，要求时间 复杂度为O(n)

判断两颗二叉树是否相等。我问有什么要求没？他说没有。我就开始写了。我的第一个思路是：如果两颗树前序 遍历和中序遍历结果都一样，那么这两棵树肯定是一样的，我就很快把程序写了。他说你这样是可以的，但能不能只遍历一次？我说我想想吧。我想了一下说，其实 可以很简单，如果这两棵树的数据结构定义成数组形式，那么只要这两个数组的每个元素相等就可以了啊。他说这种情况只能针对完全二叉树吧？我说如果不是完全 二叉树只要把原先的两颗二叉树变成完全二叉树不就行了吗，缺失的结点就是空元素嘛？虽然有点浪费空间，但也是可行的吧。然后我加了一句，当然还可以用层次 遍历啊，只要层次遍历的结果一样那么这两棵树肯定是一样的。不过层次遍历可能有点复杂，

实现strcopy(char \* dst, char \* csr, int len)要求考虑各种情况。写这个程序其实还好，只要考虑到一些特殊情况就行了。比如：是不是空指针，内存重叠等情况。

char \* strcpy(char \*dst,const char \*src)//源字符串参数用const修饰，防止修改源字符串。

{

assert(dst != NULL && src != NULL);//检查指针的有效性时使用assert

char \*ret = dst;//保存原始的strDest值

my\_memcpy(dst, src, strlen(src)+1);

return ret;

}

char \*my\_memcpy(char \*dst, const char\* src, int cnt)

{

assert(dst != NULL && src != NULL);

char \*ret = dst;

if (dst >= src && dst <= src+cnt-1) //内存重叠，从高地址开始复制

{

dst = dst+cnt-1;

src = src+cnt-1;

while (cnt--)

\*dst-- = \*src--;

}

else //正常情况，从低地址开始复制

{

while (cnt--)

\*dst++ = \*src++;

}

return ret;

}

百度中输入关键字，我们可能点击某个页面，或者点击下一页，我们可以将这些点击操作记录在日志文件 中，问如何通过这些日志文件分析两个关键字的行为相识度。我的回答是这样的：每个关键字对应一个url列表，例如关键字1对应list1(n个 url) ，关键字2对应list2(m个url)，计算对url进行结构分析，再计算list1中每个url与list2中url的相似度，求和在求平 均。

### 1、给你一个数组，设计一个既高效又公平的方法随机打乱这个数组（此题和洗牌[算法](http://www.xuebuyuan.com/category/%E7%AE%97%E6%B3%95" \o "算法" \t "http://www.xuebuyuan.com/_blank)的思想一致）

方法比较简单，基本思想是每次随机取一个数，然后把它交换到最后的位置。然后对前（n-1）个数使用递归的算法。

递归实现：

1. **void** suffle\_dfs(**int** ar[], **int** n)
2. {
3. **if**(n<=1)**return**;
4. swap(ar[n-1], ar[rand()%n]);
5. shuffle\_dfs(ar,n-1);
6. }

非递归实现：

1. void suffle(int ar[], int n)
2. {
3. while(n**>**1){
4. swap(ar[n-1], ar[rand()%n]);
5. n--;
6. }
7. }

所谓洗牌算法，就是给你一个1到n的序列，让你随机打乱，保证每个数出现在任意一个位置的概率相同，也就是说在n!个的排列中，每一个排列出现的概率相同。

我们考虑，当一个数被选之后，我们是没有必要在下一次随机的时候再考虑它的，因此，我们每次只从可选的数的集合中进行随机，也就不用考虑是否会碰到已经选过的数了，这样子直接将算法的复杂度降了一个数量级。

1. **void** MySwap(**int** &x, **int** &y)
2. {
3. **int** temp = x;
4. x = y;
5. y = temp;
6. }
8. **void** Shuffle(**int** n)
9. {
10. **for**(**int** i=n-1; i>=1; i--)
11. {
12. MySwap(num[i], num[rand()%(i+1)]);
13. }
14. }

 参照上面的代码，我们可以看到，我们每次都是随机的从没有选的数中选择一个，该洗牌算法的时间复杂度为O(n)，空间复杂度为O(0)。

.一百个灯泡排成一排，第一轮将所有灯泡打开；第二轮每隔一个灯泡关掉一个。即排在偶数的灯泡被关掉，第三轮每隔两个灯泡，将开着的灯泡关掉，关掉的灯泡打开。依次类推，第100轮结束的时候，还有几盏灯泡亮着。

思路：   
  
1．对于每盏灯，拉动的次数是奇数时，灯就是亮着的，拉动的次数是偶数时，灯就是关着的。   
2．每盏灯拉动的次数与它的编号所含约数的个数有关，它的编号有几个约数，这盏灯就被拉动几次。   
3．1—100这100个数中有哪几个数，约数的个数是奇数。   
  
**我们知道：一个数的约数都是成对出现的，只有完全平方数约数的个数才是奇数个。**

**def Light**(n):  
 bulk = [0]\*101;  
 **for** i **in** range(1,101):  
 j=0  
 **while**(j<101):  
 **if**(bulk[j]==0):  
 bulk[j] = 1  
 **else**:  
 bulk[j] = 0  
 j+=i  
 **for** k **in** range(101):  
 **if**(bulk[k]==1):  
 **print**(k)

所以这100盏灯中有10盏灯是亮着的，它们的编号分别是：1、4、9、16、25、36、49、64、81、100

2:公司组织一次羽毛球比赛，采用淘汰机制，假设公司有1001个人，如果要评出“公司羽毛球第一高手”的称号，至少需要进行多少场比赛？请简述设计过程，并写出代码模拟比赛过程。

int f500(int n)  
{  
 int N=0;  
 int temp;  
 while(n >1)  
 {  
  n=n/2+n%2;  
  N+=n;  
 }  
 return N;  
   
}

void main()  
{  
 cout<<f500(100)<<endl;  
   
}

1.求两个有序数组A和B的中位数，m、n为两个数组的长度，时间复杂度为O(log(m+n))

比较A[m/2] 和 B[n/2]大小，处理不同的比较结果

2）x-y坐标系中有一些点，估计这些点中的两两的最大长度

先求所有点的中心（横坐标为所有点横坐标的均值，纵坐标为所有点纵坐标的均值），求样本点到中心的最大距离d，那么2d为估计的两两点的最大距离，误差为d/2