习题参考答案

习题 2.1

1. 请写出一段代码来实现 Randomf(a, b) 和 Randomi(i,j)， 可以利用 C++或者 Java 提供的随机函数。

   C++提供了 rand()函数，Java 提供了 Math.random()，利用它们便可以达到目的。

   注意 Randomf(a,b)随机地返回一个实数 x，使得 a ≤ x ≤ b

   注意 Randomi(i,j)随机地返回一个实数 x，使得 i ≤ x ≤ j。

2. 假设我们不把 $B[i]$与子数组 $B[i..n]$中的任何一个随机元素交换，而把它与数组 $B[1..n]$中的任何一个元素交换：

   PermuteWithAll(B)

   1 $n \leftarrow$ length[B]

   2 **for** $i \leftarrow$ 1 **to** n **do**

   3 swap $B[i] \leftrightarrow B$[Randomi(1,n)]

   请问这段代码是否会产生一个一致的随机数组排列，为什么？

一致的随机数组排列指得是每种排列出现的概率是相等的。即对于一个数组，在算法所能产生的所有排列中，每种排列出现的次数是相等的。

由于上述算法能产生 n^n 个排列，而 n 个数的排列，共有 $n!$。因此上述算法如能产生一致随机排列，则存在整数 k，使得 $n^n = k\ n!$。当 $n = 1,2$ 时，这样的 k 是存在的，当 $n > 2$ 时，我们可以分解 n^n 为

$$n^n = (n^n - n^{n-1}) + (n^{n-1} - n^{n-2}) + \cdots + (n-1) + 1$$

n^n 除 $n-1$ 的余数为 1，而 $k\ n!$ 除 $n-1$ 的余数为 0，所以不存在 k 使得 $n^n = k\ n!$ 成立。因此上述算法能否产生一致排列，取决于 n 的大小。

3. 假定所有的优先权互不相同，PermuteBySorting(B)算法能产生数组 B 的一致随机排列吗？

   能。

我们首先考虑一个特殊的排列，每个数组元素 $A[i]$ 得到第 i 个最小的优先值，则此排列出现的概率为 $\frac{1}{n!}$。证明如下：令 X_i 表示数组元素 $A[i]$ 得到第 i 个最小的优先值，则有 $\Pr\{X_1\} = \frac{1}{n}$，则 $A[1]$ 被确定后，余下的 $n-1$ 个元素都有同样的机会得到第 2 个最小的优先值，因此 $\Pr\{X_2\} = \frac{1}{n-1}$，一般的，在前 i 个元素 $A[1]$，$A[i]$ 确定后，余下 $n-i$ 个元素都有同样的机会得到第 i+1 个最小的优先值，即 $\Pr\{X_{i+1} / X_1 \cap X_2 \cap \cdots \cap X_i\} = \frac{1}{n-i}$ 元素个元素

我们要计算的是，每个事件 X_i 都发生的概率，即 $\Pr\{X_1 \cap X_2 \cap \cdots \cap X_n\}$，利用概率知识

可得

$$\Pr\{X_1 \cap X_2 \cap \cdots \cap X_n\}$$

$$= \Pr\{X_1\} \times \Pr\{X_2 / X_1\} \times \cdots \times \Pr\{X_{i+1} / X_1 \cap X_2 \cap \cdots \cap X_i\} \times \cdots \times \Pr\{X_n / X_1 \cap X_2 \cap \cdots \cap X_{n-1}\}$$

因此有

$$\Pr\{X_1\} \times \Pr\{X_2 / X_1\} \times \cdots \times \Pr\{X_{i+1} / X_1 \cap X_2 \cap \cdots \cap X_i\} \times \cdots \times \Pr\{X_n / X_1 \cap X_2 \cap \cdots \cap X_{n-1}\}$$
$$= (\frac{1}{n})(\frac{1}{n-1}) \cdots (\frac{1}{2}) \cdots (1)$$
$$= \frac{1}{n!}.$$

考虑一般情形，对于 $\{1,2,\cdots,n\}$ 的一个排列 $\delta = <\delta_1, \cdots, \delta_n>$，我们可以类似上述证明获得该排列的概率为 $\frac{1}{n!}$。

4. 解释如何实现算法 PermuteBySorting(*B*)来处理两种或者更多优先权值相同的特殊情况。就是说，你的算法应该能够产生一个一致的随机排列，即使当两个或者更多个优先权值相同。
若优先级相同，则再次对这些优先级相同的数调用算法 PermuteBySorting(*B*)，或者对优先值相同时，根据 P[i]的值对数组 A 进行排序，如果数组中的两个元素或者更多的元素他们的 P[i]相同，则排序时保持它们之前在数组 A 中的相对位置不变。

5. 分析 HireAssistant(*n*)算法最好情形下和最坏情形下所需要的费用。

最好情形：第一次来面试就是最好的员工，费用为 $nC_i + C_h$ ；

最坏情形：来面试的一个比一个好，费用为 $n(C_i + C_h)$

6. 有时随机算法也称为概率算法，这两种称呼有区别吗？按照你自己的理解，能给出随机算法的新的分类吗？
随机算法和概率算法是相同的，他们在算法执行过程中进行某种随机性的选择，只是随机算法是从算法设计的角度来研究问题，而概率算法则是从算法分析的角度来研究问题。许多情况下，当算法在执行过程中面临一个选择时，随机性选择常常比最优性选择省时，因此，随即算法和概率算法可在很大程度上降低算法的复杂度。 它们的一个基本特征是对所求解的问题的同一输入用一个随机（概率）算法来求解两次可能得到完全不同的效果，这两次求解问题所需的时间甚至会有相当大的差别。

7. 思考一下，怎么样衡量随机算法的性能？
利用概率分析

习题 2.2

1． 请给出别的方法来估计积分

$$\Gamma = \int_a^b f(x)dx$$

的值，其中 $f(x)$ 是有界函数，且 $0 \le f(x) \le c, a \le x \le b.$

根据积分的定义，把区间[a,b]分成 n 个相等的小区间。所求的积分即所有这些小区间及函数图形所包围而成的图形的面积和，近似地认为这些图形都是长为（b-a）/n 的长方形。即

$$\Gamma = \int_a^b f(x)dx = \sum_{i=1}^{n}[f(a+(b-a)*i/n)*(b-a)/n]$$

该解的精确度与 n 的取值有关，n 越大，求得的积分的值越精确。这种方法我们可以很容易的在计算机上实现。还有别的方法，这里略。

2. 如何设计一个数值随机算法求解一个给定的线性方程组？

在解区域内随机选定一个点 X_0 作为初始解，然后随机扰动该解，得到新解 X_1，如果线性方程组 $f(X_1) < \varepsilon$，ε 为指定的一个非常小的常数，则停止搜索，否则令 $X_0 = X_1$，重复上述过程。

3. 请设计一个利用 Buffon 针近似 π 的模拟程序。

注意不得使用 π 的值。

习题 2.3

1. 实践中，当输入的数组已经基本有序的时候，我们可以利用插入排序运行速度快的优势来改进快速排序（QuickSort）的性能。当快速排序运行在一个元素个数比 k 小的子数组上的时候，我们无须对子数组进行排序，而是简单地直接返回。在最上层调用快速排序算法返回后，我们再运行插入排序完成对整个数组的排序过程。讨论这个排序算法的期望运行时间为 $O(nk + n\lg(n/k))$。在理论和实践上，我们应该如何选取 k 的值？

首先，对于一个差不多已经排好了序的数组，插入排序的运行时间为 $O(n)$。依题意，对于数组元素小于等于 k 的数组，快速排序算法将输入直接返回。所以，经过快速排序算法后，所有的划分的元素个数最大个数为 k，共划分为 n/k 个子数组。对每个子数组，采用插入排序，所需要的时间为 $O(k^2)$，因此经过快速排序之后，再调用插入排序算法所需要的时间最大不超过 n/k $O(k^2)$，即 $T_1(n) = O(nk)$。

接着，我们讨论最上层的部分快速排序的运行时间 $T_2(n)$。我们知道完整的快速排序的运行时间为 $O(n\lg(n))$， 依据题意，我们可以假设经过一次不完整的快速排序算法后

所得的子数组的长度最大为 k，共 n/k 个子问题，因此可得

$$T_2(n) = O(n\lg n) - n/kO(k\lg k)$$
$$= O(n\lg\frac{n}{k})$$

综上所述，这个排序算法的期望运行时间为 $O(nk + n\lg(n/k))$。

$T_2(n)$ 还可以如下计算：

假定最好情形划分，我们需要 $\lg n$ 次划分，可将原问题划分为 n 个大小为 1 的子问题，类似的，我们需要 $\lg k$ 次划分，可将大小为 k 的子问题划分为每个大小均为 1 的子问题，因此，我们需要 $\lg n - \lg k$ 次划分，可将原问题划分为每个大小为 k 的子问题，每次划分需要 $O(n)$，因此共需要时间 $T_2(n) = O(n)(\lg n - \lg k) = O(n\lg\frac{n}{k})$。

要获得最优的运行时间，也就是要使 $O(nk + n\lg(n/k))$ 最小，所以理论上 k 的值应该取小于等于 $\lg n$ 的整数，否则，运行时间将超过 $n\lg n$。实际上，k 一般取为 10，16，25，应该视具体情况来定。

2. 另外一种分析随机快速排序算法运行时间的方法，着重于每次调用 QuickSort 的期望运行时间，而不是进行比较的次数。
   a) 讨论对于一个给定的大小为 n 的数组，其中任何一个元素被选择为支点的概率为 $1/n$。定义随机变量 X_i =I{第 i 个最小元素被选择为支点}。请问 $E[X_i] = ?$
   b) 设随机变量 $T(n)$表示对一个大小为 n 的数组调用快速排序算法所花费的时间。讨论它的期望运行时间 $E[T(n)]$。

a) $E[X_i] = 1/n$

b) 依题意假设第 i 个最小元素被选择为支点，则 $T(n) = T(i - 1) + T(n- i) + O(n)$，因此总的期望时间为

$$
\begin{aligned}
E[T(n)] &\le E[\sum_{i=1}^{n} X_i \cdot (T(i-1)+T(n-i)+O(n)] \\
&= \sum_{i=1}^{n} E[X_i \cdot (T(i-1)+T(n-i))]+O(n) \\
&= \sum_{i=1}^{n} E[X_i] \cdot E(T(i-1)+T(n-i))+O(n) \\
&= \sum_{i=1}^{n} \frac{1}{n} \cdot E(T(i-1)+T(n-i))+O(n) \\
&= \frac{2}{n}\sum_{k=0}^{n-1} E(T(k))+O(n)
\end{aligned}
$$

类似选择问题的求解，可得 $E[T(n)] = O(n\lg n)$

3. 写一个迭代（iterative）版本的随机选择(RandomizedSelect)排序算法。

```
IterativeRandomizedSelect(A, p, r, i)
1   if p = r then return A[p]
2   q ← RandomizedPartition(A, p, r)
3   k ← q - p + 1
4   while (p!=r)||(i!=k) do
5       if i < k then
6           r ← q -1
7       else
8           p ← q + 1
9           i ← i - k
10      q ← RandomizedPartition(A, p, r)
11      k ← q - p +1
12  if p = r then
13      return A[p]
14  else
15      return A[q]
```

4. 假如我们使用随机选择（RandomizedSelect）算法来选择数组 a=（3，2，9，0，7，5，4，8，6，1）的最小元素。请你给出一个划分序列，它导致了随机选择算法的最坏情形的发生。

每次恰好选到最大元素

第一次运行：randomi(1,10) = 3 ,划分结果为：(3,2,1,0,7,5,4,8,6)(9)

第二次运行：randomi(1, 9) = 8, 划分结果为：(3,2,1,0,7,5,4,6)(8)

第三次运行: randomi(1, 8) = 5, 划分结果为：(3,2,1,0,6,5,4)(7)

第四次运行：randomi(1, 7) = 5, 划分结果为：(3,2,1,0,4,5)(6)

第五次运行: randomi(1, 6) = 6, 划分结果为：(3,2,1,0,4)(5)

第六次运行：randomi(1, 5) = 5, 划分结果为：(3,2,1,0)(4)

第七次运行：randomi(1, 4) = 1, 划分结果为：(0,2,1)(3)

第八次运行：randomi(1, 3) = 2，划分结果为：(0,1)(2)
第九次运行: randomi(1, 2) = 2，划分结果为：(0)(1)
第十次运行：仅剩下最后一个元素 0，此元素为 A[]中的最小元素。

5. 除了随机快速排序以及随机选择算法外，你能想出一个将确定性算法改造为 Shewood 算法的例子吗？
随机合并排序算法，可以类似对输入序列随机化，然后调用合并排序算法。
局部搜索算法，随机初始化解。

习题 2.4

1. 给定一个数组 $a[0..n\text{-}1]$, 请问如何查找数组中的元素 X，它至少出现 k 次。请给出这个问题的一个 Las Vegas 算法。
这个题目指的是数组中有元素 X，它至少出现 k 次，因此，可以设计出如下算法：

```
RandomSearch(L[0:n – 1],X)
1   Found ← false
2   while Found= false do
3       Choice ← Randomi(1, n)
4       if X = L[Choice] then
5           Found ←true
6   return (Choice)
```

上述算法期望最好情形是，数组中每个元素都等于 X。当 X 精确出现 k 次时，期望最坏情形出现。对于这样的输入，找到 X 的概率为 k/n，执行第 i 次 while 循环的概率为 $p_i = (1-p)^{i-1}p, i = 1,2,\cdots$，因此期望最坏情形的运行时间为

$$W\exp(n) = \tau\exp(I\text{w}) = p1 + 2p2 + \ldots + ip\text{i} + \ldots$$
$$= p[(1-p) + 2(1-p) + \ldots + i(1-p)i\text{-}1) + \ldots \quad]$$

$$W_{\exp}(n) = \tau_{\exp}(I) = p_1 + 2p_2 + \cdots + ip_i + \cdots$$

利用几何级数公式可得 $W_{\exp}(n) = \tau_{\exp}(I) = 1/p = n/k$。

如果不清楚题目意思，则可能给出如下算法。
S[i]：表示列表 L[0:n-1]中的第 i 个元素是否已经查找过了，当 S[i]=ture，第 i 个元素已经统计过了，我们应该忽略，避免重复统计元素出现的次数；否则，第 i 个元素还没有被统计过。
p: 表示列表 L[0:n-1]中已经被统计过的元素的个数，当 p=n 时，表示所有元素都已经被统计过，程序结束运行并退出。
q: 表示已经统计过的元素中不相同的元素的个数。
A[i]:存放列表 L[0:n-1]的元素互不相同的元素。
B[i]:表示 A[i]元素出现的个数
Flag：标示元素是否在 A[i]中出现，如果没有，则把值存入 A[i]中，并使 q 增加 1.详细算法如下

```
RandomSearch()
1   for i ← 0 to n do
```

```
2          S[i] ← false
3   p ← 0
4   q ← 0
5   while p<n do
6          r ← randomi(0, n-1)
7          if not S[r] then
8               S[r] ← true
9                p ← p +1
10               flag ← false
11               for j ← 0 to q do
12                    if A[j]=L[r] then
13                         flag ← true
14                         B[j] ← B[j] + 1
15                         if B[j]>=k then
16                              return A[j]    //求得问题的解
17               if flag = false then
18                    A[q] ← L[r]
19                    B[q] ← 1
20                    q ← q +1
21     return "列表中不存在 X, 它出现的次数至少为 K".
```

2. 有 *N* 位哲学家围坐在一张桌子前面。在桌子的中间放着一盆意大利面条（很多，足够大家吃饱）。每位哲学家的面前放着一个盘子，在每两个盘子中间放着一把餐叉，也就是桌上放着 *n* 个盘子 *n* 把餐叉。一位哲学家要么吃要么思考。如果哲学家旁边的两把餐叉都空闲，他可以拿起他们开始吃直到吃饱为止。但是如果只有一把餐叉空闲呢？每位哲学家是否应该保留手上的餐叉并等待另一把餐叉也空闲呢？如果每位哲学家都这样做，那么可能会发生一种情况，那就是每个人右手上都有一把餐叉并且他们之间相互等待另一把餐叉空闲，这样谁也别想吃，而永远的等待下去，如何解决？

每个哲学家抛一个硬币，如果头朝上，则等待，否则拿起餐叉吃。

如果有两个哲学家试着拿同样的餐叉，则看其旁边是否有可用的，或者再抛硬币。

重复上述过程，则哲学家们有极高的概率吃到饭，而不是永远等待。

```
Philosophers()
1 repeat
2     repeat think until hungry
3     repeat
4         x←randomi(0,1)//0 denotes right, 1 denotes left
5         repeat wait until Fork on x hand side is free
6         Take fork from the x hand side
7          if the other fork y is free then take the fork on y
8          else free the fork on x
9      until fork in both hands
10    repeat eat until satisfien
11    free the forks
12 until false
```

另一种思路：初始时，哲学家的都处于饥饿状态，但是，我们通过随机变量使他们在不同的时间做出判断，是否有两把餐叉可用，如果可用，则他们就吃饱，然后一直处于思考状态，把餐叉让出来，具体的实现代码如下：

```
LEFT ← (i-1+n)%n
RIGHT ← (i+1)%n
THINKING ← 0
HUNGRY ← 1
EATING ← 2
```

State[i]:记录第 i 个哲学家当时的状态：思考，饥饿，喝汤, 初始时 State[i]=HUNGRY

```
Philosopher(i) //i：哲学家的号码，从 0 到 n-1
1   while true do
2         think()   //哲学家正在思考
3         takeforks(i) //需要同时拥有两只餐叉，否则等待
4         eat() //进餐
5         putforks(i) //把两把餐叉同时放回桌子

takeforks(i)
1   if state[i]=HUNGRY then
2         while test()=false do //测试两把餐叉是否同时可用
3               wait(randomi(1, 3600)) //等待随机时间后，再做判断。

putforks(i)
1   state[i]←THINKING

test(i) // i：哲学家号码从 0 到 n-1
1    if state[LEFT]!=EATING && state[RIGHT]!=EATING then
2        state[i]←EATING
3        return true
4    return false
```

3. 类似 8 皇后问题，请为 SAT 问题设计一个 Las Vegas 算法。

SAT 问题：对于有 n 个变元，m 个子句的合取范式或析取范式（一般考虑 CNF）的布尔函数，问是否存在一组赋值能使得该布尔函数可满足。

$$\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$$

$$c_i = x_{i1} \vee x_{i2} \vee \cdots \vee x_{ik_i}, \text{其中} x_{ij} \in \left\{ x_1, \overline{x_1}, x_2, \overline{x_2}, \cdots, x_n, \overline{x_n} \right\}$$

算法思想：

1、随机选取若干个变元，并对他们进行随机赋值为 0 或 1；

2、回剩下的变元采用回溯法进行赋值；

3、若所有赋值都不是可满足的，那么返回 1。

进一步，如果是 k-SAT(即每个子句精确的有 k 个文字)，请思考，如何分析。

4. 编程序实现 QueensLVB(*x*[], success)，并对结果进行分析。
略

5. 根据图 2.4.2 的执行结果显示，PollardRho(*n*)什么时候打印 1387 的因子 73？
当 y=x8=814, k=16, 且程序运行到 x16=595 时，此时，gcd(y-x16,1387)=gcd(219,1387)=73, 因为，d=73 满足 d<>1 和 d<>n,所以此时，Pollard-rho 算法打印输出 73.

6. 编程实现 PollardRho 启发式算法。
略

习题 2.5

1．给定一个符号多项式(就是说，通过它们的系数数组来表示) $f(x)$，$g(x)$ 和 $h(x)$，它们的次数分别为 $2n$，n，n，考虑如下问题：测试 $f(x)$ 是否等于 $g(x)h(x)$，即 $f(x) \equiv g(x)h(x)$，请给出一个 Monte Carlo 算法来测试多项式的恒等性。

我们考虑$f(x)=g(x)h(x)$是否成立。显然，如果有一个x，它使得等式两边的值不等，那么这个等式肯定是不成立的。反之，如果x使得等式成立了，我们可以做出断言说这个等式是成立的，这个时候我们就有可能做出了错误的判断。因此可以按上述方法设计一个偏false的Monte Carlo算法。

```
TestPolyEqual(f(x),g(x),h(x))
1    j=Randomi(1,3n)
2    if f(j) = g(j)h(j) then
3         return true
4    else
5         return false
```

现在分析上述算法犯错误的概率，也就是$f(j)\neq g(j)h(j)$，但是我们运气不好找到了这样的一个j，它满足$f(j) = g(j)h(j)$。容易看出，这样的j是方程$f(j) - g(j)h(j)=0$的一个根。对于这个方程，它的最高次数是2n，所以这个方程最多有2n个根。一般地，我们只需要从[0，m]（m>2n）随机选取一个整数，代入方程验证一下，如果不等，返回false，我们确保这个答案一定是正确的；如果相等，返回true，则我们有可能犯错误了。这个算法犯错的概率最大是2n/m。这里m=3n,所以犯错的概率为2/3，因此是1/3-正确的Monte Carlo算法

可以重复s次，减少犯错误的概率。

```
Boolean IsItEquality(f, g, h, Length)
1   for i ← 1 to s do
2        j← Randomi(1, 3n)
3        if f(j)≠g(j)*h(j) then
4             return false
```

5 **return** true

2．给定三个$n \times n$的矩阵A, B和C，验证AB是否等于C。考虑A和B相乘得到结果C的标准计算方式为：

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

这个算法的时间复杂度为$O(n^3)$，Strassen的算法是这个算法的时间复杂度为$O(n^{2.81})$的一个变种。 更进一步的算法通过减少乘法和计算开销，它的时间复杂度下降到$O(n^{2.376})$。请给出它的一个Monte Carlo算法。

由于*AB*本身的乘积已经需要$O(n^3)$，所以我们要避免直接计算。 另一方面，对n*n矩阵运算，至少需要O(n)，因此，算法目的就是设计一个复杂度为$O(n^2)$ 的Monte Carlo算法。

设X是一个n维行向量，则*XAB*=(*XA*)*B*，

若直接计算*XAB*，那么时间复杂度将会是$O(n^3)$，若转化为(*XA*)*B*，时间复杂度将降为$O(n^2)$，问题就是将*AB*=*C*？转化为：*XAB*=(*XA*)*B*？很明显后者是前者的必要条件。

注：用必要条件来判定问题可设计出MC算法，用充分条件来判定问题可设计出LV算法，用充要条件来判定问题可设计出SHERWOOD算法。

现在用*XAB*=(*XA*)*B*？来作为判断*AB*=*C*？的方法，因此可设计出一个相应的MC算法。

MultiAB()

1 **for** *j*=1 **to** *n* **do**

2 x_j=randomi(0,1)

3 **if** (*XA*)*B*=*XC* **then return** true

4 **else return** false

该算法是一个偏false的MC算法。

如果$AB \neq C$，则犯错误的概率 **Pr**[*XAB*=*XC*]≤1/2。

令$D=AB-C \neq 0$，*XAB*=*XC*，即*XD*=*0*，由于$D \neq 0$，不失一般性，令$d_{11} \neq 0$

$$\sum_{k=1}^{n} x_k d_{k1} = 0 \quad \Rightarrow \quad x_1 = \frac{\sum_{k=2}^{n} x_k d_{k1}}{d_{11}}$$

除了x_1，其他x_i固定，x_1的值要么为0要么为1，只有两种选择，因此所证成立。

3．给定一个图$G = (V, E)$，问题是这个图形是否包含一个三角形，即三个顶点通过边两两

相连。请设计出该问题的 Monte Carlo 求解算法。

随机选择一条边 (x,y) 以及除了 x,y 外的另一个顶点 z。如果图至少有一个三角形，则上述选择命中的概率为 $(3/|E|)(1/(|V|-2))$，也就是说，$|E|$ 条边里有 3 条属于三角形的边，3 条边里任选一条，选择三角形另一个顶点的概率为 $1/(|V|-2)$。虽然上述概率比较小，但是如果重复 k 次，k 次重复里，没有一次返回三角形的概率为 $1-(3/|E|)(1/(|V|-2))^k$。

我们可以用下列结果。对于小的 x，$(1-x)^k$ 近似 e^{-kx}，其中 e=2.718 为自然对数的底。因此，如果我们选择 k，使得 $kx=1$，则 e^{-kx} 将会小于 1/2，$1-e^{-kx}$ 将会比 1/2 大。这样我们选择 $k=e(|V|-2)/3$，我们将有足够的理由相信，图中有一个三角形。

4．假设在最小切割算法的每一步，不是随机选择一条边压缩，而是随机选择两个顶点而后把它们压缩为一个顶点。说明修改后的算法发现一个最小切割的概率是指数阶小的。

当包含 n 个结点时，假设最小分割的结点集合 S1、S2 所含结点数目分别是 m，$n-m$，则发生错误结点结合的概率是 $\frac{C_m^1 * C_{n-m}^1}{C_n^2}=\frac{2*m*(n-m)}{n*(n-1)}\le\frac{2*(\frac{n}{2})^2}{n*(n-1)}=\frac{n}{2*(n-1)}$。

因此，随机运行算法成功获得最小分割的概率是：$\prod_{i=3}^{n}(1-\frac{i}{2*(i-1)})=\frac{1}{2^{n-2}*(n-1)}$。

因此，修改后的算法发现一个最小分割的概率是指数阶小的。

5．请分析直接通过计算 a^b 模 n 和通过二进制表示 b 计算 a^b 的时间复杂性。

直接计算 $a^b \bmod n$ 需 $O(b)$，如果利用二进制表示，则 b 需要 $\lg b$ 位，其时间复杂度为 $O(\lg b)$

6．用 C++实现 MillerRabin(*n*, *s*)算法。

略

7．证明定理 2.5.4。

如果 x 满足 $x^2\equiv 1 \pmod n$，即存在一个 m，使得 $(x-1)(x+1)=nm$，由于 n 是一个素数，所以它必能被 $(x-1)$ 或者 $(x+1)$ 整除。又因为 $0<x<n$，则 $0<x-1<n-2, 2<x+1<n$ 都不能整除 n，所以 x 只能取 1 或者 $n-1$。

8．请为 2-SAT 问题设计一个 Monte Carlo 算法

2-SAT问题即子句的长度为2，设命题公式为2SAT(X)。我们可以采取如下算法：

```
MC2SAT()
1   for j=1 to n do
2           x_j=randomi(0,1)
3   if 2SAT(X)=1 then return true
4   else return false
```

上述算法只有在返回 false 时，有可能做出错误的判断。最坏情形出现在只有一种指派 X 满足 2SAT(X)，此时，这是一个 $\frac{1}{2^n}$ 正确的 Monte Carlo 算法。

习题 2.6

1. 归纳总结前面介绍的问题与随机复杂性类之间的关系。
2. 证明定理 2.6.1。
3. 证明定理 2.6.2。

略