5.2-5.

设$E[X_i]$表示1~i的随机排列，产生的逆序对的期望.

$$E[X_k] = E[X_{k-1}] + \sum_{i=1}^{k-1} (k-i)\cdot\frac{1}{k}$$

$$= E[X_{k-1}] + \frac{1}{k}\sum_{i=1}^{k-1} i = E[X_{k-1}] + \frac{k-1}{2}$$

$$\therefore E[X_k] = \sum_{i=1}^{k-1} \frac{i}{2} = \frac{1}{4}k(k-1).$$

5.3-3

该算法并不能生成均匀随机排列.

假设当前处理序列长度为n.

随机数生成器将返回n^n种随机结果

而序列共$n!$种排列

任取质数p，使$n=p$.

则 $\frac{n^n}{n!} = \frac{n^{n-1}}{(n-1)!}$，分母中不存在$p$的因子

故必定不能整除.

即不可能均匀分配随机结果，

不能使生成每个排列的概率一致.

5-2

a)

```
int Random_Search(int A[], int n, int x) {
    Initialize array b[n] with all False;
    int visited = 0;
    while (visited < n) {
        int pos = Rand(1, n);
        if(A[pos] == x)
            return pos;
        else if (b[pos] == False) {
            b[pos] = True;
            visited = visited + 1;
        }
    }
    return Pos_Not_Found;
}
```

b)

由于目标元素有且仅有1个，且目标元素在序列中的位置是等可能的，因此1次随机查找可以找到该元素的概率为$\frac{1}{n}$。

即单次查找找到目标元素的期望个数为$\frac{1}{n}$，由期望的线性性可知，k次查找找到该元素的期望个数为$\frac{k}{n}$。

因此，找到该元素的期望查找次数为n。

c)

单次找到一个目标元素的概率为$\frac{k}{n}$，即单次找到目标元素的期望个数为$\frac{k}{n}$。

若要找到一个目标元素，期望查找次数为$\frac{1}{\frac{k}{n}}=\frac{n}{k}$。

d)

问题转化为，长度为n的序列，每次随机访问一个元素，求每个元素都被访问至少1次的期望次数。

考虑递推解决这个问题。经尝试，正推会遇到重复统计的问题，倒推法更为简单。

设f[i]表示当前访问过i种不同的元素，到达访问过所有元素这一状态所需的期望步数。

$f[i]=(f[i+1]+1)\frac{n-i}{n}+(f[i]+1)\frac{i}{n},f[n]=0$

解方程得$ans=f[0]=\sum_{i=0}^{n-1}\frac{n}{n-i}=\Theta(nlnn)$

e)

设X_i表示该元素出现在位置i的概率，显然$P[x_i]=\frac{1}{n}$。

设X表示出现的位置，则$E[x]=\sum_{i=1}^{n}iP[x_i]=\frac{n+1}{2}$

最坏情况，该元素出现在最后一个，需要n次计算才能找到。

f)

最坏情况下，所有k个目标元素在序列末尾，运行时间$f_w(n,k)=n-k+1$。

平均情况下，问题转化为，在一个长度为n的序列中随机选择k个点，选中的点的最小下标的期望值，答案可以通过枚举最小下标的位置算出。

$ans=\sum_{i=1}^{n-k+1}i*\frac{\binom{n-i}{k-1}}{\binom{n}{k}}$

$=\frac{k(n-k)!}{n!}\sum_{i=1}^{n-k+1}\frac{i(n-i)!}{(n-i-(k-1))!}$

$=\frac{n+1}{k+1}$

g)

此时平均情况和最坏情况一样，都需要完整遍历一次整个数组，需要n次计算。

h)

k=0时，最坏情况和平均情况一致，需要2n的计算量进行序列重排和遍历。

k=1时，最坏情况是所需元素在最后一位，需要2n-1次计算进行重排和遍历。平均情况下，目标元素的期望位置为$\frac{n}{2}$，因此期望进行$\frac{3}{2}n$次计算得到结果。

i)

我会选择第二种（Deterministic Search）。不论是平均还是最坏情况，这种算法的实际计算量均最小。第一种算法过于随机，第三种算法不论如何都需要n的计算量进行序列重排列，而且也有随机因素。只有第二种算法是确定的，而且不会浪费计算量进行序列随机。