Q1. 7.2-6 习题

不妨假设这一随机输入数组中的数字为从1-n的正整数（由于我们只需要考虑数字间的相对大小关系，实际输入的数据大小不影响~~结果~~对快速排序的分析）

对 $0<\alpha\le\frac{1}{2}$ 假设数组比 A[n] 小的数有 k 个.

那么要令 PARTITION 产生比 $1-\alpha:\alpha$ 更平衡的划分即要求 $k>\alpha n$（有多于 αn 个比 A[n] 小的数）

$k<(1-\alpha)n$

即比 A[n] 大或小的数均在 $\alpha n \sim (1-\alpha)n$ 个范围内

这一事件发生的概率为：$\frac{[\lfloor(1-\alpha)n\rfloor-\alpha n]}{n}\approx(1-2\alpha)$

Q2. 首先先求有限数列 $a_1, a_2, \cdots, a_n$ 的加权中位数（加权中位数定义为满足 $\sum_{x_i<x_k} w_i<\frac{1}{2}$ 及 $\sum_{x_i>x_k} w_i\le\frac{1}{2}$ 的在数列 $a_1, a_2 \cdots, a_n$ 中的元素），先定义SELECT函数：

1. 将输入数组的 n 个元素划分为 $[\frac{n}{5}]$ 个组，每组5个元素，且至多只有一组由剩下的 n mod 5 个元素组成。

2. 寻找这 $[\frac{n}{5}]$ 组中每一组的中位数：首先对每组元素作插入排序，然后确定每组有序元素的中位数

3. 对2中找出的 $[\frac{n}{5}]$ 个中位数，递归调用SELECT以找出其中位数（若有偶数个中位数，为了方便，定义 x 是较小的中位数）

至此 SELECT 函数结束，求出中位数的中位数 mid 下面继续：

4. 以 x 为主元对数组划分作 PARTITION 操作，得到新数列 $x_1, \cdots x_n$，其中 $x_j=$mid

5. 对 x_j 左右两半部分分别求其对应权重和，即分别对 $x_1 \sim x_{j-1}$、$x_{j+1} \sim x_n$ 求对应权重和，若满足加权中位数的定义，则 x 为所求，返回这个主元。若不满足，则：①若左半部分大于 $\frac{1}{2}$，则对左半部分 $x_1 \sim x_{j-1}$ 调用步骤 1.2.3.4 直到新的左半部分刚好小于 $\frac{1}{2}$ ②若右半部分大于 $\frac{1}{2}$，则对右半部分 $x_{j+1} \sim x_n$ 调用步骤 1.2.3.4，直到新的右半部分不大于 $\frac{1}{2}$.

5个步骤结束后，求得加权中位数。此处求出加权中位数的算法与求中位数的SELECT算法时间复杂度相同，为 $O(n)$。

下面简要证明加权中位数为满足 $\min\limits_{x} D(x)$ 对应的：

要求 $\min\limits_{x} D(x)$，即求 $\min \sum\limits_{i=1}^{n} w_i |a_i - x|$.

令 $f(x) = \sum\limits_{i=1}^{n} w_i |a_i - x|$

$S = S_1, S_2, \cdots, S_n$ 为数列 $a_1, \cdots a_n$ 按从小到大顺序重排后的数列，$k_1, k_2 \cdots, k_n$ 为权重 $w_i (i=1,2,\cdots n)$ 按重排后 a_i 在 S 中的位置 S_j 赋值给 k_j 后得到的 S 对应的权重

则易知 $g(x) = \sum\limits_{i=1}^{n} k_i |S_i - x| = f(x)$

∴ 下面证明 $\forall y \in R$ 有 $g(y) > g(mid)$

一共有4种情况需要考虑：

① $mid \le S_{p-1} < y < S_p$ $p \le n$ ② $S_{p-1} < y < S_p \le mid$ $p \ge 2$

③ $y < S_1$ ④ $y > S_n$

① $mid \le S_{p-1} < y < S_p$ $p \le n$

则对于 $i \in [1, p-1]$，$i \in N$ 来说有

$|S_i - y| - |S_i - mid| = y - mid$

对于 $i \in [p, n]$ $i \in N$ 来说有

$|S_i - y| - |S_i - mid| = mid - y$

$$g(y)-g(mid)=\left(\sum_{i=1}^{p-1}k_i-\sum_{i=p}^{n}k_i\right)(y-mid)$$

由 mid 的定义易知 $\sum_{i=1}^{p-1}k_i>\frac{1}{2}$, $\sum_{i=p}^{n}k_i<\frac{1}{2}$

又 $\because$ $y>mid$

$\therefore$ $g(y)>g(mid)$

② $S_{p-1}<y<S_p\leq mid$, $p-1\geq 1$

则对于 $i\in[1,p-1]$, $i\in N$ 来说有:

$|S_i-y|-|S_i-mid|=y-mid$

对于 $i\in[p,n]$, $i\in N$ 来说有.

$|S_i-y|-|S_i-mid|=mid-y$.

$$g(y)-g(mid)=\left(\sum_{i=1}^{p-1}k_i-\sum_{i=p}^{n}k_i\right)(y-mid)$$

由 mid 的定义知 $\sum_{i=1}^{p-1}k_i>\frac{1}{2}$, $\sum_{i=p}^{n}k_i<\frac{1}{2}$

又 $\because$ $y<mid$

$\therefore$ $g(y)>g(mid)$

③ $y<S_1$,

$$g(y)=\sum_{i=1}^{n}k_i(S_i-y)$$

不妨设 $mid=S_p$, 则有

$$g(y)-g(mid)=-\sum_{i=1}^{p-1}k_i(S_i-y-mid+S_i)+\sum_{i=p}^{n}k_i(mid-y)$$

由 mid 定义可知. $\sum_{i=p}^{n}k_i\geq\frac{1}{2}\geq\sum_{i=1}^{p-1}k_i$

即有 $g(y)-g(mid)\geq\sum_{i=1}^{p-1}k_i\left[(mid-y)-(2S_i-mid+y)\right]$

$$=\sum_{i=1}^{p-1}k_i(2mid-2S_i)$$

$\because$ $k_i>0$ $i\in[1,p-1]$, $i\in N$

$2mid-2S_i>0$ $i\in[1,p-1]$

$\therefore$ $g(y)>g(mid)$

④ $y > S_n$,

$$g(y) = \sum_{i=1}^{n} k_i (y - S_i)$$

不妨设 $mid = S_p$，则有

$$g(mid) = \sum_{i=1}^{p} k_i (mid - S_i) + \sum_{i=p+1}^{n} k_i (S_i - mid)$$

$$g(y) - g(mid) = \sum_{i=1}^{p} k_i (y - mid) + \sum_{i=p+1}^{n} k_i (y + mid - 2S_i)$$

由 mid 的定义有 $\sum_{i=1}^{p} k_i \geq \frac{1}{2} \geq \sum_{i=p+1}^{n} k_i$

则 $g(y) - g(mid) \geq \sum_{i=p+1}^{n} k_i (2y - 2S_i)$

又∵ $k_i > 0$，$i \in [p+1, n]$，$i \in N$

$2y - 2S_i > 0$，$i \in [p+1, n]$，$i \in N$

∴ $g(y) - g(mid) > 0$

$g(y) - g(mid)$

综上，对所有 $y \neq x$ 的情况均有 $g(y) > g(mid)$

所以 $x = mid$ 时，

$D(x)$ 有最小值.

结合前面所述的求 mid 的算法，我们有求 $\min_x D(x)$ 的算法为：

① 用前述算法求出 mid，复杂度为 $O(n)$

② 求出 $\sum_{i=1}^{n} w_i$，复杂度为 $O(n)$

③ 求出 $\sum_{i=1}^{n} w_i |a_i - mid|$，复杂度为 $O(n)$

④ $$\min_x D(x) = \frac{\sum_{i=1}^{n} w_i |a_i - mid|}{\sum_{i=1}^{n} w_i}$$

其中②、③步均为单个for循环中累加，因此复杂度为 $O(n)$

(2) 对数列 $\{a_n\}$、$\{b_n\}$

$\Delta\{a_n\} = \{a_{n+1} - a_n\}$

$\Delta^2\{b_n\} = \{b_{n+2} - 2b_{n+1} + b_n\}$.

由 $\Delta\{a_n\} = \Delta^2\{b_n\}$ 可知.

$a_{n+1} - a_n = b_{n+2} - 2b_{n+1} + b_n$.

令 $b_{n+2} - b_{n+1} = a_{n+1} + C$.

则 $b_{n+1} - b_n = a_n + C$

$\vdots$

即有 $\begin{cases} b_T - b_{T-1} = a_{T-1} + C \\ \vdots \\ b_3 - b_2 = a_2 + C \\ b_2 - b_1 = a_1 + C \\ b_1 - b_T = a_T + C \end{cases}$ C为一个常数

$b_i - b_1 = (a_1 + \cdots + a_{i-1}) + C(i-1)$,

$\begin{cases} b_i = (a_1 + \cdots + a_{i-1}) + C(i-1) + b_1 \quad i \in [2, T] \\ b_1 = b_1 \end{cases}$

由于 $b_T - b_1 = (a_1 + \cdots + a_{T-1}) + C(T-1)$

且 $b_1 - b_T = a_T + c$

所以有：$c = \dfrac{-(a_1+a_2+\cdots+a_T)}{T}$

即有

$$\begin{cases} b_i = \dfrac{\sum_{j=0}^{i-1}(T-i+1)a_j - \sum_{j=i}^{T}(i-1)a_j}{T} + b_1 \\ b_1 = b_1 + 0 \end{cases}$$

b_1 为其中唯一的变量，影响 $|b_i|$

令 c 为一个长度为T的数组，其中 $c[1]=0$，

$c[i] = -\dfrac{\sum_{j=0}^{i-1}(T-i+1)a_j - \sum_{j=i}^{T}(i-1)a_j}{T}$，$i=2,\cdots n$

易知当 b_1 取 $c[1] \sim c[n]$ n个数的中位数时，$\sum_{1\le i\le T}|b_i|$ 取到最小值.

故设计算法如下：

① 令 c 为一个长度为T的数组，其中 $c[1]=0$，

$c[i] = -\dfrac{\sum_{j=0}^{i-1}(T-i+1)a_j - \sum_{j=i}^{T}(i-1)a_j}{T}$　$i=2,3\cdots n$

这步骤时间复杂度为 $O(n)$

② 用 $O(n)$ 时间内求出中位数的SELECT算法，求出 c 中元素的中位数.

③ $d[i] = -c[i] +$ 中位数.
d 的长度也为n

④ 长度为n的数组e　$e[i] = |d[i]|$

⑤ 对e中 n个元素求和，即得

$\sum_{1\le i\le T} |b_i|$ 的最小可能值.

Q3. (1) 设计算法如下:

```
while (i < A.length && j < B.length)
    if (A[i] > B[j])
      count ← count + (n-i)
      j++;
    else
      i++;        count即为|S|
```

最坏情况即 A、B数组中 A[i]、B[i] 交替更大，如 A 1 3 5 7，B 2 4 6 的情况，

共比较 (m+n-1) 次

故此算法是线性最坏时间复杂度.

(2) 长度为 n的数组L

算法分为两个大步骤:

① 借助栈，求出数组中每个数的后面比它小的第一个数，并保存索引，存入一个新数组

② 根据这一数组对原数组中的每一元素求出在它后面比它小的元素的个数

第一步:

① 初始化栈，里面为第一个元素的索引1

② 遍历到下一个元素 L[i]

1) 若栈不为空且当前遍历的元素值 L[i] 大于栈顶的元素值 A[stack.peek()]，说明当前元素是栈顶元素右边第一个比它大的元素，将栈顶元素弹出，result[stack.pop()] = A[i]

(检查)继续遍历的元素A[i]是否小于新栈顶元素 A[stack.peek()]，如果小于，说明 A[i] 也是 A[stack.peek()] 右边第一个比它小的元素，将栈顶元素弹出，result[stack.pop()] = A[i],

一直… 或是… 2)… 小… 3)… 的… 元… 元… ③ 但… 由… 右…

一直循环，直到不满足条件1)，即栈顶为空或是当前遍历的元素值大于栈顶元素索引处的值

2) 若栈为空，说明前面的元素都找到了比它右边小的元素，则直接将当前元素的索引放入栈中

3) 若当前遍历的元素值 A[i] 大于栈顶元素索引的值 A[stack.peek()]，说明还未找到栈顶元素中右边第一个比它小的元素，直接将当前遍历的元素的索引入栈即可

将 i++，重复步骤 2)

③ 直到遍历完所有元素，若栈不为空，说明栈中保存的全是未找到右边第一个比它小的数组索引，依次将这些栈元素出栈，并赋值 result[stack.pop()] = -1。

输出新的数组

由上述的单调栈操作，实现了求出数组中每个元素右边第一个比它小的元素在数组中的索引。

由于只需要遍历数组一次，所以这个单调栈操作时间复杂度为 o(n)

然后借助这个新数组，不妨设为 b，定义一个函数 ~~for to~~ find，其作用是对 i = 1 -- to n，迭代寻找 b[i] 的右边第一个小于 b[i] 的元素（若 b[i] 非0，把 b[i] 作为新的索引找 b[b[i]] ---），并累加计数，可以得到原数组的每一元素右侧有多少个比它小的元素。全部求和即为 |I|，这一步的时间复杂度为 o(n)

综上即为线性时间复杂度下求 |I| 的算法.

(3) 证明：设 $C=\{<c_{11},c_{12}>,\cdots\ <c_{s1},c_{s2}>\}$

统计在 $c_{11},c_{21},\cdots c_{s1}$ 中出现的 $1\sim n$ 每个数的次数，记为 $d[i]$ $i=1,2,\cdots,n$

统计在 $c_{12},c_{22},\cdots c_{s2}$ 中出现的 $1\sim n$ 每个数的次数，记为 $e[i]$ $i=1,2,\cdots,n$

1° 设 A_1 为一个排列服从于C的数列。若存在 使 $d[i]=0$ 或 $e[i]=0$ 的 i，$i=1,2,\cdots,n$，则可以调换 $A_1[i]$ 和 数列 A_1 中所有比 $A_1[i]$ 大的数中最小的数，这样由于不影响该数在所有参与C中大小比较的所有数的大小顺序，所以新得到的数列 A_2 也服从于C。~~由此，得到满足~~

这里的 A_1、A_2 满足：A_1、A_2 都服从于C，A_1、A_2 只差一次对换。

2° 若 $\forall i\in[1,n],i\in N$

$d[i]\neq 0$ $e[i]\neq 0$.

~~易知对于~~ 设 A_1 为A的一组服从于C的排列

易知对于 $A_1[i]=a_i$，若 a_i 是A中第k大的数 则 $d[i]\leq(n-k)$ $e[i]\geq k-1$

若存在 $i,j\in[1,n]$ $i,j\in N$

使得 ~~$A[i]=$~~ 其中 $A_1[i]$、$A_1[j]$ 分别为A中第 k_1、k_2 大的数，且满足

$$\begin{cases} d[i]\leq n-k_2 \\ e[i]\geq k_2-1 \\ d[j]\leq n-k_1 \\ e[i]\geq k_1-1 \\ k_2=k_1+1 \end{cases}$$

则 ~~对调 $A_1[i]$、$A_1[j]$ 后得到的新排列 A_2 仍服从于C。~~

~~(因为其它数该受影响，对~~

则对调 $A_1[i]$、$A_1[j]$ 后得到新排列 A_2，$A_2[i]$、$A_2[j]$ 相关的大小比较中（由于改变了 $A_1[i]$、$A_1[j]$ 只改变了 $A_2[i]$、$A_2[j]$ 之间的比较，

不影响 $A_2[i]$、$A_2[j]$ 与其它 A_2 中元素的大小关系，所以 A_2 也服从于C

这里的 A_1、A_2 为服从条件的一组排列

下面证明：若存在至少2组不同的排列均服从于C，则C中一定存在满足

$$\begin{cases} d[i] \leq n-k_2 \\ e[i] \geq \cancel{n-}k_2-1 \\ d[i] \leq n-k_1 \\ e[i] \geq k_1-1 \\ k_2=k_1+1 \end{cases}$$

$A_1[i]$ 为A中第 k_1 大数

$A_2[j]$ 为A中第 k_2 大数

的 i,j

由于A有至少2个服从C的排列，设为 A_3、A_4.

由Q4中证明的结论，A_4 可以写成 A_3 中元素若干次两两对换后的结果

~~若不存在满足条件的 A_1、A_2，则对 A_3 中任两元素作对换后得到的排列 A_5 都不可能服从于C.~~

~~设对换的2个元素为 $A_3[i]$ 和 $A_3[j]$，~~

~~则由~~ 所以所求 A_1、A_2 存在

①

Q4. (1) 算法1是完备的，证明如下：

(数学归纳法)

首先，当 $n=2$ 时，要得到所有随机排列可以用算法1得到.

即 i,j 相同一次，相异一次. 实现交换

i,j 两次都相同/相异 实现不变.

下面证明：已知对 $n_0>2$ 时，算法1能得到所有 $(n_0!)$ 种排列时，对 (n_0+1) 也能得到所有随机打乱的结果。

假设对 $|L|=n_0+1$ 使用算法1时，前 n_0 次循环只在 $[1,n_0]$ 范围内取 j (即舍弃其中一些情况)。则对 $|L|=n_0+1$ 相当于在 $|L|=n_0$ 执行后 得到对L的前n项打乱后的L'后 对L'中前 n_0 项中任一项与 (n_0+1) 项交换(或不作任何改变)

交换前，L'有 $n_0!$ 种可能排列，对前 n_0 项的任一种排列，选择 $L[n_0+1]$ 项是否与这 n_0 项交换(共一次)或完全不作交换一共有 (n_0+1) 种排列。那么对于 $n_0!$ 种排列，这一步共产生 $n_0!*(n_0+1)=(n_0+1)!$ 种排列，且没有重复排列(因为第 (n_0+1) 项不与前面的任一项相同)

由此得 $|L|=n_0+1$ 时算法1完备

故证得算法1完备性。

② 算法2是完备的，证明如下：

前面已经证明了算法1的完备性，算法2相当于算法1的拓展情况。例如对L作多次算法1操作，得到所有随机打乱数列的情况。同样对L作同样次算法2操作，第i次中 i,j 交换当且仅当 (randint(0,1)=1) <i,j> 对在第i次算法1操作中交换过。那么就得到与第i次算法1的结果。综上，算法2完备。

③ 算法3完备，证明如下：

（数学归纳法）易知 $n=2$ 时，共有4种情况

$\begin{cases} i=1 \\ j=1 \end{cases}$ $\begin{cases} i=1 \\ j=2 \end{cases}$ $\begin{cases} i=2 \\ j=1 \end{cases}$ $\begin{cases} i=2 \\ j=2 \end{cases}$

易知可得2元素交换、不交换的结果

在 $n=2$ 时算法3完备

假设已知在 $n=n_0$ 时算法3完备，下面证明在 $n=n_0+1$ 时算法3也完备

即经过 $k=1$ 到 $k=n_0$ 的 n_0 次交换后，能得到 $|L|=n_0$ 时元素打乱后的 $(n_0!)$ 种情况

那么考虑 $|L|=n_0+1$ 时，假设前 n_0 次交换不涉及 (n_0+1) 项（即舍弃部分情况），结束后能得到 L 的前 n_0 项随机打乱后的所有 $(n_0!)$ 种情况。

当 $\begin{cases} i=n_0+1 \\ j=m, m\in[1,n_0]\ m\in N \end{cases}$ 时

交换第 (n_0+1) 项和前 n_0 项中任一项

与算法1情况相同。由算法1中得到新 $[(n_0+1)!]$ 种排列知，$n=n_0+1$ 时算法3完备

即有算法3完备

④ 算法4完备，证明如下：

```
for i ← 1 to n do
    j ← randint(i,n)
    swap (i,j);
end
```

可以理解为对第1项，它可以取所有n个数中任一数（与之交换即可）；对第2项，可以取剩下 $(n-1)$ 个数中任一数（与之交换后得）……

对第n项，只能选择前一步操作后的自身。

故共有 $(n!)$ 种结果，且互相不重复。

又∵ n个数随机打乱排列，也只有 n!
种不同排列
∴ 算法4完备.

~~②算法2~~

(2) ① 算法1不均匀，因为：

i ~~一共有n种情况~~，j一共有 n种情况

~~共有 n×n种~~

~~共有~~ 对每个 i，i=1,2,⋯n,

有n种j（每个j概率相等）

完成n次交换共有 n^n种 交换情况，

每个情况概率均为 $\frac{1}{n^n}$

~~③算法5~~

由于共有$(n!)$种排列，$(n!)$不总能整除n^n（例如，n为质数），故$(n!)$种排列出现情况不可能概率相等，故算法1不均匀。

② 算法2不均匀，因为：

一共有 $n\times n=n^2$种$\langle i,j\rangle$对，

对每对$\langle i,j\rangle$，交换或不交换各有$\frac{1}{2}$的概率。

一共有 2^{n^2} 种输出（考虑重复），但只有$(n!)$种不重复排列，$(n!)$不总能整除2^{n^2}，故$(n!)$种排列出现情况不可能概率相等，故算法2不均匀。

③ 算法3不均匀。

有n^2种$\langle i,j\rangle$对，从中可重复地选n次，一共有$(C_{n^2}^{1})^n=(n^2)^n=n^{2n}$种输出，但有$(n!)$种不重复排列，因为$(n!)$不总能整除$n^{2n}$，故$(n!)$种排列出现情况不可能概率相算，故算法2不均匀。

④ 算法4均匀，证明：

算法4等价于

```
for i ← n downto 1 do
    j ← radiant (1, i)
    swap(i, j);
end.
```

此时$|L|=n_0+1$相比$|L|=n_0$相当于多加了一步$i=n_0+1$与$j=radiant(1,n_0+1)$之间的交换，每种情况出现的概率均为$\frac{1}{n_0+1}$

(数学归纳法) 假设已知 $|L|=n_0$ 时，每个排列等概率，则再经过 $i=n_0+1$ 与 $j=\text{randiant}(1, n_0+1)$ 之间的交换后，得到 $(n_0+1)!$ 种排列，且每种排列出现的概率为 $\frac{1}{n_0!}\times\frac{1}{n_0+1}$

$$=\frac{1}{(n_0+1)!}$$

又∵当 $n_0=3$ 时，~~有~~ 用算法4有6种 ~~…~~ 输出

$$\left.\begin{matrix} L_1 & L_2 & L_3 \\ L_1 & L_3 & L_2 \\ L_2 & L_1 & L_3 \\ L_2 & L_3 & L_1 \\ L_3 & L_1 & L_2 \\ L_3 & L_2 & L_1 \end{matrix}\right\}$$

正好与6种不同排列对应

综上有算法4均匀.

(3)

(3.1) 正确，因为对于所有 $(n!)$ 种排列中的任意一种，它为最终结果的概率为：

~~它要求的元素1在位置1的不~~

$$\prod_{i=1}^{n} P_{\text{它要求的元素}i\text{在位置}i\text{的概率}}$$

所以，每种排列出现的概率相等，算法是均匀的

(3.2) 错误，重复执行多次仍不改变每种排列出现的概率。

(4) 算法1：$i=1$ $j=1\ 2\ 3$
$i=2$ $j=1\ 2\ 3$
$i=3$ $j=1\ 2\ 3$

3次交换后 共27种

KJQ 4　JKQ 5
KQJ 4　QKJ 5
JQK 4　QJK 5

策略：先分别猜 JKQ、QKJ、QJK（顺序任意）

再分别猜 KJQ、KQJ、JQK

故有 $E_1 = \frac{5}{27}\times(1+2+3) + \frac{4}{27}\times(4+5+6)$

$= \frac{90}{27} = \frac{10}{3}$

算法2：$3\times3=9$ 次可能交换。

共 (1,2)、(1,3)、(2,1)、(2,3)、(3,1)、(3,2) 有效

(1,2) } 共换1次 $\frac{1}{2}$
(2,1) } 共换2次或0次 $\frac{1}{2}$

(2,3)、(3,2) 同理。
(1,3)、(3,2)

JQK、KJQ 均为 $\frac{1}{4}$

JKQ、QKJ、QJK、KQJ 均为 $\frac{1}{8}$

策略：先猜 JQK、KJQ（顺序无影响）

再猜 JKQ、QKJ、QJK、KQJ

$E_2 = \frac{1}{4} \times (1+2) + \frac{1}{8} \times (3+4+5+6)$（顺序无影响）

$= 3$

③ 算法3. 共3次交换.

$<i,j>$ 共 9种可能，其中

$<1,2>$、$<1,3>$ $<2,1>$ $<2,3>$ $<3,1>$ $<3,2>$

有效.

JQK: $\frac{5}{27}$ JKQ: $\frac{14}{81}$ QJK: $\frac{14}{81}$

KQJ: $\frac{14}{81}$ QKJ: $\frac{4}{27}$ KJQ: $\frac{4}{27}$

策略：先 JQK，再 JKQ/QJK/KQJ，

最后 QKJ/KJQ $E_3 = \frac{55}{27}$

其中 $<1,1>$ $<2,2>$ $<3,3>$ 可合并，$[<1,2><2,1>]$ $[<1,3><3,1>]$ $[<2,3><3,2>]$ 可合并.

④ 算法4 6种排列均匀分布.

故策略就是随意猜即可

$E_4 = \frac{1}{6} \times (1+2+3+4+5+6)$

$= \frac{7}{2}$.

(5) 不会做.

Q5. 全序：集合内的任何一对元素在这个关系下都是可比较的

预序：对集合　及其上的二元关系，具有自反性和传递性.　易知$<$和$\sim$具有自反性和传递性.

(1) 证明$\lesssim_m$是X^m上的全预序.

① 首先证明是全序，由 $\lesssim_m$ 的定义可知，任意 $\langle x_1, x_2, \cdots, x_m \rangle, \langle y_1, y_2, \cdots, y_m \rangle \in X^m$，

都可以通过比较 x_i、y_i　$i \in [1, m]$　$i \in N$

来判断是否满足

$\langle x_1, x_2, \cdots, x_m \rangle \lesssim_m \langle y_1, y_2, \cdots, y_m \rangle$

即$\lesssim_m$是X^m上的全序.

② 接下来证明是预序

若 $\langle x_1, x_2, \cdots, x_m \rangle \in X^m$

由 $x_1 = x_1, x_2 = x_2, \cdots x_m = x_m$

可知

$\langle x_1, x_2, \cdots, x_m \rangle$ 满足$(\bigwedge_{i=1}^{m} x_i \sim x_i)$定义

故$\langle x_1, x_2, \cdots, x_m \rangle \lesssim_m \langle x_1, x_2, \cdots, x_m \rangle$

自反性得证.

若$\langle x_1, x_2, \cdots, x_m \rangle, \langle y_1, \cdots, y_m \rangle$

$\langle z_1, \cdots, z_m \rangle \in X^m$且

$\langle x_1, x_2, \cdots, x_m \rangle \lesssim_m \langle y_1, \cdots, y_m \rangle$

$\langle y_1, y_2, \cdots, y_m \rangle \lesssim_m \langle z_1, \cdots, z_m \rangle$同时成立

则有

$(\bigwedge_{i=1}^{m} x_i \sim y_i)$ 或 $\bigvee_{i=k}^{m-1}((x_{i+1} < y_{i+1}) \wedge (\bigwedge_{j=1}^{i} x_i \sim y_i))$

成立其中 $k \in [0, m-1]$　$k \in N$

及$(\bigwedge_{i=1}^{m} y_i \sim z_i)$ 或 $\bigvee_{i=p}^{m-1}((y_{i+1} < z_{i+1}) \wedge (\bigwedge_{j=1}^{i} y_i \sim z_i))$

成立，其中 $p \in [0, m-1]$，$p \in N$

情况1：$\bigwedge_{i=1}^{m} x_i \sim y_i$ 且 $\bigwedge_{i=1}^{m} y_i \sim z_i$，

则由 ~ 的传递性 $x_i \sim z_i$ $i\in[1,m]$ 且 $i\in N$

则有 $\bigwedge_{i=1}^{m} x_i \sim z_i$ 成立.

情况2：$\bigwedge_{i=1}^{m} x_i \sim y_i$ 且 $\bigvee_{i=p}^{m-1}((y_{i+1}<z_{i+1})\wedge(\bigwedge_{j=1}^{i} y_i\sim z_i))$

则由 ~的传递性有

$x_i\sim z_i$ $i\in[1,p]$，$i\in N$

由~的传递性有

$x_{i+1}<z_{i+1}$ $i\in[p,m-1]$ $i\in N$

故有 $\bigvee_{i=p}^{m-1}((x_{i+1}<z_{i+1})\wedge(\bigwedge_{j=1}^{i}x_i\sim z_i))$ 成立

情况3：$\bigwedge_{i=k}^{m-1}((x_{i+1}<y_{i+1})\wedge(\bigwedge_{j=1}^{i}x_i\sim y_i))$ 且 $(\bigwedge_{i=1}^{m} y_i\sim z_i)$

由 ~的传递性有.

$\bigwedge_{i=k}^{m-1}((x_{i+1}<z_{i+1})\wedge(\bigwedge_{j=1}^{i}x_i\sim z_i))$ 成立

情况4：

$\bigwedge_{i=k}^{m-1}((x_{i+1}<y_{i+1})\wedge(\bigwedge_{j=1}^{i}x_i\sim y_i))$ 且 $\bigwedge_{i=p}^{m-1}((y_{i+1}<z_{i+1})\wedge(\bigwedge_{j=1}^{i}y_i\sim z_i))$

1) $k\le p$ 时.

由 <和~的传递性有：当 $x_i\sim y_i$ 且 $y_i<z_i$ 时 $x_i<z_i$

$\bigwedge_{i=p}^{m-1}((x_{i+1}<z_{i+1})\wedge(\bigwedge_{j=1}^{i}y_i\sim z_i))$ 成立

2) $k>p$ 时.

由<和~的传递性有：

当 $x_i<y_i$ 且 $y_i\sim z_i$ 时，$x_i<z_i$

则有 $\bigwedge_{i=k}^{m-1}((x_{i+1}<z_{i+1})\wedge(\bigwedge_{j=1}^{i}y_i\sim z_i))$ 成立

综上，证得

$\langle y_1, y_2, \cdots, y_m \rangle \precsim_m \langle z_1, z_2, \cdots, z_m \rangle$

即 $\precsim_m$ 具有传递性

结合前面对 $\precsim_m$ 自反性的证明知：$\precsim_m$ 是全预序

(2) $X = \{a, b, c, d, e\}$.

共5个元素，有 $2^5 = 32$ 个子集，可以有

~~有~~32个全预序。