数据结构与算法I 作业6

2019201409 于倬浩

9.3-1

当分块长度为k时，假设存在c、a满足$T(n) \leq c\lceil \frac{n}{k} \rceil + c(n - \lfloor \frac{k+1}{2} \rfloor (\lceil \frac{1}{2} \lceil \frac{n}{k} \rceil \rceil - 2)) + an$

当$k = 7$时，$T(n) \leq \frac{cn}{7} + c(n - 4(\frac{n}{14} - 2)) + an$

$\therefore T(n) \leq \frac{c}{7}n + \frac{5c}{7}n + 8c + an$，
$= \frac{6c}{7}n + an + 8c$
$= cn + (-\frac{1}{7}cn + 8c + an)$

显然$(-\frac{1}{7}cn + 8c + an) \leq 0$可以通过调整c的值，使得n大于常数后满足该式恒成立，因此k=7时依旧可以使该算法维持线性时间复杂度。

然而，当$k = 3$时，若假设$T(n) \leq \frac{cn}{3} + c(n - 2(\frac{n}{6} - 2)) + an$

$$\begin{aligned} &\leq \frac{c}{3}n + \frac{2c}{3}n + 4c + an \\ &= cn + 4c + an \end{aligned}$$

这样直观地可以感觉k=3时并不可行，但是并不能说明$k = 3$时确实无法满足条件，于是尝试严格证明一下：

已知$T(n) = T(\lceil n/3 \rceil) + T(\frac{2}{3}n) + n$

则有$T(n) = T(\lceil n/3 \rceil) + T(\frac{2}{3}n) + an$

$$\begin{aligned} &\geq c\frac{n}{3}lg(\frac{n}{3}) + c\frac{2}{3}nlg(\frac{2}{3}n) + an \\ &= \frac{c}{3}nlgn + \frac{2c}{3}nlgn + an - lg(\frac{1}{3})c\frac{n}{3} - lg(\frac{2}{3})c\frac{2}{3} \\ &\geq cnlgn + O(n) \end{aligned}$$

即当$k = 3$时，$T(n)$增长速度比线性快。

9-3

1. 首先，如果`i`大于等于`n/2`，则直接返回调用`select`的结果。否则，将n个数两两分组，进行$\lfloor \frac{n}{2} \rfloor$次比较，将较小的$\lfloor \frac{n}{2} \rfloor$个元素放到序列的前一半的位置，较大的放到后一半。然后对前一半序列进行递归调用。递归返回后，原序列的前`i`个元素一定为所有较小元素的前`i`小的元素，然而不一定是所有元素中前`i`小的元素。注意到和较小元素中前`i`小的元素同组的、被放到较大元素一组的元素也有可能是原序列前`i`小的元素。因此，将原序列前一半中前`i`小的元素及其在序列后一半中对应的同组元素（共`2i`个）放在一起，调用`select`算法，即可得到原序列中第`i`小的元素。
2. $U_i(n) = \lfloor \frac{n}{2} \rfloor + \lfloor \frac{n}{4} \rfloor + ... + \lfloor \frac{n}{2^k} \rfloor + T(2i) + ... + T(2i)$

   其中$\lfloor \frac{n}{2^k} \rfloor$为大于$i$的最小项。

   因此，序列中共有$lgn - lgi = lg(n/i)$项$T(2i)$，而序列前一半为$(1-\epsilon)n$

   $\therefore U_i(n) = n + O(lg(n/i)T(2i))$
3. 当`i`为一个小于n/2的常数时，则将T(2i)视为常数，则

   $lg(n/i)T(2i) = lg(n/O(1)) * O(1) = O(lgn)$

   $\therefore U_i(n) = n + O(lg(n/i)T(2i)) = n + O(lgn)$
4. 当$i = n/k$时，$lg(n/i)T(2i) = lg(k)T(2n/k)$

   $\therefore O(lg(n/i)T(2i)) = O(lg(k)T(2n/k))$

   $\therefore U_i(n) = n + O(lg(n/i)T(2i)) = n + O(lg(k)T(2n/k))$

9-4

1. 考虑i、j在整个算法执行过程中出现比较的条件：假设当前$i < j \le k$，则排名在i~k中的元素，不能在i或j被选为主元之前被选中，即$z_i, ..., z_k$中，z_i或z_j相对最早地被选为主元，在该区间中，每个元素最早被选为主元的概率相等，因此这种情况下$P\{X_{ijk}\} = \frac{2}{k-i+1}$。同理，若$i \le k < j$，则$P\{X_{ijk}\} = \frac{2}{j-i+1}$；若$k \le i < j$，则$P\{X_{ijk}\} = \frac{2}{j-k+1}$。

   因此，$E[X_{ijk}] = \frac{2}{max(i,j,k) - min(i,j,k) + 1}$。
2. $X_k = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ijk}$

   $\therefore E[X_k] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ijk}]$

   $= \sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} \frac{2}{k-i+1} + \sum_{i=1}^{k} \sum_{j=k}^{n} \frac{2}{j-i+1} + \sum_{i=k+1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-k+1} - 2$

   注意到将$E[X_{ijk}]$拆开为三项时，实际上是讨论了枚举的$\binom{n}{2} - n$对数对(i, j)与k的大小关系，且需要保证i严格小于j，但是k有可能和i或j相等。为了简便起见，将相等的情况放到第二种里考虑（因为此时恰好可以同时考虑到i=k和j=k的情况），然而又引入了i=j=k这一种非法的边界情况，因此需要在答案里减去2（带入$i = j$，算得2）。这样就不需要在其他两种情况里考虑相等的情况。

   第一项$\sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} \frac{2}{k-i+1} = \sum_{i=1}^{k-2} \frac{2(k-1-(i+1)+1)}{k-i+1} = 2\sum_{i=1}^{k-2} \frac{k-i-1}{k-i+1}$

第二项$\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{2}{j-i+1}=2\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1}$

第三项

$\sum_{i=k+1}^{n}\sum_{j=i+1}^{n}\frac{2}{j-k+1}=\sum_{j=k+2}^{n}\sum_{i=k+1}^{j-1}\frac{2}{j-k+1}=2\sum_{j=k+2}^{n}\frac{j-1-(k+1)+1}{j-k+1}=2\sum_{j=k+2}^{n}\frac{j-k-1}{j-k+1}$

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

$\therefore E[X_k]\le 2(\sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1}+\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1}+\sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1})$

注意到这样写出来以后，最后一项求和符号的下界j=k+2与原题要求的j=k+1不一致，然而带入j=k+1时，分子为0，因此对答案没有贡献，因此两种形式等价。

3. $\sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1}+\sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1}\le\sum_{i=1}^{k-2}1+\sum_{j=k+1}^{n}1=k-2+n-(k+1)+1=n-2$

   对于第二问中中间一项，$\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1}=\sum_{t=1}^{n}\frac{1}{t}f(t)$，其中f(t)为在一个长度为n的序列中，覆盖了元素k且长度为t的区间的个数，显然有$f(t)\le t$，因此该项$\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1}\le n$

   求和，得$E[X_k]\le 2(n-2+n)\le 4n$

4. 显然该算法的各项运算量与比较大小的次数同阶，即时间复杂度取决于partition过程中的比较大小操作，而比较大小次数有线性上界$4n$，因此可知，该算法的期望时间复杂度为$O(n)$。