hw5

> 7.1-2，7.4-5；8.2-4，8.4-2；9.1-1；9.3-6；

7.1-2

7.1-2 当数组 $A[p..r]$中的元素都相同时，PARTITION 返回的 q 值是什么？修改 PARTITION，使得当数组 $A[p..r]$中所有元素的值都相同时，$q=\lfloor(p+r)/2\rfloor$。

如果数组中的所有元素值都相同，PARTITION 返回 r。为了在所有元素值相同时使 PARTITION 返回 $q=\lfloor(p+r)/2\rfloor$，请修改算法的第4行，使其变为：如果 $A[j]\le x$ 并且 $j(\bmod 2)=(p+1)(\bmod 2)$。这使得算法将相同值的一半实例视为小于，另一半视为大于。

7.4-5

7.4-5 当输入数据已经"几乎有序"时，插入排序速度很快。在实际应用中，我们可以利用这一特点来提高快速排序的速度。当对一个长度小于 k 的子数组调用快速排序时，让它不做任何排序就返回。当上层的快速排序调用返回后，对整个数组运行插入排序来完成排序过程。试证明：这一排序算法的期望时间复杂度为 $O(nk+n\lg(n/k))$。分别从理论和实践的角度说明我们应该如何选择 k？

当问题大小变为 $\le k$ 时执行快速排序，需要采取 $\lg(n/k)$ 步，因为预期递归树有 $\lg(n)$ 层级。对整个数组进行快速排序后，每个元素与其最终位置的差距在 k 以内。这意味着插入排序对于需要改变位置的每个元素，最多需要移动 k 个元素。

理论上，为了最小化这个表达式，对 k 取导数后应求值为零。得到 $n-\frac{n}{k}=0$，进而 $k\sim\frac{1}{n^2}$。比例常数受 nk 项和 $n\lg(n/k)$ 项中常数的影响。实际操作时，考虑到机器具体属性如缓存行大小，会为各种 k 值使用大量的输入大小进行尝试。

8.2-4

8.2-4 设计一个算法，它能够对于任何给定的介于 0 到 k 之间的 n 个整数先进行预处理，然后在 $O(1)$时间内回答输入的 n 个整数中有多少个落在区间$[a..b]$内。你设计的算法的预处理时间应为 $\Theta(n+k)$。

预处理

1. 创建一个长度为$k+1$的数组C，其中每个位置初始值都为`0`。
2. 遍历给定的n个整数，对于每个整数val，增加$C[val]$的值。

3. 遍历数组C，对于每个位置i，设置$C[i]=C[i]+C[i-1]$。这样$C[i]$现在表示小于或等于i的整数的数量。

预处理的时间复杂度为$O(n+k)$。

查询

要查询区间$[a..b]$内的整数数量，执行以下操作：

1. 如果$a=0$，返回$C[b]$。
2. 否则，返回$C[b]-C[a-1]$。

查询操作的时间复杂度为$O(1)$。

8.4-2

8.4-2 解释为什么桶排序在最坏情况下运行时间是 $\Theta(n^2)$？我们应该如何修改算法，使其在保持平均情况为线性时间代价的同时，最坏情况下时间代价为 $O(n\lg n)$？

在最坏情况下，一个桶可能包含数组的所有 n 个值。因为插入排序的最坏情况运行时间是 $O\left(n^2\right)$，桶排序也同样如此。避免这一情况的方法是使用归并排序来排序每个桶，其最坏情况的运行时间为 $O(n\lg n)$。

9.1-1

9.1-1 证明：在最坏情况下，找到 n 个元素中第二小的元素需要 $n+\lceil\lg n\rceil-2$ 次比较。（提示：可以同时找最小元素。）

在此问题中，通过将数组分成两个大小相等的元素集来进行递归，忽略取底和取顶。如果不假设 n 是2的幂，分析仍然相同，但会稍微复杂些。

将元素分成不相交的对，并比较每一对，只考虑每对中的较小元素。在这组元素中，原问题的结果将是与最小元素配对的元素，或是子问题的第二小的元素。这样可以获得最小和第二小的元素。因此，得到递归式 $T(n)=T(n/2)+n/2+1$ 且 $T(2)=1$。利用替代法解这个递归式，当 $T(n)\le n+\lceil\lg(n)\rceil-2$ 时，它满足基本情况，并且，
$T(n)=n/2+T(n/2)+1\le n/2+n/2+\lceil\lg(n/2)\rceil-2+1=n+\lceil\lg(n)\rceil-2$。

9.3-6

9.3-6 对一个包含 n 个元素的集合来说，k 分位数是指能把有序集合分成 k 个等大小集合的第 $k-1$ 个顺序统计量。给出一个能找出某一集合的 k 分位数的 $O(n\lg k)$ 时间的算法。

假设 n 和 k 都是2的幂。首先使用 SELECT 在 $O(n)$ 时间内找到第 $n/2^{th}$ 个顺序统计量，然后将问题简化为在较小的 $n/2$ 个元素中找到 $k/2^{\text{th}}$ 的分位数，并在较大的 $n/2$ 个元素中找到 $k/2^{th}$ 的分位数。设 $T(n)$ 表示算法在输入大小为 n 时的运行时间。那么 $T(n) = cn + 2T(n/2)$ 对于某个常数 c，基本情况是 $T(n/k) = O(1)$。接下来有：

$$\begin{aligned}
T(n) &\le cn + 2T(n/2) \\
&\le 2cn + 4T(n/4) \\
&\le 3cn + 8T(n/8) \\
&\vdots \\
&\le \log(k)cn + kT(n/k) \\
&\le \log(k)cn + O(k) \\
&= O(n \log k).
\end{aligned}$$