算分第二次作业:

2.7 (21分)

2.7 设 A 是含有 n 个元素的数组，如果元素 x 在 A 中出现的次数大于 $n/2$，则称 x 是 A 的主元素.

(1) 如果 A 中元素是可以排序的，设计一个 $O(n\log n)$ 时间的算法，判断 A 中是否存在主元素.

(2) 对于(1)中可排序的数组，能否设计一个 $O(n)$ 时间的算法？

(3) 如果 A 中元素只能进行“是否相等”的测试，但是不能排序，设计一个算法判断 A 中是否存在主元素.

解答:

(1) (共6分)

- 方法1 排序: 设计思想:把A按照从小到大的顺序排序。那么相同的元素是连续分布的。令x为A 中最多的元素。初始x=A[1]，顺序检查 A中元素，计数z的个数。如果发现比z更多的新元素，就替换。检索完成后，若x的个数大于n/2，x就是主元素;否则 A中没有主元素**(思路5分)**。 该算法的时间包括排序时间和通过比较进行计数的时间，总计为 T(n)=O(nlogn)+O(n)= O(nlogn) **(复杂度2分)**
- 方法2分治: **(算法设计5分，算法复杂度2分)**

分治算法的主要步骤是：

1. 若 A 只含一个元素，则此元素就是主元素，返回此数.
2. 将 A 分为大致相等的两部分 L_1 和 L_2，分别递归调用此算法求其主元素 m_1 和 m_2.
3. 若 m_1 和 m_2 都存在且相等，则这个数就是 A 的主元素，返回此数.
4. 若 m_1 和 m_2 都不存在，则 A 无主元素.
5. 若 m_1 和 m_2 都存在但不等，或者只有一个存在，则分别检查这些数在 A 中出现的次数，如果超过 $n/2$，则是主元素；否则不存在.

该算法的时间分析：第 2 行的两次递归调用，其中子问题规模是原来规模的一半；第 5 行的检查时间为 $O(n)$，于是得到时间复杂度的递推方程

$$\begin{cases} T(n) = 2T\left(\frac{n}{2}\right) + O(n) \\ T(1) = 0 \end{cases}$$

该方程的解是 $T(n)=O(n\log n)$.

(2) 适用于元素可排序的数组.

算法思路 由于 A中的主元素一定是中位数。按照从小到大排序 A的元素.若存在主元素x，那么x应该至少占有连续的 n/2+1个位置，如果其中不包含中位数的位置在内，那么这些连续位置只能分布在中位

数的单侧.中位数任何一侧的元素数都不超过n/2.这与主元素的定义矛盾。

算法描述: 1.找出 A的中位数x。 2.把x与A 中元素顺序比较并计数x在A 中出现的次数j。如果i>n/2，则x就是主元素;否则不存在。**(算法思路5分，酌情扣分)**

该算法的时间是第1行找中位数的时间和第2行的检查时间，两者都是 O(n)，于是T(n)=O(n)+O(n)=O(n) **(2分)**

(3) 如果A中元素不能排序，可采用基于两两元素的比较从而把非主元素淘汰的算法其实现方法有两种.

方法一 (思路5分，酌情扣分，没有最后的检验-1分)

设计思想如下 算法 Delete(A)

1. 如果A中元素个数小于1，则结束;否则将 A的元素两两分组。
2. 每组内两个元素进行比较: 如果相等则把1个放到B中，否则全部淘汰。
3. B作为新的A 并回到第一步继续处理。

运行上述 Delete 淘汰过程，如果最后没有元素留下，那么没有主元素，如果剩下1个元素，接着检查该元素在A中出现的次数，若次数大于n/2，则是主元素;否则没有主元素。

复杂度分析: (2分)

最后考虑算法的运行时间，经过1次 Delete 过程子问题规模至少减半，Delete 过程需要比较 n/2 次，加上对轮空元素的检查也不超过 O(n)次，于是得到递推方程: W(n)=W(n/2)+O(n) W(1)=0，即复杂度为O(n)。

方法2 (思路5分，酌情扣分，没有检验-1分)

设计思想:设立一个栈S，大小为n/2.下面顺序对 A中的元素进行处理.对任何元素x，处理原则是:

1.如果栈为空，则x进栈

2.如果栈不空，则比较x与栈顶元素，如果相等，则x进栈，否则x与栈顶元素一起淘汰.

在A中元素全部处理完成后，如果栈内剩有元素，那么检查栈顶元素在A中出现的次数.次数大于”/2的是主元素;否则没有主元素.如果栈为空，则没有主元素.

复杂度分析: (2分)

W(n) = O(n) + O(n) = O(n)

2.9 (11分)

2.9 在 $n(n\geq 3)$ 枚硬币中有一枚重量不合格的硬币(重量过轻或者过重)，如果只有一架天平可以用来称重且称重的硬币数没有限制，设计一个算法找出这枚不合格的硬币，使得称重的次数最少？给出算法的伪码描述. 如果每称 1 次就作为 1 次基本运算，分析算法的最坏情况下的时间复杂度.

算法设计: 算法设计思想:采用分治策略。 将n枚硬币分成大致相等的3份，如果n(mod 3)≠0， 那么令两份少的硬币相等。取两份相等的硬币放到天平上，如果天平不平衡，那么这两份里包含着不合格的硬币；否则不合格的硬币在剩下的一份中。递归对归约后的子问题进行处理。当处理到n小于3时，那么将剩下的硬币与拿走的硬币(合格硬币)比较，不等的那枚就是不合格的硬币。**(思路对给5分，思路不对如果提到了分治可以给2分，未处理n小于3的情况扣一分，其他情况酌情扣分。)**

伪码: 评分标准: 伪码一共4分，可以视情况酌情扣分。

```
算法 Coin(A,n)                    //A 是 n 个硬币的集合
1.  k←n/3
2.  将 A 中硬币分成 X,Y,Z 三个集合,使得|X|=|Y|=k,|Z|=n−2k
3.      if W(X)≠W(Y)              //W(X),W(Y)分别为 X 或 Y 的重量
4.      then A←X∪Y
5.      else A←Z
6.      n←|A|
7.  if n>2 then goto 1
8.  else 将 A 中的硬币与拿走的硬币比较. 如果不等,则是不合格硬币.
```

复杂度分析: (2分，结果对可以直接给，结果不对看递推方程，如果对可以给一分)。

该算法递推方程： T(n) = T(2n/3) +O(1) ; T(1)=0，T(2)=1 。该方程的解是T(n)=O(logn).

其他

如果将硬币分成个数大致相等的4份，取个数相等的两份放到天平上。 如果天平平衡，那么不合格的硬币在剩下的硬币中；否则在被称重的硬币中。1次称重后待检测的硬币数减半，直到边界条件为止。这种分治算法的时间也是n(logn)。 **(如果按照这种思路，大概的评分标准同上，可以根据情况对应，并酌情扣分)**

2.12 (13分)

2.12 设 $A=\{a_1,a_2,\cdots,a_n\}$,$B=\{b_1,b_2,\cdots,b_m\}$是整数集合,其中 $m=O(\log n)$. 设计一个算法找出集合 $C=A\cap B$. 要求给出伪码描述.

算法设计思想: 在A与B中选择一个数组排序，然后顺序对另一个数组的每个元素使用二分查找算法，检查其是否出现在排序的数组中是否出现。如果出现，则将它放入C。**(算法设计5分)**。关键是选择A还是B进行排序，算法的主要时间消耗是排序。因此选择较小的数组进行排序是合适的。考虑到B的元素数量要比A要小得多，因此对 B排序。**(对B的排序2分，在伪码指出也可得分)**

伪码:

算法

输入：数组 $A[1..n]$和 $B[1..m]$

输出：数组 C，使得 $C=A\cap B$

1. $C\leftarrow\varnothing$
2. $L\leftarrow$Sort(B)　　//对 B 排序
3. for $i\leftarrow1$ to n do
4. 　$x\leftarrow A[i]$
5. 　$j\leftarrow$BinarySearch(L,x)　//在 L 中二分搜索 x，若 x 在 L 中，j 为序标；否则 j 为 0
6. 　if $j>0$
7. 　then $C\leftarrow C\cup\{x\}$

评分标准: 整体思路4分。如果整体思路对可以酌情在5分上进行扣分(例如排序不对-1分，循环不对-1分，二分搜索-1分，输入输出不对可以-1分)。

复杂度分析（2分）: 以比较作为基本运算，该算法在第2行的排序需要 O(mlogm) 时间，第3行的循环运行n次，每次做的工作量是第5行的二分查找，查找时间是 O(logm)，因此第3行的 for 循环总时间是 O(nlogm) ，于是算法时间复杂度是 W(n)=O(mlogm)+O(nlogm)=O((m+n)logm) = O(nlogm)=O(nloglogn)。**(2分，结果不对如果前面分析对了可以给1分，或者酌情扣分**

其他情况说明: 如果简单的暴力对比，最多给不超过10分。如果两个都排序，类似于归并的方式获取并集，可以酌情给分，但是不超过12分。（实际上这两个的复杂度都是nlogn)

2.15 (15分)

2.15 给定 n 个不同数的数组 S 和正整数 i，$i\leqslant n^{1/2}$，求 S 中最大的 i 个数，并且按照从大到小的次序输出. 有下述算法：

算法 A：调用 i 次找最大算法 Findmax，每次从 S 中删除一个最大的数.

算法 B：对 S 排序，并输出 S 中最大的 i 个数.

(1) 分析 A、B 两个算法在最坏情况下的时间复杂度.

(2) 试设计一个最坏情况下时间复杂度的阶更低的算法，要求给出伪码.

(1) **复杂度分析:**

算法A **(2分，结果错误但是给出Findmax O(n) 可以给1分)** 每次调用 Findmax需要 O(n)时间，调用i次，于是T(n)= O(in)= 0(n^(3/2))

算法B **(2分，结果错误如果给出排序是O(nlogn) 给1分)** 排序时间是 O(nlogn)，输出时间是 O(i)，于是T(n)=O(nlogn)+O(i)=O(nlogn)

(2) **算法C的设计思想**: **(5分，思路对即可，酌情扣分)** 设k表示第i大元素在从小到大排好序数组中的下标，即k=n-i+1。用选择算法确定这个元素x;然后用z划分S，将比x大的放到后边;排序S中从k 到n的元素，倒序输出。

伪码：(4分，思路对即可，可以酌情扣分)

算法 C
输入：n 个数的数组 S
输出：倒序排列的前 i 个大的数
1. $k \leftarrow n-i+1$
2. $x \leftarrow \text{Select}(S,k)$
3. 采用划分算法 Partition，将 S 中比 x 大的数交换到 $S[k+1..n]$
4. 对 $S[k..n]$排序
5. 从后向前输出 $S[k..n]$

算法C复杂度分析:(2分) 以比较作为基本运算，该算法的第2行用 O(n)时间，第3行用O(n)时间，第4行用O(ilogi)时间，于是总时间为 T(n)=O(n)+O(n)+O(ilogi) = O(n)

2.16 （14分）

2.16 设 S 为 n 个不同数的集合，
(1) 设计算法找出 S 中的数 x 和 y 使得 $\forall u,v \in S, |x-y| \geqslant |u-v|$.
(2) 设计算法找出 S 中的数 x 和 y 使得 $\forall u,v \in S, |x-y| \leqslant |u-v|$.

(1) 算法设计思想:利用找最大和最小的算法 FindMaxMin，输出S中最大值max和最小值min，令x = max，y = min。**(5分，思路对即可)**

该算法最坏情况下的时间复杂度是FindMaxMin算法复杂度，即W(n)=3n/2-2。**(2分，如果给出的O(n)的时间复杂度也可以给分，没有分析酌情扣分。)**

(2)算法设计思想: **(5分，思路对即可)**

1.按照递增顺序排序S中的数

2.令c是第二个数减去第一个数的差

3.从第三个数开始，顺序计算与前一个数的差并与c比较

4.n-1个差中的最小差所对应的两个数就是所求的数

时间复杂度分析: 该算法的第一步需要 O(nlogn)时间，第2步常数时间，第3步需要 O(n)时间，因此总时间是 W(n)=O(nlogn)+O(1)+O(n)=O(nlogn)。 **(2分，没有分析酌情扣分)**

2.18 (11分)

2.18 设平面直角坐标系中有 n 个点 $(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$，每个点到原点 $(0,0)$ 的距离彼此不等. 设计一个算法找到距离原点 $(0,0)$ 最近的 $\sqrt{n}$ 个点，并按照距原点从远到近的顺序输出点的标号. 要求给出伪码描述.

解答: (算法设计思想5分，整体思路对可以给分，算法伪码4分)

2.18 算法设计思想是：先依次计算每个点到原点的距离 $d_1,d_2,\cdots,d_n$，从其中找出第 $\sqrt{n}$ 小的距离 d_i，依次将每个点的距离与 d_i 比较，如果小于 d_i，则是满足要求的点.

算法 FindPoint(A,B)

输入：$A=\{x_1,x_2,\cdots,x_n\}$，$B=\{y_1,y_2,\cdots,y_n\}$ 分别为 n 个点的横、纵坐标

输出：距离原点最近的 $\sqrt{n}$ 个点的标号

1. for $i\leftarrow 1$ to n
2. $\quad d_i\leftarrow\sqrt{x_i^2+y_i^2}$
3. $k\leftarrow\sqrt{n}$
4. 在 $\{d_1,d_2,\cdots,d_n\}$ 中找第 k 小的元素 b
5. 把每个 d_i 与 b 比较，把 k 个小于等于 b 的 d_i 及其下标 i 存放到 C 中
6. 对 C 中 d_i 按照从大到小顺序排序，若移动 d_i 时其下标 i 同时移动
7. for $j\leftarrow 1$ to k return $C[j]=d_i$ 的下标 i

复杂度分析 :(2分，结果不对可酌情扣分)

上述算法在第1行和第2行的时间是 O(n)，第4行调用 Select 算法，时间也是 O(n).第5 行的比较执行 O(n)次，每次执行用常数时间，共执行 O(n)时间，第6行的排序是O(klogk)时间，而 k=O(n^(1/2))，第7行输出是 O(k)时间，以比较做基本运算，总复杂度是:

W(n)=O(n)+O(klogk)=O(n)+O($\sqrt{}$nlog$\sqrt{}$n)= O(n)

2.23（7分）

2.23 设 A 是 n 个数的序列，如果 A 中的元素 x 满足以下条件：小于 x 的数的个数$\geqslant n/4$，且大于 x 的数的个数$\geqslant n/4$，则称 x 为 A 的近似中值. 设计算法求出 A 的一个近似中值. 说明算法的设计思想和最坏情况下的时间复杂度.

1. 用 Select 算法找第 $n/4$ 小的数 a 和第 $3n/4$ 小的数 b
2. if $a=b$ then return “无解”
3. else 用 a 和 b 划分数组 A 为 A_1,A_2,A_3,A_4,A_5，其中 A_1 的数$<a$，A_2 的数$=a$，A_3 的数$>a$ 且小于 b，A_4 的数$=b$. A_5 的数$>b$
4. if A_3 非空，则 A_3 中的任意数都为近似中值，否则无解

算法的时间复杂度为 $O(n)$.

(算法思想5分，思路对即可，如果思路不对select给1分，划分数组给1分，其他步骤各给1分，根据情况酌情扣分，时间复杂度2分)

2.27 (8分)

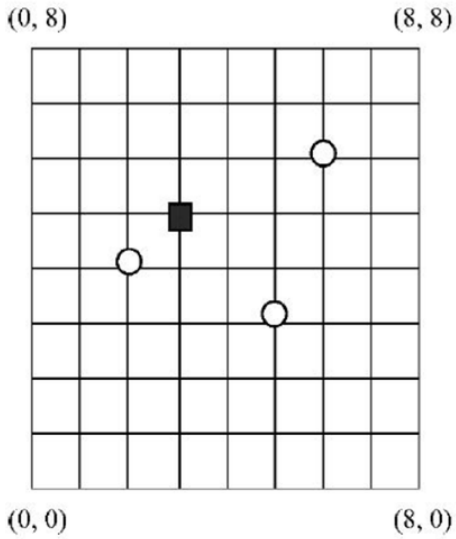

图 2.1　街道图

2.27　如图 2.1(注：主教材图 2.8)所示，城市街道都是水平或垂直分布，有 $m+1$ 条，不妨设任何两个相邻位置之间的距离都是 1. 在街道的十字路口有 n 个商店，图中的 $n=3, m=8$，3 个商店的坐标位置(图中的圆点)分别是(2,4),(5,3),(6,6). 现在需要在某个路口位置建一个合用的仓库. 若仓库选择(3,5)位置，那么这 3 个商店到仓库的路程(只能沿街道行进)总长至少是 10. 设计一个算法找到仓库的最佳位置，使得所有商店到仓库路程的总长达到最短.

2.27　设 n 个商店的坐标是 $(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$，问题的解 (x,y) 满足

$$\min\{\,|x-x_1|+|y-y_1|+|x-x_2|+|y-y_2|+\cdots+|x-x_n|+|y-y_n|\;|\;0\leqslant x,y\leqslant m\}$$

由于横、纵坐标是互相独立的，于是得到

$$\min\{|x-x_1|+|x-x_2|+\cdots+|x-x_n|\;|\;x=0,1,\cdots,m\}$$

$$\min\{|y-y_1|+|y-y_2|+\cdots+|y-y_n|\;|\;y=0,1,\cdots,m\}$$

(相互独立2分)

考虑在同一个y轴上，将所有的y轴的数据拼接到一个数组中，可以观察到中位数是一个最优解。

(算法设计包括证明。4分，复杂度2分)

证明: 考虑中位数为y， 然后其上方存在n/2各，在其上方有n/2个。假设放在另一个Y的位置（假设仓库的新坐标Y小于中位数y)，假设Y与y之间存在k个商店。不妨设置d=y-Y；则原来上方的增加的数量为d*n/2,下方减少的数量为(n/2-k)d, 中间的部分减少的量(即为delta)一般少于k*d(甚至部分还会增加）。增加的总距离可以计算为:

d*n/2 - (n/2-k)*d - delta = kd-delta > 0; 所以在中位数是一个最优解。

算法设计思想：取 $x_1,x_2,\cdots,x_n$ 的中位数作为 x，$y_1,y_2,\cdots,y_n$ 的中位数作为 y 即可，算法时间复杂度为 $O(n)$.

一般算法设计给5分，伪码给4分，然后复杂度给2分，大部分都是这个逻辑来给分的。