Homework #1

211220019 陈骏桢

1.2

1) 算法：

```
Algorithm 1: mid(a,b,c)
Input: int a,b,c
Output: mid(a,b,c)
if a<b then
    if a>c then
        return a;
    else if b<c then
        return b;
    else
        return c;
    end
else
    if a<c then
        return a;
    else if b>c then
        return b;
    else
        return c;
    end
end
```

2) 在最坏的情况下，需要进行 3 次比较；
在平均的情况下，需要进行 $3 \times \frac{4}{6} + 2 \times \frac{2}{6} = \frac{8}{3}$ 次比较。

3) 至少需要 3 次比较，证明如下：
   设 3 个数为 a,b,c，假设存在只需要 2 次比较就能得到中位数的算法，则
   设第一次比较 a 和 b，且 $a < b$；
   若第二次比较 a 和 c，当 $a < c$ 时无法确定中位数为 b 还是 c；
   若第二次比较 b 和 c，当 $b > c$ 时无法确定中位数为 a 还是 c。
   所以不存在只需要 2 次比较的算法，
   又因为第一问的算法在最坏的情况下只需要 3 次比较，
   所以至少需要三次比较。

1.3

1) 失败例子：
   U={1,2,3,4},$S_1 = \{1, 2, 3\}, S_2 = \{1, 2\}, S_3 = \{3\}$
   用该方法得到的最小覆盖为 $\{S_1, S_2, S_3\}$，而实际上最小覆盖是 $\{S_1, S_3\}$。

2) 算法：

Algorithm 2: mincover $(S_1, S_2, \ldots, S_k)$

```
Input: U,S={S_1, S_2, ..., S_m}
Output: mincover
found:=false;
for each x_i ∈ U do
    x_i is unvisited;
end
for each x_i ∈ U do
    if x_i is unvisited then
        found=false;
        for each S_k ∈ S do
            if x_i ∈ S_k then
                make each x_j ∈ S_k visited;
                join S_k into mincover;
                found=true;
                break;
            end
        end
        if found==false then
            return NULL;
        end
    end
end
return mincover;
```

算法正确性证明：该算法中，每次循环，若找不到覆盖 x_i 的子集 S_k，则直接返回 NULL，否则找到一个子集 S_k，将该子集中未被访问到的元素标记为已访问，几从未被覆盖的元素中标记至少 1 个元素为已覆盖，又因为未覆盖的元素个数有限，所以算法必然能终止。同时，更具覆盖的定义，最后正常返回的子集族一定是一个集合覆盖，故该算法正确。

3) 不能，反例如下：
U={1,2,3},S_1={1},S_2={2},S_3={3},S_4={1,2}, 则该算法的输出为 $\{S_1, S_2, S_3\}$，但正确答案是 $\{S_3, S_4\}$。

1.7

证明：

i) 当 $n=0$ 时，$p=a_0=P(x)$，正确；

ii) 假设当 $n=k$ 时算法正确，即 $p=P(x)=\sum_{i=0}^{k}a_ix^i$，则当 $n=k+1$ 时，有假设可知，在算法中循环的倒数第二次中：$p=\sum_{i=1}^{k+1}a_ix^{i-1}$，故最后一次循环中：$p=p\cdot x+A[0]=\sum_{i=1}^{k+1}a_ix^i+a_0=\sum_{i=0}^{k+1}a_ix^i$；

iii) 综上所述：归纳可得该算法对一切 $n\in\mathbb{N}$ 成立。

1.8

1) 证明：

i) 当 $z=1$ 时，$INT-MULT(y,1)=INT-MULT(2y,0)+y=y$，结论成立；

ii) 假设当 $z\leq k(k\geq 1)$ 时算法成立，即 $INT-MULT(y,z)==y\cdot z$，则当 $z=k+1$ 时，$INT-MULT(y,k+1)=INT-MULT(2y,\lfloor\frac{k+1}{2}\rfloor)+y\cdot((k+1)\mod 2)$，对 k 的奇偶性进行讨论：

① 当 $k=2n$ 时，

$$
\begin{aligned}
&INT-MULT(y,2n+1)=INT-MUTL(2y,\lfloor\frac{2n+1}{2}\rfloor)+y\\
&=INT-MUTL(2y,n)+y\\
&=2ny+y=(2n+1)y\\
&=y(k+1)\\
&=y\cdot z
\end{aligned}
$$

结论成立；

② 当 $k = 2n + 1$ 时，

$$INT - MULT(y, 2n + 2) = INT - MUTL(2y, \lfloor \frac{2n+2}{2} \rfloor)$$
$$= INT - MUTL(2y, n + 1)$$
$$= 2(n + 1)y$$
$$= y(k + 1)$$
$$= y \cdot z$$

结论成立；

iii) 综上所述：该算法对所有的 $z \in \mathbb{N}$ 成立，故该算法正确。

2) 证明：

i) 当 $z = 1$ 时，$INT - MULT(y, 1) = INT - MULT(c \cdot y, 0) + y = y$，结论成立；

ii) 假设当 $z \leq k(k \geq 1)$ 时算法成立，即 $INT - MULT(y, z) == y \cdot z$，则当 $z = k+1$ 时，$INT - MULT(y, k+1) = INT - MULT(cy, \lfloor \frac{k+1}{c} \rfloor) + y \cdot ((k+1) \mod c)$，令 $\lfloor \frac{k+1}{c} \rfloor = m, (k+1) \mod c = n$，则 $k+1 = mc+n$，则：
$$INT - MULT(y, k+1) = INT - MULT(cy, m) + ny$$
$$= mcy + ny = (mc + n)y$$
$$= y \cdot (k + 1)$$
$$= y \cdot z$$

iii) 综上所述：该算法对所有的 $z \in \mathbb{N}$ 成立，故该算法正确。

1.9

设平均情况时间复杂度为 $A(n)$，则：

$$A(n) = \sum_{i=1}^{n} Pr(r = i) f(i)$$
$$= \frac{1}{n} \cdot \frac{n}{4} \cdot 10 + \frac{2}{n} \cdot \frac{n}{4} \cdot 20 + \frac{1}{2n} \cdot \frac{n}{4} \cdot 30 + \frac{1}{2n} \cdot \frac{n}{4} \cdot n$$
$$= \frac{n + 130}{8}$$
$$\in O(n);$$

1.10

UNIQUE 算法用于判断数组中是否所有元素都互不相同，时间复杂度为 $O(n^2)$。

1) 算法中有两层循环，最坏情况下会遍历所有元素对，最坏情况时间复杂度为：$\sum_{i=0}^{n-2}(\sum_{j=i+1}^{n-1}(1)) = \sum_{i=0}^{n-2}(n-i-1) = \frac{n^2-n}{2} \in O(n^2)$;

2) 设平均情况时间复杂度为 $A(n)$，则：

$$
\begin{aligned}
A(n) =&(\sum_{i=0}^{n-2}(\sum_{j=i+1}^{n-1}(j-i+\sum_{k=0}^{i-1}(n-1-k))))\Big/C_n^2 \\
=&(\sum_{i=0}^{n-2}(\sum_{j=i+1}^{n-1}(j-i+i\cdot(n-1)-\frac{i(i-1)}{2})))\Big/C_n^2 \\
=&(\sum_{i=0}^{n-2}(\sum_{j=i+1}^{n-1}(j-\frac{i^2}{2}+i(n-\frac{3}{2}))))\Big/C_n^2 \\
=&(\sum_{i=0}^{n-2}(\frac{(n+i)(n-i-1)}{2}-(n-i-1)(\frac{i^2}{2}-i(n-\frac{3}{2})))\Big/C_n^2 \\
=&(\sum_{i=0}^{n-2}(\frac{(n-i^2+(2n-2)i)(n-i-1)}{2}))\Big/C_n^2 \\
=&(\sum_{i=0}^{n-2}((n-i^2+(2n-2)i)(n-i-1)))\Big/(n(n-1)) \\
=&(\sum_{i=0}^{n-2}(i^3-(3n-3)i^2+(2n^2-5n+2)i+n^2-n))\Big/(n(n-1)) \\
=&((\frac{(n-1)(n-2)}{2})^2-\frac{(3n-3)(n-2)(n-1)(2n-3)}{6} \\
&+\frac{(2n^2-5n+2)(n-2)(n-1)}{2}+(n-1)(n^2-n))\Big/(n(n-1)) \\
=&(\frac{n}{4}(n-1)(n^2-n+2))\Big/(n(n-1)) \\
=&\frac{n^2-n+2}{4} \\
\in&O(n^2);
\end{aligned}
$$

3) 设平均情况时间复杂度为 $A(n)$，则：

$$\begin{aligned}A(n)=&\sum_{i=0}^{n-2}(\sum_{j=i+1}^{n-1}((j-i+\sum_{s=0}^{i-1}(n-1-s))(\prod_{p=0}^{i-1}((1-\frac{1}{k-p})^{(n-1-p)}))(1-\frac{1}{k-i})^{(j-i-1)}\frac{1}{k-i}))\\ \in&O(n^2\cdot(1-\frac{1}{k})^{n^2});\end{aligned}$$

2.2

设 $k\in\mathbb{N}$ 满足 $2^k\leq n\leq 2^{k+1}-1$，即 $\log_2(n+1)-1\leq k\leq\log_2 n$；因为 $n\geq 1$，所以 $\log_2(n+1)-1=\log_2(\frac{n+1}{2})\leq\log_2 n$，所以 k 存在。对于 $\lceil\log(n+1)\rceil$：

$$\begin{aligned}&2^k\leq n\leq 2^{k+1}-1\\ \implies&k<\log(n+1)\leq k+1\\ \implies&k<\lceil\log(n+1)\rceil\leq k+1\\ \implies&\lceil\log(n+1)\rceil=k+1;\end{aligned}$$

对于 $\lfloor\log n\rfloor+1$：

$$\begin{aligned}&2^k\leq n\leq 2^{k+1}-1\\ \implies&k\leq\log n<k+1\\ \implies&k\leq\lfloor\log n\rfloor<k+1\\ \implies&k+1\leq\lfloor\log n\rfloor+1<k+2\\ \implies&\lfloor\log n\rfloor+1=k+1;\end{aligned}$$

所以 $\lceil\log(n+1)\rceil=\lfloor\log n\rfloor+1$。

2.5

1) 若 T 为 $2-tree$，则 T 中 $n_1=0$，设 T 共有 n 个节点，则 $n=n_0+n_2$，除了根节点外的其他节点都有一条边向下延伸得到，则该二叉树总共有 $n-1$ 条边，同时，度为 2 的节点向下延伸出两条边，则 $n-1=2n_2$，联合两个式子可得：$2n_2+1=n_0+n_2$，则 $n_0=n_2+1$。

2) 设 T 共有 n 个节点，则 $n = n_0 + n_1 + n_2$，除了根节点外的其他节点都有一条边向下延伸得到，则该二叉树总共有 $n-1$ 条边，同时，度为 1 的节点会向下延伸出一条边，度为 2 的节点向下延伸出两条边，则 $n-1 = 2n_2 + n_1$，联合两个式子可得：$2n_2 + n_1 + 1 = n_0 + n_1 + n_2$，则 $n_0 = n_2 + 1$。

2.7

1) 证明：

   i) O：

$$\left.\begin{aligned} f(n) = O(g(n)) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)} = c_1 < \infty \\ g(n) = O(h(n)) &\iff \lim_{n\to\infty}\frac{g(n)}{h(n)} = c_2 < \infty \end{aligned}\right\} \implies \lim_{n\to\infty}\frac{f(n)}{h(n)} = c_1c_2 < \infty$$

$$\iff f(n) = O(h(n));$$

   所以 $(f,g) \in O, (g,h) \in O \implies (f,h) \in O$，所以 O 满足传递性。

   ii) Ω：

$$\left.\begin{aligned} f(n) = \Omega(g(n)) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)} = c_1 > 0 \\ g(n) = \Omega(h(n)) &\iff \lim_{n\to\infty}\frac{g(n)}{h(n)} = c_2 > 0 \end{aligned}\right\} \implies \lim_{n\to\infty}\frac{f(n)}{h(n)} = c_1c_2 > 0$$

$$\iff f(n) = \Omega(h(n));$$

   所以 $(f,g) \in \Omega, (g,h) \in \Omega \implies (f,h) \in \Omega$，所以 Ω 满足传递性。

   iii) Θ：

$$\left.\begin{aligned} f(n) = \Theta(g(n)) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)} = c_1 \in (0,\infty) \\ g(n) = \Theta(h(n)) &\iff \lim_{n\to\infty}\frac{g(n)}{h(n)} = c_2 \in (0,\infty) \end{aligned}\right\} \implies \lim_{n\to\infty}\frac{f(n)}{h(n)} = c_1c_2 \in (0,\infty)$$

$$\iff f(n) = \Theta(h(n));$$

   所以 $(f,g) \in \Theta, (g,h) \in \Theta \implies (f,h) \in \Theta$，所以 Θ 满足传递性。

iv) o：

$$\left.\begin{aligned} f(n)=o(g(n)) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)}=0 \\ g(n)=o(h(n)) &\iff \lim_{n\to\infty}\frac{g(n)}{h(n)}=0 \end{aligned}\right\} \implies \lim_{n\to\infty}\frac{f(n)}{h(n)}=0$$
$$\iff f(n)=o(h(n));$$

所以 $(f,g)\in o,(g,h)\in o \implies (f,h)\in o$，所以 o 满足传递性。

v) ω：

$$\left.\begin{aligned} f(n)=\omega(g(n)) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty \\ g(n)=\omega(h(n)) &\iff \lim_{n\to\infty}\frac{g(n)}{h(n)}=\infty \end{aligned}\right\} \implies \lim_{n\to\infty}\frac{f(n)}{h(n)}=\infty$$
$$\iff f(n)=\omega(h(n));$$

所以 $(f,g)\in\omega,(g,h)\in\omega \implies (f,h)\in\omega$，所以 ω 满足传递性。

2) i) O：$\lim_{n\to\infty}\frac{f(n)}{f(n)}=\lim_{n\to\infty}(1)=1<\infty$，所以 $f(n)=O(f(n))$，$(f,f)\in O$，所以 O 关系满足自反性；

ii) Ω：$\lim_{n\to\infty}\frac{f(n)}{f(n)}=\lim_{n\to\infty}(1)=1>0$，所以 $f(n)=\Omega(f(n))$，$(f,f)\in\Omega$，所以 Ω 关系满足自反性；

iii) Θ：$\lim_{n\to\infty}\frac{f(n)}{f(n)}=\lim_{n\to\infty}(1)=1\in(0,\infty)$，所以 $f(n)=\Theta(f(n))$，$(f,f)\in\Theta$，所以 Θ 关系满足自反性。

3) 对于两个不同的函数 $f(n),g(n)$：

$$\begin{aligned}(f,g)\in\Theta &\iff f(n)=\Theta(g(n)) \\ &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)}=c\in(0,\infty) \\ &\iff \lim_{n\to\infty}\frac{g(n)}{f(n)}=\frac{1}{c}\in(0,\infty) \\ &\iff g(n)=\Theta(f(n)) \\ &\iff (g,f)\in\Theta\end{aligned}$$

所以 Θ 关系满足对称性，又由上面两问可得，Θ 关系满足传递性和自反性，所以 Θ 关系是一个等价关系。

4)

$$\begin{aligned}
f(n) = \Theta(g(n)) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)} = c \in (0,\infty) \\
&\iff (\lim_{n\to\infty}\frac{f(n)}{g(n)} = c < \infty) \land (\lim_{n\to\infty}\frac{f(n)}{g(n)} = c > 0) \\
&\iff (f(n) = O(g(n))) \land (f(n) = \Omega(g(n)))
\end{aligned}$$

所以 $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$。

5)

$$\begin{aligned}
f = O(g) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)} = c < \infty \\
&\iff \lim_{n\to\infty}\frac{g(n)}{f(n)} = \frac{1}{c} > 0 \\
&\iff g = \Omega(f); \\
f = o(g) &\iff \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0 \\
&\iff \lim_{n\to\infty}\frac{g(n)}{f(n)} = \infty \\
&\iff g = o(f).
\end{aligned}$$

6) i)

$$o(g(n)) = \{f| \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0\}$$

$$\omega(g(n)) = \{f| \lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty\}$$

$$o(g(n)) \cap \omega(g(n)) = \{f| \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0 \land \lim_{n\to\infty}\frac{f(n)}{g(n)} == \infty\} = \varnothing$$

ii)

$$\Theta(g(n)) = \{f| \lim_{n\to\infty}\frac{f(n)}{g(n)} = c \in (0,\infty)\}$$

$$o(g(n)) = \{f| \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0\}$$

$$\Theta(g(n)) \cap o(g(n)) = \{f| \lim_{n\to\infty}\frac{f(n)}{g(n)} = c \in (0,\infty) \land \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0\} = \varnothing$$

iii)

$$
\begin{aligned}
&\Theta(g(n)) = \{f|\lim_{n\to\infty}\frac{f(n)}{g(n)} = c \in (0,\infty)\}\\
&\omega(g(n)) = \{f|\lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty\}\\
&\Theta(g(n)) \cap \omega(g(n)) = \{f|\lim_{n\to\infty}\frac{f(n)}{g(n)} = c \in (0,\infty) \wedge \lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty\} = \varnothing
\end{aligned}
$$

2.8

1) $\log n < n < n\log n < n^2 \le (n^2+\log n) < n^3 < (n-n^3+7n^5) < 2^n$
   n^2 与 $n^2+\log n$ 渐近增长率相同

2) $\log\log n < \ln n < \log n < (\log n)^2 < \sqrt{n} < n < n\log n < n^{1+\varepsilon} < n^2 \le (n^2+\log n) < n^3 < (n-n^3+7n^5) < 2^{n-1} \le 2^n < \mathrm{e}^n < n!$
   n^2 和 $n^2+\log n$ 的渐近增长率相同，2^{n-1} 和 2^n 的渐近增长率相同。

2.16

1) $T(n) = 2T(n/3)+1$
   $a=2, b=3, f(n)=1$
   $\exists\varepsilon>0$，令 $\varepsilon \in (0,\log_3 2)$，使得：
   $\lim_{n\to\infty}\frac{f(n)}{n^{\log_b a-\varepsilon}} = \lim_{n\to\infty}\frac{1}{n^{\log_3 2-\varepsilon}} = 0, f(n) = O(n^{\log_b a-\varepsilon})$，
   所以 $T(n) = \Theta(n^{\log_3 2})$。

2) $T(n) = T(n/2)+c\log n$
   无法使用主定理求渐近增长率。
   可将该递归函数转化为递归树，则共有 $\log_2 n$ 层，第 k 层的代价为 $c\log\frac{n}{2^k}(0\le k<\log_2 n)$，
   所以总代价为 $\sum_{k=0}^{\log_2 n-1}(c\log\frac{n}{2^k}) = c\sum_{k=0}^{\log_2 n-1}(\log n-k) = c(\log n)^2 - c\frac{(\log n)(\log n-1)}{2}$，
   所以 $T(n) = \Theta((\log n)^2)$。

3) $T(n) = T(n/2)+cn$
   $a=1, b=2, f(n)=cn$
   $\exists\varepsilon>0$，令 $\varepsilon\in(0,1)$，使得：

$\lim_{n\to\infty}\frac{f(n)}{n^{\log_b a+\varepsilon}}=\lim_{n\to\infty}\frac{cn}{n^{\varepsilon}}>0, f(n)=\Omega(n^{\log_b a+\varepsilon})$，
又因为 $\exists k<1$，令 $k\in(\frac{1}{2},1)$，对所有充分大的 n，$af(\frac{n}{b})=\frac{cn}{2}\leq kf(n)$，
所以 $T(n)=\Theta(cn)=\Theta(n)$。

4) $T(n)=2T(n/2)+cn$
$a=2, b=2, f(n)=cn$
$\lim_{n\to\infty}\frac{f(n)}{n^{\log_b a}}=\lim_{n\to\infty}\frac{cn}{n}=c, f(n)=\Theta(n^{\log_b a})$，
所以 $T(n)=\Theta(n\log n)$。

5) $T(n)=2T(n/2)+cn\log n$
无法使用主定理求渐近增长率。
可将该递归函数转化为递归树，则共有 $\log_2 n$ 层，第 k 层的代价为 $2^k c\frac{n}{2^k}\log\frac{n}{2^k}=cn\log\frac{n}{2^k}(0\leq k<\log_2 n)$，
所以总代价为 $\sum_{k=0}^{\log_2 n-1}(cn\log\frac{n}{2^k})=cn\sum_{k=0}^{\log_2 n-1}(\log n-k)=cn(\log n)^2-cn\frac{(\log 2n)(\log n)}{2}$，
所以 $T(n)=\Theta(n(\log n)^2)$。

6) $T(n)=3T(n/3)+n\log^3 n$
无法使用主定理求渐近增长率。
可将该递归函数转化为递归树，则共有 $\log_3 n$ 层，第 k 层的代价为 $3^k\frac{n}{3^k}\log^3\frac{n}{3^k}=n\log^3\frac{n}{3^k}(0\leq k<\log_3 n)$，
所以总代价为 $\sum_{k=0}^{\log_3 n-1}(n\log^3\frac{n}{3^k})=n\sum_{k=0}^{\log_2 n-1}((\log n-k\log 3)^3)$
$=n(\log^4 n-\frac{3\log 3\log^3 n(\log n-1)}{2}+\frac{\log^2 3\log^2 n(\log n-1)(2\log n-1)}{2}-\frac{\log^3 3\log^2 n(\log n-1)^2}{4})$
$=\frac{n}{4}\log^2 n[(\log\frac{4}{3})\log n+\log 3][(\log^2 3-\log\frac{9}{4})\log n-\log^2 3+2\log 3]$
所以 $T(n)=\Theta(n\log^4 n)$。

7) $T(n)=2T(n/2)+cn^2$
$a=2, b=2, f(n)=cn^2$
$\exists\varepsilon>0$，令 $\varepsilon\in(0,1)$，使得：
$\lim_{n\to\infty}\frac{f(n)}{n^{\log_b a+\varepsilon}}=\lim_{n\to\infty}\frac{cn^2}{n^{1+\varepsilon}}=\lim_{n\to\infty}\frac{cn}{n^{\varepsilon}}>0, f(n)=\Omega(n^{\log_b a+\varepsilon})$，
又因为 $\exists k<1$，令 $k\in(\frac{1}{2},1)$，对所有充分大的 n，$af(\frac{n}{b})=\frac{cn^2}{2}\leq kf(n)$，
所以 $T(n)=\Theta(cn^2)=\Theta(n^2)$。

8) $T(n)=49T(n/25)+n^{3/2}\log n$
$a=49, b=25, f(n)=n^{3/2}\log n$
$\exists\varepsilon>0$，令 $\varepsilon\in(0,\frac{3}{2}-\log_5 7)$，使得：

$\lim_{n\to\infty}\frac{f(n)}{n^{\log_b a+\varepsilon}} = \lim_{n\to\infty}\frac{n^{3/2}\log n}{n^{\log_5 7+\varepsilon}} = \lim_{n\to\infty}\frac{n^{(\frac{3}{2}-\log_5 7)}\log n}{n^{\varepsilon}} > 0, f(n) = \Omega(n^{\log_b a+\varepsilon})$,
又因为 $\exists k < 1$, 令 $k \in (\frac{49}{125}, 1)$, 对所有充分大的 n, $af(\frac{n}{b}) = \frac{49}{125}n^{3/2}\log(n/25) \leq kf(n)$,
所以 $T(n) = \Theta(n^{\frac{3}{2}}\log n)$。

9) $T(n) = T(n-1) + 2$
无法使用主定理求渐近增长率。
可将该递归函数转化为递归树，则共有 $n-1$ 层，第 k 层的代价为 $2(0 \leq k < n-1)$,
所以总代价为 $\sum_{k=0}^{n-2}(2) = 2(n-1)$,
所以 $T(n) = \Theta(n)$。

10) $T(n) = T(n-1) + n^c$
无法使用主定理求渐近增长率。
可将该递归函数转化为递归树，则共有 $n-1$ 层，第 k 层的代价为 $(n-k)^c(0 \leq k < n-1)$,
所以总代价为 $\sum_{k=0}^{n-2}((n-k)^c)$,
所以 $T(n) = \Theta(n^{c+1})$。

11) $T(n) = T(n-1) + c^n$
无法使用主定理求渐近增长率。
可将该递归函数转化为递归树，则共有 $n-1$ 层，第 k 层的代价为 $c^{n-k}(0 \leq k < n-1)$,
所以当 $c = 1$ 时，总代价为 $n-1$，当 $c \neq 1$ 时，总代价为 $\sum_{k=0}^{n-2}(c^{n-k}) = \frac{c^2-c^{n+1}}{1-c}$,
所以当 $0 < c < 1$ 时，$T(n) = \Theta(1)$;
当 $c = 1$ 时，$T(n) = \Theta(n)$;
当 $c > 1$ 时，$T(n) = \Theta(c^n)$。

12) $T(n) = T(n-2) + 2n^3 - 3n^2 + 3n - 1, T(1) = 1, T(2) = 9$
可将该递归函数转化为递归树，则共有 $\lfloor\frac{n-1}{2}\rfloor$ 层，第 k 层的代价为 $2(n-2k)^3 - 3(n-2k)^2 + 3(n-2k) - 1(0 \leq k < \lfloor\frac{n-1}{2}\rfloor)$,
所以总代价为 $\sum_{k=0}^{\lfloor\frac{n-1}{2}\rfloor-1}(2(n-2k)^3 - 3(n-2k)^2 + 3(n-2k) - 1)$,
所以 $T(n) = \Theta(n^3\lfloor\frac{n-1}{2}\rfloor)$。

13) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

先证：$T(n)=O(n)$，即存在常数 $c_1>0$ 和 $n_1>0$，使得 $T(n)\le c_1 n$ 对所有 $n\ge n_1$ 恒成立。

i) 当 $n=1$ 时，$T(n)=T(1)=1$，$c_1\ge 1$ 时 $T(n)\le c_1 n$；

ii) 假设对于某常数 $c_1>0$，当 $n<k$ 时命题均成立，则当 $n=k$ 时，$T(k) = T(k/2)+T(k/4)+T(k/8)+k \le (c_1/2+c_1/4+c_1/8+1)k=(\frac{7c_1}{8}+1)k\le c_1 k($ 当 $c_1\ge 8$ 时 $)$；

iii) 因此对于任意常数 $c_1\ge 8$，命题均成立，即 $T(n)=O(n)(c_1\ge 8)$。

再证：$T(n)=\Omega(n)$，

$$\begin{aligned}
\lim_{n\to\infty}\frac{T(n)}{n} &= \lim_{n\to\infty}\frac{T(n/2)+T(n/4)+T(n/8)+n}{n}\\
&= \lim_{n\to\infty}\frac{T(n/2)}{n}+\lim_{n\to\infty}\frac{T(n/4)}{n}+\lim_{n\to\infty}\frac{T(n/8)}{n}+1\\
&> 0
\end{aligned}$$

所以 $T(n)=\Omega(n)$
所以 $T(n)=\Theta(n)$。

2.18

$T(n)=\sqrt{n}\cdot T(\sqrt{n})+O(n)$
无法使用主定理计算时间复杂度；
可将该递归函数转化为递归树。每层做的工作为 $n^{\sum_{i=1}^{k}(2^{-i})}\cdot n^{2^{-k}}=O(n)$，其中 k 为第几层，因此，现在的问题是递归树的深度。
在每个递归调用中将 n 取平方根，因此，经过 k 次迭代，得：$n^{2^{-k}}\ge 2$，则 $k\le \log\log n$，因此，递归树的深度为 $\log\log n$。
又因为在递归树的每一层上做的工作都为 $O(n)$，
所以 $T(n)=O(n\log\log n)$。

2.19

$a=2, b=2, f(n)=n\log n$

2.22

1) ALG1 和 ALG2 算法输出的都是数组 $A[1..n]$ 中的最小值。

2) ALG1：时间复杂度 $T(n) = T(n-1) + 1 = n - 1 = \Theta(n)$
ALG2：时间复杂度 $T(n) = 2T(n/2) + 1$
$a = 2, b = 2, f(n) = 1$,
$\exists \varepsilon > 0$，令 $\varepsilon \in (0,1)$，使得 $\lim_{n\to\infty} \frac{1}{n^{1-\varepsilon}} = 0 \iff f(n) = O(n^{1-\varepsilon})$，则 $T(n) = \Theta(n)$

2.24

1) MYSTERY:
结果：$\sum_{i=1}^{n-1}(\sum_{j=i+1}^{n}(\sum_{k=1}^{j}(1))) = \sum_{i=1}^{n-1} \frac{n^2+n-i^2-i}{2} = \frac{n(n+1)(n-1)}{3}$
最坏情况下运行时间：$O(n^3)$

2) PRESKY:
结果：$\sum_{i=1}^{n}(\sum_{j=1}^{i}(\sum_{k=j}^{i+j}(1))) = \frac{n(n+1)(n+2)}{3}$
最坏情况下运行时间：$O(n^3)$

3) PRESTIFEROUS:
结果：$\sum_{i=1}^{n}(\sum_{j=1}^{i}(\sum_{k=j}^{i+j}(\sum_{l=1}^{i+j-k}(1)))) = \frac{n(n+1)(n+2)(3n+1)}{24}$
最坏情况下运行时间：$O(n^4)$

4) CONUNDRUM:
结果:$\sum_{i=1}^{n}(\sum_{j=i+1}^{n}(\sum_{k=i+j-1}^{n}(1))) = \sum_{i=1}^{n}(\sum_{j=i+1}^{n}(\max(n+2-i-j,0))) = \sum_{i=1}^{n}(\sum_{j=i+1}^{n+1-i}(n+2-i-j,0)) = \frac{n(n+2)(2n-1)}{24}$
最坏情况下运行时间：$O(n^3)$

3.2

1) 归纳证明如下：

   i) 当 $i = n$ 时，$j : 1 \to n-1$，$n-1$ 次比较后易知 $A[n]$ 为数组最大值，$A[i..n]$ 即 $A[n]$ 显然有序；

ii) 假设对于第 $n-i+1$ 次循环，$j:1\rightarrow i-1$，$i-1$ 次比较后 $A[i..n]$ 有序且 $A[i]\leq A[i+1]\leq ...\leq A[n]$，同时对任意 m，有 $A[m]\leq A[i](1\leq m\text{e}i-1)$，则对第 $n-i+2$ 次循环，$j:1\rightarrow i-2$，$i-2$ 次比较后易知 $A[i-1]$ 为前 $i-1$ 个数的最大值，即对任意 $m(1\leq m\leq i-2)$，$A[m]\leq A[i-1]$，且 $A[i-1]\leq A[i]\leq ...\leq A[n]$，即 $A[i-1..n]$ 有序；

iii) 综上所述，当 $i=2$，即循环结束时，$A[2,3,..,n]$ 有序且 $A[1]\leq A[2]$，即 $A[1,2,..,n]$ 也是有序的，故已经完成排序，算法正确。

2) 最坏情况：当 $A[1,..n]$ 按从大到小排序时，需要 $\sum_{i=n}^{2}(i-1)=\frac{n(n-1)}{2}$ 次比较，故最坏情况时间复杂度为 $\Theta(n^2)$；
平均情况：也需要 $\sum_{i=n}^{2}(i-1)=\frac{n(n-1)}{2}$ 次比较，故平均情况时间复杂度为 $\Theta(n^2)$。

3) 算法改进后：
最坏情况任然需要经过所有循环，即需要 $\frac{n(n-1)}{2}$ 次比较，故最坏情况时间复杂度仍为 $\Theta(n^2)$；
平均情况：时间复杂度仍为 $\Theta(n^2)$，归纳证明：

i) 设有 k 个数的数组 $A[1..k]$ 的某一种排列用改进后的算法排序需要的比较次数为 $a_m(m=1,..,k!)$，则 k 个数的数组平均时间复杂度为 $A(k)=\sum_{m=1}^{k!}p_k a_m$，其中，$p_k$ 为初始序列为 $k!$ 种排序中的某一种的概率，$p_k=\frac{1}{k!}$；

ii) 假设 k 个数的数组排序的平均时间复杂度为 $A(k)=\Theta(k^2)$，则将其表示为 $A(k)=ak^2+bk+c(a>0)$。将有 $k+1$ 个数的数组排序可看做在原来 k 个数的数组中插入了一个元素，则其排序的平均时间复杂度为 $\sum_{m=1}^{!}p_k(a_m+\Delta a_m)$，其中 Δa_m 为将第 $k+1$ 个数插入原来第 m 种序列后平均增加的比较次数，显然，$\Delta a_m\leq\frac{k(k+1)}{2}-\frac{k(k-1)}{2}=k$，则 $A(k+1)=\sum_{m=1}^{!}p_k(a_m+\Delta a_m)\leq\sum_{m=1}^{!}p_k(a_m+k)=ak^2+b(k+1)+c=\Theta((k+1)^2)$；

iii) 综上所述，改进算法后平均情况时间复杂度仍为 $\Theta((k+1)^2)$。

3.5

将第 5 行的 "$j := j-1$" 改为 "$j := P[j]$" 即可，伪代码如下：

Algorithm 3: PREVIOUS-LARGER(A[1..n])

```
1 for i := 1 to n do
2     j := i − 1;
3     while j > 0 and A[j] ⩽ A[i] do
4         j := P[j];
5     end
6     P[i] := j;
7 end
8 return P[1..n];
```

证明：

i) 当 $i=1$ 时，while 循环结束后 $j=0$，正确；

ii) 假设在前 k 次循环后，$P[1..k]$ 都是 $A[1..k]$ 向左第一个大于 $A[1..k]$ 的元素的下标，则当第 $k+1$ 次循环时，对于每一个 $A[j](j<i)$，向左第一个大于它的元素为 $A[P[j]]$，所以在 $A[P[j]]$ 和 $A[j]$ 之间的元素 $A[k]$ 一定满足：$A[k] \leqslant A[j]$，而由 while 循环中判断条件可知进入 while 循环的 j 一定满足 $A[j] \leqslant A[i]$，所以 $A[k] \leqslant A[i]$，故无需将 $A[k]$ 和 $A[i]$ 比较，所以第 $k+1$ 次循环结束后，$A[j]$ 时 $A[k+1]$ 向左第一个大于它的元素，即 $P[k+1]$ 正确；

iii) 综上所述，该算法正确。

时间复杂度：
假设对于某一元素 $A[i](1<i<n)$ 需要进行 k_i 次比较，则：
若 $A[i-1] > A[i]$，则 $k_i = 1$；
若 $A[i-1] \leq A[i]$< 则 $P[i]$ 一定在 $P[1..i-1]$ 之中，则 $k_i > 1$，且 $P[i] \leq P[i-k_i+1]$。
则对于 $A[i+1]$，同理有：$k_{i+1} = 1(A[i] > A[i+1])$ 或 $k_{i+1} > 1(A[i] \leq A[i+1])$，且有关系：$k_{i+1} \leq i-k_i+2$，即 $k_i + k_{i+1} \leq i+2$。
同理可得：$k_{i+m} = 1$ 或 $1 < k_{i+m} \leq i-(k_i+...+k_{i+m-1})$ 即 $k_i+...+k_{i+m} \leq$

$i+m+1$，则 $k_i+...+k_n \le n+1$，所以 $k_1+...+k_n \le n+1$；
所以时间复杂度为 $\Theta(n)$。

3.6

1) 算法伪代码如下：

```
Input: A[1..n], k
for i := k to 1 do
    for j := i to i + n − k − 1 do
        SWAP(A[j], A[j + 1]);
    end
end
```

2) 算法伪代码如下：

```
Input: A[1..n], k
B[1..n];
for i := 1 to k do
    B[(n − k + i)] = A[i];
end
for i := k + 1 to n do
    B[(i − k)] = A[i];
end
```

3) 算法伪代码如下：

```
Algorithm 4: MYSWAP
Input: A[1..n], k
if k = 0 or k = n then
    return;
end
if k ≤ n/2 then
    for i := 1 to k do
        SWAP(A[i], A[n − k + i]);
        return MYSWAP(A[1..n − k], k);
    end
end
for i := 1 to n − k do
    SWAP(A[i], A[n − k + i]);
    return MYSWAP(A[n − k + 1..n], 2k − n);
end
```

3.8

1) 在一群共 n 个人中，至多只可能有 1 个名人，因为可能没有名人，而如果有名人，则其不关注其他任何人，并且被其他所有人关注，则其他任何人都不可能做到“不关注其他任何人，并且被其他所有人关注”这一成为名人的条件，故至多只有 1 个名人；

2) 算法伪代码如下：

```
Input: A[1..n]
famous := true;
pos[1..n] := {true};
for i := 1 to n do
    if pos[i] then
        famous := true;
        for j := 1 to n do
            if j ≠ i then
                if A[i] does not follow j then
                    pos[j] := false;
                else
                    famous := false;
                end
            end
        end
        if famous = true then
            for k := 1 to n do
                if k ≠ i and A[k] does not follow i then
                    return 0;
                end
            end
            return i;
        end
    end
end
return 0;
```

3.9

1) 算法伪代码如下:

Algorithm 5: MAX-SUBARRAY1($S[1..n]$)

```
Input: S[1..n]
maxs := 0; temp = 0;
for i := 1 to n do
    for j := i + 1 to n do
        for k := i to j do
            temp := temp + S[k];
        end
        if temp > maxs then
            maxs := temp;
        end
        temp := 0;
    end
end
return maxs;
```

2) 算法伪代码如下:

Algorithm 6: MAX-SUBARRAY2($S[1..n]$)

```
Input: S[1..n]
maxs := 0; temp = 0;
for i := 1 to n do
    for j := i + 1 to n do
        temp := temp + S[j];
        if temp > maxs then
            maxs := temp;
        end
    end
    temp := 0;
end
return maxs;
```

3) 算法伪代码如下：

Algorithm 7: MAX-SUBARRAY3($S[1..n]$)

```
Input: S[1..n]
if n ≤ 1 then
    return S[1];
end
midi = (n + 1)/2;
maxl := MAX − SUBARRAY3(S[1..midi − 1]);
maxr := MAX − SUBARRAY3(S[midi + 1..n]);
midl := 0; midr := 0; temp := 0;
for i := midi to 1 do
    temp := temp + S[i];
    if temp > midl then
        midl := temp;
    end
end
temp := 0;
for i := midi + 1 to n do
    temp := temp + S[i];
    if temp > midl then
        midl := temp;
    end
end
midmax := midl + midr;
return max(maxl, maxr, midmax);
```

4) 算法伪代码如下：

Algorithm 8: MAX-SUBARRAY4($S[1..n]$)

```
Input: S[1..n]
maxs := 0; temp = 0;
for i := 1 to n do
    temp := temp + S[i];
    if temp > maxs then
        maxs := temp;
    end
    if temp < 0 then
        temp := 0;
    end
end
return maxs;
```

5) 算法伪代码如下:

Algorithm 9: MAX-SUBARRAY5($S[1..n]$)

```
Input: S[1..n]
maxs := 0; temp = 0;
for i := 1 to n do
    temp := max(temp, temp + S[i]);
    maxs := max(temp, maxs);
end
return maxs;
```