

Homework1

231220030 邢俊书

2025 年 3 月 18 日

Solution 1.2.

(1) 代码如下:

```
int getMedian(int a, int b, int c) {  
    if (a > b) {  
        if (b > c) return b;  
        else {  
            if (a > c) return c;  
            else return a;  
        }  
    }  
    else {  
        if (b > c) {  
            if (a < c) return c;  
            else return a;  
        }  
        else return c;  
    }  
}
```

(2) 最坏情况下，需要比较 3 次；

平均情况下，需要比较 $\frac{1}{3} * 2 + \frac{2}{3} * 3 = \frac{8}{3}$ 次。

(3) 在最坏情况下，要找出三个不同整数的中位数至少需要进行 3 次比较。显然，三个不同整数的大小关系共有 $3! = 6$ 种不同情况，仅进行 2 次比较，只能得到 4 种情况；而进行 3 次比较，则可以得到 8 种情况，故至少需要进行 3 次比较。

Solution 1.3.

(1) 取 $U = \{1, 2, 3, 4, 5\}$, $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 3, 4\}$, $S_3 = \{4, 5\}$ ，题述算法得到的最小覆盖为 $\{S_1, S_2, S_3\}$ ，实际上最小覆盖为 $\{S_1, S_2\}$ 。

(2) 每次都从子集中选取覆盖最多未被覆盖元素的集合，直至所有元素均被覆盖，若遍历完所有子集均未覆盖所有元素，则不存在覆盖。

算法至多遍历完所有子集，所以总能终止；

不存在覆盖的情况下，算法正确性显然，故以下讨论均基于至少存在一个可行解的情况。

当 $|U| = 1$ ，则所有子集中至少有一个子集包含这个唯一元素，算法会选择该子集，正确性显然；

假设当 $|U| \leq n$ 时，算法总能找到一个可行解，下证当 $|U| = n + 1$ 时，算法可以找到一个可行解：

根据算法描述，算法首先会选取一个覆盖了最多未被覆盖元素的子集，假设该子集大小为 m ，则问题被转化为规模为 $n + 1 - m \leq n$ 的子问题，根据归纳假设，算法总能给出该子问题的一个可行解，所以当 $|U| = n + 1$ 时，算法可以找到一个可行解。

根据强数学归纳法，当输入存在可行解时，算法总能找到一个可行解。

综上，算法正确性得证。

(3) 不能保证总是得出最小覆盖。

例如取 $U = \{1, 2, 3, 4, 5, 6\}$, $S_1 = \{1, 2, 3, 4\}$, $S_2 = \{2, 3, 5\}$, $S_3 = \{1, 4, 6\}$ ，根据 (2) 中算法得到的覆盖为 $\{S_1, S_2, S_3\}$ ，实际上最小覆盖为 $\{S_2, S_3\}$ 。

Solution 1.7.

不难发现 $P(x) = (\dots((a_n x + a_{n-1})x + a_{n-2})x + \dots + a_1)x + a_0$ 。

记已循环次数为 k ，当 $k = 1$ 时， $p = a_n x + a_{n-1}$ ；

假设当 $k = m$ 时，算法可以正确计算由内至外第 m 个括号内的值，下证当 $k = m + 1$ 时，算法可以正确计算出由内至外第 $m + 1$ 个括号内的值：

由归纳假设，第 $m + 1$ 次循环前， $p = a_n x^m + a_{n-1} x^{m-1} + \dots + a_{n-m}$ ，则第 $m + 1$ 次循环后， $p = a_n x^{m+1} + a_{n-1} x^m + \dots + a_{n-m-1}$ 。

根据数学归纳法，算法可以正确计算出多项式的值。

Solution 1.8.

(1) $c = 2$ 是 (2) 的特殊情况，由下可知正确性显然。

(2) 不妨考虑对 z 进行归纳。

当 $z = 0$ 时，根据算法有 $\text{INT-MULT}(y, 0) = 0$ ，算法可以正确计算出两数乘积；

当 $0 < z < c$ 时， $\text{INT-MULT}(y, z) = \text{INT-MULT}(cy, 0) + yz = yz$ ，算法可以正确计算出两数乘积；

假设当 $c \leq z \leq n$ 时，算法可以正确计算出两数乘积，下证当 $z = n + 1$ 时，算法可以正确计算出两数乘积：

记 $n + 1 = sc + t, 0 \leq t < c$ ，由归纳假设可知，当 $z \leq n$ 时，算法总能正确计算出两数乘积，则有

$$\begin{aligned} \text{INT-MULT}(y, n + 1) &= \text{INT-MULT}(cy, \lfloor \frac{n + 1}{c} \rfloor) + y * ((n + 1) \bmod c) \\ &= \text{INT-MULT}(cy, s) + yt \\ &= y(sc + t) \\ &= y(n + 1) \end{aligned}$$

根据强数学归纳法，算法正确总能计算出两数乘积，算法正确。

Solution 1.9.

平均时间复杂度为

$$\begin{aligned}
 A(n) &= \sum_{I \in D_n} Pr(I) * f(I) \\
 &= \frac{1}{4} * 10 + \frac{1}{2} * 20 + \frac{1}{8} * 30 + \frac{1}{8} * n \\
 &= \frac{n}{8} + 16.25
 \end{aligned}$$

Solution 1.10.

(1) UNIQUE 算法用于判断数组中是否有两元素相等，最坏情况即不存在相等元素，需要遍历完所有 i, j ，时间复杂度为 $O(n^2)$ 。

(2) 平均时间复杂度为 $A(n) = \frac{1+2+\dots+\frac{n(n-1)}{2}}{\frac{n(n-1)}{2}} = \frac{n^2-n+2}{4}$ 。

(3) 数组中任意两个元素相等的概率为 $Pr(A[i] == A[j]) = k * (\frac{1}{k})^2 = \frac{1}{k}$ ，不妨假设每次比较均消耗 1 个代价，则消耗了 i 个代价后算法终止的概率为 $(1 - \frac{1}{k})^{i-1} * \frac{1}{k}$ ，不难发现算法停止时消耗的代价 $X \sim g(\frac{1}{k})$ ， $EX = k$ ，若 $k < \frac{n(n-1)}{2}$ ，则时间复杂度近似于 $O(k)$ ；否则，时间复杂度近似于 $O(n^2)$ 。

Solution 2.2.

对于任意 $2^k \leq n \leq 2^{k+1} - 1, k \geq 0$ ， $\lceil \log(n+1) \rceil = k+1 = \lfloor \log n \rfloor + 1$ 。

Solution 2.5.

(1) 2-tree 是 (2) 的特殊情况，由下可知等式成立。

(2) 假设总结点数为 n ，则有

$$\begin{cases} n = n_0 + n_1 + n_2 \\ n - 1 = n_1 + 2n_2 \end{cases}$$

消去 n_1 即得： $n_0 = n_2 + 1$ 。

Solution 2.7.

为了方便讨论,不妨先约定三个渐进增长率 $f(n)$ 、 $g(n)$ 、 $h(n)$ 。

(1) O : $f(n) = O(g(n))$ 、 $g(n) = O(h(n))$, 根据定义, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c_1 < \infty$ 且 $\lim_{n \rightarrow \infty} \frac{g(n)}{h(n)} = c_2 < \infty$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{h(n)} = c_1 c_2 < \infty$, $f(n) = O(h(n))$, O 满足传递性。

Ω : $f(n) = \Omega(g(n))$ 、 $g(n) = \Omega(h(n))$, 根据定义, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c_1 > 0$ 且 $\lim_{n \rightarrow \infty} \frac{g(n)}{h(n)} = c_2 > 0$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{h(n)} = c_1 c_2 > 0$, $f(n) = \Omega(h(n))$, Ω 满足传递性。

Θ : $f(n) = \Theta(g(n))$ 、 $g(n) = \Theta(h(n))$, 根据定义, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c_1, 0 < c_1 < \infty$ 且 $\lim_{n \rightarrow \infty} \frac{g(n)}{h(n)} = c_2, 0 < c_2 < \infty$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{h(n)} = c_1 c_2, 0 < c_1 c_2 < \infty$, $f(n) = \Theta(h(n))$, Θ 满足传递性。

o : $f(n) = o(g(n))$ 、 $g(n) = o(h(n))$, 根据定义, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ 且 $\lim_{n \rightarrow \infty} \frac{g(n)}{h(n)} = 0$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{h(n)} = 0$, $f(n) = o(h(n))$, o 满足传递性。

ω : $f(n) = \omega(g(n))$ 、 $g(n) = \omega(h(n))$, 根据定义, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ 且 $\lim_{n \rightarrow \infty} \frac{g(n)}{h(n)} = \infty$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{h(n)} = \infty$, $f(n) = \omega(h(n))$, ω 满足传递性。

(2) O : $\lim_{n \rightarrow \infty} \frac{f(n)}{f(n)} = 1 < \infty$, $f(n) = O(f(n))$, O 满足自反性。

Ω : $\lim_{n \rightarrow \infty} \frac{f(n)}{f(n)} = 1 > 0$, $f(n) = \Omega(f(n))$, Ω 满足自反性。

Θ : $\lim_{n \rightarrow \infty} \frac{f(n)}{f(n)} = 1, 0 < 1 < \infty$, $f(n) = \Theta(f(n))$, Θ 满足自反性。

(3) 由 (1)、(2) 可知: Θ 满足自反性与传递性, 故仅需再证其满足对称性即可。 $f(n) = \Theta(g(n))$, 根据定义, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c, 0 < c < \infty$, 则有 $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \frac{1}{c}, 0 < \frac{1}{c} < \infty$, $g(n) = \Theta(f(n))$, Θ 满足对称性, 综上, Θ 是一个等价关系。

(4) \Rightarrow : 因为 $f(n) = \Theta(g(n))$, 故 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c, 0 < c < \infty$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0 \wedge \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c < \infty$, 即 $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$;

\Leftarrow : 因为 $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$, 故 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c_1 > 0 \wedge \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c_2 < \infty$, 显然 $c_1 = c_2$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c, 0 < c < \infty$, 即 $f(n) = \Theta(g(n))$ 。

综上, 得证。

(5) \Rightarrow : 因为 $f = O(g)$, 故 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c < \infty$, 则有 $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \frac{1}{c} > 0$, $g = \Omega(f)$;

\Leftarrow : 因为 $g = \Omega(f)$, 故 $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = c > 0$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \frac{1}{c} < \infty$, $f = O(g)$ 。

综上, $f = O(g)$ iff $g = \Omega(f)$ 。

\Rightarrow : 因为 $f = o(g)$, 故 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$, 则有 $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \infty$, $g = \omega(f)$;

\Leftarrow : 因为 $g = \omega(f)$, 故 $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \infty$, 则有 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$, $f = o(g)$ 。

综上, $f = o(g)$ iff $g = \omega(f)$ 。

(6) $\forall f(n) \in o(g(n))$, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \neq \infty$, 故 $f(n) \notin \omega(g(n))$,

$o(g(n)) \cap \omega(g(n)) = \emptyset$;

$\forall f(n) \in \Theta(g(n))$, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$, 故 $f(n) \notin o(g(n))$,

$\Theta(g(n)) \cap o(g(n)) = \emptyset$;

$\forall f(n) \in \Theta(g(n))$, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c < \infty$, 故 $f(n) \notin \omega(g(n))$,

$\Theta(g(n)) \cap \omega(g(n)) = \emptyset$ 。

Solution 2.8.

(1) 渐进增长率从低到高依次为:

$$\log n < n < n \log n < n^2 = n^2 + \log n < n^3 < n - n^3 + 7n^5 < 2^n。$$

(2) 渐进增长率从低到高依次为:

$$\log \log n < \ln n = \log n < (\log n)^2 < \sqrt{n} < n < n \log n < n^{1+\epsilon} < n^2 = n^2 + \log n < n^3 < n - n^3 + 7n^5 < 2^{n-1} = 2^n < e^n < n!。$$

Solution 2.16.

(1) $f(n) = 1 = n^0 = O(n^{\log_3 2 - 0.5})$, $T(n) = \Theta(n^{\log_3 2})$ 。

(2) 为方便讨论, 不妨假设 $n = 2^k$, $k = \log n$, 则有

$$\begin{aligned}
 T(n) &= T(n/2) + c \log n \\
 &= T(n/4) + c \log n + c \log \frac{n}{2} \\
 &= \dots \\
 &= T(1) + c \log^2 n - c(\log 2 + \log 4 + \dots + \log(2^k)) \\
 &= \frac{c}{2} \log^2 n - \frac{c}{2} \log n + 1
 \end{aligned}$$

$$T(n) = \Theta(\log^2 n)。$$

(3) $f(n) = cn = \Omega(n^{\log_2 1 + 0.5})$, $f(n/2) = \frac{cn}{2} < \frac{2}{3}f(n)$, $T(n) = \Theta(n)$ 。

(4) $f(n) = cn = \Theta(n^{\log_2 2})$, $T(n) = \Theta(n \log n)$ 。

(5) 为方便讨论, 不妨假设 $n = 2^k$, $k = \log n$, 则有

$$\begin{aligned}
 T(n) &= 2T(n/2) + cn \log n \\
 &= 4T(n/4) + cn \log n + cn \log \frac{n}{2} \\
 &= \dots \\
 &= nT(1) + cn \log^2 n - cn(\log 2 + \log 4 + \dots + \log 2^k) \\
 &= n + \frac{c}{2} cn \log^2 n - \frac{c}{2} n \log n
 \end{aligned}$$

$$T(n) = \Theta(n \log^2 n)。$$

(6) 为方便讨论, 不妨假设 $n = 3^k$, $k = \log_3 n$, 则有

$$\begin{aligned}
 T(n) &= 3T(n/3) + n \log^3 n \\
 &= 9T(n/9) + n \log^3 n + n \log^3 \frac{n}{3} \\
 &= \dots \\
 &= nT(1) + \sum_{i=0}^k n \log^3 \frac{n}{3^i}
 \end{aligned}$$

$$T(n) = \Theta(n \log^4 n)。$$

$$(7) f(n) = cn^2 = \Omega(n^{\log_2 2 + 0.5}), \quad 2f(n/2) = \frac{cn^2}{2} < \frac{2}{3}f(n), \quad T(n) = \Theta(n^2)。$$

$$(8) f(n) = n^{3/2} \log n = \Omega(n^{\log_5 7 + 0.2}), \quad 49T(n/25) < \frac{49}{125}n^{3/2} \log n = \frac{49}{125}f(n), \\ T(n) = \Theta(n^{3/2} \log n)。$$

$$(9) T(n) = T(n-1) + 2 = T(1) + 2n - 2 = 2n - 1, \quad T(n) = \Theta(n)。$$

$$(10) \text{不妨猜测 } \frac{1}{c+1}n^{c+1} \leq T(n) \leq n^{c+1}, \text{ 代入可得:}$$

$$T(n) = T(n-1) + n^c \leq (n-1)^{c+1} + n^c \leq (n-1)n^c + n^c = n^{c+1}$$

$$\begin{aligned} T(n) &= T(n-1) + n^c \\ &\geq \frac{1}{c+1}(n-1)^{c+1} + n^c \\ &= \frac{1}{c+1}(n^{c+1} - \binom{c+1}{1}n^c + \binom{c+1}{2}n^{c-1} + \cdots + (-1)^{c+1}) + n^c \\ &> \frac{1}{c+1}n^{c+1} \end{aligned}$$

猜想成立，故 $T(n) = \Theta(n^{c+1})$ 。

$$(11) T(n) = T(n-1) + c^n = \sum_{i=2}^n c^i + 1, c > 1, \quad T(n) = \Theta(c^n)。$$

$$(12) \text{不妨猜测 } \frac{1}{4}n^4 \leq T(n) \leq n^4, \text{ 代入可得:}$$

$$\begin{aligned} T(n) &= T(n-2) + 2n^3 - 3n^2 + 3n - 1 \\ &\leq (n-2)^4 + 2n^3 - 3n^2 + 3n - 1 \\ &= n^4 - 6n^3 + 21n^2 - 29n + 16 \\ &< n^4 \end{aligned}$$

$$\begin{aligned} T(n) &= T(n-2) + 2n^3 - 3n^2 + 3n - 1 \\ &\geq \frac{1}{4}(n-2)^4 + 2n^3 - 3n^2 + 3n - 1 \\ &= \frac{1}{4}n^4 + 3n^2 - 5n + 2 \\ &> \frac{1}{4}n^4 \end{aligned}$$

猜想成立，故 $T(n) = \Theta(n^4)$ 。

(13) 不妨猜测 $4n \leq T(n) \leq 8n$ ，代入可得：

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n \leq 7n + n \leq 8n$$

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n \geq \frac{9}{2}n \leq 4n$$

猜想成立，故 $T(n) = \Theta(n)$ 。

Solution 2.18.

由题意得：

$$\begin{aligned} T(n) &= n^{\frac{1}{2}}T(n^{\frac{1}{2}}) + cn \\ &= n^{\frac{1}{2}+\frac{1}{4}}T(n^{\frac{1}{4}}) + 2cn \\ &= n^{\frac{1}{2}+\frac{1}{4}+\frac{1}{8}}T(n^{\frac{1}{8}}) + 3cn \\ &= \dots \\ &= n^{\sum_{i=1}^k \frac{1}{2^i}}T(n^{\frac{1}{2^k}}) + kcn \\ &= \Theta(kcn) \end{aligned}$$

显然， $n^{\frac{1}{2^k}} = C$ ，得： $k = \log(\log n - \log C)$ ，故 $T(n) = \Theta(n \log \log n)$ 。

Solution 2.19.

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

Solution 2.22.

(1) 两个算法的输出均为数组中最小元素。

(2) 两个算法的时间复杂度均为 $\Theta(n)$ 。

Solution 2.24.

MYSTERY:

$$\begin{aligned}
 r &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^j 1 \\
 &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n j \\
 &= \sum_{i=1}^{n-1} \frac{n^2 + n - i^2 - i}{2} \\
 &= \frac{(n-1)n(n+1)}{3}
 \end{aligned}$$

最坏情况下的运行时间为 $O(n^3)$ 。

PERSKY:

$$\begin{aligned}
 r &= \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^{i+j} 1 \\
 &= \sum_{i=1}^n \sum_{j=1}^i (i+1) \\
 &= \sum_{i=1}^n i(i+1) \\
 &= \frac{n(n+1)(n+2)}{3}
 \end{aligned}$$

最坏情况下的运行时间为 $O(n^3)$ 。

PRESTIFEROUS:

$$\begin{aligned}
 r &= \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^{i+j} \sum_{l=1}^{i+j-k} 1 \\
 &= \sum_{i=1}^n \sum_{j=1}^i \sum_{k=j}^{i+j} (i+j-k) \\
 &= \sum_{i=1}^n \sum_{j=1}^i \frac{i^2 + i}{2} \\
 &= \sum_{i=1}^n \frac{i^3 + i^2}{2} \\
 &= \frac{n(n+1)(3n^2 + 7n + 2)}{24}
 \end{aligned}$$

最坏情况下的运行时间为 $O(n^4)$ 。

CONUNDRUM: 当 n 为偶数时,

$$\begin{aligned}
 r &= \sum_{i=1}^{\frac{n}{2}} \sum_{j=i+1}^{n+1-i} \sum_{k=i+j-1}^n 1 \\
 &= \sum_{i=1}^{\frac{n}{2}} \sum_{j=i+1}^{n+1-i} (n - i - j + 2) \\
 &= \frac{1}{12}n^3 + \frac{1}{8}n^2 - \frac{1}{12}n
 \end{aligned}$$

当 n 为奇数时,

$$\begin{aligned}
 r &= \sum_{i=1}^{\frac{n-1}{2}} \sum_{j=i+1}^{n+1-i} \sum_{k=i+j-1}^n 1 \\
 &= \sum_{i=1}^{\frac{n-1}{2}} \sum_{j=i+1}^{n+1-i} (n - i - j + 2) \\
 &= \frac{1}{12}n^3 + \frac{1}{8}n^2 - \frac{1}{12}n - \frac{1}{8}
 \end{aligned}$$

最坏情况下的运行时间为 $O(n^3)$ 。

Solution 3.2.

(1) 首先对内层循环做证明，循环不变量：第 j 次循环开始前， $A[1 \dots j]$ 部分的最大值是 $A[j]$ 。

Initialize: $j = 1$ 的内层循环开始前， $A[1 \dots 1]$ 部分的最大值显然是 $A[1]$ 。

Maintenance: 假设 $j = k$ 的循环开始前，数组 $A[1 \dots k]$ 部分的最大值为 $A[k]$ 。在此次循环中，如果 $A[j] > A[j + 1]$ ，则会交换这两个元素，循环结束后有 $A[j + 1] = \max\{A[1 \dots j], A[j + 1]\} = \max\{A[1 \dots j + 1]\}$ ，满足循环不变式。

Termination: 当 $j = i - 1$ 次循环结束后，数组 $A[1 \dots i]$ 中的最大元素就是 $A[i]$ 。

接下来证明外层循环，循环不变量：第 i 次循环开始前，数组 $A[i + 1 \dots n]$ 部分是数组 $A[1 \dots n]$ 升序排列后的对应部分。

Initialize: 在 $i = n$ 的循环开始前， $A[n + 1 \dots n]$ 为空，满足。

Maintenance: 假设在 $i = k$ 的循环开始前，不变式满足。在此次循环中，由上述结论，内层循环会选出 $A[1 \dots k]$ 部分的最大值置于 $A[k]$ 处，故循环后满足循环不变式。

Termination: 在 $i = 2$ 的循环结束后，数组 $A[2 \dots n]$ 部分已经排好序，而显然 $A[1]$ 就是第 1 小的元素，因此 $A[1 \dots n]$ 均已排好序。

(2) 无论内层循环还是外层循环，均不会提前终止，总共需要比较 $\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$ ，故最坏情况与平均情况时间复杂度均为 $\Theta(n^2)$ 。

(3) 最坏情况下，输入为完全倒序的数组，无论是否记录每次循环最后交换位置，都会进行 $i - 1$ 次比较，故不会影响最坏情况；平均情况下，大致可以减少一半的时间复杂度，但是整体时间复杂度仍为 $\Theta(n^2)$ 。

Solution 3.5.

修改后代码如下：

```
void PREVIOUS_LARGER(int a[], int n) {
    memset(P, 0, n + 1); // a 下标从 1 开始
    stack<int> s;
    for (int i = 1; i <= n; i++) {
        while (!s.empty() && a[s.top()] <= a[i]) s.pop();
        if (!s.empty()) P[i] = s.top();
        else P[i] = 0;
        s.push(i);
    }
    return;
}
```

考虑如下循环不变式，每次循环开始前，栈中元素满足：1. 从底至顶依次递减；2. 栈顶元素为左侧最近的较大元素的索引。

Initialize：栈中没有元素，满足循环不变式；

Maintenance：假设第 i 轮循环开始之前满足循环不变式，由算法可知，循环开始后会弹出栈中所有不大于 $a[i]$ 的元素，最后将 $a[i]$ 入栈，由于循环开始前栈中元素自底至顶依次递减，故弹栈操作会在遇到第一个大于 $a[i]$ 的元素或栈空后停止，保证了栈中元素仍然自底至顶依次递减，且栈顶元素为左侧最近的较大元素的索引。

Termination：由循环不变式可知，栈中每个元素的下一个元素均是其左侧最近的较大元素的索引，因而每次循环后均能正确地为 $P[i]$ 赋值。

对于每个元素，至多入栈、出栈一次，故时间复杂度为 $\Theta(n)$ 。

Solution 3.6.

为了方便起见，不妨先约定 `swap()` 作为交换两个变量值的函数。

(1) 时间复杂度为 $O(n^2)$ ，空间复杂度为 $O(1)$ ：

```
void My_reverse(int a[], int n, int k) {
    for (int i = k + 1; i <= n; i++) {
        for (int j = i; j >= i - k + 1; j--) {
            swap(a[j - 1], a[j]);
        }
    }
    return;
}
```

(2) 时间复杂度为 $O(n)$ ，空间复杂度为 $O(n)$ ：

```
void My_reverse(int a[], int n, int k) {
    int b[n + 1]; // a 下标从 1 开始
    memset(b, 0, n + 1);
    for (int i = 1; i <= k; i++) {
        b[n - k + i] = a[i];
    }
    for (int i = k + 1; i <= n; i++) {
        b[i - k] = a[i];
    }
    for (int i = 1; i <= n; i++) {
        a[i] = b[i];
    }
    return;
}
```

(3) 时间复杂度为 $O(n)$, 空间复杂度为 $O(1)$:

```
void My_reverse(int a[], int n, int k) {
    int b[n + 1]; // a 下标从 1 开始
    memset(b, 0, n + 1);
    for (int i = 1; i <= k / 2; i++) {
        swap(a[i], a[k - i + 1]);
    }
    for (int i = k + 1; i <= (k + 1 + n) / 2; i++) {
        swap(a[i], a[n - i + k + 1]);
    }
    for (int i = 1; i <= n / 2; i++) {
        swap(a[i], a[n - i + 1]);
    }
    return;
}
```

Solution 3.8.

(1) 0 个或 1 个。假设存在大于等于 2 个名人, 任取其中两人 A、B, 由名人的定义可知: A 不认识 B, 与 B 是名人矛盾。

(2) 定义集合 S 为所有名人候选者名单, 每次从中任取两人 A、B, 询问 A 是否认识 B。若不认识, 则 B 不可能是名人; 若认识, 则 A 不可能是名人。每次询问均可以从集合 S 中删除一人, 经过 $n - 1$ 次询问, 即可得出名人, 时间复杂度为 $\Theta(n)$ 。

Solution 3.9.

(1) $O(n^3)$:

```
int MaxSubarr(int S[], int n) {
    int max = INT_MIN;
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int sum = 0;
            for (int k = i; k <= j; k++) sum += S[k];
            if (sum >= max) max = sum;
        }
    }
    return max;
}
```

(2) $O(n^2)$:

```
int MaxSubarr(int S[], int n) {
    int max = INT_MIN;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = i; j < n; j++) {
            sum += S[j];
            if (sum >= max) max = sum;
        }
    }
    return max;
}
```


(3) 将整个数组分为左右两部分，递归求解每个部分的最大和连续子序列，再以 $O(n)$ 的时间代价求解横跨左右两部分的最大和连续子序列。

(4) $O(n)$:

```
int MaxSubarr(int S[], int n) {  
    int sum = 0, max = INT_MIN;  
    for (int i = 0; i < n; i++) {  
        sum += S[i];  
        if (sum < 0) sum = 0;  
        if (sum >= max) max = sum;  
    }  
    return max;  
}
```

(5) 动态规划:

```
int MaxSubarr(int S[], int n) {  
    int ans = 0, sum = 0;  
    for (int i = 0; i < n; i++) {  
        sum = max(sum + S[i], S[i]);  
        ans = max(ans, sum);  
    }  
    return ans;  
}
```