算法设计与分析阶段一作业

唐吉宏　161240057

March 12, 2019

1　作业内容

- 1.1, 1.2, 1.3, 1.5, 1.6, 1.7
- 2.2, 2.3, 2.5, 2.6, 2.7, 2.15, 2.17, 2.18, 2.20
- 3.2, 3.4, 3.5, 3.6, 3.7

2　作业解答

Problem 1.1

解答. 各小题的解答如下:

1) 设计的算法命名为 **Sorting Three Different Integers** 并引为算法 1

2) 在最坏情况下, 即输入的三个整数开始的排列顺序为逆顺序排列（针对算法 1为降序排列）的情况下，算法 1所需比较次数为 3，分别为 $Array[1]$ 与 $Array[2]$、$Array[2]$ 与 $Array[3]$、新 $Array[1]$ 与新 $Array[2]$ 之间的比较。

   在平均情况下的比较次数计算可由以下步骤组成，首先对于各种等概率的可能输入情况进行穷举，结果如表 1。其中，用简单数字（1，2，3）之间的大小关系代表输入数据之间的大小关系。接着如下公式

$$A(n) = \sum_{I \in D(n)} Pr(I) \cdot f(I)$$

   可以计算得出平均情况下算法 1的比较次数为 $\frac{8}{3}$ 次。

Algorithm 1: Sorting Three Different Integers

Data: Three different integers stored as Array[1,2,3]

Result: Three different integers in increasing order

```
Initialization and data input;
if Array[1] > Array[2] then
    Temp ← Array[1] ;
    Array[1] ← Array[2] ;
    Array[2] ← Temp ;
end
if Array[2] > Array[3] then
    Temp ← Array[2] ;
    Array[2] ← Array[3] ;
    Array[3] ← Temp ;
    if Array[1] > Array[2] then
        Temp ← Array[1] ;
        Array[1] ← Array[2] ;
        Array[2] ← Temp ;
    end
end
return Array;
```

Table 1: 算法 1平均比较次数情况穷举表

编号	输入情况 $D(n)$	比较次数 $f(I)$	存在概率 $Pr(I)$
1	$1,2,3$	2	$\frac{1}{6}$
2	$1,3,2$	3	$\frac{1}{6}$
3	$2,1,3$	2	$\frac{1}{6}$
4	$2,3,1$	3	$\frac{1}{6}$
5	$3,1,2$	3	$\frac{1}{6}$
6	$3,2,1$	3	$\frac{1}{6}$

3) 在最坏情况下将三个不同的整数排序至少需要 3 次比较。

□

Problem 1.2

解答. 各小题的解答如下:

1) 设计的算法命名为 **Finding the Median among Three Different Integers** 并引为算法 2

Algorithm 2: Finding the Median among Three Different Integers

Data: Three different integers stored as Array[1,2,3]

Result: The median among the three integers

```
Initialization and data input;
if Array[1] > Array[2] then
    if Array[1] < Array[3] then
        return Array[1];
    else
        if Array[2] > Array[3] then
            return Array[2];
        else
            return Array[3];
        end
    end
else
    if Array[2] < Array[3] then
        return Array[2];
    else
        if Array[1] > Array[3] then
            return Array[1]
        else
            return Array[3]
        end
    end
end
```

2) 计算最坏情况与平均情况下的比较次数可以由以下步骤展开，首先对于各种等概率的可能输入情况进行穷举，结果如表 2。其中，用简单数字（1，2，3）之间的大小关系代表输入数据之间的大小关系。最坏情况下的比较次数可以由公式

$$W(n) = \max_{I \in D_n} f(I)$$

可得最坏情况下的比较次数为 3 次，由公式

$$A(n) = \sum_{I \in D(n)} Pr(I) \cdot f(I)$$

可以计算得出平均情况下算法 2的比较次数为 $\frac{8}{3}$ 次。

Table 2: 算法 1平均比较次数情况穷举表

编号	输入情况 $D(n)$	比较次数 $f(I)$	存在概率 $Pr(I)$
1	$1,2,3$	2	$\frac{1}{6}$
2	$1,3,2$	3	$\frac{1}{6}$
3	$2,1,3$	2	$\frac{1}{6}$
4	$2,3,1$	3	$\frac{1}{6}$
5	$3,1,2$	3	$\frac{1}{6}$
6	$3,2,1$	3	$\frac{1}{6}$

3) 在最坏情况下找出 3 个不同整数的中位数至少需要进行 3 次比较。

□

Problem 1.3

解答. 1) 构建使题中所述算法失败的例子如下:

全集为 $U = \{1,2,3,4,5,6,7\}$; 集合族为 $S = \{S_1, S_2, S_3, S_4, S_5\}$; 其中 $S_1 = \{1,2,3,4\}$, $S_2 = \{3,4,5,6\}$, $S_3 = \{4,5,6,7\}$, $S_4 = \{5,6,7\}$, $S_5 = \{1,4\}$。则根据所给算法得到的最小覆盖为 $\{S_1, S_2, S_3\}$, 但真实的最小覆盖可以为 $\{S_1, S_4\}$。

2) 设计的算法可以描述如下: 首先选择 S 中满足 $S_1 = 1,2,3,4$ 的最大的集合 S_i, 并从全集中将 S_i 中所有的元素删除; 然后从 S 中剩余的集合中挑选最大的满足相同条件的 S_i 并且做相同操作，重复上述过程知道全集中的所有元素都被覆盖。

3) 不能保证总是得出最小覆盖。反例如下:

全集为 $U = \{1,2,3,4,5,6,7\}$; 集合族为 $S = \{S_1, S_2, S_3, S_4, S_5\}$; 其中 $S_1 = \{1,2,3\}$, $S_2 = \{4,5,\}$, $S_3 = \{6,7\}$, $S_4 = \{1,2\}$, $S_5 = \{3,4,5,6,7\}$。则根据所给算法得到的最小覆盖为 $\{S_1, S_2, S_3\}$, 但真实的最小覆盖可以为 $\{S_4, S_5\}$。

□

Problem 1.5

证明. 利用数学归纳法证明 HORNER 算法的正确性如下：

设命题 Q 为: 在集合 **N** 上，HORNER 算法对所给输入 $A[0...n], x$ 可以给出正确的输出。对参数 n 进行如下归纳证明。

Step One：

证明基础情况 $Q(0)$ 的正确性，当 $n = 0$ 时，HORNER$(A[0], x)$ 总是返回 $a(0)$，而 $P(x)\big|_{n=0} = a(0)$，所以 $Q(0)$ 为 **TRUE**。

Step Two:

归纳假设当 $n = k$ 时，$Q(k)$ 为 **TRUE**，即 HORNER$(A[0...k], x)$= $P(x)\big|_{n=k}$，则 HORNER$(A[0...k+1], x)$ 的输出值 p_{k+1} 满足：

$$p_{k+1} = p_k + a_{k+1}x^{k+1} = P(x)\big|_{n=k} + a_{k+1}x^{k+1} = P(x)\big|_{n=k+1}$$

即

$$\forall k \in N, Q(k) \to Q(k+1)$$

根据步骤一二可知，对于所有的自然数 n，$Q(n)$ 都为正确的，即算法正确性得证。 □

Problem 1.6

证明. 利用数学归纳法分别证明不同情况下 INT-MULT 算法的正确性如下：

1) $c = 2$ 时，设命题 P 为：在集合 **N** 上，INT-MULT 算法对所给输入 y, z 可以给出正确的输出。对参数 z 进行如下归纳证明。

   Step One:

   证明基础情况 $P(0)$ 的正确性，当 $z = 0$ 时，INT-MULT(y, z) 总是返回 0，这一结果符合我们对于整数相乘的定义，所以 $P(0)$ 为 **TRUE**。

   Step Two:

   归纳假设当 $z \le k$ 时，命题 P 总是正确的，即 $\forall z \le k$, $P(z)$ 为 **TRUE**。针对 $P(k+1)$ 做以下证明：根据算法实现，INT-MULT$(y, k+1)$ 的输出结果为 INT-MULT$(2y, \lfloor\frac{k+1}{2}\rfloor)$ $+y\cdot(k+1 \mod 2)$。根据归纳假设，由于 $\lfloor\frac{k+1}{2}\rfloor \le k$，输出的前半部分必能正确整数相乘，同时针对 $k+1$ 是否为偶数的两种情况下的后半部分输出能够保证该输出可以正确表示 $y, k+1$ 两整数相乘。所以，$P(k+1)$ 为 **TRUE** 得证。

   根据步骤一二可知，对于所有的自然数 z，$P(z)$ 都为正确的，即算法正确性得证。

2) c 为任意一个不小于 2 的常数时，设命题 P 为：在集合 **N** 上，INT-MULT 算法对所给输入 y, z 可以给出正确的输出。对参数 z 进行如下归纳证明。

Step One:

证明基础情况 $P(0)$ 的正确性，当 $z = 0$ 时，INT-MULT(y, z) 总是返回 0，这一结果符合我们对于整数相乘的定义，所以 $P(0)$ 为 **TRUE**。

Step Two:

归纳假设当 $z \leq k$ 时，命题 P 总是正确的，即 $\forall z \leq k$, $P(z)$ 为 **TRUE**。针对 $P(k+1)$ 做以下证明：根据算法实现，INT-MULT$(y, k+1)$ 的输出结果为 INT-MULT$(cy, \lfloor \frac{k+1}{c} \rfloor)$ $+y \cdot (k+1 \mod c)$。根据归纳假设，由于 $\lfloor \frac{k+1}{c} \rfloor \leq k$，输出的前半部分必能正确整数相乘，同时由下式

$$\lfloor \frac{k+1}{c} \rfloor = \frac{k+1}{c} - \frac{(k+1) \mod c}{c}$$

可得

$$cy \cdot \lfloor \frac{k+1}{c} \rfloor + cy \cdot \frac{(k+1) \mod c}{c} = y \cdot (k+1)$$

所以，$P(k+1)$ 为 **TRUE** 得证。

根据步骤一二可知，对于所有的自然数 z，$P(z)$ 都为正确的，即算法正确性得证。

□

Problem 1.7

解答. 由公式

$$A(n) = \sum_{I \in D(n)} Pr(I) \cdot f(I)$$

可以计算出该算法的平均情况时间复杂度为：

$$A(n) = \frac{1}{n} \times 10 + \frac{2}{n} \times 20 + \frac{1}{2n} \times 30 + \frac{1}{2n} \times n = \frac{130 + n}{2n}$$

□

Problem 2.2

证明. 运用数学归纳法证明如下：

将 n 划分为 $2^k \leq n \leq 2^{k+1} - 1$，对 k 在 **N** 上进行归纳证明。

Step One：

证明基础情况 $k = 0$ 的正确性，当 $k = 0$ 时，$1 \leq n \leq 1$，即 $n = 1$，则 $\lceil \log(n+1) \rceil = 1$，$\lfloor \log n \rfloor + 1 = 0 + 1 = 1$，故结论 $\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$ 成立。

Step Two:

归纳假设当 $k = m$ 时，结论 $\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$ 成立，即 $2^m \leq n \leq 2^{m+1} - 1$ 时结论成立。则 $2^{m+1} \leq n \leq 2^{(m+1)+1} - 1$ 时，

$$\lceil \log(n_{m+1} + 1) \rceil = \lceil \log(n_m + 1) \rceil + 1 = \lfloor \log n_m \rfloor + 1 + 1 = \lfloor \log n_{m+1} \rfloor + 1$$

故结论在 $k = m + 1$ 时成立。

根据步骤一二可知，对于所有的自然数 k，结论 $\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$ 都成立。

□

Problem 2.3

证明. 利用斐波那契数列的递推公式与通项公式分别证明如下：

1) 由奇偶相加关系：奇奇为偶、奇偶为奇、偶偶为偶穷举可得斐波那契数列的通项奇偶关系如表 3：

Table 3: 斐波那契数列通项奇偶关系表

n	F_n	奇偶情况	奇偶递推
0	0	偶	
1	1	奇	
2	1	奇	偶 + 奇
3	2	偶	奇 + 奇
4	3	奇	奇 + 偶
5	5	奇	偶 + 奇
6	8	偶	奇 + 奇
7	13	奇	奇 + 偶
...	...	....	....

由穷举可发现奇偶递推的循环关系，同时由于循环节内单元为 3，且起始项为偶数，故结论可证。

2) 根据

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$

可得

$$\begin{aligned} F_n^2 &= \frac{1}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]^2 \\ &= \frac{1}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{2n} + \left(\frac{1-\sqrt{5}}{2}\right)^{2n} - 2\times(-1)^n\right] \end{aligned} \tag{1}$$

$$\begin{aligned} F_{n+1}F_{n_1} &= \frac{1}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right]\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n-1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n-1}\right] \\ &= \frac{1}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{2n} + \left(\frac{1-\sqrt{5}}{2}\right)^{2n} - \left(\frac{1+\sqrt{5}}{2}\right)^{2}\times(-1)^{n-1} - \left(\frac{1-\sqrt{5}}{2}\right)^{2}\times(-1)^{n-1}\right] \\ &= \frac{1}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{2n} + \left(\frac{1-\sqrt{5}}{2}\right)^{2n} + 3\times(-1)^n\right] \end{aligned} \tag{2}$$

式 1减去式 2可得

$$F_n^2 - F_{n+1}F_{n_1} = -(-1)^n = (-1)^{n+1}$$

□

Problem 2.5

证明. 在非空二叉树中，记总边数为 b, 总结点数为 n, 则有以下关系式满足

$$b = n - 1$$

$$n = n_0 + n_1 + n_2$$

1) 如果 T 为一棵 $2-tree$，则 $n_1 = 0$，边数 b 满足的关系式为

$$b = 2\times n_2 + n_0 = n - 1 = n_0 + n_2 - 1$$

所以

$$n_0 = n_2 + 1$$

2) 如果 T 为任意一棵二叉树，则 $n_1 \neq 0$，边数 b 满足的关系式为

$$b = 2\times n_2 + n_0 + 1\times n_1 = n - 1 = n_0 + n_1 + n_2 - 1$$

所以结论同样满足，即

$$n_0 = n_2 + 1$$

□

Problem 2.6

证明. 想再花些时间打磨一下证明语言，个人对于证明语言熟悉度有所欠缺，下次作业补上该题。 □

Problem 2.7

解答. 排序结果按照升序分别排列如下：

1) $\log n < n < n\log n < n^2 < n^2 + \log n < n^3 < n - n^3 + 7n^5 < 2^n$

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

□

Problem 2.15

解答. 各题解决过程如下：

1) 运用 Master 定理：

$\because a = 2, \quad b = 3, \quad \log_b a = \log_3 2$

$\therefore f(n) = 1 = \mathcal{O}(n^{\log_b a - \epsilon})$，则 $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_3 2})$

2) 运用直接展开法：

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + c\log n \\ &= T(\frac{n}{2^2}) + 2c\log n - c \\ &= T(\frac{n}{2^3}) + 3c\log n - c - 2c \\ &= ... \\ &= T(\frac{n}{2^k}) + kc\log n - \frac{k\cdot(k-1)}{2}c \\ &= ... \\ &= T(\frac{n}{2^{\log n}}) + \log n \cdot c\log n - \frac{\log n\cdot(\log n - 1)}{2}c \\ &= T(1) + \frac{c}{2}\cdot[(\log n)^2 - \log n] \\ &= \mathcal{O}((\log n)^2) \end{aligned}$$

3) 运用 Master 定理：

$\because a = 1, \quad b = 2, \quad \log_b a = 0$

$\therefore f(n) = n = \Omega(n^{\log_b a + \epsilon}), 0 < \epsilon < 1$，且 $\exists c \geq \frac{1}{2}, \to af(\frac{n}{b}) \leq cf(n)$

则 $T(n) = \Theta(f(n)) = \Theta(n)$

4) 运用 Guess and Prove 法：

猜测 $T(n) = \mathcal{O}(n \log n)$，运用数学归纳法进行证明。

Step one:

基础情况下的证明要考虑渐进增长率的定义，需要对于某个 n_0，证明对于 $n > n_0, T(n) \leq n \log n$，取 $n_0 = 2$ 即可满足。

Step two:

归纳假设对于小于 n 的参数情况，不等式 $T(n) \leq cn \log n$ 对于某常数 c 已经成立，则

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + n \\ &\leq 2c \cdot \frac{n}{2} \cdot \log(\frac{n}{2}) + n \\ &= cn \log n - cn + n \\ &\leq cn \log n (c \geq 1) \end{aligned}$$

根据步骤一和二，结论得证。

5) 运用 Guess and Prove 法：

猜测 $T(n) = \mathcal{O}(n(\log n)^2)$，运用数学归纳法进行证明。证明流程与上题类似。

6) 运用 Master 定理：

$\because a = 2, \quad b = 2, \quad \log_b a = 1$

$\therefore f(n) = n^2 = \Omega(n^{\log_b a + \epsilon}), 0 < \epsilon < 1$，且 $\exists c \geq 1, \rightarrow af(\frac{n}{b}) \leq cf(n)$

则 $T(n) = \Theta(f(n)) = \Theta(n^2)$

7) 暂未解出，下次作业补上

8) 运用直接展开法：

$$\begin{aligned} T(n) &= T(n-1) + 2 \\ &= T(n-2) + 2 \times 2 \\ &= T(n-3) + 2 \times 3 \\ &= ... \\ &= T(1) + 2 \times (n-1) \\ &= \mathcal{O}(n) \end{aligned}$$

9) 运用直接展开法：

$$\begin{aligned} T(n) &= T(n-1) + n^c \\ &= T(n-2) + n^c \times 2 \\ &= T(n-3) + n^c \times 3 \\ &= ... \\ &= T(1) + n^c \times (n-1) \\ &= \mathcal{O}(n^{c+1}) \end{aligned}$$

10) 运用直接展开法：

$$\begin{aligned} T(n) &= T(n-1) + c^n \\ &= T(n-2) + c^n \times 2 \\ &= T(n-3) + c^n \times 3 \\ &= ... \\ &= T(1) + c^n \times (n-1) \\ &= \mathcal{O}(nc^n) \end{aligned}$$

11) 暂未解出，下次作业补上

□

Problem 2.17

解答. 将题给时间复杂度的递归式进行变化得以下递归式：

$$\begin{aligned} \frac{T(n)}{n} &= \frac{T(n^{\frac{1}{2}})}{n^{\frac{1}{2}}} + 1 \\ &= \frac{T(n^{\frac{1}{4}})}{n^{\frac{1}{4}}} + 1 + 1 \\ &= \frac{T(n^{\frac{1}{8}})}{n^{\frac{1}{8}}} + 1 + 1 + 1 \\ &= ... \\ &= \frac{T(n^{\frac{1}{2^k}})}{n^{\frac{1}{2^k}}} + k \\ &= ... \\ &= \frac{T(2)}{2} + m \end{aligned} \tag{3}$$

式 3中，$2 = n^{\frac{1}{2^m}}$，则 $\frac{1}{2^m} = \log_n 2$，$2^m = \log n$，$m = \log\log n$，所以

$$\frac{T(n)}{n} = \frac{T(2)}{2} + \log\log n = \Theta(1) + \log\log n$$

即

$$T(n) = \mathcal{O}(n \log \log n)$$

□

Problem 2.18

解答. 选择 $a = 1, b = 2, f(n) = \log n$，则 $\log_b a = 0$，由于

$$f(n) \notin \mathcal{O}(n^{\log_b a - \epsilon})$$

$$f(n) \neq \Theta(n^{\log_b a})$$

$$f(n) \neq \Omega(n^{\log_b a + \epsilon})$$

Master 定理的 3 种情况均不能应用于求解该递归表达式。 □

Problem 2.20

解答. 各算法的结果与最坏情况运行时间分别如下:

1) 结果为:

$$\begin{aligned}
MYSTERY(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} j \\
&= \sum_{i=1}^{n-1} \left(\frac{(i+1+n) \times (n-i)}{2} \right) \\
&= \frac{1}{2} \sum_{i=1}^{n-1} (n^2 + n - i^2 - i) \\
&= \frac{1}{2} \left((n^2+n)(n-1) - \sum_{i=1}^{n-1} i^2 - \sum_{i=1}^{n-1} i \right) \\
&= \frac{1}{2} \left((n^2+n)(n-1) - \frac{(n-1)n}{2} - \frac{1}{3}(n-1)(n-\frac{1}{2})n \right) \\
&= \frac{2}{3}(n^3 - n)
\end{aligned}$$

最坏情况下的运行时间为 $\mathcal{O}(n^3)$

2) 结果为:

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

最坏情况下的运行时间为 $\mathcal{O}(n^3)$

3) 结果为:

$$\begin{aligned} PRESTIFEROUS(n) &= \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}(i+j-k) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{i}\left[(i+j)(i+1) - \frac{1}{2}(2j+i)(i+1)\right] \\ &= \sum_{i=1}^{n}\sum_{j=1}^{i}\left[\frac{i}{2}(i+1)\right] \\ &= \sum_{i=1}^{n}\frac{i}{2}(i+1)i \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} i^3 + \sum_{i=1}^{n} i^2\right) \\ &= \frac{1}{2}\left[\left(\frac{n(n+1)}{2}\right)^2 + \frac{1}{3}n(n+\frac{1}{2})(n+1)\right] \\ &= \frac{1}{8}n^4 + \frac{5}{12}n^3 + \frac{3}{8}n^2 + \frac{1}{12}n \end{aligned}$$

最坏情况下的运行时间为 $\mathcal{O}(n^4)$

4) 暂未解出，下次作业补上

□

Problem 3.2

解答. 1) 采用数学归纳法对算法中 i 进行归纳证明：排序根据 i 的数值可以划分为多个阶段，针对不同阶段进行归纳证明

Step one：基础情况当所需排序元素个数为 2 时，由算法定义，经过一次比较必可排列成功。

Step two: 归纳假设前 $i-1$ 种情况都排序正确，即末尾的 $i-1$ 个元素已为正确的升序排列，则由算法的定义，在经过第 i 次遍历后，末尾的 i 个元素将为正确的升序排列。

根据步骤一二可得冒泡排序的正确性。

2) 因为冒泡排序是基于遍历的排序算法，所以冒泡排序的最坏情况与平均情况的时间复杂度同样为

$$W(n) = A(n) = \sum_{i=1}^{n-1}(i-1) = \frac{n(n-1)}{2} - (n-1) = \mathcal{O}(n^2)$$

3) 不会影响算法的最坏情况和平均情况的时间复杂度，因为冒泡排序基于遍历的基本属性并未改变。

□

Problem 3.4

证明. 暂未解出，下次作业补交 □

Problem 3.5

解答. 设计的算法可以叙述如下：步骤一，从头到尾颠倒句子中所有字符的顺序；步骤二，判断出句子中各个单词的区间；步骤三，依次对每个单词从头到尾进行字符的颠倒。

由于总共做了线性次数的颠倒字符，所以该算法的时间复杂度为 $\mathcal{O}(n)$；由于字符串的颠倒并未消耗其他空间，故空间复杂度为 $\mathcal{O}(1)$。 □

Problem 3.6

解答. 1) 在一群共 n 个人中，可能有 1 个名人或者没有，因为名人的存在表明其不关注其他任何人，所以至多只能有一个名人的存在。

2) 基于遍历的暴力算法可以叙述如下：针对这 n 个人，从第一个人开始直到第 n 个人分别进行 check 操作，每次 check 其余 $n-1$ 个人是否都关注他，并最终找到名人。算法的时间复杂度为 $\mathcal{O}(n^2)$

基于遍历的改进算法叙述如下：将 n 个人进行有序比较，每次进行 check(A, B) 操作，若结果为 **YES**，即 A 关注 B，则删除 A 点不再考虑，反之删除 B 点；重复以上操作直至 n 个人只剩下一个人或都不剩下。由于每进行一次比较必可排除一个人，故在线性时间内即可解决该问题，即该算法的时间复杂度为 $\mathcal{O}(n)$

□

Problem 3.7

解答. 各题设计的算法如下所示:

1) 设计的算法命名为 **Brute Force of Max-sum Subsequence** 并引为算法 3

Algorithm 3: Brute Force of Max-sum Subsequence

Data: Given sequence of integers stored as Array[]

Result: The largers sum of one consecutive subequence

```
1  Initialization and data input;
2  MaxSum = 0;
3  for (i = 0; i < N; i++) do
4      for (j = i; j < N; j++) do
5          ThisSum = 0;
6          for (k = i; k <= j; k++) do
7              ThisSum += A[k];
8              if (ThisSum > MaxSum) then
9                  MaxSum = ThisSum;
10             end
11         end
12     end
13 end
14 return MaxSum;
```

2) 设计的算法命名为 **Improved Brute Force of Max-sum Subsequence** 并引为算法 4

3) 算法伪代码描述语言还在打磨，下次补交

4) 设计的算法命名为 **Linear Algorithm of Max-sum Subsequence** 并引为算法 5

5) 算法伪代码语言还在打磨，下次补交

□

Algorithm 4: Improved Brute Force of Max-sum Subsequence

Data: Given sequence of integers stored as Array[]

Result: The largers sum of one consecutive subequence

```
1  Initialization and data input;
2  MaxSum = 0;
3  for (i = 0; i < N; i++) do
4      ThisSum = 0;
5      for (j = i; j < N; j++) do
6          ThisSum += A[k];
7          if (ThisSum > MaxSum) then
8              MaxSum = ThisSum;
9          end
10     end
11 end
12 return MaxSum;
```

Algorithm 5: Linear Algorithm of Max-sum Subsequence

Data: Given sequence of integers stored as Array[]

Result: The largers sum of one consecutive subequence

```
1  Initialization and data input;
2  ThisSum = MaxSum = 0;
3  for (j = 0; j < N; j++) do
4      ThisSum += A[k];
5      if (ThisSum > MaxSum) then
6          MaxSum = ThisSum;
7      else
8          if ThisSum <0 then
9              ThisSum = 0;
10         end
11     end
12 end
13 return MaxSum;
```