算法设计与分析作业一

作者：吴润泽　　　**学号：**181860109

Email：181860109@smail.nju.edu.cn

2020 年 3 月 3 日

目录

PART I

problem 1.1

(1)

算法 1 对三个数进行排序

输入: 含有三个各不相同的整数的序列 $A = \{a, b, c\}$

输出: 从小到大排好序的序列 A

```
function ThreeNumSort(A)
    for i ← 2 to 3 do
        temp ← A[i]
        j ← i − 1
        while A[j] > temp and j > 0 do
            A[j + 1] ← A[j]
            j ← j − 1
        end while
        A[j + 1] ← temp
    end for
end function
```

(2)

最坏情况比较次数为： 3 次，**平均情况比较次数为:**$\frac{2+2+3+3+3+3}{6} = \frac{8}{3}$ 次。

(3)

在最坏情况下： 至少需要比较 3 次，最坏情况下，即每次进入 while 循环当前 i 指针位置都要与其前面元素进行比较。假设三个数按照大小依次为 a, b, c,则初始未排序序列共有 6 种即 $\{abc\}, \{acb\}, \{bac\}, \{bca\}, \{cab\}, \{cba\}$。其中 $\{acb\}, \{bca\}, \{cab\}, \{cba\}$ 四种序列，根据算法1，在 for 循环体中，第一次循环需要比较 1 次，第二次循环需要比较 2 次，因此最坏情况下需要比较 3 次

problem 1.2

(1)

算法 2 三个数的中位数

输入: 含有三个各不相同的整数 $\{a, b, c\}$

输出: 返回序列的中位数

```
function ThreeNumMiddle(a, b, c)
    if a < b then
        swap(a, b)
    end if
    if b > c then
        return b
    else if a < c then
        return a
    else
        return c
    end if
end function
```

(2)

在算法 2 中 最坏情况下比较次数为：3 次

平均情况比较次数为：$\frac{2+2+3+3+3+3}{6} = \frac{8}{3}$

(3)

在最坏情况下： 首先第一个 IF 语句体中，确定前两个数字的大小关系，如果之后不满足第二个数大于第三个数，则此时中位数在第一个数和第三个数之间，仍需一次比较判断，故最坏情况下需要比较三次。

problem 1.3

(1)

反例： 假设全集 $U = 1, 2, 3, 4$，U 的子集组成的集合族 $S = S_1, S_2, S_3, S_4$，其中 $S_1 = \{1, 2, 3\}, S_2 = \{1, 2, 4\}, S_3 = \{1, 2, 5\}, S_4 = \{4, 5\}$.
根据题中所给算法得到的覆盖为 $\{S_1, S_2, S_3\}$，而正确的最小覆盖应为 $\{S_1, S_4\}$.

(2)

算法 3 最小集合覆盖：贪心近似解

输入: 待覆盖的集合 U，U 的子集组成的一个集合族 S

输出: 作为覆盖集的结果 C

```
function COVER(U, S)
    members ← U
    subsets ← S
    covering ← {}
    while size(members) > 0 and size(subsets) > 0 do
        intersection ← set_intersection(members, subsets)[1]
        members ← remove(members, intersection)
        subsets ← remove(subsets, intersection)[2]
        covering ← add(covering, intersection)[3]
    end while
    if size(members) > 0 then
        return −1[4]
    end if
    return covering
end function
```

[1]在 subsets 中找出能够覆盖到 members 的最大交集

[2]在 members 中将 intersection 覆盖元素去掉，在 subsets 中将 intersection 集合去掉

[3]在 covering 中加入 intersection 作为覆盖集成员

[4]如果 members 中仍然存在未被覆盖的元素，那么不可能实现完全覆盖

(3)

不能总是得到最小覆盖。

反例： 假设全集 $U = 1, 2, 3, 4, 5, 6, 7$,U 的子集组成的集合族 $S = S_1, S_2, S_3, S_4$，其中 $S_1 = \{1, 2, 3, 4\}, S_2 = \{1, 2, 5\}, S_3 = \{3, 4, 6\}, S_4 = \{7\}$.
根据上述算法 3 得到的覆盖为 $\{S_1, S_2, S_3, S_4\}$，而正确的最小覆盖应为 $\{S_2, S_3, S_4\}$. 因此并不能总是得到正确解。

problem 1.5

Induction on n:

- **Base case**
  - $n = 0,\ for\ any$
    $x:\ HORNER(A[0], x) = a_0$
- **Assumption**
  - $n = k,\ for\ any\ x:$
    $HORNER(A[0...k], x) = a_kx^k + a_{k-1}x^{k-1} + ... + a_1x + a_0$
- **Induction**
  - $n = k + 1,\ for\ any\ x\ Assume\ B[0...k] \leftarrow A[1...k+1]:$
    $HORNER(A[0...k+1], x)$
    $= x * HORNER(B[0...k], x) + a_0$
    $= x * (a_{k+1}x^k + a_kx^{k-1} + ... + a_2x + a_1) + a_0$
    $= a_{k+1}x^{k+1} + a_kx^k + ... + a_2x^2 + a_1x + a_0$

problem 1.6

(1)

Induction on z:

- **Base case**
  - $z = 0,\ for\ any\ y \geq 0:\ INT\_MULT(y, 0) = 0$
  - $z = 1,\ for\ any\ y \geq 0:$

    $INT\_MULT(y, 1)$

    $= INT\_MULT(2y, \lfloor\frac{1}{2}\rfloor) + y \cdot (1 \mod 2)$

    $= INT\_MULT(2y, 0) + y$

    $= y$
- **Assumption**
  - $for\ any\ z \geq k,\ for\ any\ y \geq 0:$

    $INT\_MULT(y, z)$

    $= INT\_MULT(2y, \lfloor\frac{z}{2}\rfloor) + y \cdot (z \mod 2)$

    $= y \cdot z$
- **Induction**
  - $z = k + 1,\ for\ any\ y:$

    $INT\_MULT(y, k + 1)$

    $= INT\_MULT(2y, \lfloor\frac{k+1}{2}\rfloor) + y \cdot (k + 1 \mod 2)$

    $= y \cdot (2\lfloor\frac{k+1}{2}\rfloor + (k + 1) \mod 2)$

    $= y \cdot (k + 1)$

(2)

Induction on z: 与证明方法类似，将其中的 c=2 替换为 c 即可。

problem 1.7

平均时间复杂度：

$$A(n) = \sum_{I \in D(n)} Pr(I)f(I) = \frac{1}{n} \cdot 10 \cdot \frac{n}{4} + \frac{2}{n} \cdot 20 \cdot \frac{n}{4} + \frac{1}{2n} \cdot 30 \cdot \frac{n}{4} + \frac{1}{2n} \cdot n \cdot \frac{n}{4} = \frac{65}{4} + \frac{n}{8}$$

PART II

problem 2.2

证明：

$$n \geq 1, let\ 2^k \leq n \leq 2^{k+1} - 1,$$

$$\Rightarrow k < \log(2^k + 1) \leq \log(n+1) \leq \log(2^{k+1}) = k+1 \quad (1)$$

$$(1) \rightarrow \lceil \log(n+1) \rceil = k+1$$

$$\Rightarrow k+1 = \log(2^k) + 1 \leq \log(n) + 1 \leq \log(2^(k+1) - 1) + 1 < k+2 \quad (2)$$

$$(2) \rightarrow \lfloor \log(n) + 1 \rfloor = k+1$$

因此，$\lceil \log(n+1) \rceil = \lfloor \log(n) + 1 \rfloor$，证毕。

problem 2.3

(1)

证明： 对于斐波那契数列：$F_n = F_{n-1} + F_{n-2}$，对于 F_n 的奇偶性与 F_{n-1} F_{n-2} 的奇偶性有关。

奇偶性：

F_n	F_{n-1}	F_{n-2}
奇	奇	偶
奇	偶	奇
偶	奇	奇
偶	偶	偶

由上表： 斐波那契数列前两项为 1，均为奇数，因此永远不会出现偶数加偶数的情况，所以只能是前 3 种假发循环出现，因此偶数均出现在项数为 3 的倍数的位置。

(2)

Induction on n:

- **Base case**
  - $n=2,\ F_2^2-F_3F_1=-1=(-1)^3$
- **Assumption**
  - $n=k,\ F_k^2-F_{k+1}F_{k-1}=(-1)^{k+1}$
- **Induction**
  - $n=k+1:$
    $$\begin{aligned}&F_{k+1}^2-F_kF_{k+2}\\&=F_{k+1}(F_k+F_{k-1})-F_kF_{k+2}\\&=F_{k-1}F_{k+1}+F_kF_{k+1}-F_kF_{k+2}\\&=F_k^2+F_kF_{k+1}-F_kF_{k+2}-(-1)^{k+1}\\&=F_k(F_k+F_{k+1}-F_{k+2})+(-1)^{k+2}\\&=(-1)^{k+2}\end{aligned}$$

problem 2.5

(1)

证明： 2-tree 即结点的度为 0 或 2，即仅有 n_0,n_2 假设共有 n 个结点，则 $n_0+n_2=n$，同时 n 个结点则有 n-1 条边，n-1 条边由度数为 2 的结点组成，即 $2n_2=n-1=n_0+n_2-1\Rightarrow n_0=n_2+1$，证毕。

(2)

成立

证明： 同上述证明，假设 n 个结点，则 $n_0+n_1+n_2=n$，n-1 条边由度数为 1 和度数为 2 的结点组成，即 $2n_2+n_1=n-1=n_0+n_1+n_2-1\Rightarrow n_0=n_2+1$，证毕。

problem 2.6

(1)

- ***O transitivity***

  $f(n) = O(g(n)), g(n) = O(h(n)),$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty, \lim_{n\to\infty} \frac{g(n)}{h(n)} = d < \infty$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \frac{g(n)}{h(n)} = c \cdot d < \infty$

  $\rightarrow f(n) = O(h(n)),\ proved.$

- ***Ω transitivity***

  $f(n) = \Omega(g(n)), g(n) = \Omega(h(n)),$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0, \lim_{n\to\infty} \frac{g(n)}{h(n)} = d > 0$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \frac{g(n)}{h(n)} = c \cdot d > 0$

  $\rightarrow f(n) = \Omega(h(n)),\ proved.$

- ***Θ transitivity***

  $f(n) = \Theta(g(n)), g(n) = \Theta(h(n)),$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = c, \lim_{n\to\infty} \frac{g(n)}{h(n)} = d(0 < c, d < \infty)$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \frac{g(n)}{h(n)} = c \cdot d(0 < c \cdot d < \infty)$

  $\rightarrow f(n) = \Theta(h(n)),\ proved.$

- ***o transitivity***

  $f(n) = o(g(n)), g(n) = o(h(n)),$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0, \lim_{n\to\infty} \frac{g(n)}{h(n)} = 0$

  $\rightarrow \lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \frac{g(n)}{h(n)} = 0$

  $\rightarrow f(n) = o(h(n)),\ proved.$

- ***ω transitivity***

$$
\begin{aligned}
& f(n)=\omega(g(n)), g(n)=\omega(h(n)), \\
& \rightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty, \lim_{n\to\infty}\frac{g(n)}{h(n)}=\infty \\
& \rightarrow \lim_{n\to\infty}\frac{f(n)}{h(n)}=\lim_{n\to\infty}\frac{f(n)}{g(n)}\cdot\frac{g(n)}{h(n)}=\infty \\
& \rightarrow f(n)=\omega(h(n)),\ proved.
\end{aligned}
$$

(2)

证明： $\lim_{n\to\infty}\frac{f(n)}{f(n)}=1$，对于 O, Θ, Ω 三种关系的定义均符合，因此满足自反性。

(3)

证明：

对称性：

$$
\begin{aligned}
& f(n)=\Theta(g(n)) \rightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)}=c, (0<c,<\infty) \\
& \rightarrow \lim_{n\to\infty}\frac{g(n)}{f(n)}=\frac{1}{c}(0<\frac{1}{c}<\infty) \\
& \rightarrow g(n)=\Theta(f(n)),\ proved.
\end{aligned}
$$

由 (1), (2), (3) 可知，Θ 满足传递性、自反性、对称性，因此是等价关系。

(4)

证明：

充分性：

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

$$f(n) = \Theta(g(n))$$

必要性：
$$\to \lim_{n\to\infty} \frac{f(n)}{g(n)} = c, (0 < c, < \infty)$$
$$\to f(n) = O(g(n)),\ f(n) = \Omega(g(n)),\ proved.$$

(5)

a. $f(n) = O(g(n)) \Leftrightarrow \lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = c < \infty \Leftrightarrow \lim\limits_{n\to\infty} \dfrac{g(n)}{f(n)} = \dfrac{1}{c} > 0 \Leftrightarrow g(n) = \Omega(f(n))$

b. $f(n) = o(g(n)) \Leftrightarrow \lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0 \Leftrightarrow \lim\limits_{n\to\infty} \dfrac{g(n)}{f(n)} = \infty \Leftrightarrow g(n) = \omega(f(n))$

(6)

a.
$$assume A = o(g(n)) \cap \omega(g(n)) \neq \emptyset, f(n) \in A$$
$$f(n) = o(g(n)), \to \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0,\ f(n) = \omega(g(n)), \to \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

产生矛盾，即 $o(g(n)) \cap \omega(g(n)) = \emptyset$

b.
$$assume A = \Theta(g(n)) \cap o(g(n)) \neq \emptyset, f(n) \in A$$
$$f(n) = \Theta(g(n)), \to \lim_{n\to\infty} \frac{f(n)}{g(n)} = c, (0 < c < \infty)\ f(n) = o(g(n)), \to \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

产生矛盾，即 $\Theta(g(n)) \cap o(g(n)) = \emptyset$

c.
$$assume A = \Theta(g(n)) \cap \omega(g(n)) \neq \emptyset, f(n) \in A$$
$$f(n) = \Theta(g(n)), \to \lim_{n\to\infty} \frac{f(n)}{g(n)} = c, (0 < c < \infty)\ f(n) = o(g(n)), \to \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

产生矛盾，即 $\Theta(g(n)) \cap \omega(g(n)) = \emptyset$

problem 2.7

(1)

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

(2)

排序： $\log\log n < \log n < (\log n)^2 < \sqrt{n} < n < n\log n < n^{1+\varepsilon} < n^2 = n^2 + \log n < n^3 < n - n^3 + 7n^5 < 2^{n-1} = 2^n = e^n < n!$

problem 2.15

(1)

$$a = 1, b = 2, f(n) = 1$$
$$n^{log_3 2} \approx n^{0.63},\ f(n) = O(n^{\log_3 2 - 0.5}), \varepsilon > 0$$
$$master\ case1 : T(n) = \Theta(n^{\log_3 2})$$

(2)

$$a = 1, b = 2, f(n) = c\log n$$
$$n^{log_2 1} = 0, \forall\varepsilon > 0,\ f(n) \neq \Omega(n^{\log_2 1+\varepsilon})$$
$$T(n) = T(\frac{n}{2}) + c\log n$$
$$T(n) = T(\frac{n}{4}) + c\log n + c(\log n - 1)$$
$$\cdots$$
$$T(n) = T(1) + c\log n + c(\log n\log n - 1 - 2 - \cdots - k)$$
$$T(n) = \Theta((\log n)^2)$$

(3)

$$a = 1, b = 2, f(n) = cn$$
$$n^{log_2 1} = 0,\ f(n) = \Omega(n^{\log_2 1+1}), \varepsilon > 0$$
$$proving:\ af(\frac{n}{b}) \leq df(n), n \geq N_0, d > 0$$
$$f(\frac{n}{2}) = c\frac{n}{2} \leq cdn, d \geq 0.5$$
$$master\ case3 : T(n) = \Theta(n)$$

(4)

$$a = 2, b = 2, f(n) = cn$$
$$n^{log_2 2} = n,\ f(n) = \Theta(n^{\log_2 2})$$
$$master\ case2 : T(n) = \Theta(n\log n)$$

(5)

$$a = 2, b = 2, f(n) = cn\log n$$
$$n^{log_2 2} = n, \forall\varepsilon > 0,\ f(n) \neq \Omega(n^{\log_2 2+\varepsilon})$$
$$T(n) = 2T(\frac{n}{2}) + cn\log n$$
$$T(n) = 2(2T(\frac{n}{4}) + c\frac{n\log n - 1}{2}) + cn\log n$$
$$T(n) = 4T(\frac{n}{4}) + 2cn\log n - c$$
$$\cdots$$
$$T(n) = cn\log n + cn(\log n - 1) + cn(\log n - 2) + \cdots + cn(\log n - k + 1)$$
$$T(n) = n\log^2 n - nk(k-1)/2$$
$$T(n) = \Theta(n(\log n)^2)$$

(6)

$$a = 2, b = 2, f(n) = cn^2$$
$$n^{log_2 2} = n,\ f(n) = \Omega(n^{\log_2 2+1}), \varepsilon > 0$$
$$proving:\ af(\frac{n}{b}) \leq df(n), n \geq N_0, d > 0$$
$$2f(\frac{n}{2}) = c\frac{n^2}{2} \leq cdn^2, d \geq 0.5$$
$$master\ case3 : T(n) = \Theta(n^2)$$

(7)

$$a = 49, b = 25, f(n) = cn^{\frac{3}{2}} \log n$$
$$n^{log_{25} 49} \approx n^{1.21},\ f(n) = \Omega(n^{log_{25} 49+0.2}), \varepsilon > 0$$
$$proving:\ af(\frac{n}{b}) \leq df(n), n \geq N_0, d > 0$$
$$49f(\frac{n}{25}) = 49c(\frac{n^2}{625}(\log n - 2\log 5)) \leq cdn^{1.5}\log n, d \geq 0.5$$
$$master\ case3 : T(n) = \Theta(n^{\frac{3}{2}} \log n)$$

(8)

$$T(n) = T(n-1) + 2 = T(n-2) + 4 = \cdots = T(1) + 2(n-1) = \Theta(n)$$

(9)

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

(10)

$$\begin{aligned} T(n) &= T(n-1) + c^n = T(n-2) + c^n + c^(n-1) \\ &= \cdots = T(1) + c^n + c^{n-1} + \cdots + c^2 + c,\ c \geq 1,\ geometric\ series \\ &= \Theta(c^n) \end{aligned}$$

(11)

$T(n) = T(n/2) + T(n/4) + T(n/8) + n$ 递归树，每一层的 $f(n) = (\frac{7}{8})^h n$，h 为结点所在树的高度（根高度记为 0），树高为 $k = \Theta(\log n)$，则，

$$T(n) = n + \frac{7}{8}n + \cdots + (\frac{7}{8})^k n = \Theta(n)$$

problem 2.17

化简

$$\begin{aligned}
&T(n) = \sqrt{n}T(\sqrt{n}) + n \\
&\frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 1 \\
&A(n) = \frac{T(n)}{n}
\end{aligned}$$

A(n)

$$\begin{aligned}
A(n) &= A(\sqrt{n}) + 1 \\
&= A(\sqrt[4]{n}) + 2 \\
&= \cdots \\
&= A(1) + k, n = 2^{2^k}, k = \log\log n \\
&= \Theta(\log\log n)
\end{aligned}$$

解得

$$T(n) = \Theta(n \log\log n)$$

problem 2.18

$$\begin{aligned}
&a = 1, b = 2, f(n) = c\log n^{log_2 1} = 0, three\ conditions\ not\ statisfied \\
&case\ 1:\ \forall\varepsilon > 0,\ f(n) \neq O(n^{\log_2 1-\varepsilon}) \\
&case\ 2:\ \qquad\qquad f(n) \neq \Theta(n^{\log_2 1}) \\
&case\ 3:\ \forall\varepsilon > 0,\ f(n) \neq \Omega(n^{\log_2 1+\varepsilon})
\end{aligned}$$

problem 2.20

MYSTERY

result: $r = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{j} 1 = \frac{n^3}{3} - \frac{n}{3}$

最坏时间复杂度： $T(n) = O(n^3)$

PERSKY

result: $r = \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i+j} 1 = \frac{n^3}{3} + n^2 + \frac{2n}{3}$

最坏时间复杂度： $T(n) = O(n^3)$

PRESTIFEROUS

result: $r = \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i+j} \sum_{l=1}^{i+j-k} 1 = \frac{n^4}{8} + \frac{5n^3}{12} + \frac{3n^2}{8} + \frac{n}{12}$

最坏时间复杂度： $T(n) = O(n^4)$

CONUNDRUM

result: $r = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=i+j-1}^{n} 1 = \frac{n^2}{2} - \frac{n}{2}$

最坏时间复杂度： $T(n) = O(n^2)$

PART III

problem 3.2

(1)

对外层循环进行数学归纳：

Base Case 外层循环第一轮结束后，使得最后一个元素大于前 n-1 个元素。外层循环第二轮结束后，使得倒数第二个元素大于前 n-2 个元素，且满足最后两个元素有序。

Assumption 假设外层循环第 k 轮结束后，使得倒数第 k 个元素大于前 n-k 个元素，且后 k 个元素有序即 $A[n-k+1] < A[n-k+2] < \cdots < A[n]$。

Induction 当外层循环第 k+1 轮结束后，由于 $A[1 \cdots n-k-1] < A[n-k] < A[n-k+1]$，且后 k 个元素有序，则 $A[n-k] < A[n-k+1] < \cdots < A[n]$ 后 k+1 个元素有序。进行最后一轮循环后，n 个元素有序，证毕。

(2)

最坏时间复杂度与平均复杂度相同： 任何情况下，内层循环都要进行 $i-1$ 次比较，最坏与平均情况时间复杂度均为 $\sum_{i=n}^{2}\sum_{j=1}^{i-1} 1 = \frac{n^2}{2} - \frac{n}{2} = \Theta(n^2)$。

(3)

最坏时间复杂度： 不影响最坏情况时间复杂度，最坏情况下，原始序列是降序的，每次内层循环的最后一次交换都发生在待排序序列的末尾，因而每次交换次数仍未 $i-1$ 次，时间复杂度不变。

平均时间复杂度： 影响平均情况最坏时间复杂度，改进后内层循环的最后一次交换可能发生在待排段的中间位置，使得内层循环实际运行次数小于 $i-1$，交换和比较次数减少，但是不影响其量级仍为 $\Theta(n^2)$。

problem 3.4

算法 4 PREVIOUS-LARGER 改进算法

```
function :PREVIOUS-LARGER(A[1···n])
    for i ← 1 to n do
        j ← i − 1
        while j > 0 and A[j] ≤ A[i] do
            j ← p[j]
        end while
        p[i] ← j
    end for
    return p[1···n]
end function
```

证明 Induction on i:

- **Base case**
  - $i = 1 : p[1] = 0$，正确。
- **Assumption**
  - $i = k :$ $p[1 \cdots k]$ 均已被正确得到。
- **Induction**
  - $i = k + 1 :$ 在 While 循环体中，当 $A[j] \le A[k+1]$ 时，$j = p[j]$，又由于 $A[p[j] + 1 \cdots (j-1)] \le A[j]$，则 $p[k+1] \le p[j]$。

    内层循环中重复向左移动 j 指针，直到退出循环，在退出前一次循环有 $A[j] \le A[k+1]$ *and* $A[p[j]] \ge A[k+1]$ *and* $p[k+1] \le p[j]$，则 $p[k+1] = p[j]$ 正确，算法正确性证毕。

时间复杂度 最外层循环遍历 n 次，内层循环每次向前移动常数次，因此时间复杂度为 $\Theta(n)$。

problem 3.5

算法 5 颠倒单词顺序

输入: 待翻转的句子 S，句子长度 L

输出: 翻转后的句子 S

function REVERSE_STR($S, start, end$)
/**首先翻转整个句子的每个字符**/
    $low \leftarrow start,\ high \leftarrow end$
    while $low < high$ **do**
        $swap(S[low], S[high])$ \\句子头尾字符进行翻转
        $low++,\ high--$
    end while
end function
function REVERSE_WORD(S, L)
/**然后翻转句子中的每个单词**/
    $start \leftarrow 0,\ end \leftarrow 0$
    for $i \leftarrow 0$ *to* L **do**
        $end \leftarrow i$
        if $S[end] = blank$ **then**\\单词范围为 [start,end-1]
            $SWAP\_STR(S, start, end-1)$\\对这个单词进行翻转
            $start \leftarrow end + 1$
        end if
    end for
    return S
end function

时间复杂度 首先翻转整个句子前后指针同时移动 $\frac{n}{2}$ 次，之后遍历整个句子的每一个单词的分割移动 n 次，因此时间复杂度为 $\frac{n}{2} + n = \frac{3n}{2} = O(n)$。

空间复杂度 除了最开始的原始输入字符串外，只使用若干常数级的变量，因此如果不考虑输入字符串空间复杂度为 $O(1)$。

problem 3.6

(1) 可能有 1 个名人或者没有名人。

算法 6 寻找名人

输入: 人群集合 $S[1 \cdots n]$，人数 n

输出: 名人 target

```
function FIND_TARGET(S, n)
/**如果 A 关注 B 则 A 不是名人，如果 B 关注 A 则 B 不是名人，
如果 AB 互相关注则均不是名人，如果 AB 互不关注均不是名人 **/
    num ← n
    first ← 0, next ← 1
    while n > 0 do
        if S[first] YES S[next] and S[next] NO S[first] then
            first ← next, next ← next + 1 \\保证 first 指向可能的名人
            n ← n − 1
        else if S[first] NO S[next] and S[next] YES S[first] then
            next ← next + 1
            n ← n − 1
        else
            first ← next + 1, next ← next + 2
            n ← n − 2
        end if
    end while
/**如果 first 为名人则一定在 1-n 中，否则退出循环体时 first 大于 n**/
    return first ≤ num?first : 0   \\0 表示没有名人
end function
```

(2) **时间复杂度**遍历一遍所有人群即可，时间复杂度为 $\Theta(n)$。

problem 3.7

算法 7 (1) 最大连续子序列和暴力算法 $O(n^3)$

```
function MAX_SUM1(S)
/**枚举所有子序列，并根据子序列上下界，计算其子序列和，与当前最大值进行比较**/
    res ← 0, len ← S.length()
    for i ← 1 to len do
        for j ← 0 to i − 1 do
            cur_res ← 0
            for k ← i to j do
                cur_res ← cur_res + S[k]
            end for
            res ← max(cur_res, res)
        end for
    end for
    return res
end function
```

算法 8 (2) 最大连续子序列和暴力算法 $O(n^2)$

function MAX_SUM2(S)
/**枚举所有子序列，数组 SUM[i] 记录前 i 项数据和，利用 SUM 数组根据子序列头尾计算当前子序列和，与当前最大值进行比较**/
    $res \leftarrow 0, len \leftarrow S.length()$
    for $i \leftarrow 2$ *to* len **do**
        $sum[i] = sum[i-1] + S[i]$
        for $j \leftarrow 1$ *to* $i-1$ **do**
            $res \leftarrow max(sum[i] - sum[j-1], res)$
        end for
    end for
    return res
end function

算法 9 (3) 最大连续子序列和分治算法 $O(n \log n)$

```
function MAX_SUM3(S, start, end)
/**最大子序列和的位置存在三种情况：1. 在左半部分，2. 在右半部分，3. 跨越左右两部分，三者中的最大者即为所求。**/
    res ← 0, len ← S.length(),  sum[1] = S[1]
    if start = end then
        res = S[start] > 0?S[start] : 0
    else
        center ← (start + end)/2
        left_res = MAX_SUM3(S, start, center)
        right_res = MAX_SUM3(S, center + 1, end)
/**计算包含左半部分最右元素的最大和 s1**/
        left ← 0,  s1 ← 0
        for i ← center to start do
            left ← lefts + S[i]
            s1 ← max(left, s1)

        end for/**同理计算包含右半部分最左元素的最大和 s2 **/
        res = max(left_res, right_res, s1 + s2)
    end if
    return res
end function
```

算法 10 (4) 最大连续子序列和去冗余算法

```
function MAX_SUM4(S)
/**如果当前子序列和非正，那么对后续子序列和无贡献应当舍去**/
    res ← 0, len ← S.length(),  sum ← 0
    for i ← 0 to len do
        sum = sum > 0?sum + S[i] : S[i]
        res = max(sum, res)
    end for
    return res
end function
```

算法 11 (5) 最大连续子序列和动态规划算法

```
function MAX_SUM4(S)
/**sum[i] 用来记录以 S[i] 结尾的序列最大和**/
    res ← 0, len ← S.length(),  sum[0] ← 0
    for i ← 1 to len do
        sum[i] = max(sum[i], sum[i − 1] + num[i])
        res = max(sum[i], res)
    end for
    return res
end function
```