# 第一章　准备知识

## 1.1　练习

**Problem 1.1.1**
假设一个有序集中没有相同的元素，那么有序集的中值是这样的一个元素：集合中比它小的元素个数和比它大的元素个数差值小于2。

1. 编写一个从三个不同的整数a,b,c中找到中值的算法。
2. 在最坏情况下你的算法需要进行多少次比较？平均情况下呢？
3. 在最坏情况下找出三个元素的中值需要多少次比较？证明你的结论。

**Problem 1.1.2**
证明:对于任意整数$n \geq 1, \lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$（提示：将n划分为$2^k \leq n \leq 2^{k+1}-1$）。

**Problem 1.1.3**
斐波纳契数列为：$F(n) = F(n-1) + F(n-2), n \geq 2, F(0) = 0, F(1) = 1$。证明（利用归纳法）下列两个结论中正确的结论：

1. 对于$n \geq 1, F(n) \leq 100(\frac{3}{2})^n$。
2. 对于$n \geq 1, F(n) \geq 0.01(\frac{3}{2})^n$。

**Problem 1.1.4**
集合覆盖问题有如下描述：给定全集$U = \{1, ..., n\}$的子集的集合$S = \{S_1, ..., S_m\}$，找出$S$的最小子集$T(T \subseteq S)$，满足$\bigcup_{t_i \in T} t_i = U$。例如，全集$U = \{1, 2, 3, 4, 5\}$有下面几个子集$S_1 = \{1, 3, 5\}, S_2 = \{2, 4\}, S_3 = \{1, 4\}$和$S_4 = \{2, 5\}$，得到的集合覆盖就是$S_1$和$S_2$。
找出下面算法失败的例子：首先选择$S$中最大的元素$S_i$，并从全集中将$S_i$中的所有元素删除；然后从$S$中剩余元素中挑选最大的并从全集中删除对应元素；重复上述过程直到全集中的所有元素都覆盖到了。

**Problem 1.1.5 (换硬币问题)**
我们定义换硬币问题如下：给定若干硬币，它们的面值是正整数值$S = \{s_1, s_2, ..., s_n\}$；另外给定一个金额值正整数$T$。我们需要从$S$中找出若干个硬币，使得它们的面值和为$T$，或者返回不存在这样的硬币集合。我们给出三种不同的算法设计方案：

1. 依次扫描硬币$s_1, s_2, ..., s_n$，并累加金额。
2. 按面值从小到大的顺序，依次扫描硬币，并累加金额。

3. 按面值从大到大的顺序，依次扫描硬币，并累加金额。

在上述扫描过程中，如果如果金额值累积到正好为$T$，则返回已经扫描到的硬币；否则返回不存在。请将上述三种方案分别写成算法，并通过举反例的方式证明这三种算法的“不正确性”。

**Problem 1.1.6**

背包问题有如下描述：给定一个整数集合$S = \{s_1, s_2, ..., s_n\}$和一个目标数字$T$，现在需要找到$S$ 的一个子集使得子集中的整数的和为$T$。例如，$S = \{1, 2, 5, 9, 10\}$ 存在子集满足$T = 22$但是不存在满足$T = 23$ 的子集。

请找出下列解决背包问题的算法失败的例子，即给定$S$和$T$，利用给定算无法找到元素的和等于$T$的子集，但是实际上$S$存在这样的子集。

(a) $S$中的元素保持原本顺序，按照从左到右的顺序依次选入子集。如果当前子集的和小于$T$则继续挑选；否则算法结束。我们称这个算法为“$first - fit$”。

(b) $S$中的元素按照从小到大排列，然后按照从左到右的顺序依次选入子集。如果当前子集的和小于$T$则继续挑选；否则算法结束。我们称这个算法为“$best - fit$”。

(c) $S$中的元素按照从大到小排列，然后按照从左到右的顺序依次选入子集。如果当前子集的和小于$T$则继续挑选；否则算法结束。

**Problem 1.1.7**

算法1是用来求解多项式P(x) $= a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ 的，请证明算法的正确性。

**Algorithm 1:** HORNER($A[0..n], x$)

$p \leftarrow A[n]$ ; /* 数组$A[0..n]$存放系数$a_0, a_1, \cdots, a_n$ */

**for** $i \leftarrow n - 1$ **downto** $0$ **do**

  $p \leftarrow p \cdot x + A[i]$ ;

**return** p ;

**Problem 1.1.8 (Proving the correctness of *Multiply*$(y, z)$)**

算法2 是用来计算两个非负整数$y, z$的乘积.

**Algorithm 2: int** multiply(**int** y, **int** z)

**if** $z = 0$ **then**

  **return** 0;

**else**

  **return** multiply(cy,$\lfloor \frac{z}{c} \rfloor$) + y $\cdot$ ($z$ mod $c$);

(a) 令$c = 2$，证明算法2的正确性。

(b) 令$c$为任意的一个常数$c(c \geq 2)$，证明算法2的正确性。

**Problem 1.1.9**

算法3是对数组$A[1..n]$中的元素进行排序，请证明它的正确性。

**Algorithm 3:** BUBBLE-SORT($A[1..n]$)

```
for i ← n downto 1 do
    for j ← 1 to i − 1 do
        if A[j] > A[j + 1] then
            swap the values of A[j] and A[j + 1] ;
```

**Problem 1.1.10**

请分别给出下面四个算法的结果（用含有$n$的表达式表示）以及在最坏情况下的运行时间（用$O$表示）。

**Algorithm 4:** MYSTERY(n)

```
r ← 0;
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        for k ← 1 to j do
            r ← r + 1;
return r;
```

**Algorithm 5:** PERSKY(n)

```
r ← 0;
for i ← 1 to n do
    for j ← 1 to i do
        for k ← j to i + j do
            r = r + 1;
return r;
```

**Algorithm 6:** PRESTIFEROUS(n)

```
r ← 0;
for i ← 1 to n do
    for j ← 1 to i do
        for k ← j to i + j do
            for l ← 1 to i + j − k do
                r ← r + 1;
return r;
```

**Problem 1.1.11**

将下面给出的函数按照渐近阶从低到高的顺序进行排序。如果有多个函数其渐近阶相同，指出它们。

**Algorithm 7:** CONUNDRUM(n)

```
r ← 0;
for i ← 1 to n do
    for j ← i + 1 to n do
        for k ← i + j − 1 to n do
            r ← r + 1;
return r;
```

1. 将下面的函数排序：

   $n$, $2^n$, $n\log n$, $n^3$, $n^2$, $\lg n$, $n - n^3 + 7n^5$, $n^2 + \lg n$

2. 将下面的函数同a中的函数置于一起进行排序（假设$0 < \epsilon < 1$）：

   $e^n$, $\sqrt{n}$, $2^{n-1}$, $\lg\lg n$, $\lg n$, $(\lg n)^2$, $n!$, $n^{1+\epsilon}$

**Problem 1.1.12**

对于大小为n的输入，假设在最坏情况下，算法1需要执行的步数为$f(n) = n^2 + 4n$，算法2需要执行的步数为$g(n) = 29n + 3$。当n为多少时，算法1比算法2快（在最坏情况下）?

**Problem 1.1.13**

证明：一个算法的运行时间是$\Theta(g(n))$当且仅当其最坏情况运行时间为$O(g(n))$，且最佳情况运行时间为$\Omega(g(n))$。

**Problem 1.1.14**

证明：$o(g(n))\bigcap\omega(g(n))$是空集。

**Problem 1.1.15**

证明：$k\,ln\,k = \Theta(n)$可以推出$k = \Theta(n/ln n)$。

**Problem 1.1.16**

证明或者给出一个反例：对于任意从非负整数映射到非负实数的函数f，不存在一个函数g，使得g在$\Theta(f)$以及$o(f)$之间，也即，$\Theta(f) \cap o(f) = \emptyset$。

**Problem 1.1.17**

找出下列递归方程结果的渐近阶。你可以假设$T(1) = 1, n > 1$，c是正的常量。

1. $T(n) = 2T(n/3) + 1$

2. $T(n) = T(n/2) + c\lg n$

3. $T(n) = T(n/2) + cn$

4. $T(n) = 2T(n/2) + cn$

5. $T(n) = 2T(n/2) + cn\lg n$

6. $T(n) = 2T(n/2) + cn^2$

7. $T(n) = 49T(n/25) + n^{3/2}\log n$
8. $T(n) = T(n-1) + 2$
9. $T(n) = T(n-1) + n^c$, where $c \geq 1$ is some constant
10. $T(n) = T(n-1) + c^n$, where $c > 1$ is some constant

## 1.2 问题

**Problem 1.2.1**

有一个$2^n \times 2^n$个小方格组成的棋盘，现在从其中任意删除一个方格，请用归纳法证明棋盘可以由$(4^n - 1)/3$ 个$L$形的块（由三个小方块组成）无缝隙，无重叠，紧凑地组合而成。

**Problem 1.2.2**

假设你有一堆数量为$n$的大小互不不同的煎饼，现在需要对这些煎饼排序使得小的煎饼放在大的煎饼上面。你仅可以使用的操作是用一个锅铲插入到最上面的$k$（$1 \leq k \leq n$）个煎饼下面，然后将它们一起翻转过去，如图1.1所示。请设计算法，证明算法的正确性，分析并给出算法的复杂度。

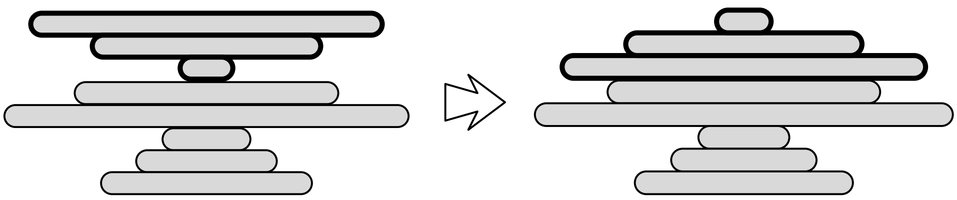

图 1.1: 翻转最上面的三块煎饼

**Problem 1.2.3**

给定一个数列$\langle a_1, ..., a_2, a_n\rangle$，对数列中的每个元素$a_i(1 \leq i \leq n)$，找到序列中位于$a_i$之前的元素即$\langle a_1, ..., a_{i-1}\rangle$ 中比$a_i$ 大的元素的最大下标，如果不存在这样的元素，则得到的下标为0。下面是一个求解这个问题的复杂度为$\Theta(n^2)$ 的算法。

**Algorithm 8:** PREVIOUS-LARGER($a[1..n]$)

**for** $i \leftarrow 1$ *to* $n$ **do**
  $j \leftarrow i - 1$;
  **while** $j > 0$ *and* $a[j] \leq a[i]$ **do**
    $j - -$;
  $p[i] \leftarrow j$;
**return** p;

导致这个算法不高效的一个比较明显的原因是语句j--，它使得每次只能向前推进一个元素。可以考虑利用已经得到的$p$中的值来提高算法的效率。(如果你不能立即明白这一点，可以尝试画图模拟这一过程。)请利用这个提示设计一个复杂度为$\Theta(n)$的算法并证明算法的正确性以及说明算法的复杂度。

**Problem 1.2.4**
假设现在需要颠倒句子中的所有单词的顺序，例如“My name is Chris”，颠倒句子中的所有单词得到“Chris is name My”。请给出相应的算法，算法的时间和空间复杂度越低越好。

**Problem 1.2.5**
给定一个数组，并将它分为左右两个部分。考虑将其左右两个部分互换位置。例如，$A = [1,2,3,4,5,6,7]$，左半部分有四个元素，其余为右半部分，交换左右两个部分的元素的结果为$A' = [5,6,7,1,2,3,4]$。请给出相应的算法。

**Problem 1.2.6 (最大和子序列)**
给定一个由一些整数组成的序列$S$，请找出和最大的连续子序列。例如，$S = \{-2, 11, -4, 13, -5, -2\}$，得到的结果应为$20 = 11 - 4 + 13$。

**Problem 1.2.7 (任务调度问题)**
假设我们有一堆任务需要解决，每个任务有指定的起始、终止时间。同时，由于任务的解决需要互斥地使用一些资源（例如打印机等），我们需要挑出时间两两互不重叠的任务集合来。我们定义其中的任务两两互不重叠的任务集合为相容的（compatible），且定义任务集合的大小为其中包含的任务的个数。我们的问题就是找出最大的相容的任务集合（不一定唯一）。记任务集合$A = \{\, a_1, a_2, \cdots, a_n \,\}$，每个任务定义为起始时间和终止时间的区间为$a_i = [s_i, f_i)$。我们定义任务调度问题为：

| | |
|---|---|
| 输入： | 任务集合$A$ |
| 输出： | 最大的相容任务集合$S$ |

**Problem 1.2.8 (矩阵链相乘问题)**
给定一个矩阵序列，我们要计算它们的乘积$A_1 \cdot A_2 \cdots A_n$，这里假设相邻两个矩阵的行列数值是可以相乘的。由于矩阵相乘满足结合律，所以无论我们以何种次序进行相乘，最终的结果是一样的。但是不同的相乘次序，相乘的代价可以有很大的差别。首先我们来计算两个矩阵相乘的代价。假设矩阵$C_{p\times r} = A_{p\times q} \times B_{q\times r}$，其中的元素$c_{i,j}$ 为：

$$c_{i,j} = \sum_{k=1}^{q} a_{ik} b_{kj} \quad (1 \le i \le p, 1 \le j \le r)$$

我们假设计算两个元素乘积的代价为1，则计算矩阵$C$中一个元素的开销为$q$。因为$C$中有$p \cdot r$个元素，则计算矩阵$C$的总开销为$p \cdot q \cdot r$，恰为两个相乘的矩阵所涉及到的三个维度值的乘积。

根据两个矩阵相乘的代价，我们很容易发现不同的矩阵相乘次序对矩阵链相乘的总开销影响很大。例如对于矩阵链：

$$\underbrace{A_1}_{30\times 1} \times \underbrace{A_2}_{1\times 40} \times \underbrace{A_3}_{40\times 10} \times \underbrace{A_4}_{10\times 25}$$

不同相乘次序的开销分别为：

$$\begin{aligned}
((A_1 \times A_2) \times A_3) \times A_4 &\quad : \quad 20700 \\
A_1 \times (A_2 \times (A_3 \times A_4)) &\quad : \quad 11750 \\
(A_1 \times A_2) \times (A_3 \times A_4) &\quad : \quad 41200 \\
A_1 \times ((A_2 \times A_3) \times A_4) &\quad : \quad 1400
\end{aligned}$$

我们发现最大开销是最小开销的约30倍。

因而对于给定矩阵序列的相乘，我们面临这样一个算法问题：给定一个矩阵序列，计算它们相乘的最低开销及相应的相乘次序。为了计算的方便，我们的算法中直接处理的是矩阵的行列的值，假设链表$D =< d_0, d_1, \cdots, d_n >$存放了所有矩阵的行列值，其中矩阵$A_i$为$d_{i-1} \times d_i (1 \le i \le n)$的矩阵。我们的算法问题为：

| | |
|---|---|
| 输入： | 矩阵序列对应的行列值序列$D =< d_0, d_1, \cdots, d_n >$ |
| 输出： | 计算矩阵序列相乘的最小代价及相应相乘次序 |

**Problem 1.2.9 (微博名人问题)**
给定$n$个人。我们称一个人为“微博名人”，如果他被其他所有人微博关注，但是自己不关注任何人。为了从给定的$n$个人中找出名人，我们唯一可以进行的操作是：针对两个人A和B，询问“A 是否微博关注B”。答案只可能是YES（A关注B）或者NO（A不关注B）。

1. 在一群共n个人中，可能有多少个名人？
2. 请设计一个算法找出名人（你可以很容易地得出一个brute force算法，然后尝试改进它）。

**Problem 1.2.10**
函数$\lceil log n \rceil !$是否多项式有界？函数$\lceil log log n \rceil !$呢？请给出证明。

**Problem 1.2.11**
请在渐进意义下比较$f(n), g(n)$。

$$f(n) = n^{\log n},\ g(n) = (\log n)^n$$

**Problem 1.2.12**
给定算法的时间复杂度的递归式$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$，但是它并不满足Master定理所需要的形式。请给出算法的时间复杂度。

**Problem 1.2.13**
给定递归表达式$T(n) = aT(\frac{n}{b}) + f(n)$。选择合适的$a$, $b$和$f(n)$使得Master定理的三个case都不能应用与求解该递归表达式。