任选5题，必须包含必选题

1.1 计算模型

Problem 1.1.1 (三个数排序)
输入三个各不相同的整数：

1. 请设计一个算法将输入的三个整数排序；
2. 在最坏情况下、平均情况下你的算法分别需要进行多少次比较？(假设所有可能的输入等概率出现。)
3. 在最坏情况下将三个不同整数排序至少需要多少次比较？证明你的结论。

Problem 1.1.2 (三个数的中位数)
输入三个各不相同的整数：

1. 请设计一个寻找三个数的中位数的算法；
2. 在最坏情况下、平均情况下你的算法分别需要进行多少次比较？(假设所有可能的输入等概率出现。)
3. 在最坏情况下找出三个不同整数的中位数至少需要多少次比较？证明你的结论。

Problem 1.1.3 (集合覆盖问题)
我们定义集合的最小覆盖如下：已知全集$U = \{1, ..., n\}$。给定U的子集组成的集合族$S = \{S_1, ..., S_m\}$，找出S的最小子集T，满足$\bigcup_{S_i \in T} S_i = U$。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$S_4 = \{2, 5\}$，则最小覆盖为$\{S_1, S_2\}$。

1. 找出下面算法失败的例子：首先选择S中最大的集合S_i，并从全集中将S_i中的所有元素删除；然后从S中剩余的集合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被覆盖；
2. 请设计一个算法计算输入全集的一个集合覆盖，并证明你所设计算法的正确性；
3. 你所设计的算法能否保证总是得出最小覆盖？如果不能，请针对你的算法设计一个反例。

Problem 1.1.4 (换硬币问题)
我们定义换硬币问题如下：给定n个硬币，它们的面值是正整数$S = \{s_1, s_2, ..., s_n\}$；另外给定一个正整数金额值$T$。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。我们给出两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. 按面值从大到小的顺序，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个算法的“不正确性”。

Problem 1.1.5 (子集和问题)
子集和(subset sum)问题定义如下：给定一个整数集合$S = \{s_1, s_2, ..., s_n\}$和一个目标数字$T$，要找$S$的一个子集使得子集中的整数的和为$T$。例如，$S = \{1, 2, 5, 9, 10\}$，存在子集其和为$T = 22$，但是不存子集其和为$T = 23$。请找出下列解决子集和问题的算法失败的例子：

1. 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”；

2. 将S中的元素按照从小到大排列，然后按照从小到大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“smallest-fit”；

3. S中的元素按照从大到小排列，然后按照从大到小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。我们称这个算法为我们称这个算法为“largest-fit”。

Problem 1.1.6 (多项式计算)
HORNER算法是用来求解多项式$P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ 的，请证明其正确性。

Algorithm: HORNER($A[0..n], x$)
$p := A[n]$; /* 数组$A[0..n]$存放系数$a_0, a_1, \cdots, a_n$ */
for $i := n - 1$ **downto** 0 **do**
  $p := p \cdot x + A[i]$;
return p ;

Problem 1.1.7 (整数相乘(必选))
INT-MULT算法用来计算两个非负整数y, z的乘积。

Algorithm: **int** INT-MULT(**int** y, **int** z)
if $z = 0$ **then**
  return 0;
else
  return multiply(cy,$\lfloor \frac{z}{c} \rfloor$) + y $\cdot$ (z mod c);

---

1. 令$c = 2$，请证明算法的正确性；

2. 令c为任意的一个不小于2的常数，请证明算法的正确性。

Problem 1.1.8 (平均情况时间复杂度的定义(必选))
算法的输入r为1到n之间的正整数，r取不同值的概率如下：

$$Pr\{r=i\}=\begin{cases}\frac{1}{n} & (1\le i\le \frac{n}{4})\\ \frac{2}{n} & (\frac{n}{4}<i\le \frac{n}{2})\\ \frac{1}{2n} & (\frac{n}{2}<i\le n)\end{cases}$$

假设n是4的倍数。请针对输入的取值情况，分析下面算法的平均情况时间复杂度(一个operation的代价记为1)。

---

if $r \le \frac{n}{4}$ **then**
    perform 10 operations ;
else if $\frac{n}{4} < r \le \frac{n}{2}$ **then**
    perform 20 operations ;
else if $\frac{n}{2} < r \le \frac{3n}{4}$ **then**
    perform 30 operations ;
else
    perform n operations ;

任选10题，必须包含必选题

1.2 数学基础

Problem 1.2.1
请计算满足下面两个条件的实数x的区间：

- $1 < x < 2$；
- $\lfloor x^2 \rfloor = \lfloor x \rfloor^2$。

Problem 1.2.2 (取整运算的性质(必选))
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 1.2.3
对于斐波那契数列，请证明：

1. F_n是偶数当且仅当n能被3整除；
2. $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$。

Problem 1.2.4 (完美二叉树(必选))
给定一棵完美二叉树，记其节点数为n，高度为h，叶节点树为l，内部节点数为t。

1. 给定上述4个量中的任意一个，请推导出其它三个量；
2. 请计算完美二叉树任意一层的节点个数：
   (a) 如果指定任意深度为p的一层节点，请计算该层节点个数n_p $(0 \leq p \leq h)$；
   (b) 如果任意指定高度为q的一层节点，请计算该层节点个数n_q $(0 \leq q \leq h)$。

Problem 1.2.5 (二叉树的数学性质(必选))
对于一棵非空的二叉树T，记其中叶节点的个数为n_0，有一个子节点的节点个数为n_1，有两个子节点的节点个数为n_2。

- 如果T为一棵2-tree，请证明$n_0 = n_2 + 1$；
- 如果T为一棵任意二叉树，n_0和n_2是否满足上述关系，请证明你的结论。

Problem 1.2.6 (函数渐近增长率的基本性质)
请证明函数渐近增长率的如下性质：

1. $O, \Omega, \Theta, o, \omega$这五种关系均满足传递性；
2. O, Ω, Θ这三种关系满足自反性；
3. Θ是一个等价关系；
4. $f(n) = \Theta(g(n))$ iff. $f(n) = O(g(n))$且$f(n) = \Omega(g(n))$；

5. O, o和Ω, ω之间有一种对偶的关系，即：$f = O(g)$ iff. $g = \Omega(f)$, $f = o(g)$ iff. $g = \omega(f)$；

6. $o(g(n)) \cap \omega(g(n)) = \emptyset$, $\Theta(g(n)) \cap o(g(n)) = \emptyset$, $\Theta(g(n)) \cap \omega(g(n)) = \emptyset$。

Problem 1.2.7
请将下面的函数按照渐近增长率从低到高的顺序进行排序。如果有多个函数其渐近增长率相同，请予以指出。

1. 将下面的函数排序：

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

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

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

Problem 1.2.8
对于Fibonacci数列，请判断下面两个结论的正误，并证明你的结论：

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 1.2.9
对于大小为n的输入，假设在最坏情况下，算法一需要执行的步数为$f(n) = n^2 + 4n$，算法二需要执行的步数为$g(n) = 29n + 3$。当n为多少时，算法一比算法二快(在最坏情况下)？

Problem 1.2.10 ((必选))
请证明$\log(n!) = \Theta(n\log n)$。(注：你可以通过Stirling公式得到一个证明；如果不能使用Stirling公式，你能否得到另一个证明？)

Problem 1.2.11
请比较$f(n), g(n)$的渐近增长率：

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

Problem 1.2.12
精确求解递归式$T(n) = \sum_{i=1}^{n-1} T(i) + k$。其中$k$为常数，$T(1) = 1$。

Problem 1.2.13 ((必选))
计算$T(n)$的渐近增长率。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. $T(n) = T(n-1) + n^c$，常量$c \geq 1$
10. $T(n) = T(n-1) + c^n$，常量$c > 1$
11. $T(n) = T(n/2) + T(n/4) + T(n/8)$

Problem 1.2.14

假设$T(1) = 0$，且对所有$n \geq 2$，$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1$，请证明：

$$T(n) = n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1$$

Problem 1.2.15 (guess and prove (必选))

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。请计算算法的时间复杂度。

Problem 1.2.16 (Master定理的“空隙”(必选))

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定理的三种情况均不能应用于求解该递归表达式。

Problem 1.2.17

假设你可以选择如下3个算法来解决当前的问题：

- 算法A将问题划分为5个规模为一半的子问题，递归地解每个子问题，合并这些子问题需要线性时间复杂度；
- 算法B将一个规模为n的问题划分为两个规模为$n-1$的子问题，递归地解这两个子问题，合并这些子问题需要常数的时间复杂度；
- 算法C将规模为n的问题划分为9个规模为$\frac{n}{3}$的子问题，递归地解每个问题，合并这些子问题的时间复杂度是$O(n^2)$。

每个算法的时间复杂度是多少(以O 来表示)，你会选择哪一个算法？

Problem 1.2.18

请分别给出下面四个算法MYSTERY、PERSKY、PRESTIFEROUS和CONUNDRUM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: 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: 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: 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;
```

```
Algorithm: 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;
```

任选5题

1.3 蛮力算法

Problem 1.3.1

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。

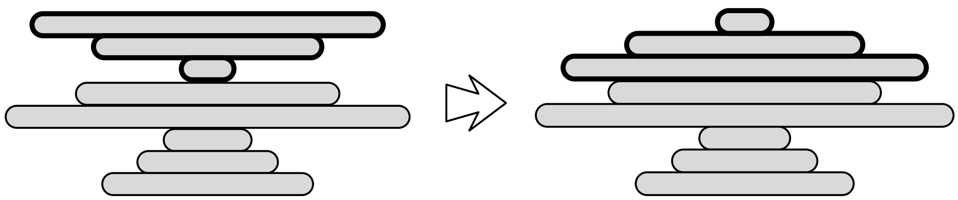

图 1.1: 翻转最上面的三块煎饼

Problem 1.3.2

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 1: 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 1.3.3

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 1.3.4

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 1.3.5 (最大和子序列(穿越问题))

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 1.3.6 (任务调度问题(穿越问题))

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 1.3.7 (矩阵链相乘问题(穿越问题))

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 1.3.8 (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）。