第3章 必备技能——基本算法设计方法

3.1 单项选择题及其参考答案

3.1.1 单项选择题

1. 穷举法的适用范围是_____。

A. 一切问题　　B. 解的个数极多的问题

C. 解的个数有限且可一一列举　　D. 不适合设计算法

2. 如果一个4位数恰好等于它的各位数字的4次方和，则这个4位数称为玫瑰花数。例如$1634=1^4+6^4+3^4+4^4$，则1634是一个玫瑰花数。若想求出4位数中所有的玫瑰花数，可以采用的解决方法是_____。

A. 递归法　　B. 穷举法　　C. 归纳法　　D. 都不适合

3. 有一个数列，递推关系是$a_1=\frac{1}{2}$，$a_{n+1}=\frac{a_n}{a_n+1}$，则求出的通项公式是_____。

A. $a_n=\frac{1}{n+1}$　　B. $a_n=\frac{1}{n}$　　C. $a_n=\frac{1}{2n}$　　D. $a_n=\frac{n}{2}$

4. 猜想$1=1$，$1-4=-(1+2)$，$1-4+9=1+2+3$，…的第5个式子是_____。

A. $1^2+2^2-3^2-4^2+5^2=1+2+3+4+5$

B. $1^2+2^2-3^2+4^2-5^2=-(1+2+3+4+5)$

C. $1^2-2^2+3^2-4^2+5^2=-(1+2+3+4+5)$

D. $1^2-2^2+3^2-4^2+5^2=1+2+3+4+5$

5. 对于迭代法，下面的说法不正确的是_____。

A. 需要确定迭代模型

B. 需要建立迭代关系式

C. 需要对迭代过程进行控制，要考虑什么时候结束迭代过程

D. 不需要对迭代过程进行控制

6. 设计递归算法的关键是_____。

A. 划分子问题　　B. 提取递归模型

C. 合并子问题　　D. 求解递归出口

7. 若一个问题的求解既可以用递归算法，也可以用迭代算法，则往往用__①__算法，因为__②__。

① A. 先递归后迭代　　B. 先迭代后递归

C. 递归　　D. 迭代

② A. 迭代的效率比递归高　　B. 递归宜于问题分解

C. 递归的效率比迭代高　　D. 迭代宜于问题分解

8. 一般来说，递归需要有递归出口和递归体，求解过程分为分解和求值，当到达递归出口时_____。

A. 进行运算　　B. 返回求值　　C. 继续分解　　D. 结束求解

9. 递归函数 $f(n)=f(n-1)+n(n>1)$的递归出口是_____。

A. $f(-1)=0$　　B. $f(1)=1$　　C. $f(0)=1$　　D. $f(n)=n$

10. 递归函数 $f(n)=f(n-1)+n(n>1)$的递归体是_____。

A. $f(1)=0$　　B. $f(1)=1$

C. $f(n)=n$　　D. $f(n)=f(n-1)+n$

11. 有以下递归算法，$f(123)$的输出结果是_____。

```
void fun(int n) {
    if(n>0) {
        System.out.printf("%d",n%10);
        f(n/10);
    }
}
```

A. 321　　B. 123　　C. 6　　D. 以上都不对

12. 有以下递归算法，$f(123)$的输出结果是_____。

```
void fun(int n) {
    if(n>0){
        f(n/10);
        System.out.printf("%d",n%10);
    }
}
```

A. 321　　B. 123　　C. 6　　D. 以上都不对

13. 整数单链表 h 是不带头结点的，结点类型 ListNode 为(val,next)，则以下递归算法中隐含的递归出口是_____。

```
void fun(ListNode h) {
    if(h!=null) {
        System.out.printf("%d",h->val);
        f(h.next);
    }
}
```

A. if(h!=null) return;　　B. if(h==null) return 0;

C. if(h==null) return;　　D. 没有递归出口

14. $T(n)$表示输入规模为 n 时算法的效率，以下算法中性能最优的是_____。

A. $T(n)=T(n-1)+1, T(1)=1$　　B. $T(n)=2n^2$

C. $T(n)=T(n/2)+1, T(1)=1$　　D. $T(n)=3n\log_2 n$

3.1.2 单项选择题参考答案

1. **答**：穷举法作为一种基本算法设计方法并非是万能的，主要适合于解个数有限且可一一列举的问题的求解。答案为 C。

2. **答**：4 位数的范围是 1000～9999，可以一一枚举。答案为 B。

3. **答**：采用不完全归纳法，$a_1=\frac{1}{2}$，$a_2=\frac{1}{3}$，$a_3=\frac{1}{4}$，$a_4=\frac{1}{5}$，…，$a_n=\frac{1}{n+1}$，可以采用数学归纳法证明其正确性。答案为 A。

4. **答**：推导如下。

$n=1,1^2=1$

$n=2,1-4=1-2^2=-(1+2)$

$n=3,1-4+9=1^2-2^2+3^2=1+2+3$

$n=3,1^2-2^2+3^2-4^2=-(1+2+3+4)$

$n=3,1^2-2^2+3^2-4^2+5^2=1+2+3+4+5$

答案为 D。

5. **答**：迭代法先确定迭代变量，再对迭代过程进行控制。答案为 D。

6. **答**：递归模型是递归算法的核心，反映递归算法的本质。答案为 B。

7. **答**：一般情况下求解同一个问题的迭代算法比递归算法的效率高。答案是①D ②A。

8. **答**：当到达递归出口时得到该子问题的解，然后返回求值。答案为 B。

9. **答**：$f(n)=f(n-1)+n(n>1)$是递归体，递归出口对应 $n=1$ 的情况。答案为 B。

10. **答**：$f(n)=f(n-1)+n(n>1)$本身就是递归体。答案为 D。

11. **答**：该算法属于先合后递的递归算法。在执行 $f(123)$时先输出 123%10=3，再调用 $f(12)$输出 2，最后调用 $f(1)$输出 1。答案为 A。

12. **答**：该算法属于先递后合的递归算法。执行过程是 $f(123)\rightarrow f(12)\rightarrow f(1)$，返回时依次输出 1，2 和 3。答案为 B。

13. **答**：任何递归算法都包含递归出口，该递归算法是正向输出单链表 h 中的结点值，递归出口是 h 为空的情况。答案为 C。

14. **答**：对于选项 A，采用直接展开法求出 $T(n)=\Theta(n)$。对于选项 B，$T(n)=\Theta(n^2)$。对于选项 C，采用主方法，$a=1,b=2,f(n)=O(1),n^{\log_b a}=1$ 与 $f(n)$的阶相同，则 $T(n)=\Theta(\log_2 n)$。对于选项 D，$T(n)=\Theta(n\log_2 n)$。答案为 C。

3.2 问答题及其参考答案

3.2.1 问答题

1. 用穷举法解题时的常用列举方法有顺序列举、排列列举和组合列举，问求解以下问题应该采用哪一种列举方法？

(1) 求 m～n 的所有素数。

(2) 在数组 a 中选择出若干元素，它们的和恰好等于 k。

(3) 有 n 个人合起来做一个任务，他们的排列顺序不同完成该任务的时间不同，求最优完成时间。

2. 许多系统用户登录时需要输入密码，为什么还需要输入已知的验证码？

3. 什么是递归算法？递归模型由哪两个部分组成？

4. 比较迭代算法与递归算法的异同。

5. 有一个含 $n(n>1)$ 个整数的数组 a，写出求其中最小元素的递归定义。

6. 有一个含 $n(n>1)$ 个整数的数组 a，写出求所有元素和的递归定义。

7. 利用整数的后继函数 succ 写出 $x+y$（x 和 y 都是正整数）的递归定义。

8. 有以下递归算法，则 $f(f(7))$ 的结果是多少？

```
int f(int n) {
    if(n<=3)
        return 1;
    else
        return f(n-2)+f(n-4)+1;
}
```

9. 有以下递归算法，则 $f(3,5)$ 的结果是多少？

```
int f(int x,int y) {
    if(x<=0 || y<=0)
        return 1;
    else
        return 3*f(x-1,y/2);
}
```

10. 采用直接展开法求以下递推式：

$$T(1)=1$$
$$T(n)=T(n-1)+n \quad \text{当 } n>1 \text{ 时}$$

11. 采用递归树方法求解以下递推式：

$$T(1)=1$$
$$T(n)=4T(n/2)+n \quad \text{当 } n>1 \text{ 时}$$

12. 采用主方法求解以下递推式：

(1) $T(n)=4T(n/2)+n$

(2) $T(n)=4T(n/2)+n^2$

(3) $T(n)=4T(n/2)+n^3$

13. 有以下算法，分析其时间复杂度。

```
void f(int n) {
    for(int i=1;i<=n;i++){
        for(int j=1;j<=i;j++)
            System.out.printf("%d %d %d\n",i,j,n);
    }
    if(n>0){
        for(int i=1;i<=4;i++)
            f(n/2);
    }
}
```

14. 分析《教程》3.4.3 节的求 1～n 全排列的递归算法 perm11(n,n)的时间复杂度。

15*. 有以下多项式：

$$f(x,n)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\cdots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}$$

给出求 $f(x,n)$值的递推式，分析其求解的时间复杂度。

3.2.2 问答题参考答案

1. **答**：(1) 采用顺序列举。

(2) 采用组合列举。

(3) 采用排列列举。

2. **答**：一般密码长度有限，密码由数字和字母等组成，可以采用穷举法枚举所有可能的密码，对每个密码进行试探。如果加入验证码，就会延迟每次试探的时间，从而使得这样破解密码变成几乎不可能。

3. **答**：递归算法是指直接或间接地调用自身的算法。递归模型由递归出口和递归体两部分组成。

4. **答**：迭代算法与递归算法的相同点是都是解决“重复操作”的机制，不同点是递归算法往往比迭代算法耗费更多的时间(调用和返回均需要额外的时间)与存储空间(用来保存不同次调用下变量的当前值的栈空间)，每个迭代算法原则上总可以转换成与它等价的递归算法，反之不然。

5. **答**：设 $f(a,i)$表示 $a[0..i]$(共 $i+1$ 个元素)中的最小元素，为大问题；设 $f(a,i-1)$表示 $a[0..i-1]$(共 i 个元素)中的最小元素，为小问题。对应的递归定义如下：

$$\begin{aligned} f(a,i)&=a[0] && \text{当 } i=0 \text{ 时}\\ f(a,i)&=\min(f(a,i-1),a[i]) && \text{其他}\end{aligned}$$

则 $f(a,n-1)$求数组 a 中 n 个元素的最小元素。

6. **答**：设 $f(a,i)$表示 $a[0..i]$(共 $i+1$ 个元素)中的所有元素和，为大问题；设 $f(a,i-1)$表示 $a[0..i-1]$(共 i 个元素)中的所有元素和，为小问题。对应的递归定义如下：

$$\begin{aligned} f(a,i)&=a[0] && \text{当 } i=0 \text{ 时}\\ f(a,i)&=f(a,i-1)+a[i] && \text{其他}\end{aligned}$$

则 $f(a,n-1)$求数组 a 中 n 个元素和。

7. **答**：设 $f(x,y)=x+y$，对应的递归定义如下。

$$\begin{aligned} f(x,y)&=y && \text{当 } x=0 \text{ 时}\\ f(x,y)&=x && \text{当 } y=0 \text{ 时}\\ f(x,y)&=f(\text{succ}(x),\text{succ}(y))+2 && \text{其他}\end{aligned}$$

8. **答**：先求 $f(7)$，如图 3.1(a)所示，求出 $f(7)=5$，再求 $f(5)$，如图 3.1(b)所示，求出 $f(5)=3$，所以 $f(f(7))$的结果是 3。

9. **答**：求 $f(3,5)$的过程如图 3.2 所示，求出 $f(3,5)=27$。

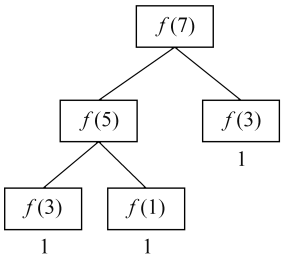
10. **答**：求 $T(n)$的过程如下。

$$\begin{aligned} T(n)&=T(n-1)+n=[T(n-2)+(n-1)]+n=T(n-2)+n+(n-1)\\ &=T(n-3)+n+(n-1)+(n-2)\end{aligned}$$

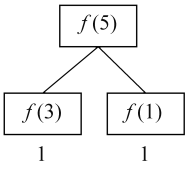
$=\cdots$

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

$=n+(n-1)+\cdots+2+1=n(n+1)/2=\Theta(n^2)$。



(a) 求$f(7)$的过程



(b) 求$f(5)$的过程

图 3.1 求 $f(f(7))$的过程

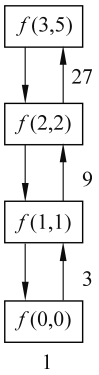


图 3.2 求 $f(3,5)$的过程

11. **解**：构造的递归树如图 3.3 所示，第 1 层的问题规模为 n，第 2 层的子问题的问题规模为 $n/2$，以此类推，当展开到第 $k+1$ 层时，其规模为 $n/2^k=1$，所以递归树的高度为 $\log_2 n+1$。

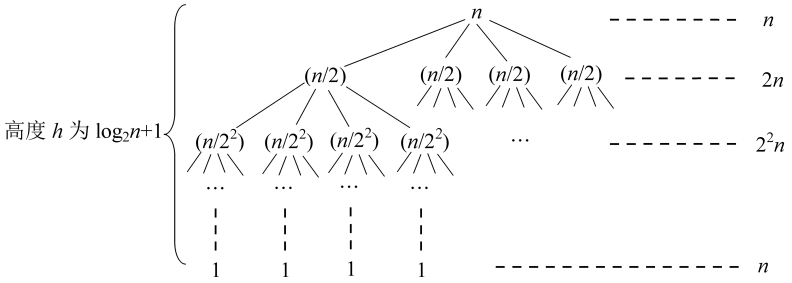


图 3.3 一棵递归树

第 1 层有一个结点，其时间为 n，第 2 层有 4 个结点，其时间为 $4(n/2)=2n$，以此类推，第 k 层有 4^{k-1} 个结点，每个子问题的规模为 $n/2^{k-1}$，其时间为 $4^{k-1}(n/2^{k-1})=2^{k-1}n$。叶子结点的个数为 n，其时间为 n。将递归树每一层的时间加起来，可得

$$T(n)=n+2n+\cdots+2^{k-1}n+\cdots+n\approx n*2^{\log_2 n}=\Theta(n^2)$$

12. **答**：(1) 这里 $a=4,b=2,f(n)=n$。$n^{\log_b a}=n^{\log_2 4}=n^2$，显然 $f(n)$的阶小于 $n^{\log_b a}$，满足主方法中的情况①，所以 $T(n)=\Theta(n^{\log_b a})=\Theta(n^2)$。

(2) 这里 $a=4,b=2,f(n)=n^2$。$n^{\log_b a}=n^{\log_2 4}=n^2$，显然 $f(n)$与 $n^{\log_b a}$ 同阶，满足主方法中的情况②，$T(n)=\Theta(n^{\log_b a}\log_2 n)=\Theta(n^2\log_2 n)$。

(3) 这里 $a=4,b=2,f(n)=n^3$。$n^{\log_b a}=n^{\log_2 4}=n^2$，显然 $f(n)$的阶大于 $n^{\log_b a}$。另外，对于足够大的 n，$af(n/b)=4\times n^3/8=n^3/2\leqslant cf(n)$，这里 $c=1/2$，满足正规性条件，则有 $T(n)=\Theta(f(n))=\Theta(n^3)$。

13. **答**：算法中两重 for 的执行次数为 $\sum_{i=1}^{n}\sum_{j=1}^{i}1=\sum_{i=1}^{n}i=\frac{n(n+1)}{2}=\Theta(n^2)$。对应的时

间递推式如下：

$$T(n)=1 \quad \text{当 } n=0 \text{ 时}$$
$$T(n)=4T(n/2)+n^2 \quad \text{其他情况}$$

采用主方法，$a=4,b=2,f(n)=n^2,n^{\log_b a}=n^2$ 与 $f(n)$ 的阶相同，则 $T(n)=\Theta(n^2\log_2 n)$。

14. **答**：perm11(n,n)用于求 1～n 的全排列 P_n，设其执行时间为 $T(n)$，它首先调用 perm1(n,$n-1$)求出 1～$n-1$ 的全排列 P_{n-1}，该小问题的执行时间为 $T(n-1)$，再对 P_{n-1} 中每个集合元素(共$(n-1)!$个集合元素)的每个位置(共 n 个位置)插入 n，合并结果得到 P_n，执行次数为 $n(n-1)!=n!$。所以递推式如下：

$$T(1)=1 \quad \text{求 } P_1 \text{ 为常量}$$
$$T(n)=T(n-1)+n! \quad \text{当 } n>1 \text{ 时}$$

令 $T(n)=n!g(n)$(因为 1～n 的全排列中的排列个数为 $n!$)，则 $T(n-1)=(n-1)!g(n-1)$，代入 $T(n)=T(n-1)+n!$中得到：

$$n!g(n)=(n-1)!g(n-1)+n!$$

两边乘以 n：$nn!g(n)=n(n-1)!g(n-1)+nn!=n!g(n-1)+nn!$。

两边除以 $n!$：$ng(n)=g(n-1)+n$，得到如下递推式。

$$g(1)=1$$
$$g(n)=g(n-1)/n+1 \quad \text{当 } n>1 \text{ 时}$$

采用直接展开法，$g(n)=g(n-1)/n+1=g(n-2)/(n(n-1))+1/n+1=\cdots\leqslant 2$。

$$T(n)=n!g(n)\leqslant 2n!$$

因此有 $T(n)=O(n!)$。

15. **答**：为了简单，省略 x 参数。

$$f(1)=x,g(1)=x$$
$$f(2)=x-\frac{x^3}{3!}=f(1)+(-1)\times g(1)\times\frac{x^2}{3\times 2},$$
$$g(2)=(-1)\times g(1)\times\frac{x^2}{2\times 3}$$
$$f(3)=x-\frac{x^3}{3!}+\frac{x5}{5!}=f(2)+(-1)\times g(2)\times\frac{x^2}{4\times 5},$$
$$g(3)=(-1)\times g(2)\times\frac{x^2}{4\times 5}$$

可以推出求 $f(n)$的递推式如下：

$$f(1)=x,\quad g(1)=x$$
$$g(n)=(-1)\times g(n-1)\times\frac{x^2}{(2n-2)(2n-1)}$$
$$f(n)=f(n-1)+g(n)$$

设求 $f(n)$的执行时间为 $T(n)$，求 $f(n)$需要求出 $g(n)$，但它们是同时计算的，也就是说 $T(n)$表示的是求 $f(n)$和 $g(n)$的时间，对应的执行时间的递推式如下：

$$T(1)=O(1)$$
$$T(n)=T(n-1)+O(1) \quad \text{当 } n>1 \text{ 时}$$

可以推出 $T(n)=O(n)$。

3.3 算法设计题及其参考答案

3.3.1 算法设计题

1. 有 3 种硬币若干个，面值分别是 1 分、2 分、5 分，如果要凑够 1 毛 5，设计一个算法求有哪些组合方式，共有多少种组合方式。

2. 有一个整数序列是 0,5,6,12,19,32,52,…，其中第 1 项为 0，第 2 项为 5，第 3 项为 6，以此类推，采用迭代算法和递归算法求该数列的第 $n(n\geqslant 1)$ 项。

3. 给定一个正整数 $n(1\leqslant n\leqslant 100)$，采用迭代算法和递归算法求 $s=1+(1+2)+(1+2+3)+\cdots+(1+2+\cdots+n)$。

4. 一个数列的首项 $a_1=0$，后续奇数项和偶数项的计算公式分别为 $a_{2n}=a_{2n-1}+2$，$a_{2n+1}=a_{2n-1}+a_{2n}-1$，设计一个递归算法求数列的第 n 项。

5. 设计一个递归算法用于翻转一个非空字符串 s(用 String 表示)。

6. 对于不带头结点的非空整数单链表 h，设计一个递归算法求其中值为 x 的结点的个数。

7. 对于不带头结点的非空单链表 h，设计一个递归算法删除其中第一个值为 x 的结点。

8. 对于不带头结点的非空单链表 h，设计一个递归算法删除其中所有值为 x 的结点。

9. 假设二叉树采用二叉链存储结构存放，结点值为整数，设计一个递归算法求二叉树 b 中所有叶子结点值之和。

10. 假设二叉树采用二叉链存储结构存放，结点值为整数，设计一个递归算法求二叉树 b 中第 k(1≤k≤二叉树 b 的高度)层的所有结点值之和(根结点的层次为 1)。

11. 设计将十进制正整数 n 转换为二进制数的迭代算法和递归算法。

12. 在《教程》的 3.2.2 节中采用迭代算法实现直接插入排序，请设计等效的递归算法。

13. 在《教程》的 3.3.2 节中采用迭代算法实现简单选择排序，请设计等效的递归算法。

14. 在《教程》的 3.4.2 节中采用递归算法实现冒泡排序，请设计等效的迭代算法。

15. 在《教程》的 3.3.4 节中采用迭代算法求 1～n 的幂集，请设计等效的递归算法。

16. 给定一个含 n 个元素的整数序列 a，设计一个算法求其中两个不同元素相加的绝对值的最小值。

17. 采用非递归和递归算法创建一个 $n(1\leqslant n\leqslant 10)$阶螺旋矩阵并输出。例如，$n=4$ 时的螺旋矩阵如下：

```
1   2   3   4
12  13  14  5
11  16  15  6
10  9   8   7
```

3.3.2 算法设计题参考答案

1. **解**：采用穷举法，设所需 1 分、2 分和 5 分硬币的个数分别为 i、j 和 k，显然有 $0 \leqslant i \leqslant 15$，$0 \leqslant j \leqslant 7$，$0 \leqslant k \leqslant 3$，约束条件 $i+2j+5k=15$，用 cnt 表示组合方式数。对应的算法如下：

```
int solve() {           //求解算法
    int cnt=0;
    for(int i=0;i<=15;i++) {
        for(int j=0;j<=7;j++) {
            for(int k=0;k<=3;k++) {
                if(i+(2*j)+(5*k)==15) {
                  System.out.printf("1分硬币%d个,2分硬币%d个,5分硬币%d个\n",i,j,k);
                  cnt++;
                }
            }
        }
    }
    return cnt;
}
```

2. **解**：设 $f(n)$为数列的第 n 项，则

$$f(1)=0$$
$$f(2)=5$$
$$f(3)=6=f(1)+f(2)+1$$
$$f(4)=12=f(2)+f(3)+1$$
$$\cdots$$

可以归纳出当 $n>2$ 时有 $f(n)=f(n-2)+f(n-1)+1$。对应的迭代算法如下：

```
int sequence1(int n) {          //迭代算法
    int a=0,b=5,c=0;
    if(n==1)
        return a;
    else if(n==2)
        return b;
    else {
        for(int i=3;i<=n;i++) {
            c=a+b+1;
            a=b;b=c;
        }
        return c;
    }
}
```

对应的递归算法如下：

```
int sequence2(int n) {          //递归算法
    if(n==1)
```

```
        return 0;
    else if(n==2)
        return 5;
    else
        return sequence2(n-2)+sequence2(n-1)+1;
}
```

3. **解**：设 curs=1+2+…+(i-1)，则 curs+i 便是 1+2+…+i，用 ans 累加所有的 curs(初始为 0)。对应的迭代算法如下：

```
int Sum1(int n){                    //迭代算法
    int ans=0;
    int curs=0;
    for(int i=1;i<=n;i++){
        curs+=i;
        ans+=curs;
    }
    return ans;
}
```

设 $f(n)=1+(1+2)+(1+2+3)+\cdots+(1+2+\cdots+n)$，则 $f(n-1)=1+(1+2)+(1+2+3)+\cdots+(1+2+\cdots+n-1)$，两式相减得到 $f(n)-f(n-1)=(1+2+\cdots+n)=n(n+1)/2$，则递归模型如下：

$$f(1)=1$$
$$f(n)=f(n-1)+n(n+1)/2$$

对应的递归算法如下：

```
int Sum2(int n){                    //递归算法
    if(n==1)
        return 1;
    else
        return Sum2(n-1)+n*(n+1)/2;
}
```

4. **解**：设 $f(m)$计算数列的第 m 项。当 m 为偶数时，不妨设 $m=2n$，则 $2n-1=m-1$，所以有 $f(m)=f(m-1)+2$。当 m 为奇数时，不妨设 $m=2n+1$，则 $2n-1=m-2$，$2n=m-1$，所以有 $f(m)=f(m-2)+f(m-1)-1$。对应的递归算法如下：

```
int sequence(int m) {                       //递归算法
    if(m==1)
        return 0;
    else if(m%2==0)
        return sequence(m-1)+2;
    else
        return sequence(m-2)+sequence(m-1)-1;
}
```

5. **解**：设 $f(\text{str},i)$返回 $s[i..n-1]$(共 $n-i$ 个字符)的翻转字符串，为大问题；$f(\text{str},i+1)$返回 $s[i..n-1]$(共 $n-i-1$ 个字符)的翻转字符串，为小问题；$i \geqslant n$ 时空串的翻转结果是空串。对应的递归模型如下：

$$f(s,i)=\text{""} \quad \text{当 } i \geqslant n \text{ 时}$$
$$f(s,i)=f(s,i+1)+s[i] \quad \text{其他情况}$$

对应的递归算法如下：

```
String reverse1(String s,int i) {          //递归算法
    if(i>=s.length())
        return "";
    else
        return reverse1(s,i+1)+s.charAt(i);
}
String reverse(String s) {                 //求解算法
    return reverse1(s,0);
}
```

6. **解**：设 $f(h,x)$返回单链表 h 中值为 x 的结点的个数，为大问题；$f(h.\text{next},x)$返回子单链表 $h.\text{next}$ 中值为 x 的结点的个数，为小问题；空单链表的结点的个数为 0。对应的递归模型如下：

$$f(h,x)=0 \quad \text{当 } h=\text{null 时}$$
$$f(h,x)=f(h.\text{next},x)+1 \quad \text{当 } h.\text{val}=x \text{ 时}$$
$$f(h,x)=f(h.\text{next},x) \quad \text{其他情况}$$

对应的递归算法如下：

```
int Count(ListNode h,int x) {              //求解算法
    if(h==null)
        return 0;
    else if(h.val==x)
        return Count(h.next,x)+1;
    else
        return Count(h.next,x);
}
```

7. **解**：设 $f(h,x)$返回删除单链表 h 中第一个值为 x 的结点后的单链表，为大问题；$f(h.\text{next},x)$返回删除子单链表 $h.\text{next}$ 中第一个值为 x 的结点后的单链表，为小问题。对应的递归模型如下：

$$f(h,x)=\text{null} \quad \text{当 } h=\text{null 时}$$
$$f(h,x)=h.\text{next} \quad \text{当 } h.\text{val}=x \text{ 时}$$
$$f(h,x)=h(h.\text{next}=f(h.\text{next},x)) \quad \text{其他情况}$$

对应的递归算法如下：

```
ListNode Delfirstx(ListNode h,int x) {     //递归算法
    if(h==null) return null;
    if(h.val==x)
```

```
        return h.next;
    else {
        h.next=Delfirstx(h.next,x);
        return h;
    }
}
```

8. **解**：设 $f(h,x)$返回删除单链表 h 中所有值为 x 的结点后的单链表，为大问题；$f(h.\text{next},x)$返回删除子单链表 $h.\text{next}$ 中所有值为 x 的结点后的单链表，为小问题。对应的递归模型如下：

$$
\begin{aligned}
&f(h,x)=\text{null} && \text{当 } h=\text{null} \text{ 时}\\
&f(h,x)=f(h.\text{next},x) && \text{当 } h.\text{val}=x \text{ 时}\\
&f(h,x)=h(h.\text{next}=f(h.\text{next},x)) && \text{其他情况}
\end{aligned}
$$

对应的递归算法如下：

```
ListNode Delallx(ListNode h,int x) {          //递归算法
    if(h==null) return null;
    if(h.val==x)
        return Delallx(h.next,x);
    else {
        h.next=Delallx(h.next,x);
        return h;
    }
}
```

9. **解**：设 $f(b)$返回二叉树 b 中所有叶子结点值之和，为大问题，$f(b.\text{left})$和$f(b.\text{right})$分别返回二叉树 b 的左、右子树中所有叶子结点值之和，为两个小问题。对应的递归模型如下：

$$
\begin{aligned}
&f(b)=0 && \text{当 } b=\text{null} \text{ 时}\\
&f(b)=b.\text{val} && \text{当 } b \text{ 结点为叶子结点时}\\
&f(b)=f(b.\text{left})+f(b.\text{right}) && \text{其他}
\end{aligned}
$$

对应的递归算法如下：

```
int LeafSum(TreeNode b) {             //递归算法
    if(b==null) return 0;
    if(b.left==null && b.right==null)
        return b.val;
    int lsum=LeafSum(b.left);
    int rsum=LeafSum(b.right);
    return lsum+rsum;
}
```

10. **解**：设 $f(b,h,k)$返回二叉树 b 中第 k 层的所有结点值之和(初始时 b 指向根结点，h 置为 1 表示结点 b 的层次)。其递归模型如下：

$f(b,h,k)=0$ 当 $b=$null 时

$f(b,h,k)=b.\text{val}$ 当 $h=k$ 时

$f(b,h,k)=f(b.\text{left},h+1,k)+f(b.\text{right},h+1,k)$ 当 $h<k$ 时

$f(b,h,k)=0$ 其他情况

对应的递归算法如下：

```
int LevelkSum1(TreeNode b,int h,int k) {        //递归算法
    if(b==null)
        return 0;
    if(h==k)
        return b.val;
    if(h<k) {
        int lsum=LevelkSum1(b.left,h+1,k);
        int rsum=LevelkSum1(b.right,h+1,k);
        return lsum+rsum;
    }
    else return 0;
}
int LevelkSum(TreeNode b,int k) {               //求解算法
    return LevelkSum1(b,1,k);
}
```

11. **解**：设 $f(n)$为 n 的二进制数。求出 $n\%2$ 和 $n/2$，$n\%2$ 作为结果二进制数的最高位，$f(n/2)$作为小问题。假设 $f(n/2)$已经求出，将 $n\%2$ 作为其最高位得到 $f(n)$的结果。对应的递归模型如下：

$$f(n)=\text{空} \quad \text{当 } n\leqslant 0 \text{ 时}$$

$$f(n)=n\%2 \oplus f(n/2) \quad \text{当 } n>0 \text{ 时}$$

其中 $x\oplus y$ 表示将 x 作为 y 的最高位。

采用数组存放转换的二进制数，每个元素表示一个二进制位，由于需要将 $n\%2$ 的二进制位插入最前面，为此改为用 Stack<Integer>集合存放转换的二进制数。对应的迭代算法如下：

```
Stack<Integer>trans1(int n) {                   //迭代算法
    Stack<Integer>ans=new Stack<>();
    while(n>0) {
        int d=n%2;                              //求出二进制位 d
        ans.push(d);                            //将 d 作为高位的元素
        n/=2;                                   //新值取代旧值
    }
    return ans;
}
```

对应的递归算法如下：

```
ArrayList<Integer>trans2(int n) {               //递归算法
    ArrayList<Integer>ans=new ArrayList<>();
    if(n<=0) return ans;
```

```
    ans=trans2(n/2);     //先递后合
    int d=n%2;
    ans.add(d);
    return ans;
}
```

12. **解**：直接插入排序递归算法的设计思路参考《教程》中的 3.2.2 节。采用先递后合和先合后递的两种递归算法如下：

```
void Insert(int R[],int i) {              //将 R[i]有序插入 R[0..i-1]中
    int tmp=R[i];
    int j=i-1;
    do {                                  //找 R[i]的插入位置
        R[j+1]=R[j];                      //将大于 R[i]的元素后移
        j--;
    } while(j>=0 && R[j]>tmp);            //直到 R[j]<=tmp 为止
    R[j+1]=tmp;                           //在 j+1 处插入 R[i]
}
/***先递后合算法*****************************/
void InsertSort21(int R[],int i) {        //递归直接插入排序
    if(i==0) return;
    InsertSort21(R,i-1);
    if(R[i]<R[i-1])                       //反序时
        Insert(R,i);
}
void InsertSort2(int R[]) {               //递归算法:直接插入排序
    int n=R.length;
    InsertSort21(R,n-1);
}
/***先合后递算法*****************************/
void InsertSort31(int R[],int i) {        //递归直接插入排序
    int n=R.length;
    if(i<1 || i>n-1) return;
    if(R[i]<R[i-1])                       //反序时
        Insert(R,i);
    InsertSort31(R,i+1);
}
void InsertSort3(int R[]) {               //递归算法:直接插入排序
    InsertSort31(R,1);
}
```

13. **解**：简单选择排序递归算法的设计思路参考《教程》中的 3.3.2 节。采用先递后合和先合后递的两种递归算法如下：

```
void Select(int R[],int i) {              //在 R[i..n-1]中选择最小元素交换到 R[i]位置
    int minj=i;                           //minj 表示 R[i..n-1]中最小元素的下标
    for(int j=i+1;j<R.length;j++) {       //在 R[i..n-1]中找最小元素
        if(R[j]<R[minj])
```

```
            minj=j;
        }
        if(minj!=i) {                       //若最小元素不是 R[i]
            int tmp=R[minj];                //交换 R[minj]和 R[i]
            R[minj]=R[i]; R[i]=tmp;
        }
    }
    /***先递后合算法*****************************/
    void SelectSort21(int R[],int i) {      //递归的简单选择排序
        if(i==-1) return;                   //满足递归出口条件
        SelectSort21(R,i-1);
        Select(R,i);
    }
    void SelectSort2(int R[]){              //递归的简单选择排序
        SelectSort21(R,R.length-2);
    }
    /***先合后递算法*****************************/
    void SelectSort31(int R[],int i) {      //递归的简单选择排序
        int n=R.length;
        if(i==n-1) return;                  //满足递归出口条件
        Select(R,i);
        SelectSort31(R,i+1);
    }
    void SelectSort3(int R[]) {             //递归的简单选择排序
        SelectSort31(R,0);
    }
```

14. **解**：冒泡排序迭代算法的设计思路参考《教程》中的 3.4.2 节。对应的算法如下：

```
boolean exchange;                       //类变量
void Bubble(int R[],int i) {            //在 R[i..n-1]中冒泡最小元素到 R[i]位置
    int n=R.length;
    for(int j=n-1;j>i;j--) {            //无序区元素的比较,找出最小元素
        if(R[j-1]>R[j]) {               //当相邻元素反序时
            int tmp=R[j];               //R[j]与 R[j-1]进行交换
            R[j]=R[j-1]; R[j-1]=tmp;
            exchange=true;              //本趟排序发生交换,置 exchange 为 true
        }
    }
}
void BubbleSort1(int R[]) {             //迭代算法:冒泡排序
    int n=R.length;
    for(int i=0;i<n-1;i++) {            //进行 n-1 趟排序
        exchange=false;                 //本趟排序前置 exchange 为 false
        Bubble(R,i);
        if(exchange==false)             //本趟未发生交换时结束算法
            return;
    }
}
```

15. **解**：用 M_i 表示 1～i 的幂集，对应的递归模型如下。

$$M_1=\{\{\},\{1\}\}$$

$$M_i=M_{i-1}\cup A_i \quad 当 i>1 时$$

其中，A_i=appendi(M_{i-1},i)。幂集用 List<List<Integer>>类型的 Java 集合存放，其中每个 List<Integer>类型的元素表示幂集中的一个集合。大问题是求{1～i}的幂集，小问题是求{1～$i-1$}的幂集。采用先递后合和先合后递的两种递归算法如下：

```
List<List<Integer>>deepcopy(List<List<Integer>>A) { //返回 A 的深拷贝
    List<List<Integer>>B=new ArrayList<>();
    for(List<Integer>x : A)
        B.add(new ArrayList<>(x));
    return B;
}
List<List<Integer>>appendi(List<List<Integer>>Mi_1,int i) {
//向 Mi_1 中每个集合元素的末尾添加 i
    List<List<Integer>>Ai=deepcopy(Mi_1);
    for(int j=0;j<Ai.size();j++)
        Ai.get(j).add(i);
    return Ai;
}
/***先递后合算法*****************************/
List<List<Integer>>pset(int n,int i){
    if(i==1) {
        List<List<Integer>>tmp=new ArrayList<>();  //建立 tmp={{},{1}}并返回
        List<Integer>e1=new ArrayList<>();
        tmp.add(e1);                               //添加{}
        List<Integer>e2=new ArrayList<>();
        e2.add(1);
        tmp.add(e2);                               //添加{1}
        return tmp;
    }
    else {
        List<List<Integer>>Mi_1=pset(n,i-1);       //递归求出 Mi_1
        List<List<Integer>>Mi=Mi_1;                //Mi 置为 Mi_1
        List<List<Integer>>Ai=appendi(Mi_1,i);
        for(int j=0;j<Ai.size();j++)               //将 Ai 的所有集合元素添加到 Mi 中
            Mi.add(Ai.get(j));
        return Mi;                                 //返回 Mi
    }
}
List<List<Integer>>subsets2(int n){                //递归算法
    return pset(n,n);
}
```

```
/***先合后递算法*****************************/
List<List<Integer>>pset(List<List<Integer>>M,int n,int i) {
    List<List<Integer>>A=deepcopy(appendi(M,i));  //求 A
    for(int j=0;j<A.size();j++)                   //将 A 的所有集合元素添加到 M 中
        M.add(A.get(j));
    if(i==n)                                      //已经求出结果时返回 M
        return M;
    else                                          //否则递归调用
        return pset(M,n,i+1);
}
List<List<Integer>>subsets3(int n) {              //递归算法
    List<List<Integer>>M=new ArrayList<>();
    List<Integer>e1=new ArrayList<>();
    M.add(e1);                                    //添加{}
    List<Integer>e2=new ArrayList<>();
    e2.add(1);
    M.add(e2);                                    //添加{1},置 M={{},{1}}
    if(n==1)
        return M;
    else
        return pset(M,n,2);
}
```

16. **解法 1**：采用穷举法，对任意两个不同元素求相加的绝对值，比较求最小值 ans，算法的时间复杂度为 $O(n^2)$。对应的算法如下：

```
int minabs1(int a[]) {                    //解法 1
    int n=a.length;
    int ans=0x3f3f3f3f;                   //初始置为∞
    for(int i=0;i<n-1;i++) {
        for(int j=i+1;j<n;j++) {
            ans=Math.min(ans,Math.abs(a[i]+a[j]));
            if(ans==0) return ans;        //当结果为 0 时不必继续
        }
    }
    return ans;
}
```

解法 2：改进穷举法算法。如果 a 中元素全部是正数，只需要找到其中两个不同的最小元素，答案就是它们相加的结果，对应的时间复杂度为 $O(n)$。但这里 a 中可能有负数，那么在这种情况下就变成了求差的绝对值，而差的绝对值最小的两个整数一定是大小最相近的，为此先对数组 a 递增排序，用 low 和 high 前后遍历，求 $f=a[\text{low}]+a[\text{high}]$，将最小绝对值保存在 ans 中，如果 $f>0$，除去 $a[\text{high}]$；如果 $f<0$，除去 $a[\text{low}]$。算法的时间主要花费在排序上，时间复杂度为 $O(n\log_2 n)$。对应的算法如下：

```
int minabs2(int a[]) {                    //解法 2
    int n=a.length;
```

```
        int ans=0x3f3f3f3f;                    //初始置为∞
        Arrays.sort(a);                        //递增排序
        int low=0,high=n-1;
        while(low<high) {
            int f=a[low]+a[high];
            ans=Math.min(ans,Math.abs(f));
            if(ans==0) return ans;             //当结果为 0 时不必继续
            if(f>0) high--;
            if(f<0) low++;
        }
        return ans;
    }
```

17. **解**：采用递归解法。设 $f(x,y,\text{start},n)$用于创建左上角为(x,y)、起始元素值为 start 的n 阶螺旋矩阵，共 n 行 n 列，它是大问题；$f(x+1,y+1,\text{start},n-2)$用于创建左上角为$(x+1,y+1)$ 、起始元素值为 start 的 $n-2$ 阶螺旋矩阵，共 $n-2$ 行 $n-2$ 列，它是小问题。图 3.4 所示为 $n=4$ 时的大问题和小问题。对应的递归模型如下：

$$
\begin{aligned}
f(x,y,\text{start},n) &\equiv \text{不做任何事情} && \text{当 } n \leqslant 0 \text{ 时}\\
f(x,y,\text{start},n) &\equiv \text{产生只有一个元素的螺旋矩阵} && \text{当 } n=1 \text{ 时}\\
f(x,y,\text{start},n) &\equiv \text{产生}(x,y)\text{ 的那一圈；} && \text{当 } n>1 \text{ 时}\\
& \quad f(x+1,y+1,\text{start},n-2)
\end{aligned}
$$

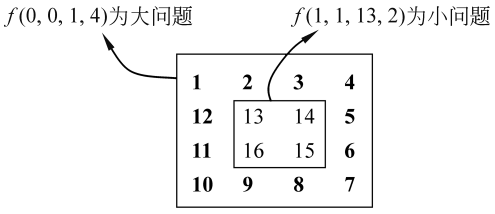


图 3.4　$n=4$ 时的大问题和小问题

非递归解法则是采用循环语句代替递归调用。对应的算法如下：

```
class Solution {
    int s[][];
    int n;
    int start;
    void CreateaLevel(int ix,int iy,int ex,int ey) {    //产生一圈的螺旋矩阵元素
        if(ix==ex)                                      //该圈只有一个元素时
            s[ix][iy]=start++;
        else {
            int curx=ix;
            int cury=iy;
            while(curx!=ex) {                           //上一行
                s[iy][curx]=start++;
                curx++;
            }
            while(cury!=ey) {                           //右一列
                s[cury][ex]=start++;
```

```
                cury++;
            }
            while(curx!=ix) {                    //下一行
                s[ey][curx]=start++;
                curx--;
            }
            while(cury!=iy) {                    //左一列
                s[cury][ix]=start++;
                cury--;
            }
        }
    }
    void Spiral1(int n){                         //非递归求解算法
        this.n=n;
        s=new int[n][n];
        start=1;
        int ix=0,iy=0;
        int ex=n-1,ey=n-1;
        while(ix<=ex && iy<=ey)
            CreateaLevel(ix++,iy++,ex--,ey--);
    }
    void Spiral2(int x,int y,int n) {            //递归创建螺旋矩阵
        if(n<=0)                                 //递归结束条件
            return;
        if(n==1) {                               //矩阵大小为1时
            s[x][y] =start;
            return;
        }
        CreateaLevel(x,y,x+n-1,y+n-1);           //产生一圈的螺旋矩阵元素
        Spiral2(x+1,y+1,n-2);                    //递归调用
    }
    void Spiral2(int n) {                        //递归求解算法
        this.n=n;
        s=new int[n][n];
        start=1;
        Spiral2(0,0,n);
    }
}
```

3.4 在线编程题及其参考答案

3.4.1 LeetCode647——回文子串★★

问题描述：给定一个字符串 s(1≤s.length≤1000，s 由小写英文字母组成)，设计一个算法求这个字符串中回文子串的数目。注意，具有不同开始位置或结束位置的子串，即使是由相同的字符组成的，也会被视作不同的子串。例如，s="aaa"，有 6 个回文子串，即 "a"、