第2章 分 治 法

当输入数据的规模很小时，比如只有一个或两个数字，则绝大多数的问题都很容易解。可是当输入规模增大时，问题往往变得很难。因此，算法设计的一个基本方法就是寻找大规模问题解与小规模问题解之间的关系。分治法（divide and conquer）是这种方法之一。后面要讲的贪心法和动态规划也是基于这种方法，但技巧上有不同之处。

2.1 分治法原理

简单地说，分治法（亦称分治术）的做法是将一个规模为 n 的问题分解为若干个规模小些的子问题，然后找出这些子问题或者一部分子问题的解并由此得到原问题的解。在解决这些子问题时，分治法要求用同样的方法递归解出。也就是说，我们必须能把这些子问题用同样的方法再分为更小的问题直至问题的规模小到可以直接解出。这个不断分解的过程看起来很复杂，但用递归的算法表达出来却往往简洁明了。通常，分治法只要讲明三件事即可：

1）底（bottom case）：对足够小的输入规模，如何直接解出。

2）分（divide）：如何将一个规模为 n 的输入分为整数个规模小些的子问题。

3）合（conquer）：如何从子问题的解中获得原规模为 n 的解。

下面我们看一个例子。

2.1.1 二元搜索的例子

假定要在一个已排好序的数组 $A[1] \leqslant A[2] \leqslant \cdots \leqslant A[n]$中查找是否有一个数等于要找的数 x，如果有 $A[i]=x$，$1 \leqslant i \leqslant n$，则报告序号 i，否则报告无（nil）。我们可以用以下的算法。

【例 2-1】二元搜索算法。

```
Binary-Search (A, p, r, x)
输入：  要寻找的数字 x 和数组 A 中任一段子序列，A[p] ≤ A[p+1] ≤ … ≤ A[r]。
输出：  i（如果 A[i] = x），否则 nil。
1   if p > r                    //说明被搜索的集合为空集
2       then return (nil)
3   endif
4   midpoint ← ⌊(p+r)/2⌋
5   if A[midpoint] = x
6       then return (midpoint)
7       else if x < A[midpoint]
8               then Binary-Search (A, p, midpoint–1, x)
9               else Binary-Search (A, midpoint+1, r, x)
10          endif
11  endif
12  End
```

注意，上面的算法只是一个子程序，搜索数组 $A[1]$, $A[2]$, …, $A[n]$时，主程序需要调用

Binary-Search $(A, 1, n, x)$。很清楚，上面算法用的是分治法。它先找出序列的中位数（即位于序列中间的数字）$A[midpoint]$，如果它等于 x，则问题解决。否则，原序列一分为二，前一半为数值小的子序列而后一半为数值大的子序列。如果 $x < A[midpoint]$，则只需要搜索数值小的子序列，否则搜索数值大的子序列。搜索子序列的算法是递归进行的。如果存在某个数 $A[i] = x$，则在某一步递归时会发现。否则，要搜索的子序列在某一步递归时变为空集。这时算法遇到了底（bottom case）而报告 *nil*。我们注意到，在上面算法中，输入的序列是从 $A[p]$到 $A[r]$，而不是 $A[1]$到 $A[n]$。这是因为分治或递归过程中，虽然递归算法本身的做法是一样的，但子问题所对应的输入数据（即子序列）的长短以及在原序列中的位置是变化的，我们必须用变量而不是常量来指明子问题搜索的范围，这是在写递归算法时要注意的地方。另外，递归算法只需说清楚相邻两层之间的关系即可，即说清楚如何把一个问题分为几个子问题。至于如何把子问题再分为更小的子问题是由计算机编译软件按照算法说明的分法自动完成，写算法时千万不要画蛇添足。

以后会看到，如果要搜索的数 x 等于序列中任一数的概率相等，那么二元搜索是最佳算法（见第 4 章习题），所以它是一个很重要的算法而受到广泛应用。因为许多分治法的算法需要先把输入数据排序等预处理，而排序在下一章才讲，这里就不多举例了。在后续的章节中我们还会看到许多用分治法设计的算法。

2.1.2 表示复杂度的递推关系

对一个用分治法设计的算法，该如何计算其复杂度呢？一般来说，可以先建立一个表示复杂度的递推关系，然后求解得到。以二元搜索算法为例，我们把序列 A 中的数和 x 之间的比较作为主要的基本运算，把序列 A 中数字的个数 n 作为输入的规模。假设在最坏情况下，对一个规模为 n 的序列，二元搜索算法所需的比较次数为 $T(n)$，那么我们有以下的递推关系：

$$\begin{aligned} T(0) &= 0 \\ T(n) &= T(\lfloor n/2 \rfloor) + 1 \end{aligned} \qquad (2.1)$$

这是因为当序列为空时，我们不需任何比较，而当序列非空时，我们作一次 x 和中位数的比较。如果它们相等，运算中止。但在最坏情况下，它们不相等而我们需要在含有$\lfloor n/2 \rfloor$个数字的子序列中继续递归。

对其他用分治法设计的算法，其复杂度均可以用类似的递推关系表示。假设某个分治算法把规模为 n 的输入数据分为规模为 n/b（b 为整数）的一组子问题，然后解出其中 a 个子问题，从而获得原问题的解。那么，该算法的复杂度 $T(n)$可以用下面的递推关系表示：

$$\begin{aligned} T(1) &= c \\ T(n) &= aT(n/b) + f(n) \end{aligned} \qquad (2.2)$$

其中 c 是一常数，$T(1) = c$ 表示算法在遇到底时（$n = 1$）所需时间为 c，而 $f(n)$是算法在做分解工作时所需要的时间以及在做合并工作时所需要的时间的总和，也就是除了解决子问题本身以外所需要的前序工作和收尾工作所花的时间总和。底的规模因算法不同而不同，有时会有 $T(2) = c$，$T(3) = c$ 等。我们以后会看到，算法在遇到底时所需的时间不影响算法的（渐近）复杂度。

绝大部分分治算法的复杂度均可用（2.2）式表示，但有些复杂度的递推关系会在形式上略有不同，例如 $T(n) = aT(\sqrt{n}) + f(n)$。这时往往需要先作一个变量替换，把它变为（2.2）的形式来解。在下节中我们探讨如何解（2.2）式这样的递推关系。

2.2 递推关系求解

在这一节中，我们探讨如何解（2.2）式这样的递推关系，递推关系的其他形式及更为全面的

一般性讨论可在离散数学书中找到。

2.2.1 替换法

这个方法是先猜想一个复杂度，然后用数学归纳法证明其正确性。

【例 2-2】确定由以下递推关系表示的算法复杂度。

当 $1 \leqslant n \leqslant 4$ 时，$T(n) = \Theta(1)$，否则为 $T(n) = 2T(\lfloor n/2 \rfloor) + n$。

解：我们猜想这个解是 $T(n) = \Theta(n\lg n)$。

先证明存在常数 c 使得当 $n \geqslant 4$ 时，$T(n) \leqslant cn\lg n$。

归纳基础：

当 $n = 4$ 时，因为 $T(n) = \Theta(1)$，存在常数 c 使 $T(n) \leqslant c \leqslant cn\lg n$。可假定 $c > 1$。

归纳步骤：

假定当 $n = 4, 5, \cdots, (k-1)$时，$T(n) \leqslant cn\lg n$，现在证明当 $n = k\ (k \geqslant 5)$时，$T(n) \leqslant cn\lg n$ 仍然正确。我们注意到，当 $n = k$ 时，$\lfloor n/2 \rfloor = \lfloor k/2 \rfloor \leqslant k - 1$。由归纳假设得到

$$T(\lfloor n/2 \rfloor) \leqslant c(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leqslant c(n/2)\lg(n/2) = (cn/2)(\lg n - 1)$$

进而从递推关系得到

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leqslant 2(cn/2)(\lg n - 1) + n \\ &= cn(\lg n - 1) + n \\ &= cn\lg n - cn + n \\ &\leqslant cn\lg n \qquad (\text{因为 } c > 1) \end{aligned}$$

这就证明了 $T(n) = O(n\lg n)$。

为了证明 $T(n) = \Omega(n\lg n)$，我们证明存在一常数 $d > 0$ 使得当 $n \geqslant 4$ 时，$T(n) \geqslant d(n\lg n + n)$。

归纳基础：

当 $n = 4$ 时，$(n\lg n + n) = 12$。因为 $T(n) = \Theta(1)$，所以存在常数 $c > 0$ 使 $T(n) \geqslant c$。那么，显然存在常数 d 使得 $0 < d \leqslant c/12$，从而有 $T(n) \geqslant c \geqslant 12d = d(n\lg n + n)$。可假定 $0 < d < 1/4$。

归纳步骤：

假定当 $n = 4, 5, \cdots, (k-1)$时，$T(n) \geqslant d(n\lg n + n)$成立。那么当 $n = k$ 时，因为$\lfloor n/2 \rfloor = \lfloor k/2 \rfloor \leqslant k-1$，所以由归纳假设得到：

$$\begin{aligned} T(\lfloor n/2 \rfloor) &\geqslant d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) + d(\lfloor n/2 \rfloor) \\ &\geqslant d(n/2 - 1)\lg(n/4) + d(n/2 - 1) \\ &= d(n/2 - 1)(\lg n - 2) + d(n/2 - 1) \\ &= (dn/2)\lg n + 2d - d\lg n - dn + d(n/2 - 1) \\ &= (dn/2)\lg n + d - d\lg n - dn/2 \\ &> (dn/2)\lg n - dn - dn/2 \qquad (\text{因为 } \lg n < n) \\ &= (dn/2)\lg n - 3dn/2 \end{aligned}$$

进而从递推关系得到

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\geqslant 2[(dn/2)\lg n - 3dn/2] + n \\ &= dn\lg n - 3dn + n \end{aligned}$$

$$> dn\lg n + dn \qquad (\text{因为 } d < 1/4)$$
$$= d(n\lg n + n)$$

这就证明了 $T(n) = \Omega(n\lg n + n) = \Omega(n\lg n)$，从而有 $T(n) = \Theta(n\lg n)$。 ■

用替换法解递推关系需要猜测出正确的复杂度，并且往往需要分开证明上界和下界。当然，很多时候我们只需对上界作出估计从而简化了工作。另外，该方法需要一定的数学技巧。本书不常用这个方法。下面看一个需要变量变换的例子。

【例 2-3】确定由以下递推关系表示的算法复杂度。

$$T(n) = 2T(\sqrt{n}) + \lg n$$

解：置 $n = 2^k$ 并代入原递推关系后得到

$$T(2^k) = 2T(2^{k/2}) + k$$

这样一来，T 是变量 n 的函数，而 n 又是 k 的函数。我们定义 $S(k) = T(2^k)$，那么 $T(2^{k/2}) = S(k/2)$。这样，得到一个关于变量 k 的递推关系：$S(k) = 2S(k/2) + k$。从例 2-2，我们知道 $S(k) = \Theta(k\lg k)$。因为 $n = 2^k$，$k = \lg n$，我们得到 $T(n) = T(2^k) = S(k) = \Theta(k\lg k) = \Theta(\lg n\lg\lg n)$。

2.2.2 序列求和法和递归树法

序列求和法和递归树法的本质是相同的，只不过递归树法用一棵树把序列产生的过程显示出来，而序列求和法中的序列是直接从递推关系一步一步展开后得到，然后，对该序列求和即可。下面看一个例子。

【例 2-4】用序列求和法确定由以下递推关系表示的算法复杂度。

$$T(1) = \Theta(1)$$
$$T(n) = 2T(n/2) + n\lg n$$

解：我们先置 $n = 2^k$ 把原递推关系简化，得到 $T(2^k) = 2T(2^{k-1}) + k2^k$。

定义 $W(k) = T(2^k)$后，我们得到 $W(k) = 2W(k-1) + k2^k$。下面我们一步一步展开这个递推关系。

$$\begin{aligned}
W(k) &= 2W(k-1) + k2^k \\
&= 2[2W(k-2) + (k-1)2^{k-1}] + k2^k \\
&= 2^2W(k-2) + (k-1)2^k + k2^k \\
&= 2^2[2W(k-3) + (k-2)2^{k-2}] + (k-1)2^k + k2^k \\
&= 2^3W(k-3) + (k-2)2^k + (k-1)2^k + k2^k \\
&\cdots \\
&= 2^{k-1}W(1) + 2\times 2^k + 3\times 2^k + \cdots + (k-1)2^k + k2^k \\
&= 2^{k-1}W(1) + 2^k[2 + 3 + \cdots + (k-2) + (k-1) + k] \\
&= 2^{k-1}W(1) + 2^k[k(k+1)/2 - 1] \\
&= \Theta(k^2 2^k) \qquad (\text{因为 } W(1) = \Theta(1))
\end{aligned}$$

因而我们得到：

$$T(n) = T(2^k) = W(k) = \Theta(k^2 2^k) = \Theta(n\lg^2 n)$$

从上面例子可看到，底的复杂度 $W(1)$不影响算法复杂度。这里，我们要解释一个问题。那就是，当变量 k 取整数值 1，2，3，…，k，…时，变量 n 取值为 2，4，8，…，2^k，…，也就是说，n 的取值是一些特殊的整数值且间隔会越来越大。那么以上结果还正确吗？是的，结果仍然正确。这是因为，对任一 n，存在一个 k，使得 $2^k \leqslant n < 2^{k+1}$。从递推关系可知，$T(n)$是一个单调递增函数，故有 $T(2^k) \leqslant T(n) \leqslant T(2^{k+1})$，或者 $\Theta(k^2 2^k) \leqslant T(n) \leqslant \Theta((k+1)^2 2^{k+1})$。因为$(k+1)^2 2^{k+1} = \Theta(k^2 2^k)$，

所以 $T(n) = \Theta(k^2 2^k) = \Theta(n\lg^2 n)$。

序列求和法中一步一步展开的过程可以用一棵递归树来描述。在递归树中，每个结点对应着序列中的一项，而所有结点的表达式的总和等于 $T(n)$。对应于以上例子的递归树在图 2-1 中给出。图 2-1a 显示的是第一步展开后的结果，表示 $W(k)$等于 $k2^k$ 加上两个 $W(k-1)$的总和。图 2-1b 显示的是第二步展开后的结果。这一步把第一步中叶结点的表达式再用递归关系展开一层。图 2-1c 显示的是经过（$k-2$）步展开后得到的树。这时每个叶结点代表一个底的情形，递归停止。从图中可以看到，树根的表达式即是序列求和中最右边一项，树中第一层两结点表达式之和就等于序列求和中右边第二项，树中第二层四结点表达式之和就等于序列求和中右边第三项。总之，递归树每向下发展一层就相当于序列求和法中展开一步。所以，这两种方法本质上相同。

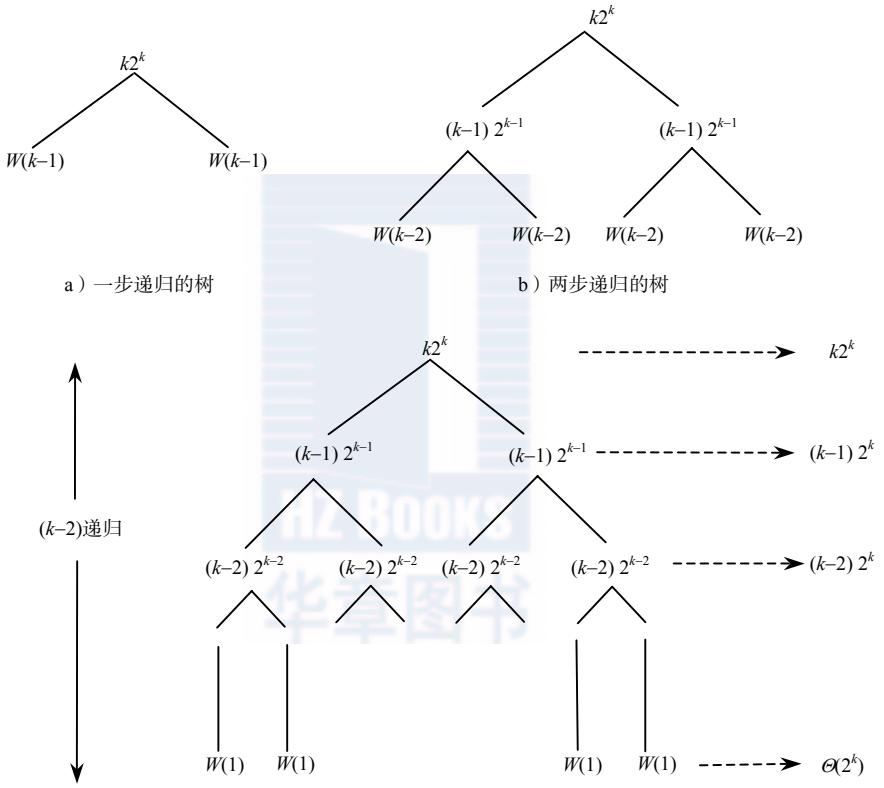


图 2-1　递推关系 $W(k) = 2W(k-1) + k2^k$ 对应的递归树

2.2.3　常用序列和公式

因为用序列求和法来解递归关系的关键是对序列求和，所以有必要熟悉一些常用的序列求和公式。下面是一些最常见的序列和（级数和）公式。它们的证明大部分已为读者熟知或留作练习而不在此赘述，我们只对个别的公式给予证明。

1．等差级数

一个序列$\{a_n\}$称为等差级数，如果 $a_{n+1} = a_n + d$，其中 d 为常数。等差级数的求和公式为 $\sum_{i=1}^{n} a_i =$

$a_1 + (a_1+d) + (a_1+2d) + \cdots + [a_1+(n-1)d] = n\,a_1 + d\,\dfrac{n(n-1)}{2}$。

最简单也是最常用的等差级数就是{n}，其和为

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{2.3}$$

2．多项式级数

一个序列{a_n}称为多项式级数，如果 a_n 是一个以 n 为自变量的 k 阶多项式，其中 k 是一个常数，例如，{n^2}，常用的求和公式有：

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \tag{2.4}$$

$$\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2 \tag{2.5}$$

$$\sum_{i=1}^{n} i^k = 1^k + 2^k + 3^k + \cdots + n^k = \Theta(n^{k+1}) \tag{2.6}$$

我们证明一下式（2.6）。首先，因为 $\sum_{i=1}^{n} i^k \leqslant n^k + n^k + \cdots + n^k = n \times n^k = n^{k+1}$，所以有 $\sum_{i=1}^{n} i^k = O(n^{k+1})$。然后，又因为 $\sum_{i=1}^{n} i^k \geqslant \sum_{i=\lceil n/2 \rceil}^{n} i^k \geqslant \sum_{i=\lceil n/2 \rceil}^{n} (n/2)^k \geqslant (n/2)^{k+1} = \left(\frac{1}{2^{k+1}}\right) n^{k+1}$，所以有 $\sum_{i=1}^{n} i^k = \Omega(n^{k+1})$。因此有 $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$。这里用的方法往往可用于其他序列求和的问题。

3．等比级数

一个序列{a_n}称为等比级数，如果 $a_n = ra_{n-1} = a_0 r^n$（$r \neq 1$）。用数学归纳法可证明其求和公式为：

$$\sum_{i=0}^{n} a_0 r^n = a_0 \frac{r^{n+1} - 1}{r - 1} \tag{2.7}$$

例如，$1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1$。

4．调和级数

序列{$1/n$}称为调和级数，而 $H_n = \sum_{i=1}^{n} \frac{1}{i}$ 称为第 n 个调和数。一个有名的结果是 $H_n = \Theta(\ln n)$。这个结果可以用积分的办法证明。

因为 $H_n = \sum_{k=1}^{n} \frac{1}{k} > \sum_{k=1}^{n} \int_{k}^{k+1} \frac{1}{x} \mathrm{d}x = \sum_{k=1}^{n} [\ln(k+1) - \ln k] = \ln(n+1)$，又因为

$$H_n = \sum_{k=1}^{n} \frac{1}{k} = 1 + \sum_{k=2}^{n} \frac{1}{k} < 1 + \sum_{k=2}^{n} \int_{k-1}^{k} \frac{1}{x} \mathrm{d}x = 1 + \sum_{k=2}^{n} [\ln k - \ln(k-1)] = 1 + \ln n$$

我们得到

$$\ln(n+1) < H_n < 1 + \ln n$$

所以有 $H_n = \Theta(\ln n)$。

5．其他级数

$$\sum_{k=1}^{n} k2^k = (n-1)2^{n+1} + 2 \tag{2.8}$$

2.2.4 主方法求解

这里的主方法指的不是一个新的方法，而是用序列求和法解递推关系（2.2）式时得到的一些结果的总结。在（2.2）式中，影响结果的参数有 3 个，即 a、b 和函数 $f(n)$。当它们满足一定关系时，其解可以立即得到。因此，当我们用主方法解（2.2）式时，只需要检查一下这 3 个参数满足哪一个条件即可对号入座地给出结果，而省去烦琐重复的求解过程。具体来说，给定递推关系 $T(n)=aT(n/b)+f(n)$，这里 $a\geqslant 1$，$b>1$，要先比较一下 n^k 和 $f(n)$哪个大，这里 $k=\log_b a$。下面我们介绍最常用的三条规则。

规则 1：如果存在一个正数 $\varepsilon>0$ 使得 $f(n)=O(n^{k-\varepsilon})$，那么 $T(n)=\Theta(n^k)$。

【例 2-5】用主方法确定由以下递推关系表示的算法复杂度。

$$T(n)=9T(n/3)+n\lg n$$

解：在这个递推关系中，$a=9$，$b=3$，$k=\log_b a=\log_3 9=2$。

如果我们用 $\varepsilon=0.2$，那么 $n^{k-\varepsilon}=n^{2-0.2}=n^{1.8}$。因为 $f(n)=n\lg n=O(n^{1.8})$，所以 $T(n)=\Theta(n^2)$。

规则 2：如果 $f(n)=\Theta(n^k)$，那么 $T(n)=\Theta(n^k\log n)$。

【例 2-6】用主方法确定由以下递推关系表示的算法复杂度。

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

解：在这个递推关系中，$a=1$，$b=3/2$，$k=\log_b a=0$。

因为 $f(n)=1=\Theta(n^0)$，所以 $T(n)=\Theta(\lg n)$。

【例 2-7】用主方法确定由以下递推关系表示的算法复杂度。

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

解：在这个递推关系中，$a=2$，$b=2$，$k=\log_b a=\log_2 2=1$。因为 $f(n)=n=\Theta(n^1)$，所以 $T(n)=\Theta(n\lg n)$。

规则 3：如果存在一个正数 $\varepsilon>0$ 使得 $f(n)=\Omega(n^{k+\varepsilon})$，并且存在一个正数 $c<1$ 使得 $af(n/b)\leqslant cf(n)$，那么 $T(n)=\Theta(f(n))$。

【例 2-8】用主方法确定由以下递推关系表示的算法复杂度。

$$T(n)=3T(n/2)+n^2\lg n$$

解：在这个递推关系中，$a=3$，$b=2$，$k=\log_2 3\approx 1.58$。显然，我们可以用 $\varepsilon=0.2$ 使得 $f(n)=n^2\lg n=\Omega(n^{1.58+0.2})$。现在需要找到一个正数 $c<1$ 使得 $af(n/b)\leqslant cf(n)$。因为 $af(n/b)=3(n/2)^2\lg(n/2)=\frac{3}{4}n^2\lg(n/2)<\frac{3}{4}n^2\lg n=\frac{3}{4}f(n)$，所以 $c=\frac{3}{4}<1$ 可以满足要求。因此，$T(n)=\Theta(n^2\lg n)$。

以上 3 条主方法规则的正确性可以用序列求和法证明。感兴趣的读者可以自己试着证明或在其他教科书中找到。这里我们就省去这个证明。另外要说明的是，以上 3 条规则不能解决所有的情况。除了这 3 条规则外，人们还发展了一些更强的规则，但以上 3 条规则基本上够用。当遇到以上 3 条规则不适用的情况时，我们则需要用序列求和法或找出针对具体问题的方法去解。

习题

1. 用分治法设计一个算法找出数组 $A[1..n]$中最大的数，并分析所需的比较次数。
2. 用分治法设计一个算法同时找出数组 $A[1..n]$中最大和第二大的数，$n\geqslant 2$，并分析所需的比较次数。
3. 假设 Google 公司在过去 n 天中的股票价格记录在数组 $A[1..n]$中。我们希望从中找出两天的价格，其价格的增幅最大。也就是说，我们希望找到 $A[i]$和 $A[j]$，$i<j$，使得 $M=A[j]-A[i]$的值

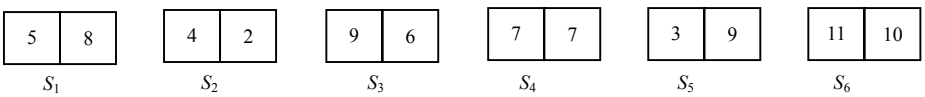
最大，即 $M = \text{Max}\{A[j] - A[i] \mid 1 \leqslant i < j \leqslant n\}$。试设计一个复杂度为 $O(n\lg n)$或更好的分治算法。

4. 设 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 是两个 $n \times n$ 矩阵。众所周知，计算其乘积 $C = AB$ 通常需要做$\Theta(n^3)$的乘法和加法。基于分治法的 Strassen 算法可以改进这个复杂度，下面是这个算法。为简化起见，我们假设 $n = 2^k$。

> **Strassen's algorithm** (A, B, n);
>
> 1 当 $n = 1$ 时，输出 $C = [a_{11} \cdot b_{11}]$，否则，按以下步骤进行。
>
> 2 将 A 和 B 各自划分为 4 个 $n/2 \times n/2$ 的矩阵如下。
>
> $$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},\ B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$
>
> 3 按下面公式递归计算出 7 个 $n/2 \times n/2$ 的矩阵。
>
> $$P = (A_{11} + A_{22})(B_{11} + B_{22}) \qquad Q = (A_{21} + A_{22})B_{11}$$
> $$R = A_{11}(B_{12} - B_{22}) \qquad S = A_{22}(B_{21} - B_{11})$$
> $$T = (A_{11} + A_{12})B_{22} \qquad U = (A_{21} - A_{11})(B_{11} + B_{12})$$
> $$V = (A_{12} - A_{22})(B_{21} + B_{22})$$
>
> 4 按下面公式计算出 4 个 $n/2 \times n/2$ 的矩阵。
>
> $$C_{11} = P + S - T + V \qquad C_{12} = R + T$$
> $$C_{21} = Q + S \qquad C_{22} = P + R - Q + U$$
>
> 5 输出结果如下。
>
> $$C = AB = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

请分析 Strassen 算法的复杂度。不必证明其正确性。

*5. 每个多米诺骨牌有两个正整数。假设我们把 n 个多米诺骨牌 $S_1, S_2, \cdots, S_n$ 水平地放成一排，如下图所示。假设我们不能改变骨牌之间的相对位置，但可以原地翻转每个骨牌。用 $L[i]$和 $R[i]$ 分别表示 S_i $(1 \leqslant i \leqslant n)$左边和右边的数。例如，在下例中 $L[1] = 5$, $R[1] = 8$, $L[2] = 4$, $R[2] = 2$ 等。如果 $L[i] < R[i]$, 我们称 S_i 被置为状态 0 并记为 $W[i] = 0$，翻转后则为状态 1 并记为 $W[i] = 1$。例如，下例中 S_1 为状态 0 而 S_2 为状态 1。如果 $L[i] = R[i]$，S_i 的状态可记为 $W[i] = 0$ 或 $W[i] = 1$。



请用分治法设计一个算法来确定每个骨牌的状态使得 $M = \sum_{i=1}^{n-1} R[i] \times L[i+1]$ 取得最大值。分析该算法的复杂度。

6. 用替换法获得以下递推关系的一个渐近上界。

（a）$T(n) = T(n/2) + 2T(n/4)$

（b）$T(n) = 2T(\lfloor n/2 \rfloor + 5) + n$

7. 证明以下递推关系有 $T(n) = O(2^n)$。

（a）$T(n) = nT(n/2) + n$

$T(1) = 1$

（b）$T(n) = T(n-1) + T(n-2) + n^2$

$T(1) = 1$，$T(2) = 2$

（c）$T(n) = 5n^2 T(n/2) + n^3$

$T(1)=1$

8．用序列求和法解以下递推关系。

（a）$T(n)=4T(n/2)+n^2\lg n$

（b）$T(n)=3T(n/3)+\dfrac{n}{\lg n}$

9．用主方法解以下递推关系。

（a）$T(n)=4T(n/2)+n$

（b）$T(n)=4T(n/2)+n^2$

（c）$T(n)=4T(n/2)+n^3$

（d）$T(n)=7T(n/2)+n^2$

（e）$T(n)=4T(n/3)+n$

（f）$T(n)=3T(n/9)+5\sqrt{n}$

（g）$T(n)=4T(n/2)+n^2\sqrt{n}$

10．解递推关系 $T(n)=2T(\sqrt{n})+\lg n$。

11．证明以下序列和的公式的正确性。

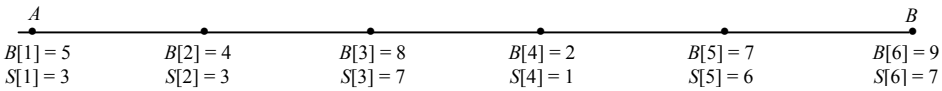
（a）$\sum_{k=1}^{n}k2^k=(n-1)2^{n+1}+2$（（2.8）式）

（b）$\sum_{k=1}^{n}k\lg k=\Theta(n^2\lg n)$

（c）$\sum_{k=2}^{n}\dfrac{k}{\lg k}=\Theta\left(\dfrac{n^2}{\lg n}\right)$

12．用积分法证明 $1^k+2^k+3^k+\cdots+n^k=\Theta(n^{k+1})$，这里 k 是任一个固定的正整数。

13．假设我们开一辆卡车从城市 A 到城市 B，中间一共经过 n 个苹果市场，包括城市 A 和城市 B 的苹果市场，并且编号为 1～n。在市场 i，$1\leqslant i\leqslant n$，从顾客的观点看，其每斤的买入价 $B[i]$ 和卖出价 $S[i]$都已知，单位是元。下图给出了一个 $n=6$ 的例子。



现在我们计划找一个市场 i 买苹果，然后再找一个市场 $j\geqslant i$ 把苹果卖掉使得赚的钱最多（如果根本赚不到钱，则使亏损越小越好）。我们假设卡车不可以向回开，并且只做一次买卖。例如，在上面的例子中，最好的方案是在市场 4 买苹果而在市场 6 卖出去，这样做每斤可赚 7–2=5 元钱。请设计一个分治算法找出市场 i 和 j 使得利润最大或亏损最小，并分析算法复杂度。

14．假设 n 个学生 $S[i]$ 的身高 $height[i]$不同（$1\leqslant i\leqslant n$），并且已排序为 $height[1]<height[2]<\cdots<height[n]$。另外，他们的性别对应地记录在数组 $sex[1..n]$ 中，$sex[i]=F$ 表示 $S[i]$是女生，$sex[i]=M$ 表示 $S[i]$是男生（$1\leqslant i\leqslant n$）。如果 $sex[i]=F$，$sex[j]=M$，并且 $height[i]<height[j]$，那么 $S[i]$ 和 $S[j]$ 可组成一对合格的舞伴。请用分治法设计一个复杂度为 $O(n)$的算法来计算在这 n 个学生中有多少个不同的合格的配对方案？