练习

4.3-1 证明：$T(n)=T(n-1)+n$ 的解为 $O(n^2)$。

4.3-2 证明：$T(n)=T(\lceil n/2\rceil)+1$ 的解为 $O(\lg n)$。

4.3-3 我们看到 $T(n)=2T(\lfloor n/2\rfloor)+n$ 的解为 $O(n\lg n)$。证明 $\Omega(n\lg n)$ 也是这个递归式的解。从而得出结论：解为 $\Theta(n\lg n)$。

4.3-4 证明：通过做出不同的归纳假设，我们不必调整归纳证明中的边界条件，即可克服递归式(4.19)中边界条件 $T(1)=1$ 带来的困难。

4.3-5 证明：归并排序的“严格”递归式(4.3)的解为 $\Theta(n\lg n)$。

4.3-6 证明：$T(n)=2T(\lfloor n/2\rfloor+17)+n$ 的解为 $O(n\lg n)$。

4.3-7 使用 4.5 节中的主方法，可以证明 $T(n)=4T(n/3)+n$ 的解为 $T(n)=\Theta(n^{\log_3 4})$。说明基于假设 $T(n)\leqslant cn^{\log_3 4}$ 的代入法不能证明这一结论。然后说明如何通过减去一个低阶项完成代入法证明。

4.3-8 使用 4.5 节中的主方法，可以证明 $T(n)=4T(n/2)+n$ 的解为 $T(n)=\Theta(n^2)$。说明基于假设 $T(n)\leqslant cn^2$ 的代入法不能证明这一结论。然后说明如何通过减去一个低阶项完成代入

87 法证明。

4.3-9 利用改变变量的方法求解递归式 $T(n)=3T(\sqrt{n})+\log n$。你的解应该是渐近紧确的。不必担心数值是否是整数。

4.4 用递归树方法求解递归式

虽然你可以用代入法简洁地证明一个解确是递归式的正确解，但想出一个好的猜测可能会很困难。画出递归树，如我们在 2.3.2 节分析归并排序的递归式时所做的那样，是设计好的猜测的一种简单而直接的方法。在**递归树**中，每个结点表示一个单一子问题的代价，子问题对应某次递归函数调用。我们将树中每层中的代价求和，得到每层代价，然后将所有层的代价求和，得到所有层次的递归调用的总代价。

递归树最适合用来生成好的猜测，然后即可用代入法来验证猜测是否正确。当使用递归树来生成好的猜测时，常常需要忍受一点儿“不精确”，因为稍后才会验证猜测是否正确。但如果在画递归树和代价求和时非常仔细，就可以用递归树直接证明解是否正确。在本节中，我们将使用递归树生成好的猜测，并且在 4.6 节中，我们将使用递归树直接证明主方法的基础定理。

我们以递归式 $T(n)=3T(\lfloor n/4\rfloor)+\Theta(n^2)$ 为例来看一下如何用递归树生成一个好的猜测。首先关注如何寻找解的一个上界。因为我们知道舍入对求解递归式通常没有影响(此处即是我们需要忍受不精确的一个例子)，因此可以为递归式 $T(n)=3T(\lfloor n/4\rfloor)+cn^2$ 创建一棵递归树，其中已将渐近符号改写为隐含的常数系数 $c>0$。

图 4-5 显示了如何从递归式 $T(n)=3T(\lfloor n/4\rfloor)+cn^2$ 构造出递归树。为方便起见，我们假定 n 是 4 的幂(忍受不精确的另一个例子)，这样所有子问题的规模均为正数。图 4-5(a)显示了 $T(n)$，它在图 4-5(b)中扩展为一棵等价的递归树。根结点中的 cn^2 项表示递归调用顶层的代价，根的三棵子树表示规模为 $n/4$ 的子问题所产生的代价。图 4-5(c)显示了进一步构造递归树的过程，将图 4-5(b)中代价为 $T(n/4)$ 的结点逐一扩展。我们继续扩展树中每个结点，根据递归式确定的关

88 系将其分解为几个组成部分(孩子结点)。

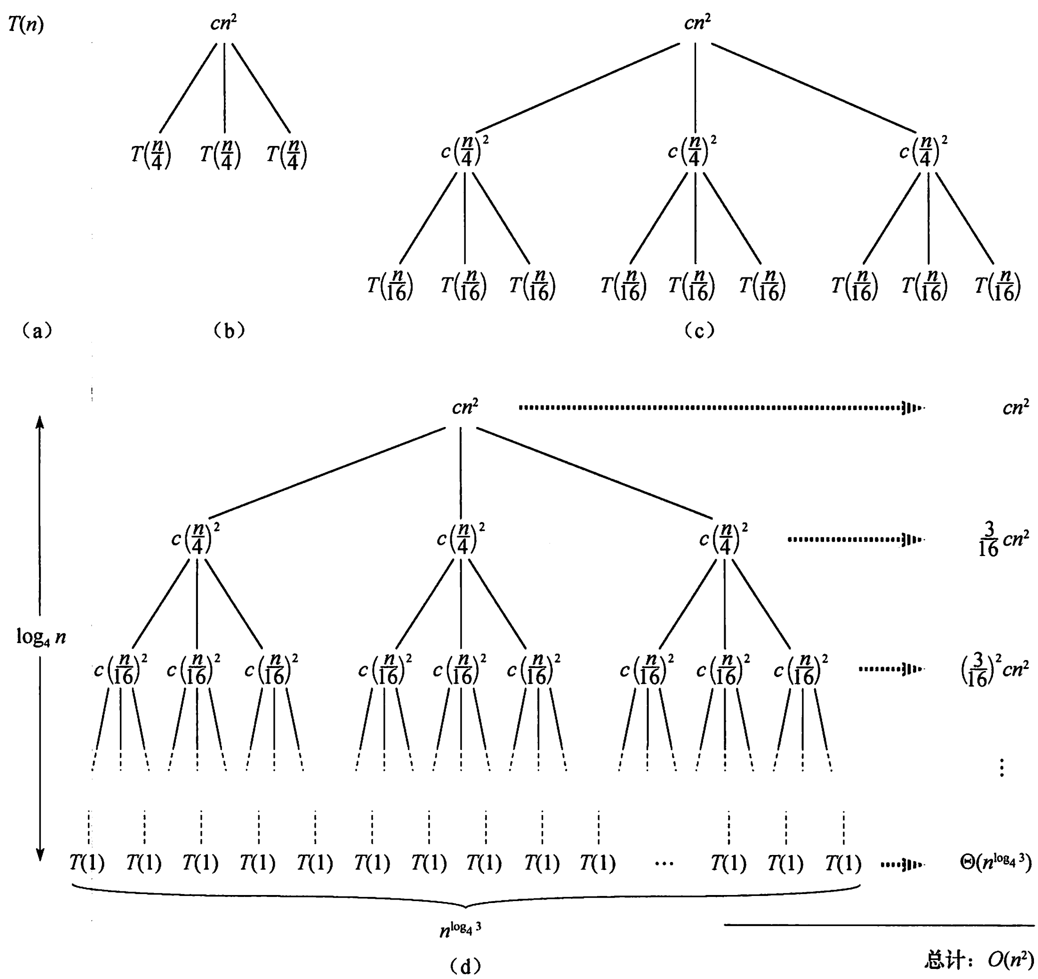

图 4-5　为递归式 $T(n)=3T(\lfloor n/4 \rfloor)+cn^2$ 构造递归树。(a)显示了 $T(n)$，在(b)～(d)中逐步扩展为递归树的形式。(d)中显示了扩展完毕的递归树，其高度为 $\log_4 n$(有 $\log_4 n+1$ 层)　89

因为子问题的规模每一步减少为上一步的 1/4，所以最终必然会达到边界条件。那么根结点与距离为 1 的子问题距离多远呢？深度为 i 的结点对应规模为 $n/4^i$ 的子问题。因此，当 $n/4^i=1$，或等价地 $i=\log_4 n$ 时，子问题规模变为 1。因此，递归树有 $\log_4 n+1$ 层(深度为 0，1，2，…，$\log_4 n$)。

接下来确定树的每一层的代价。每层的结点数都是上一层的 3 倍，因此深度为 i 的结点数为 3^i。因为每一层子问题规模都是上一层的 1/4，所以对 $i=0$，1，2，…，$\log_4 n-1$，深度为 i 的每个结点的代价为 $c(n/4^i)^2$。做一下乘法可得，对 $i=0$，1，2，…，$\log_4 n-1$，深度为 i 的所有结点的总代价为 $3^i c(n/4^i)^2=(3/16)^i cn^2$。树的最底层深度为 $\log_4 n$，有 $3^{\log_4 n}=n^{\log_4 3}$ 个结点，每个结点的代价为 $T(1)$，总代价为 $n^{\log_4 3}T(1)$，即 $\Theta(n^{\log_4 3})$，因为假定 $T(1)$ 是常量。

现在我们求所有层次的代价之和，确定整棵树的代价：

$$
\begin{aligned}
T(n) &= cn^2+\frac{3}{16}cn^2+\left(\frac{3}{16}\right)^2 cn^2+\cdots+\left(\frac{3}{16}\right)^{\log_4 n-1} cn^2+\Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2+\Theta(n^{\log_4 3}) \\
&= \frac{(3/16)^{\log_4 n}-1}{(3/16)-1}cn^2+\Theta(n^{\log_4 3}) \quad (\text{根据公式(A.5)})
\end{aligned}
$$

最后的这个公式看起来有些凌乱，但我们可以再次充分利用一定程度的不精确，并利用无限递减几何级数作为上界。回退一步，应用公式(A.6)，我们得到

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2+\Theta(n^{\log_4 3}) < \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2+\Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)}cn^2+\Theta(n^{\log_4 3}) \\
&= \frac{16}{13}cn^2+\Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

这样，对原始的递归式 $T(n)=3T(\lfloor n/4\rfloor)+\Theta(n^2)$，我们推导出了一个猜测 $T(n)=O(n^2)$。在本
90 例中，cn^2 的系数形成了一个递减几何级数，利用公式(A.6)，得出这些系数的和的一个上界——常数 16/13。由于根结点对总代价的贡献为 cn^2，所以根结点的代价占总代价的一个常数比例。换句话说，根结点的代价支配了整棵树的总代价。

实际上，如果 $O(n^2)$确实是递归式的上界(稍后就会证明这一点)，那么它必然是一个紧确界。为什么？因为第一次递归调用的代价为 $\Theta(n^2)$，因此 $\Omega(n^2)$必然是递归式的一个下界。

现在用代入法验证猜测是正确的，即 $T(n)=O(n^2)$是递归式 $T(n)=3T(\lfloor n/4\rfloor)+\Theta(n^2)$的一个上界。我们希望证明 $T(n)\leqslant dn^2$ 对某个常数 $d>0$ 成立。与之前一样，使用常数 $c>0$，我们有

$$
\begin{aligned}
T(n) &\leqslant 3T(\lfloor n/4\rfloor)+cn^2 \leqslant 3d\lfloor n/4\rfloor^2+cn^2 \leqslant 3d(n/4)^2+cn^2 \\
&= \frac{3}{16}dn^2+cn^2 \leqslant dn^2
\end{aligned}
$$

当 $d\geqslant(16/13)c$ 时，最后一步推导成立。

在另一个更复杂的例子中，图 4-6 显示了如下递归式的递归树：

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

(为简单起见，再次忽略了舍入问题。)与之前一样，令 c 表示 $O(n)$项中的常数因子。对图中显示出的递归树的每个层次，当求代价之和
91 时，我们发现每层的代价均为 cn。从根到叶的最长简单路径是 $n\to(2/3)n\to(2/3)^2n\to\cdots\to1$。由于当 $k=\log_{3/2}n$ 时，$(2/3)^kn=1$，因此树高为 $\log_{3/2}n$。

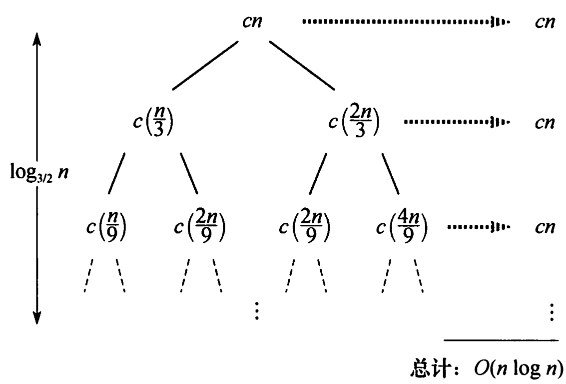

图 4-6　递归式 $T(n)=T(n/3)+T(2n/3)+cn$

直觉上，我们期望递归式的解最多是层数乘以每层的代价，即 $O(cn\log_{3/2}n)=O(n\lg n)$。但图 4-6 仅显示了递归树的顶部几层，并不是递归树中每个层次的代价都是 cn。考虑叶结点的代价。如果递归树是一棵高度为 $\log_{3/2}n$ 的完全二叉树，则叶结点的数量应为 $2^{\log_{3/2}n}=n^{\log_{3/2}2}$。由于每个叶结点的代价为常数，因此所有叶结点的总代价为 $\Theta(n^{\log_{3/2}2})$，由于 $\log_{3/2}n$ 是严格大于 1 的常数，因此叶结点代价总和为 $\Omega(n\lg n)$。但递归树并不是完全二叉树，因此叶结点数量小于 $n^{\log_{3/2}2}$。而且，当从根结点逐步向下走时，越来越多的内结点是缺失的。因此，递归树中靠下的层次对总代价的贡献小于 cn。我们可以计算出所有代价的准确值，但记住我们只是希望得到一个猜测，用于代入法。我们还是忍受一些不精确，尝试证明猜测的上界 $O(n\lg n)$是正确的。

我们确实可以用代入法验证 $O(n\lg n)$是递归式解的一个上界。我们来证明 $T(n)\leqslant dn\lg n$，其中 d 是一个适当的正常数。我们有

$$
\begin{aligned}
T(n) &\leqslant T(n/3)+T(2n/3)+cn \\
&\leqslant d(n/3)\lg(n/3)+d(2n/3)\lg(2n/3)+cn
\end{aligned}
$$

$$
\begin{aligned}
&= (d(n/3)\lg n - d(n/3)\lg 3) + (d(2n/3)\lg n - d(2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2) + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\leqslant dn\lg n
\end{aligned}
$$

只要 $d \geqslant c/(\lg 3-(2/3))$。因此，无需对递归树的代价进行更精确的计算。

练习

4.4-1 对递归式 $T(n)=3T(\lfloor n/2\rfloor)+n$，利用递归树确定一个好的渐近上界，用代入法进行验证。

4.4-2 对递归式 $T(n)=T(n/2)+n^2$，利用递归树确定一个好的渐近上界，用代入法进行验证。 92

4.4-3 对递归式 $T(n)=4T(n/2+2)+n$，利用递归树确定一个好的渐近上界，用代入法进行验证。

4.4-4 对递归式 $T(n)=T(n-1)+1$，利用递归树确定一个好的渐近上界，用代入法进行验证。

4.4-5 对递归式 $T(n)=T(n-1)+T(n/2)+n$，利用递归树确定一个好的渐近上界，用代入法进行验证。

4.4-6 对递归式 $T(n)=T(n/3)+T(2n/3)+cn$，利用递归树论证其解为 $\Omega(n\lg n)$，其中 c 为常数。

4.4-7 对递归式 $T(n)=4T(\lfloor n/2\rfloor)+cn$（$c$ 为常数），画出递归树，并给出其解的一个渐近紧确界。用代入法进行验证。

4.4-8 对递归式 $T(n)=T(n-a)+T(a)+cn$，利用递归树给出一个渐近紧确解，其中 $a\geqslant 1$ 和 $c>0$ 是常数。

4.4-9 对递归式 $T(n)=T(\alpha n)+T((1-\alpha)n)+cn$，利用递归树给出一个渐近紧确解，其中 $0<\alpha<1$ 和 $c>0$ 是常数。

4.5 用主方法求解递归式

主方法为如下形式的递归式提供了一种“菜谱”式的求解方法

$$T(n) = aT(n/b) + f(n) \tag{4.20}$$

其中 $a\geqslant 1$ 和 $b>1$ 是常数，$f(n)$是渐近正函数。为了使用主方法，需要牢记三种情况，但随后你就可以很容易地求解很多递归式，通常不需要纸和笔的帮助。 93

递归式(4.20)描述的是这样一种算法的运行时间：它将规模为 n 的问题分解为 a 个子问题，每个子问题规模为 n/b，其中 a 和 b 都是正常数。a 个子问题递归地进行求解，每个花费时间 $T(n/b)$。函数 $f(n)$包含了问题分解和子问题解合并的代价。例如，描述 Strassen 算法的递归式中，$a=7$，$b=2$，$f(n)=\Theta(n^2)$。

从技术的正确性方面看，此递归式实际上并不是良好定义的，因为 n/b 可能不是整数。但将 a 项 $T(n/b)$都替换为 $T(\lfloor n/b\rfloor)$或 $T(\lceil n/b\rceil)$并不会影响递归式的渐近性质(我们将在下一节证明这个断言)。因此，我们通常发现当写下这种形式的分治算法的递归式时，忽略舍入问题是很方便的。

主定理

主方法依赖于下面的定理。

定理 4.1(主定理) 令 $a\geqslant 1$ 和 $b>1$ 是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将 n/b 解释为 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$。那么 $T(n)$ 有如下渐近界：

1. 若对某个常数 $\varepsilon>0$ 有 $f(n)=O(n^{\log_b a-\varepsilon})$，则 $T(n)=\Theta(n^{\log_b a})$。

2. 若 $f(n)=\Theta(n^{\log_b a})$，则 $T(n)=\Theta(n^{\log_b a}\lg n)$。

3. 若对某个常数 $\varepsilon>0$ 有 $f(n)=\Omega(n^{\log_b a+\varepsilon})$，且对某个常数 $c<1$ 和所有足够大的 n 有 $af(n/b)\leqslant cf(n)$，则 $T(n)=\Theta(f(n))$。 ■

在使用主定理之前，我们花一点儿时间尝试理解一下它的含义。对于三种情况的每一种，我们将函数 $f(n)$ 与函数 $n^{\log_b a}$ 进行比较。直觉上，两个函数较大者决定了递归式的解。若函数 $n^{\log_b a}$ 更大，如情况 1，则解为 $T(n)=\Theta(n^{\log_b a})$。若函数 $f(n)$ 更大，如情况 3，则解为 $T(n)=\Theta(f(n))$。若两个函数大小相当，如情况 2，则乘上一个对数因子，解为 $T(n)=\Theta(n^{\log_b a}\lg n)=\Theta(f(n)\lg n)$。

在此直觉之外，我们需要了解一些技术细节。在第一种情况中，不是 $f(n)$ 小于 $n^{\log_b a}$ 就够了，
94 而是要多项式意义上的小于。也就是说，$f(n)$ 必须渐近小于 $n^{\log_b a}$，要相差一个因子 n^{ε}，其中 ε 是大于 0 的常数。在第三种情况中，不是 $f(n)$ 大于 $n^{\log_b a}$ 就够了，而是要多项式意义上的大于，而且还要满足“正则”条件 $af(n/b)\leqslant cf(n)$。我们将会遇到的多项式界的函数中，多数都满足此条件。

注意，这三种情况并未覆盖 $f(n)$ 的所有可能性。情况 1 和情况 2 之间有一定间隙，$f(n)$ 可能小于 $n^{\log_b a}$ 但不是多项式意义上的小于。类似地，情况 2 和情况 3 之间也有一定间隙，$f(n)$ 可能大于 $n^{\log_b a}$ 但不是多项式意义上的大于。如果函数 $f(n)$ 落在这两个间隙中，或者情况 3 中要求的正则条件不成立，就不能使用主方法来求解递归式。

使用主方法

使用主方法很简单，我们只需确定主定理的哪种情况成立，即可得到解。

我们先看下面这个例子

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

对于这个递归式，我们有 $a=9$，$b=3$，$f(n)=n$，因此 $n^{\log_b a}=n^{\log_3 9}=\Theta(n^2)$。由于 $f(n)=O(n^{\log_3 9-\varepsilon})$，其中 $\varepsilon=1$，因此可以应用主定理的情况 1，从而得到解 $T(n)=\Theta(n^2)$。

现在考虑

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

其中 $a=1$，$b=3/2$，$f(n)=1$，因此 $n^{\log_b a}=n^{\log_{3/2} 1}=n^0=1$。由于 $f(n)=\Theta(n^{\log_b a})=\Theta(1)$，因此应用情况 2，从而得到解 $T(n)=\Theta(\lg n)$。

对于递归式

$$T(n) = 3T(n/4)+n\lg n$$

我们有 $a=3$，$b=4$，$f(n)=n\lg n$，因此 $n^{\log_b a}=n^{\log_4 3}=O(n^{0.793})$。由于 $f(n)=\Omega(n^{\log_4 3+\varepsilon})$，其中 $\varepsilon\approx0.2$，因此，如果可以证明正则条件成立，即可应用情况 3。当 n 足够大时，对于 $c=3/4$，$af(n/b)=3(n/4)\lg(n/4)\leqslant(3/4)n\lg n=cf(n)$。因此，由情况 3，递归式的解为 $T(n)=\Theta(n\lg n)$。

主方法不能用于如下递归式：

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

虽然这个递归式看起来有恰当的形式：$a=2$，$b=2$，$f(n)=n\lg n$，以及 $n^{\log_b a}=n$。你可能错误地
95 认为应该应用情况 3，因为 $f(n)=n\lg n$ 渐近大于 $n^{\log_b a}=n$。问题出在它并不是多项式意义上的大于。对任意正常数 ε，比值 $f(n)/n^{\log_b a}=(n\lg n)/n=\lg n$ 都渐近小于 n^{ε}。因此，递归式落入了情况 2 和情况 3 之间的间隙(此递归式的解参见练习 4.6-2)。

我们利用主方法求解在 4.1 节和 4.2 节中曾见过的递归式(4.7)，

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

它刻画了最大子数组问题和归并排序的分治算法的运行时间(按照通常的做法，我们忽略了递归

式中基本情况的描述)。这里，我们有 $a=2$，$b=2$，$f(n)=\Theta(n)$，因此 $n^{\log_b a}=n^{\log_2 2}=n$。由于 $f(n)=\Theta(n)$，应用情况 2，于是得到解 $T(n)=\Theta(n\lg n)$。

递归式(4.17)，

$$T(n) = 8T(n/2) + \Theta(n^2)$$

它描述了矩阵乘法问题第一个分治算法的运行时间。我们有 $a=8$，$b=2$，$f(n)=\Theta(n^2)$，因此 $n^{\log_b a}=n^{\log_2 8}=n^3$。由于 n^3 多项式意义上大于 $f(n)$(即对 $\varepsilon=1$，$f(n)=O(n^{3-\varepsilon})$)，应用情况 1，解为 $T(n)=\Theta(n^3)$。

最后，我们考虑递归式(4.18)，

$$T(n) = 7T(n/2) + \Theta(n^2)$$

它描述了 Strassen 算法的运行时间。这里，我们有 $a=7$，$b=2$，$f(n)=\Theta(n^2)$，因此 $n^{\log_b a}=n^{\log_2 7}$。将 $\log_2 7$ 改写为 $\lg 7$，由于 $2.80<\lg 7<2.81$，我们知道对 $\varepsilon=0.8$，有 $f(n)=O(n^{\lg 7-\varepsilon})$。再次应用情况 1，我们得到解 $T(n)=\Theta(n^{\lg 7})$。

练习

4.5-1 对下列递归式，使用主方法求出渐近紧确界。

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

b. $T(n)=2T(n/4)+\sqrt{n}$

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

d. $T(n)=2T(n/4)+n^2$ 96

4.5-2 Caesar 教授想设计一个渐近快于 Strassen 算法的矩阵相乘算法。他的算法使用分治方法，将每个矩阵分解为 $n/4\times n/4$ 的子矩阵，分解和合并步骤共花费 $\Theta(n^2)$时间。他需要确定，他的算法需要创建多少个子问题，才能击败 Strassen 算法。如果他的算法创建 a 个子问题，则描述运行时间 $T(n)$的递归式为 $T(n)=aT(n/4)+\Theta(n^2)$。Caesar 教授的算法如果要渐近快于 Strassen 算法，a 的最大整数值应是多少？

4.5-3 使用主方法证明：二分查找递归式 $T(n)=T(n/2)+\Theta(1)$的解是 $T(n)=\Theta(\lg n)$。(二分查找的描述见练习 2.3-5)。

4.5-4 主方法能应用于递归式 $T(n)=4T(n/2)+n^2\lg n$ 吗？请说明为什么可以或者为什么不可以。给出这个递归式的一个渐近上界。

***4.5-5** 考虑主定理情况 3 的一部分：对某个常数 $c<1$，正则条件 $af(n/b)\leqslant cf(n)$是否成立。给出一个例子，其中常数 $a\geqslant 1$，$b>1$ 且函数 $f(n)$满足主定理情况 3 中除正则条件外的所有条件。

*4.6 证明主定理

本节给出主定理(定理 4.1)的证明。但如果只是为了使用主定理，你不必理解这个证明。

证明分为两部分。第一部分分析主递归式(4.20)，为简单起见，假定 $T(n)$仅定义在 $b(b>1)$的幂上，即仅对 $n=1$，b，b^2，…定义。这一部分给出了为理解主定理是正确的所需的所有直觉知识。第二部分显示了如何将分析扩展到所有正整数 n；这一部分应用了处理向下和向上取整问题的数学技巧。

在本节中，我们有时会稍微滥用渐近符号，用来描述仅仅定义在 b 的幂上的函数的行为。回忆一下，渐近符号的定义要求对所有足够大的数都证明函数的界，而不是仅仅对 b 的幂。因为可 97
以定义出仅仅应用于集合$\{b^i: i=0, 1, 2, \cdots\}$上而不是所有非负数上新的渐近符号，所以这种滥用问题不大。