> 分治法是将一个复杂的问题分成一些规模较小而结构与原问题相似的子问题，递归地解这些子问题，然后将各子问题的解合并得到原问题的解。

分治法在每一层的递归上都有三个步骤：

- **分解（Divided）**：将原问题分解成一系列子问题。
- **解决（Conquer）**：递归地解各子问题。
- **合并（Combine）**：将子问题的结果合并成原问题的解。

假设我们将原问题分解成a个子问题，每一个的大小是原问题的1/b。如果分解和合并的时间各为D(n)和C(n)，则可得到递归式：

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n \leq c \\ aT(n/b) + D(n) + C(n) & \text{else} \end{cases}$$

合并排序（Merge Sort）

当合并排序的运行时间如下分解：

- **分解**：仅计算出子数组的中间位置，需要常量时间，故$D(n) = \Theta(1)$。
- **解决**：递归地解两个规模为n/2的子问题，时间为$2T(n/2)$。
- **合并**：在一个含有n个元素的子数组上，MERGE过程的运行时间为$\Theta(n)$，则$C(n) = \Theta(n)$。

因此，合并排序的最坏运行时间$T(n)$的递归表示是：

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n = 1 \\ 2T(n/2) + \Theta(n) & \text{if} \quad n > 1 \end{cases}$$

由之前第一章介绍的主定理，我们可以得到$T(n) = \Theta(nlgn)$，这里的lgn代表$log_2 n$。同样，我们也可以通过递归树得到相同的答案。

最大子数组问题（Maximum Subarray Problem）

对于一个具有连续连续值的数组A，寻找A的和最大的非空连续子数组，**我们称这样的连续子数组为最大子数组。**例如，对于下图的数组，A[1...16]的最大子数组是A[8...11]，其和为43。在实际的例子中可表示在第8天买入股票，并在第11天卖出，获得的收益为43美元。

使用分治法可以求解最大子数组问题。首先找到数组的中间点，将数组分为左右两个数组，那么最大子数组可能存在于下列三种情况之一：

- 完全位于左边的数组中。
- 完全位于右边的数组中。
- 跨越了左右两个数组。

对于前两种情况，使用同样的方式递归地划分为规模最小的子数组求解即可。对于第三种情况，我们采用的算法是从中间点向两边遍历，分别求出两边的最大子数组，然后将左右两边的子数组相加即为跨越中点的最大子数组。

下面给出算法的伪代码：

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
1  if high == low
2      return (low, high, A[low])          // base case: only one element
3  else mid = ⌊(low + high)/2⌋
4      (left-low, left-high, left-sum) =
           FIND-MAXIMUM-SUBARRAY(A, low, mid)
5      (right-low, right-high, right-sum) =
           FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6      (cross-low, cross-high, cross-sum) =
           FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
8          return (left-low, left-high, left-sum)
9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10         return (right-low, right-high, right-sum)
11     else return (cross-low, cross-high, cross-sum)
```

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4      sum = sum + A[i]
 5      if sum > left-sum
 6          left-sum = sum
 7          max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

对FIND-MAXIMUM-SUBARRAY算法的运行时间进行分析：

首先计算出子数组的中间位置，需要常量时间$\Theta(1)$，然后递归地解两个规模为n/2的子问题，时间为$2T(n/2)$，调用FIND-MAX-CROSSING-SUBARRAY花费了$\Theta(n)$时间，则$T(n)=\Theta(1)+2T(n/2)+\Theta(n)$，用主方法或递归树求解此递归式可得$T(n)=\Theta(nlgn)$。

斐波那契数列（Fibonacci Number）

对于斐波那契数，我们有$F_0=0,F_1=1,F_n=F_{n-1}+F_{n-2}$。利用分治策略，我们可以将斐波那契数列转换为矩阵乘幂的问题，如下图所示：

$$\begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}$$

使用分治法，我们将矩阵递归地分解成两个相同的矩阵，再将这两个矩阵相乘即可。

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n}$$

$$= \left(\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n/2} \right)^{2}$$

$$= \left(\left(\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n/4} \right)^{2} \right)^{2}$$

故
$T(n)=T(n/2)+O(1)=T(n)/4+2O(1)=\ldots=T(n/2^{logn})+O(logn)\times O(1)$
。

整数乘法（Integer Multiplication）

假设x, y分别为两个n-bit的整数，如果要将它们相乘，模拟使用手动乘法得到的时间复杂度是$\Theta(n^2)$，考虑分治法。

令$x = (10^m a + b), y = (10^m c + d)$，如
$x = 1234567890, m = 5, a = 12345, b = 67890$。那么
$x \times y = (10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$，这里的时间复杂度$T(n) = 4T(n/2) + O(n)$，使用主方法可得$T(n) = O(n^2)$。

Anatolii, Karatsuba在1962年提出了一个只需要三次子乘法就可以完成运算的算法，其时间复杂度的递归表示为$T(n) = 3T(n/2) + O(n)$，使用主方法可得$T(n) = O(n^{lg3})$。

矩阵乘法（Matrix multiplication）

给定一个n维矩阵X和Y，计算Z=XY。我们可以使用分治法求解这个问题。

- **分解**：将X和Y分解为n/2维的矩阵。
- **解决**：使用8次矩阵乘法递归地将这些n/2维的矩阵相乘。
- **合并**：使用4次矩阵加法将矩阵合并。

下面给出一个例子：

$$X=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad Y=\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$X=\begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad Y=\begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$a=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad d=\begin{bmatrix} a_{33} & a_{34} \\ a_{44} & a_{44} \end{bmatrix} \quad e=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \quad h=\begin{bmatrix} b_{33} & b_{34} \\ b_{43} & b_{44} \end{bmatrix}$$

$$X=\begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad Y=\begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$XY=\begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

$$\mathrm{T(n)}=8T(n/2)+O(n^2)$$

$$\mathrm{T(n)}=O(n^3)$$

1969年Strassen提出了一个只需要7次矩阵乘法就可以完成运算的算法，算法将原矩阵分为7个新的子矩阵如下图所示：

$$
\begin{aligned}
P_1 &= a(f-h) \\
P_2 &= (a+b)h \\
P_3 &= (c+d)e \\
P_4 &= d(g-e) \\
P_5 &= (a+d)(e+h) \\
P_6 &= (b-d)(g+h) \\
P_7 &= (a-c)(e+f)
\end{aligned}
$$

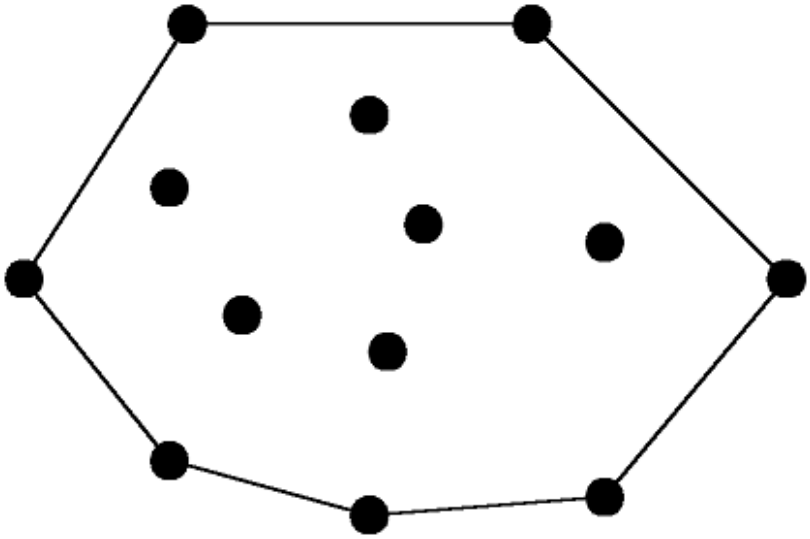
然后进行计算：

$$
XY = \begin{bmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{bmatrix}
$$

此算法的时间复杂度为$T(n) = 7T(n/2) + \Theta(n^2) = O(nlog_2 7) = O(n^{2.81})$。

凸包问题（The Convex Hull Problem）

假设平面上有一系列点，过某些点作一个多边形，使这个多边形能把所有点都“包”起来，当这个多边形是凸多边形的时候，我们就叫它**凸包**，凸包问题就是求构成凸包的点，如下图所示。



使用蛮力法是最容易想到的，思路是由两点确定一条直线，如果剩余的点都在这条直线的同一侧，那么认为这两个点是构成凸包的点。蛮力法的时间复杂度为$O(n^3)$。

下面我们介绍解决凸包问题的分治法，下面为具体步骤和图例：

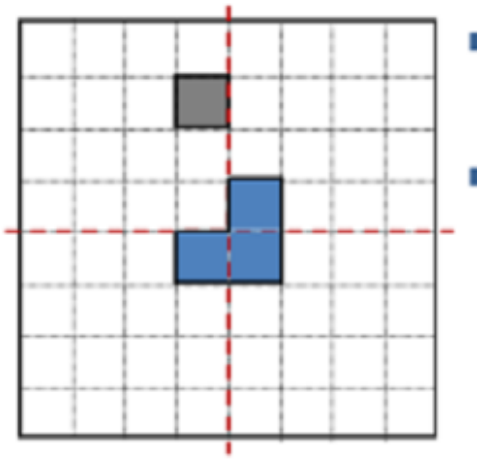
1. 将所有点放在二维坐标系里，那么横坐标最大的两个点p_1、p_n一定是凸包上的点（具体可以用反证法证明，这里不展开说）。直线p_1p_n将点集合分为了两部分，分别叫上包和下包。
2. 对于上包，求距离直线p_1p_n最远的点，即下图中的p_{max}。
3. 作直线p_1p_{max}和p_np_{max}，把p_1p_{max}左侧的点当作上包，把p_np_{max}右侧的点也当作是上包。
4. 重复步骤2、3。
5. 对下包也做类似的操作。

分治法的时间复杂度为$T(n)=2T(n/2)+O(n)=O(nlogn)$。

三格骨牌问题（Tromino Tiling）

对于三格骨牌问题，同样可以用分治法求解，解决这个问题的思想是每次都将平板分成四块同等大小的子平板。

例如，在插入三格骨牌时，将平板分成四块，由于洞位于左上方的子平板，因此将三个骨牌放置为如图所示的位置，以确保四块子平板的大小相等。递归地进行这个过程即可得出结果。



最邻近点问题（Finding the Closest Pair of Points）

顾名思义，最邻近点问题即在平面点集中找出距离最近的两个点。使用时间复杂度为$O(n^2)$的蛮力法可以解决这个问题。

下面我们介绍解决这个问题的分治法，首先将点集划分为两个部分，然后递归地寻找最近的点，若找到最近的两个点之间的距离为δ，则寻找是否存在分别属于两个部分的点之间的距离小于δ，算法的时间复杂度为
$T(n) = O(n) + 2T(n/2) + O(n) = O(nlgn)$。