武漢大學 WUHAN UNIVERSITY

第2章　分治策略

林 海

Lin.hai@whu.edu.cn

分治法

- **To solve P:**

  -- 分解 P into smaller problems P_1, P_2, …, P_k.

  -- 解决 by solving the (smaller) subproblems recursively.

  -- 合并 the solutions to P_1, P_2, …, P_k into the solution for P.

由于子问题与原问题是同类的,故分治法可以很自然地应用递归。

分治算法形式

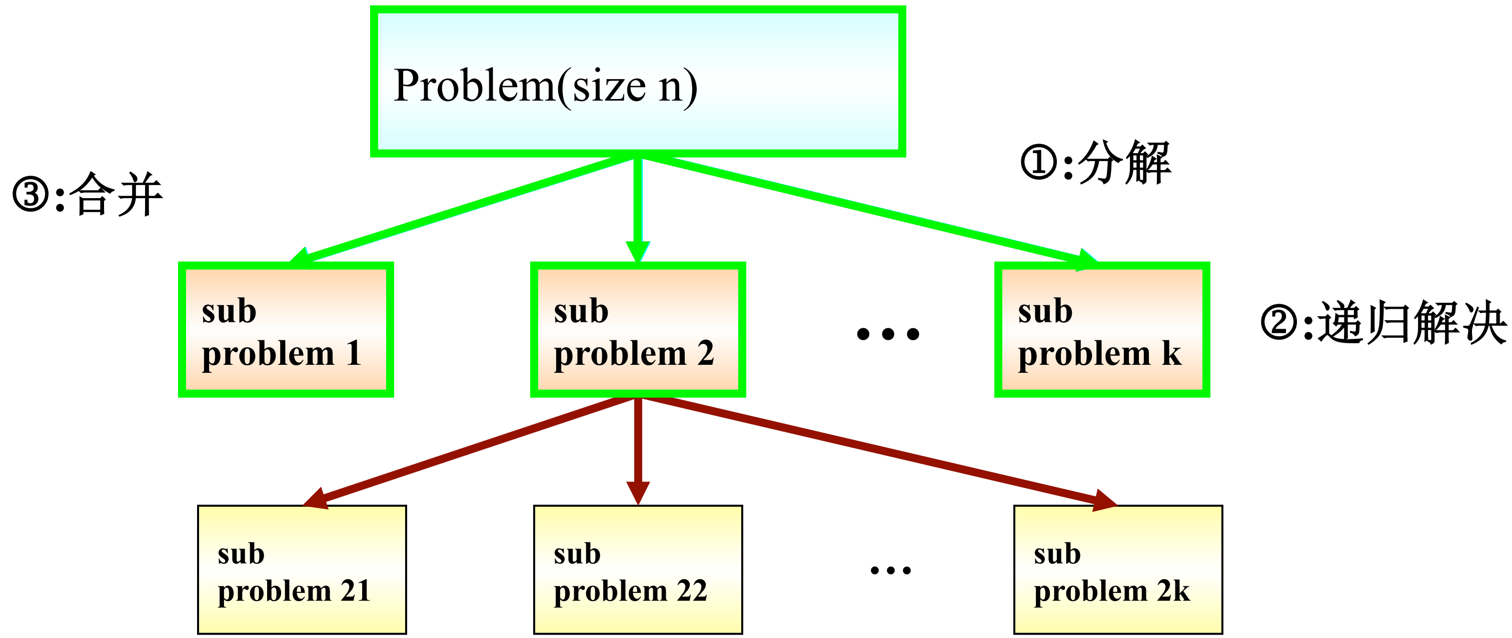
(1) 如果实例 I 的规模是“小”的，则使用直接的方法求解问题并返回其答案，否则继续做下一步。

(2) 把实例 I 分割成 p 个大小几乎相同的子实例 $I_1, I_2, \cdots, I_p$。

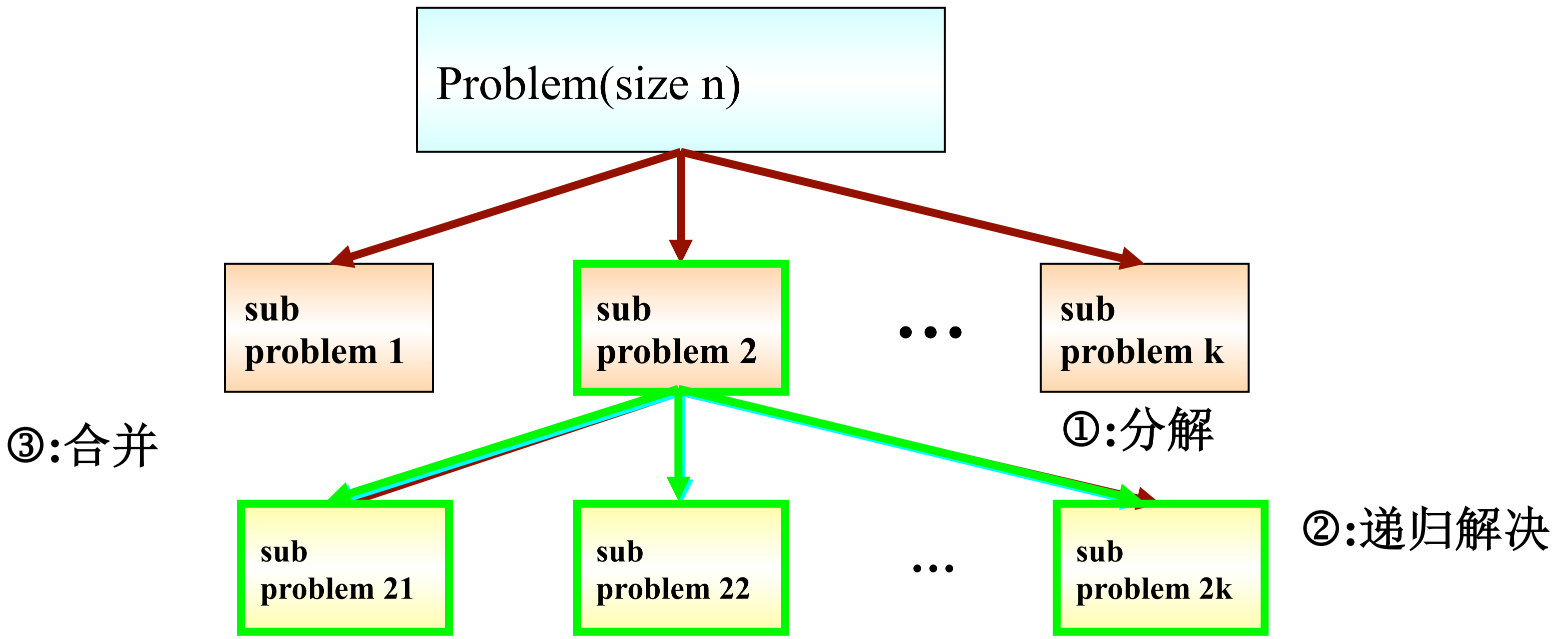
(3) 对每个子实例 I_j，$1 \leq j \leq p$，递归调用算法，并得到 p 个部分解。

(4) 组合 p 个部分解的结果得到原实例 I 的解，返回实例 I 的解。

分治法


Problem(size n)
①:分解
③:合并
sub problem 1
sub problem 2
…
sub problem k
②:递归解决
sub problem 21
sub problem 22
…
sub problem 2k

分治法


Problem(size n)
sub problem 1
sub problem 2
• • •
sub problem k
①:分解
③:合并
sub problem 21
sub problem 22
• • •
sub problem 2k
②:递归解决

6.3 合并排序

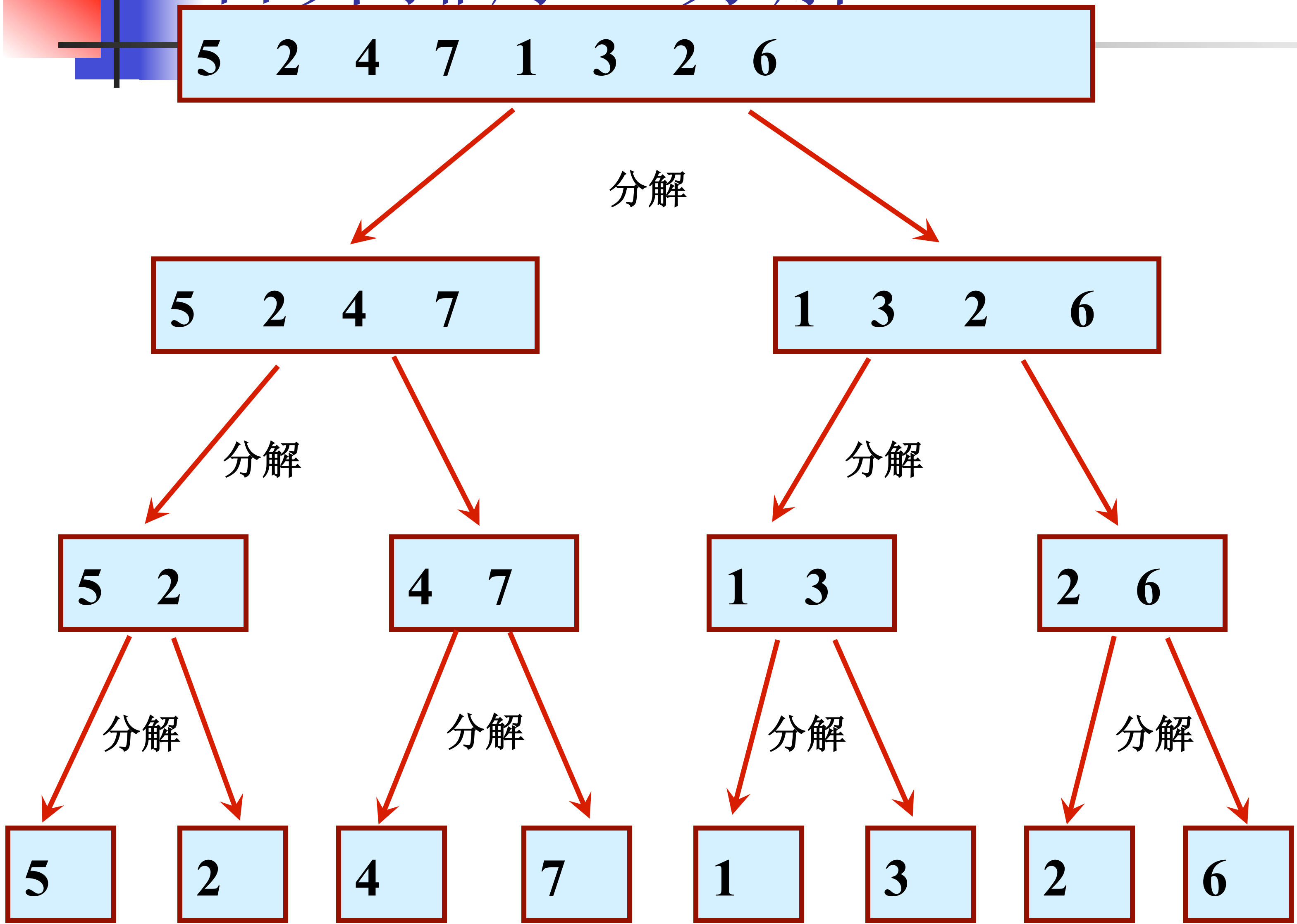
- **Using 分解-解决-合并，we can obtain a merge-sort algorithm**

  -- 分解: Divide the n elements into two subsequences of n/2 elements each.

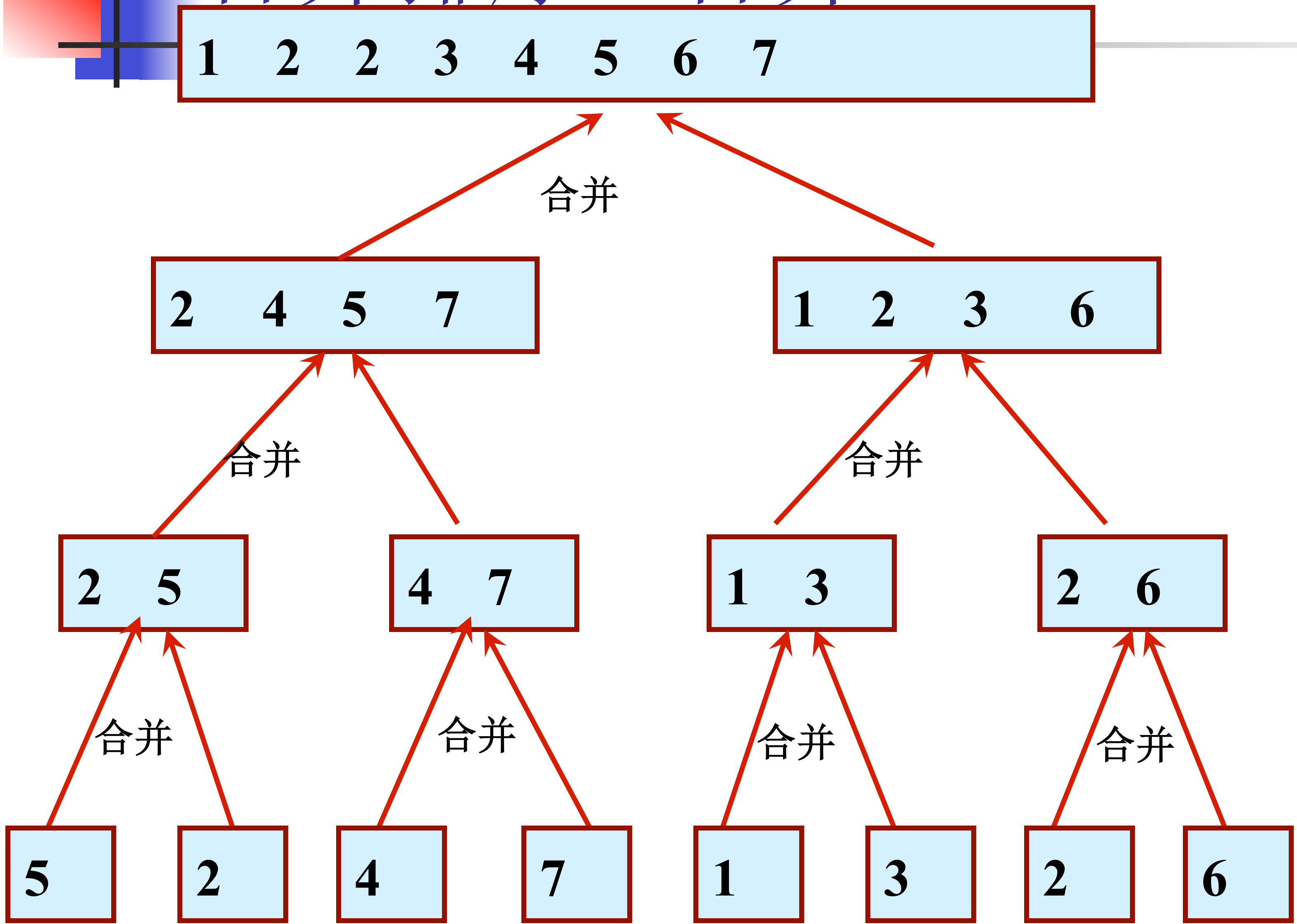
  -- 解决: Sort the two subsequences recursively.

  -- 合并: Merge the two sorted subsequences to produce the sorted answer.

合并排序—分解



合并排序—合并



算法分析

- Merge的复杂度为 $\Theta(n)$
- 合并排序的复杂度为

分解：分解步骤仅仅计算子数组的中间位置，需要常量时间，因此，$D(n)=\Theta(1)$。

解决：我们递归地求解两个规模均为 $n/2$ 的子问题，将贡献 $2T(n/2)$的运行时间。

合并：我们已经注意到在一个具有 n 个元素的子数组上过程 MERGE 需要 $\Theta(n)$的时间，所以 $C(n)=\Theta(n)$。

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

解此方程得出复杂度为 $\Theta(n\lg n)$

6.6 快速排序

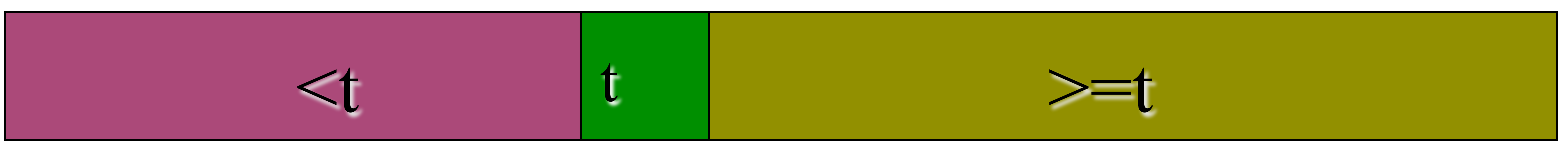
快速排序是一个非常流行而且高效的算法，其平均时间复杂度为 $\Theta(n\log n)$. 其优于合并排序之处在于它在原位上排序，不需要额外的辅助存贮空间(合并排序需 $\Theta(n)$的辅助空间)。

Charles A. R. Hoare 1960 年发布了使他闻名于世的快速排序算法(Quicksort)，这个算法也是当前世界上使用最广泛的算法之一，当时他供职于伦敦一家不大的计算机生产厂家。1980 年，Hoare 被授予图灵奖，以表彰其在程序语言定义与设计领域的根本性的贡献。在 2000 年，Hoare 因其在计算机科学和教育方面的杰出贡献被英国皇家封为爵士。

快速排序思想

1） 寻找一个中心元素（通常为第一个数）

2）将小于中心点的元素移动至中心点之前，大于中心点的元素移动至中心点之后。



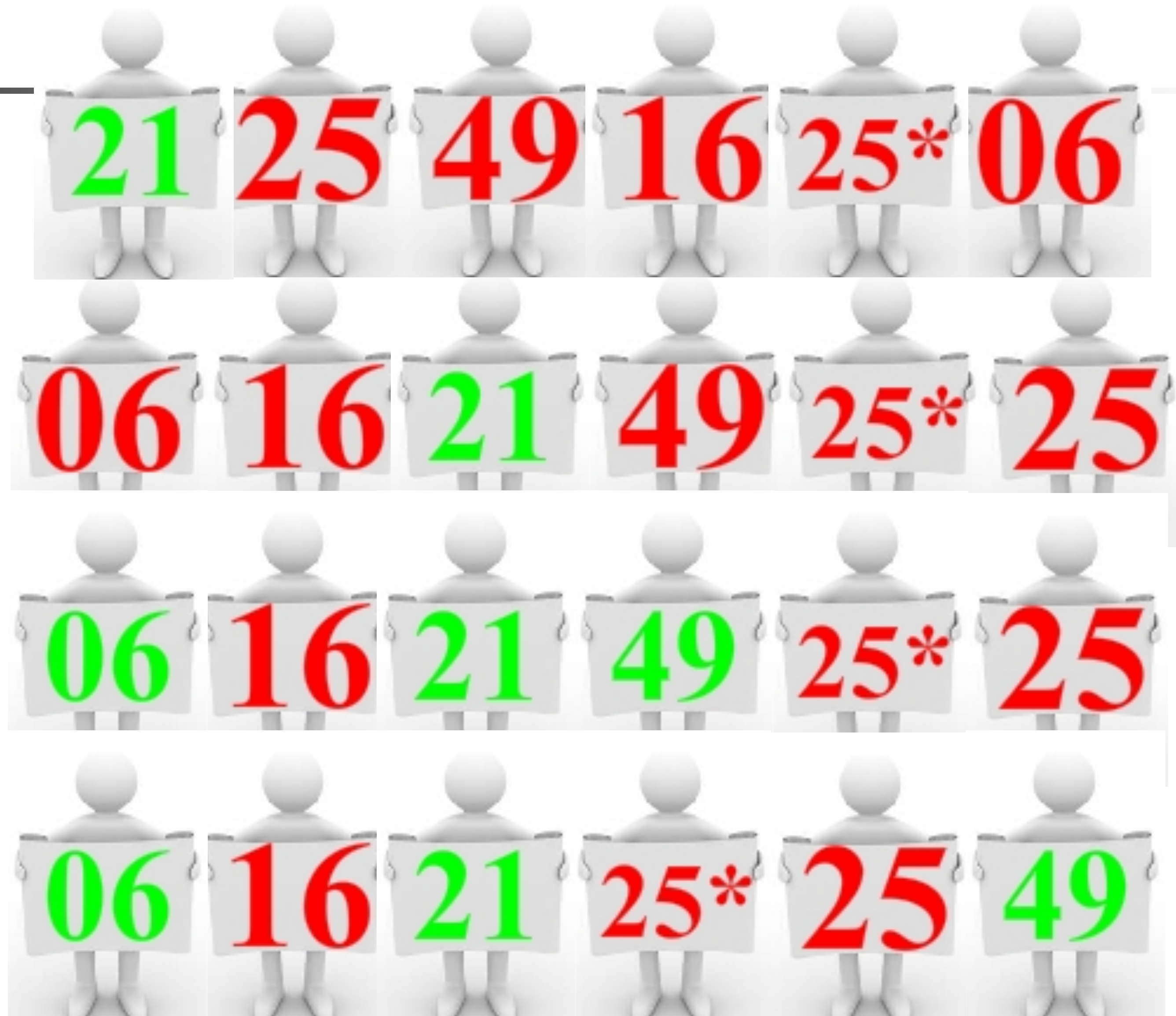
3）对上步分成的两个无序数组段重复1）、2）操作直到段长为1。

以21为中心元素

划分可得：

以06、49为中心元素

划分可得：



快速排序讲解

①选取中心元素的问题

选取第一个数为中心元素

②如何划分问题

③如何重复步骤①②将所有数据排序

使用递归

快速排序讲解

需要解决的问题（Partition划分算法）

当已知中心元素的前提下，怎样将其他元素划分好？（即：大于中心点在之后，小于中心点在之前）

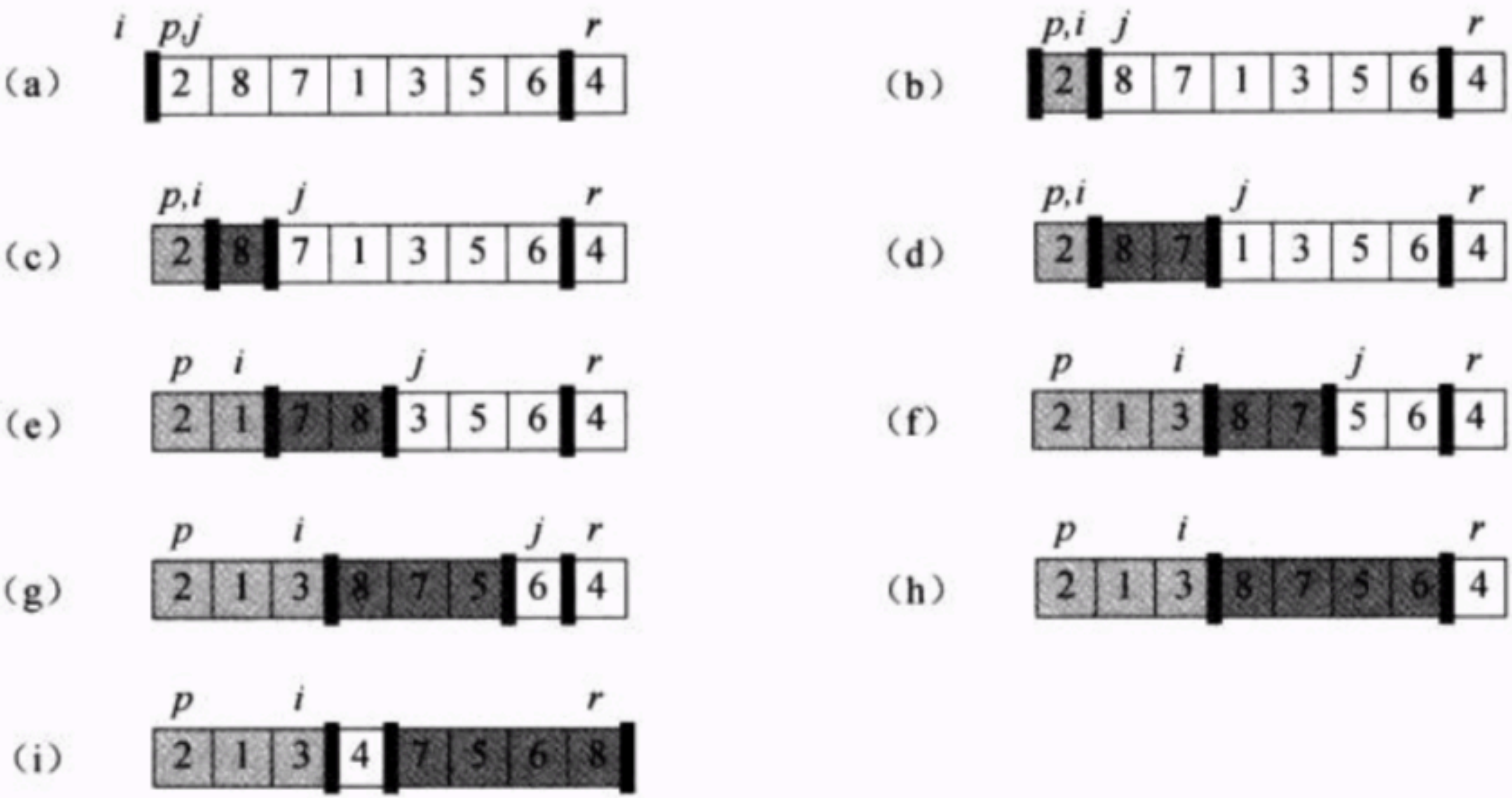
[0] [1] [2] [3] [4] [5]

i=0	j=5
i=1	j=5
i=1	j=4
i=1	j=3
i=2	j=3
i=2	j=2

算法终止

武漢大學 WUHAN UNIVERSITY

快速排序讲解



快速排序讲解

```
PARTITION(A, p, r)
x = A[r]
i = p−1
for j = p to r−1
        if A[j]≤x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i+ 1] with A[r]
return i + 1
```

快速排序讲解

请同学们思考

该算法有没有可以改进的地方

通过动画，可以看出每次中心元素都要交换。
根据划分的思想最后位置一定是中心元素

可以申请一个变量保存中心元素，以避免交换

i=0	j=5
i=1	j=5
i=1	j=4
i=1	j=3
i=2	j=3
i=2	j=2

[0] [1] [2] [3] [4] [5]

算法终止

快速排序讲解

left,right用于限定要排序数列的范围,temp即为中心元素

```
i=left;j=right;int temp=a[left];
do
{       //从右向左找第1个不小于中心元素的位置j
        while(      >       && i<j)     j--;
        if(i<j)
        {       a[  ]  =  a[  ];
                i++;
        }
```

当前元素小于中心元素结束循环时，应当在中心元素的左边

移至左边

程序填空

快速排序讲解

```
        //从左向右找第1个不大于中心元素的位置i
        while(a[i]<temp && i<j）      i++;
        if(i<j)
        {       a[j]=a[i];
                j--;
        }
}while(i<j);
                /将中心元素t填入最终位置
w=i;
```

快速排序

- 对原数组进行分割
- 对分割后的左、右子数组进行递归调用

Algorithm: QUICKSORT(A[low…high])
输入: n个元素的数组A[low…high]
输出：按非降序排列的数组A[low…high]
1. if low<high then
2.   w ← Patition(A[low…high]) {w为基准元素A[low]的新位置}
3.   quicksort(A, low, w-1)
4.   quicksort(A, w+1, high)
5. end if

时间复杂度分析

理想情形：每次SPLIT后得到的左右子数组规模相当，因此有：

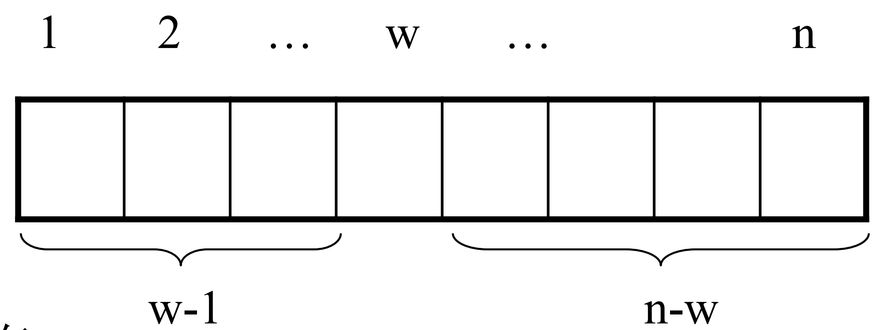
$$T(n)=\begin{cases}O(1) & if\ n=1\\ 2T(n/2)+\Theta(n) & if\ n>1\end{cases} \Longrightarrow T(n)=\Theta(n\log n)$$

最差情形(已经排好序或是逆序的数组)：每次SPLIT后，只得到左或是右子数组，因此有：

$$T(n)=\begin{cases}O(1) & if\ n=1\\ T(n-1)+\Theta(n) & if\ n>1\end{cases} \Longrightarrow T(n)=\Theta(n^2)$$

平均情形：

我们用 $C(n)$ 表示对一个n个元素的数组进行快速排序所需要的总的比较次数。



因此，我们有：

$$C(n) = (n-1) + \tfrac{1}{n}\sum_{w=1}^{n}\left(C(w-1) + C(n-w)\right)$$

$$\because \sum_{w=1}^{n} C(n-w) = C(n-1) + C(n-2) + \cdots + C(0) = \sum_{w=1}^{n} C(w-1)$$

$$\therefore C(n) = (n-1) + \tfrac{2}{n}\sum_{w=1}^{n} C(w-1)$$

$$n \cdot C(n) = n(n-1) + 2\sum_{w=1}^{n} C(w-1) \ldots\ldots (a)$$

n-1替换n

$$(n-1)C(n-1) = (n-1)(n-2) + 2\sum_{w=1}^{n-1} C(w-1) \ldots\ldots (b)$$

(a)-(b),并适当变换

$$\frac{C(n)}{n+1} = \frac{C(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

令$D(n) = \frac{C(n)}{n+1}$

$$D(n) = D(n-1) + \frac{2(n-1)}{n(n+1)}, D(1) = 0$$

$$D(n) = 2\sum_{j=1}^{n} \frac{j-1}{j(j+1)} = 2\sum_{j=1}^{n} \frac{2}{(j+1)} - 2\sum_{j=1}^{n} \frac{1}{j}$$

$$= 4\sum_{j=2}^{n+1} \frac{1}{j} - 2\sum_{j=1}^{n} \frac{1}{j} = 2\sum_{j=1}^{n} \frac{1}{j} - \frac{4n}{n+1} = \Theta(\log n)$$

$$\therefore C(n) = (n+1)D(n) = \Theta(n\log n)$$

分治法：最大子数组问题

问题描述：给出一个数组，找出此数组的最大子数组，即子数组的和最大

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

最大子数组

暴力求解的复杂度为 $\Theta(n^2)$

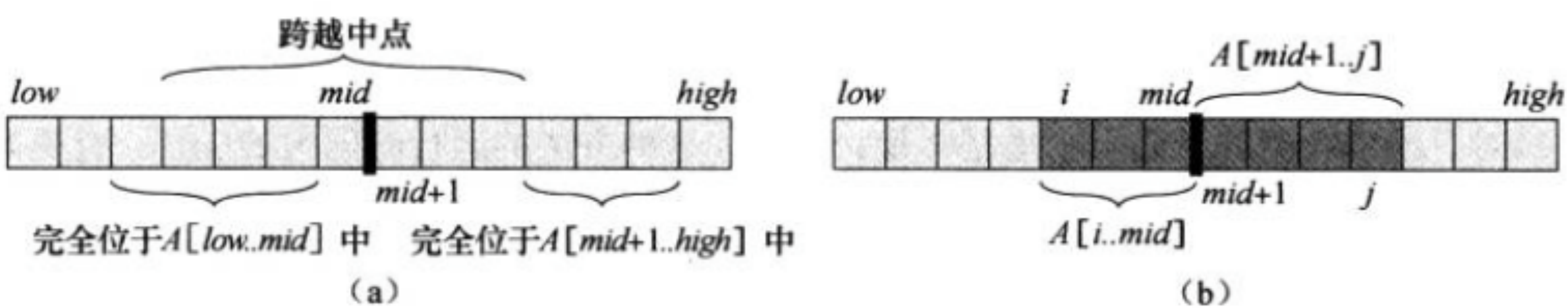
分治法：最大子数组问题

求解方法：将数组划分为两个规模相等的子数组

$A[low..high] \Rightarrow A[low..mid]$ 和 $A[mid+1..high]$

最大子数组必然是以下三种情况之一

- 完全位于子数组 $A[low..mid]$ 中，因此 $low \leqslant i \leqslant j \leqslant mid$。
- 完全位于子数组 $A[mid+1..high]$ 中，因此 $mid < i \leqslant j \leqslant high$。
- 跨越了中点，因此 $low \leqslant i \leqslant mid < j \leqslant high$。



分治法：最大子数组问题

- 寻找跨越中点的最大子数组

我们可以很容易地在线性时间(相对于子数组 $A[low..high]$的规模)内求出跨越中点的最大子数组。此问题并非原问题规模更小的实例，因为它加入了限制——求出的子数组必须跨越中点。如图 4-4(b)所示，任何跨越中点的子数组都由两个子数组 $A[i..mid]$和 $A[mid+1..j]$组成，其中 $low \leqslant i \leqslant mid$ 且 $mid < j \leqslant high$。因此，我们只需找出形如 $A[i..mid]$和 $A[mid+1..j]$的最大子数组，然后将其合并即可。过程 FIND-MAX-CORSSING-SUBARRAY 接收数组 A 和下标

分治法：最大子数组问题

- 寻找跨越中点的最大子数组

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid downto low
    sum=sum+A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum=sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

分治法：最大子数组问题

- 求解最大子数组问题的分治算法

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])            // base case: only one element
else mid=⌊(low+high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid+1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif rightr-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

分治法：最大子数组问题

- 算法分析
  - FIND-MAX-CROSSING-SUBARRAY的复杂度为Θ(n)

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

$$T(n)=\Theta(n\lg n)$$

分治法：矩阵乘法（Strassen算法）

- 矩阵相乘

若 A=(a_{ij})和 B=(b_{ij})是 $n\times n$ 的方阵，则对 i，j=1，2，…，n，定义乘积 $C=A\cdot B$ 中的元素 c_{ij} 为：

$$c_{ij}=\sum_{k=1}^{n}a_{ik}\cdot b_{kj}$$

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n×n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

复杂度为 $\Theta(n^3)$

分治法：矩阵乘法（Strassen算法）

- 矩阵相乘：一个简单的分治算法

假定将 A、B 和 C 均分解为 4 个 $n/2\times n/2$ 的子矩阵：

$$A=\begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix},\quad B=\begin{bmatrix}B_{11} & B_{12}\\ B_{21} & B_{22}\end{bmatrix},\quad C=\begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}$$

因此可以将公式 $C=A\cdot B$ 改写为：

$$\begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}=\begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix}\cdot\begin{bmatrix}B_{11} & B_{12}\\ B_{21} & B_{22}\end{bmatrix}$$

等价于如下 4 个公式：

$$C_{11}=A_{11}\cdot B_{11}+A_{12}\cdot B_{21}$$
$$C_{12}=A_{11}\cdot B_{12}+A_{12}\cdot B_{22}$$
$$C_{21}=A_{21}\cdot B_{11}+A_{22}\cdot B_{21}$$
$$C_{22}=A_{21}\cdot B_{12}+A_{22}\cdot B_{22}$$

分治法：矩阵乘法（Strassen算法）

- 矩阵相乘：一个简单的分治算法

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n×n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

分治法：矩阵乘法（Strassen算法）

- 一个简单的分治算法复杂度分析
  - 1到5行 $\Theta(1)$
  - 6到9行 每次递归T(n/2)，共8次，所以8*T(n/2)
  - 6到9行 的矩阵加法Θ(n^2/4)，即Θ(n^2)

$$T(n) = \Theta(1) + 8T(n/2) + \Theta(n^2) = 8T(n/2) + \Theta(n^2)$$

$$T(n) = \begin{cases} \Theta(1) & \text{若 } n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{若 } n > 1 \end{cases}$$

$$T(n) = \Theta(n^3)$$

分治法：矩阵乘法（Strassen算法）

- 关键是如何减少子矩阵相乘的次数：Strassen算法

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$S_1 = B_{12} - B_{22}$$
$$S_2 = A_{11} + A_{12}$$
$$S_3 = A_{21} + A_{22}$$
$$S_4 = B_{21} - B_{11}$$
$$S_5 = A_{11} + A_{22}$$
$$S_6 = B_{11} + B_{22}$$
$$S_7 = A_{12} - A_{22}$$
$$S_8 = B_{21} + B_{22}$$
$$S_9 = A_{11} - A_{21}$$
$$S_{10} = B_{11} + B_{12}$$

$$\Theta(n^2)$$

$$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$$
$$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$$
$$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$$
$$P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}$$
$$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$$
$$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$$
$$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$$

7 次 $n/2 \times n/2$

分治法：矩阵乘法（Strassen算法）

- 关键是如何减少子矩阵相乘的次数：Strassen算法

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = P_5 + P_4 - P_2 + P_6$$
$$C_{12} = P_1 + P_2$$
$$C_{21} = P_3 + P_4$$
$$C_{22} = P_5 + P_1 - P_3 - P_7$$

共进行了 8 次 $n/2 \times n/2$ 矩阵的加减法，因此花费 $\Theta(n^2)$时间

分治法：矩阵乘法（Strassen算法）

- 关键是如何减少子矩阵相乘的次数：Strassen算法

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$T(n) = \begin{cases} \Theta(1) & 若\ n = 1 \\ 7T(n/2) + \Theta(n^2) & 若\ n > 1 \end{cases}$$

$$T(n) = \Theta(n^{\lg 7})$$