



第六章 分治策略



引例：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储空间(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 Turing 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。

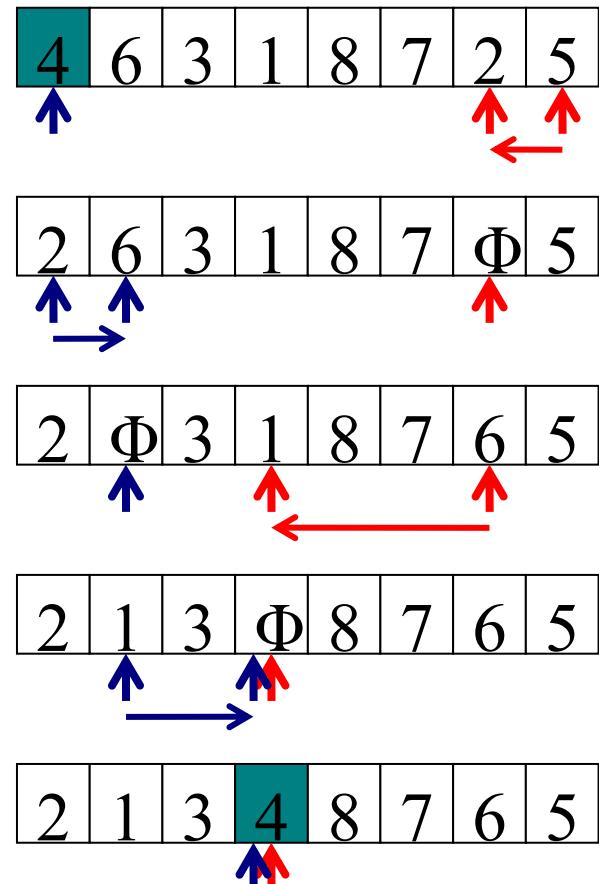


Algorithm: SPLIT(A[low,...high])

输入: 数组A[low,...high]

输出: 用A[low]作基准元素划分后的数组A
及基准元素新的位置w

1. $x \leftarrow A[low]$
2. while ($low < high$)
 3. while ($low < high \ \&\& \ A[high] > x$) $--high;$
 4. $A[low] \leftarrow A[high]$
 5. while ($low < high \ \&\& \ A[low] \leq x$) $++low;$
 6. $A[high] \leftarrow A[low]$
7. end while
8. $A[low] \leftarrow x$
9. $w \leftarrow low$
10. return A and w //新数组A与x的新位置w





Algorithm: QUICKSORT($A[low \dots high]$)

输入: n 个元素的数组 $A[low \dots high]$

输出: 按非降序排列的数组 $A[low \dots high]$

1. if $low < high$ then
2. $w \leftarrow \text{SPLIT}(A[low \dots high])$ { w 为基准元素 $A[low]$ 的新位置}
3. $\text{quicksort}(A, low, w-1)$
4. $\text{quicksort}(A, w+1, high)$
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：

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

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：

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases} \quad \xrightarrow{\hspace{1cm}} \quad T(n) = \Theta(n^2)$$



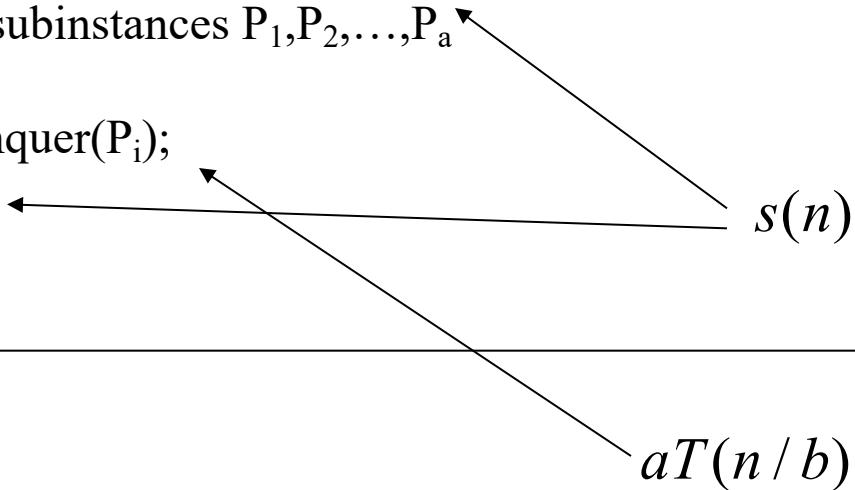
分治策略的思想

- 把规模较大的问题分解为若干个规模较小的子问题，这些子问题相互独立且与原问题同类；（该子问题的规模减小到一定的程度就可以容易地解决）
- 依次求出这些子问题的解，然后把这些子问题的解组合起来得到原问题的解。
- 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。



使用分治策略的算法设计模式

```
divide_and_conquer(P)
{
    if(|P|<=n0)
        direct_process(P);
    else
    {
        divide P into smaller subinstances P1,P2,...,Pa
        for(int i=1;i<=a;i++)
            yi=divide_and_conquer(Pi);
        merge(y1,y2,...,ya);
    }
}
```





使用分治策略的算法的时间复杂度分析

- 从分治法的一般设计模式可以看出，用它设计出的算法通常可以是递归算法。因而，算法的效率通常可以用递归方程来分析。
- 假设算法将规模为 n 的问题分解为 $a(a \geq 1)$ 个规模为 $n/b(b > 1)$ 的子问题解决。分解子问题以及合并子问题的解耗费的时间为 $s(n)$ ，则算法的时间复杂度可以递归表示为：

$$T(n) = \begin{cases} c & , n \leq n_0 \\ aT(n/b) + s(n) & , n > n_0 \end{cases}$$



合并排序Merge Sort

例：给定数组A [1...8]=

8	4	3	1	6	2	9	7
---	---	---	---	---	---	---	---

1. 将其分分成左右两个子数组：

8	4	3	1		6	2	9	7
---	---	---	---	--	---	---	---	---

2. 对子数组进行排序(可采用任何排序方法)。

3. 对排序后的子数组进行合并 :两个已排序的子数组用A[p...q]和A[q+1...r]表示.

设两个指针s和t, 初始时各自指向A[p]和A[q+1], 再设一空数组B[p...q, q+1...r]做暂存器, 比较元素A[s]和A[t], 将较小者添加到B, 然后移动指针,

若A[s]较小, 则s+1, 否则t+1,

直到s=q+1 或 t=r+1 为止

将剩余元素A[t...r] 或 A[s...q] 拷贝到数组B, 然后令A←B.



Algorithm: MERGE(A, p, q, r)

输入: 数组 $A[p \dots q]$ 和 $A[q+1 \dots r]$, 各自按升序排列

输出: 将 $A[p \dots q]$ 和 $A[q+1 \dots r]$ 合并成一个升序排序的新数组

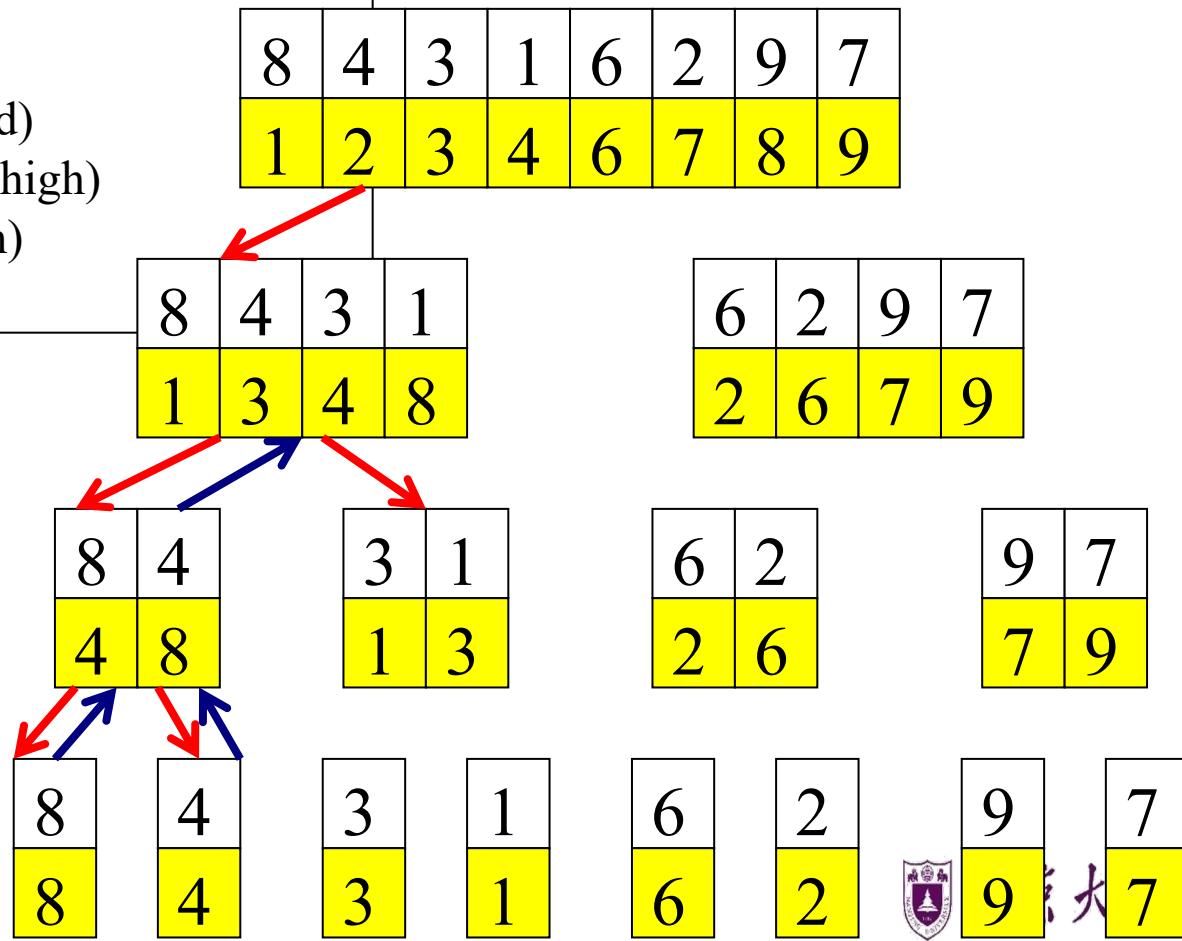
1. $s \leftarrow p$; $t \leftarrow q+1$; $k \leftarrow p$; $\{s, t, p$ 分别指向 $A[p \dots q], A[q+1 \dots r]$ 和 $B\}$
2. while $s \leq q$ and $t \leq r$
3. if $A[s] \leq A[t]$ then
4. $B[k] \leftarrow A[s]$
5. $s \leftarrow s+1$
6. else
7. $B[k] \leftarrow A[t]$
8. $t \leftarrow t+1$
9. end if
10. $k \leftarrow k+1$
11. end while
12. if $s = q+1$ then $B[k \dots r] \leftarrow A[t \dots r]$
13. else $B[k \dots r] \leftarrow A[s \dots q]$
14. end if
15. $A[p \dots q] \leftarrow B[p \dots q]$



Algorithm: MERGESORT(A[low...high])

输入: 待排序数组A[low,...high]
输出: A[low...high]按非降序排列

1. if $low < high$ then
2. $mid \leftarrow \lfloor (low+high)/2 \rfloor$
3. MERGESORT(A, low, mid)
4. MERGESORT(A, mid+1, high)
5. MERGE(A, low, mid, high)
6. end if



NANJING UNIVERSITY



矩阵乘法

- 设 A, B 是两个 $n \times n$ 的矩阵, 求 $C=AB$.
- 方法1: 直接相乘法
- 方法2: 分块矩阵法(直接应用分治策略)
- 方法3: Strassen算法(改进的分治策略)



方法1: 直接相乘

$$C = [c_{ij}]_{i=1,2,\dots,n; j=1,2,\dots,n} \quad c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

时间复杂度分析: 假设每做一次标量乘法耗费时间为 \mathbf{m} , 每做一次标量加法耗费时间为 \mathbf{a} , 那么直接相乘算法的时间复杂度为:

$$T(n) = n^2 \cdot n \cdot \mathbf{m} + n^2 \cdot (n-1) \cdot \mathbf{a} = \Theta(n^3)$$



方法2:分块矩阵法(直接应用分治策略)

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix}$$

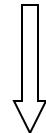
$$\mathbf{C}_{11} = \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21}$$

$$\mathbf{C}_{12} = \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22}$$

$$\mathbf{C}_{21} = \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21}$$

$$\mathbf{C}_{22} = \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22}$$

$$T(n) = \begin{cases} m & \text{if } n = 1 \\ 8T(n/2) + 4(n/2)^2 a & \text{if } n \geq 2 \end{cases}$$



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

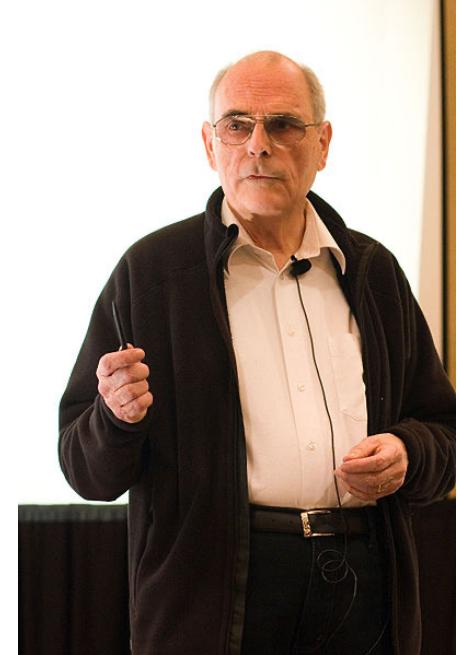


Strassen算法

Strassen was born on April 29, 1936, in Germany. In 1969, Strassen shifted his research efforts towards the analysis of algorithms with a paper on Gaussian elimination, introducing Strassen's algorithm, **the first algorithm** for performing **matrix multiplication faster than the $O(n^3)$** time bound that would result from a naive algorithm. In the same paper he also presented an asymptotically-fast algorithm to perform matrix inversion, based on the fast matrix multiplication algorithm. **This result was an important theoretical breakthrough**, leading to much additional research on fast matrix multiplication, and despite later theoretical improvements it remains a practical method for multiplication of dense matrices of moderate to large sizes.

—From Wikipedia, the free encyclopedia

Volker Strassen
giving the Knuth Prize
lecture at SODA 2009.





引入下列 $\mathbf{M}_i (i=1, 2, \dots, 7)$:

$$\mathbf{M}_1 = (\mathbf{A}_{12} - \mathbf{A}_{22})(\mathbf{B}_{21} + \mathbf{B}_{22})$$

$$\mathbf{M}_2 = (\mathbf{A}_{11} + \mathbf{A}_{22})(\mathbf{B}_{11} + \mathbf{B}_{12})$$

$$\mathbf{M}_3 = (\mathbf{A}_{11} - \mathbf{A}_{21})(\mathbf{B}_{11} + \mathbf{B}_{12})$$

$$\mathbf{M}_4 = (\mathbf{A}_{11} + \mathbf{A}_{12})\mathbf{B}_{22}$$

$$\mathbf{M}_5 = \mathbf{A}_{11}(\mathbf{B}_{12} - \mathbf{B}_{22})$$

$$\mathbf{M}_6 = \mathbf{A}_{22}(\mathbf{B}_{21} - \mathbf{B}_{11})$$

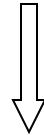
$$\mathbf{M}_7 = (\mathbf{A}_{21} + \mathbf{A}_{22})\mathbf{B}_{11}$$

则有: $\mathbf{C}_{11} = \mathbf{M}_1 + \mathbf{M}_2 - \mathbf{M}_4 + \mathbf{M}_6$, $\mathbf{C}_{12} = \mathbf{M}_4 + \mathbf{M}_5$,

$$\mathbf{C}_{21} = \mathbf{M}_6 + \mathbf{M}_7$$

$$\mathbf{C}_{22} = \mathbf{M}_2 - \mathbf{M}_3 + \mathbf{M}_5 - \mathbf{M}_7$$

$$T(n) = \begin{cases} m & \text{if } n = 1 \\ 7T(n/2) + 18(n/2)^2 a & \text{if } n \geq 2 \end{cases}$$



$$T(n) = \Theta(n^{\log_b^a}) = \Theta(n^{\log_2^7}) = \Theta(n^{2.81})$$



小结

分治法的适用条件

- 该问题的规模缩小到一定的程度就可以容易地解决；
- 该问题可以分解为若干个规模较小的同类问题；
- 利用该问题分解出的子问题的解可以合并为该问题的解；
- 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。这条特征涉及到分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然也可用分治法，但一般用动态规划更为合适。