



武汉大学

WUHAN UNIVERSITY

递归与分治1

算法设计与分析

武汉大学
国家网络安全学院
李雨晴



课时安排

- 算法基础 5学时
- 数学基础与数据结构 3学时
- 递归与分治 7 学时
- 动态规划 8学时
- 贪心算法 6学时
- 图的遍历 6学时
- 回溯与分支界限 6学时
- NP完全 3 学时



递归的概念

例1 阶乘函数

阶乘函数可递归地定义为：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

边界条件

递归方程

边界条件与递归方程是递归函数的二个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。



递归的概念

例1 阶乘函数

阶乘函数可递归地定义为：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

边界条件

递归方程

Factorial(n)

 If n==0

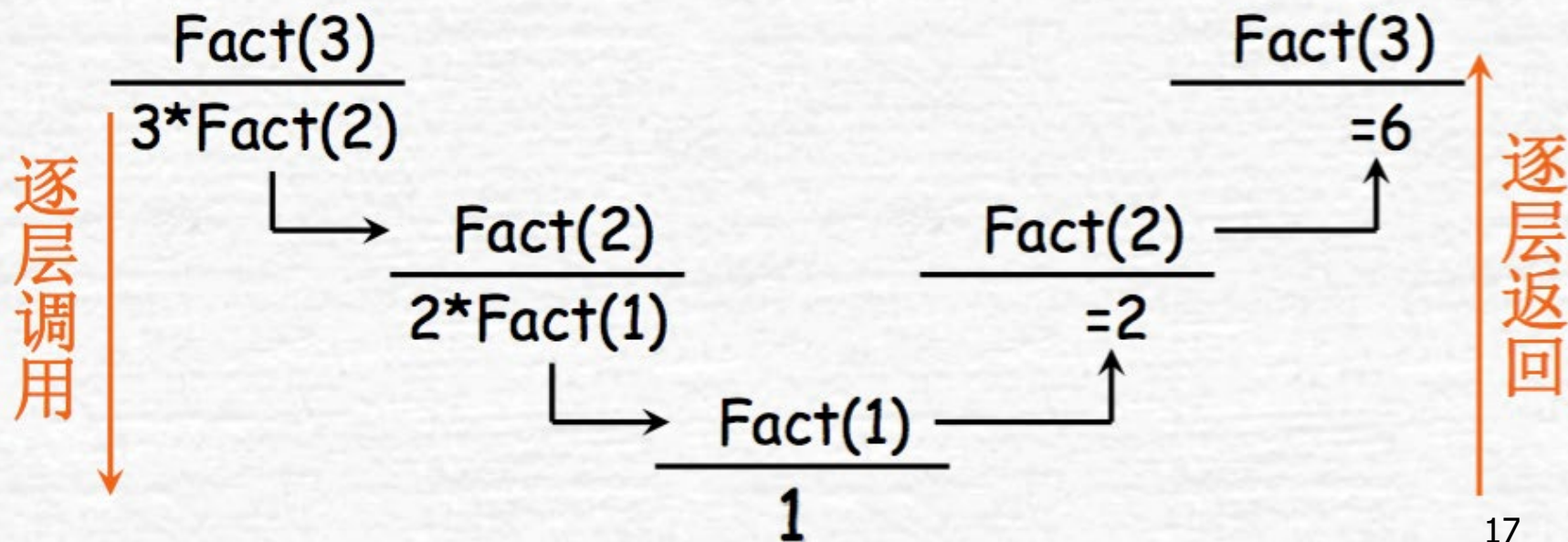
 return 1

 Return n*Factorial(n-1)



递归的概念

- n 阶乘的定义:
$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n > 0 \end{cases}$$
- 以求3! 为例的计算过程





递归的概念

例2 Fibonacci数列

无穷数列1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ..., 被称为Fibonacci数列。它可以递归地定义为:

$$F(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

边界条件

递归方程

第n个Fibonacci数可递归地计算如下:

Fibonacci(n)

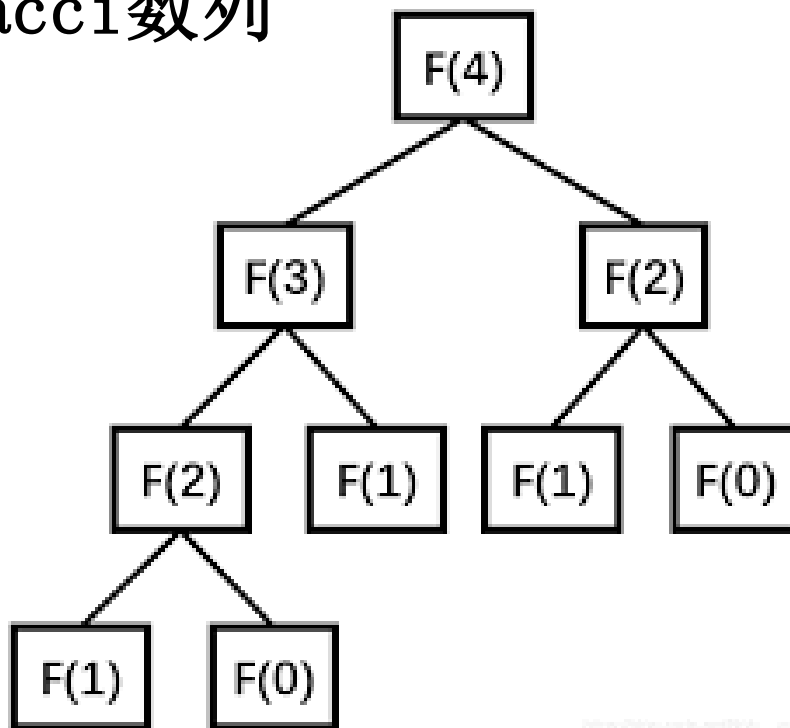
if (n <= 1) return 1;

return Fibonacci(n-1)+Fibonacci(n-2);



递归的概念

例2 Fibonacci数列





递归小结

- **优点：**结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。
- **缺点：**递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。



分治算法

- 引例：二分搜索
- 合并排序
- 快速排序
- 矩阵乘法
- 大整数相乘



习题：二分搜索

- 给定已按升序排好序的 n 个元素 $a[1:n]$ ，现要在这 n 个元素中找出一特定元素 x 。
- 二分搜索：将 n 个元素分成个数大致相同的两半，取 $a[n/2]$ 与 x 比较，如果 $x=a[n/2]$ ，则找到 x ；如果 $x<a[n/2]$ ，则只要在数组 a 的左半部分搜索 x ；如果 $x>a[n/2]$ ，则只要在右半部分搜索



习题：二分搜索

输入：非降序排列的数组 $A[1...n]$ 和元素 x

输出：如果 $x=A[j]$, $1 \leq j \leq n$, 则输出 j , 否则输出 0 .

1. $low \leftarrow 1; high \leftarrow n; j \leftarrow 0$
2. while($low \leq high$) and ($j=0$)
3. $mid \leftarrow \lfloor (low+high)/2 \rfloor$
4. if $x=A[mid]$ then $j \leftarrow mid$
5. else if $x < A[mid]$ then $high \leftarrow mid-1$
6. else $low \leftarrow mid+1$
7. end while
8. return j



习题：二分搜索（递归）

```
binarySearch(A, x, low, high)
  if low > high then return 0
  else
    middle = [ (low + high)/2];
    if (x == A[middle]) return middle;
    if (x < A[middle]) return binarySearch(A,
x, low, middle-1);
    else return binarySearch(A, x, middle+1, high);
  end if
```



二分搜索分析

$$C(n) \leq \begin{cases} 1 & \text{若 } n = 1 \\ 1 + C(\lfloor n/2 \rfloor) & \text{若 } n \geq 2 \end{cases}$$

$$\begin{aligned} C(n) &\leq 1 + C(\lfloor n/2 \rfloor) \\ &\leq 2 + C(\lfloor n/4 \rfloor) \\ &\vdots \\ &\leq (k-1) + C(\lfloor n/2^{k-1} \rfloor) \\ &= (k-1) + 1 \\ &= k \end{aligned}$$

$$\lfloor n/2^{k-1} \rfloor = 1$$

$$2^{k-1} \leq n < 2^k$$

$$k \leq \log n + 1 \leq k + 1$$

$$k = \lfloor \log n \rfloor + 1$$

$$C(n) \leq \lfloor \log n \rfloor + 1$$



分治策略

Divide and Conquer



合并排序Mergesort

例：给定数组A[1...8]=

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

1. 将其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。



合并Merge

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

输入：数组A[p...q]和A[q+1...r], 各自按升序排列

输出：将A[p...q]和A[q+1...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...q], A[q+1...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 $B[k...r] \leftarrow A[t...r]$
13. else $B[k...r] \leftarrow A[s...q]$
14. end if
15. $A[p...q] \leftarrow B[p...q]$



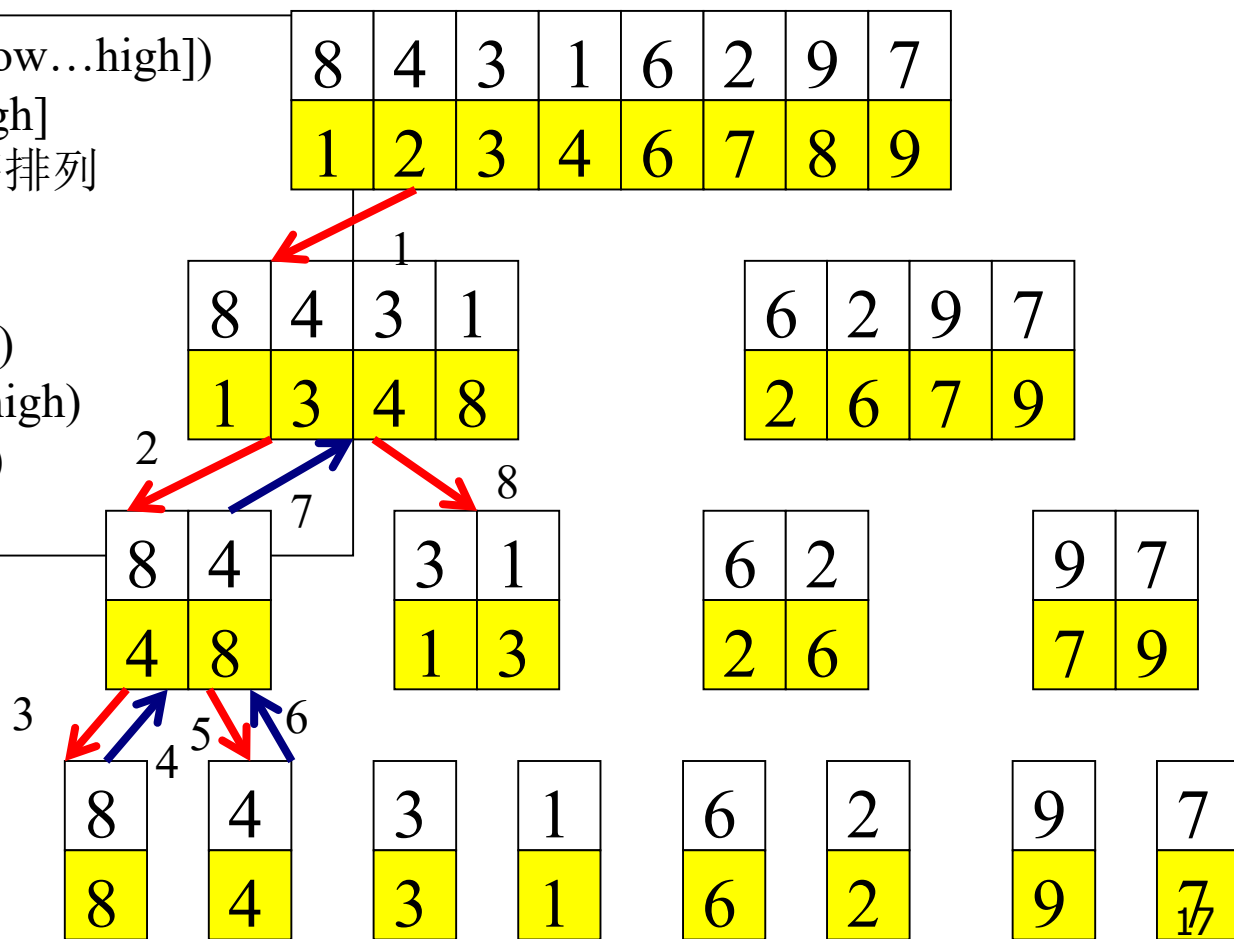
合并排序Mergesort

Algorithm: MERGESORT($A[\text{low} \dots \text{high}]$)

输入：待排序数组 $A[\text{low}, \dots, \text{high}]$

输出： $A[\text{low} \dots \text{high}]$ 按非降序排列

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





时间复杂度分析

- 如果两数组大小分别为 $\lfloor n/2 \rfloor$ 和 $\lfloor n/2 \rfloor$, 则比较的次数是 $\lfloor n/2 \rfloor$ 到 $n-1$

最小比较次数

$$C(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2C(n/2) + n/2 & \text{if } n \geq 2 \end{cases} \longrightarrow C(n) = \frac{n \log n}{2}$$

最大比较次数

$$C(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2C(n/2) + n - 1 & \text{if } n \geq 2 \end{cases} \longrightarrow C(n) = n \log n - n + 1$$



较大比较次数

$$\begin{aligned}C(n) &= 2C(n/2) + n - 1 \\&= 2(2C(n/2^2) + n/2 - 1) + n - 1 \\&= 2^2 C(n/2^2) + n - 2 + n - 1 \\&= 2^2 C(n/2^2) + 2n - 2 - 1 \\&\vdots \\&= 2^k C(n/2^k) + kn - 2^{k-1} - 2^{k-2} - \cdots - 2 - 1 \\&= 2^k C(1) + kn - \sum_{j=0}^{k-1} 2^j \\&= 2^k \times 0 + kn - (2^k - 1) \quad (\text{等式 2.10}) \\&= kn - 2^k + 1 \\&= n \log n - n + 1\end{aligned}$$



合并排序算法复杂度

- 算法mergesort对于一个n个元素的数组排序所需要的时间是 $\Theta(n \log n)$ 空间是 $\Theta(n)$



分治策略的思想

- **划分**：把规模较大的问题(n)分解为若干(通常为2)个规模较小的子问题($<n$)，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)
- **治理**：依次求出这些子问题的解
- **组合**：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。



分治算法形式

- 如果实例 I 规模是小的，则直接求解，否则继续做下一步
- 把实例 I 分割成 p 个大小几乎相同的子实例 $I_1, I_2 \dots I_p$
- 对每个子实例 $I_j, 1 \leq j \leq p$, 递归调用算法，并得到个 p 部分解
- 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



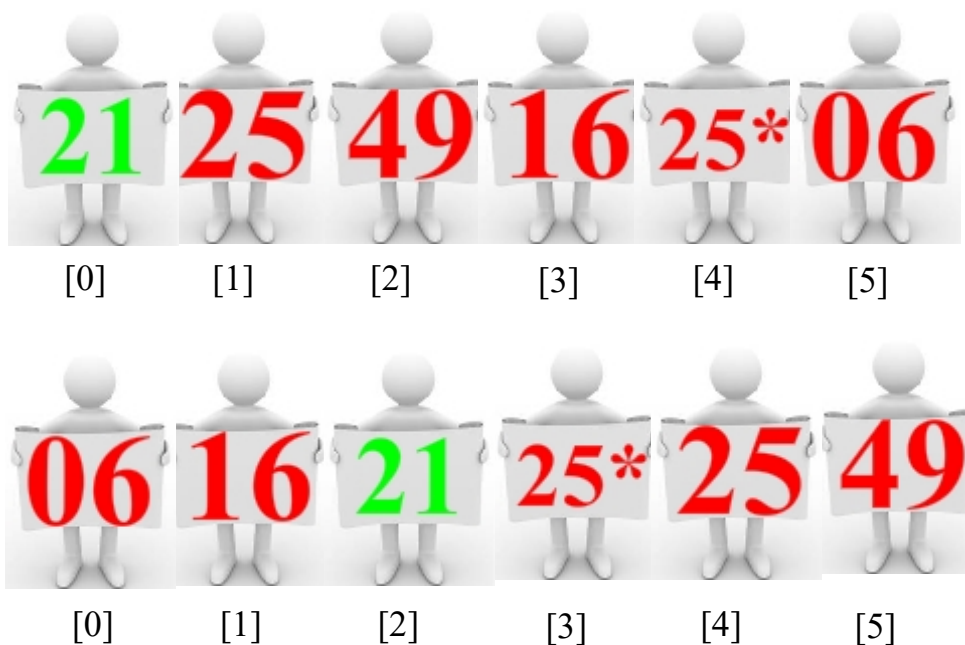
快速排序



- 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
- 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

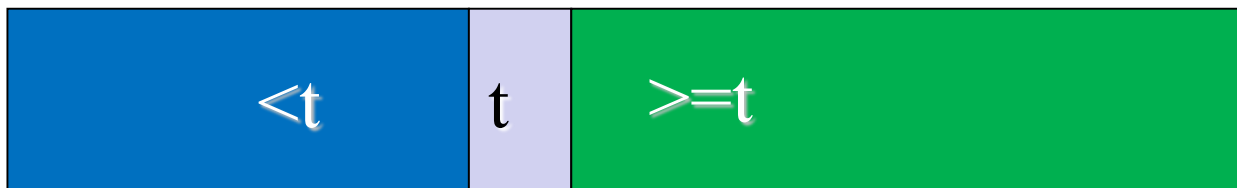




快速排序思想

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) 和 2) 操作直到段长为1



快速排序

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

- 选取中心元素的问题
 - 选取第一个数为中心元素
- 如何划分的问题
- 如何重复步骤1和2将所有数据排序
 - 使用递归



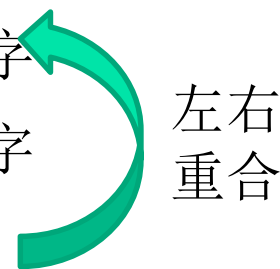
快速排序

■ 需要解决的问题（Split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，小于中心点在前）

• 解法

- 左边向右找第一个大于等于中心点的数字
- 右边向左找第一个小于等于中心点的数字
- 两个数字交换





快速排序实例讲解

$i=0$ $j=5$





快速排序实例讲解

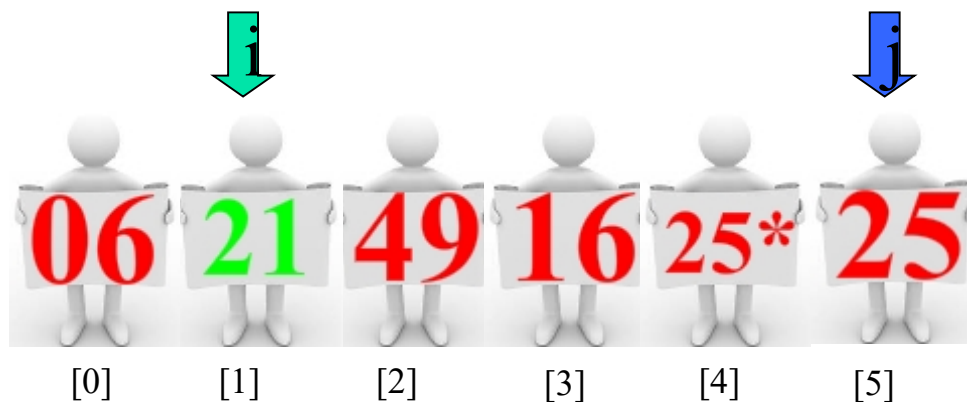
$i=1$ $j=5$





快速排序实例讲解

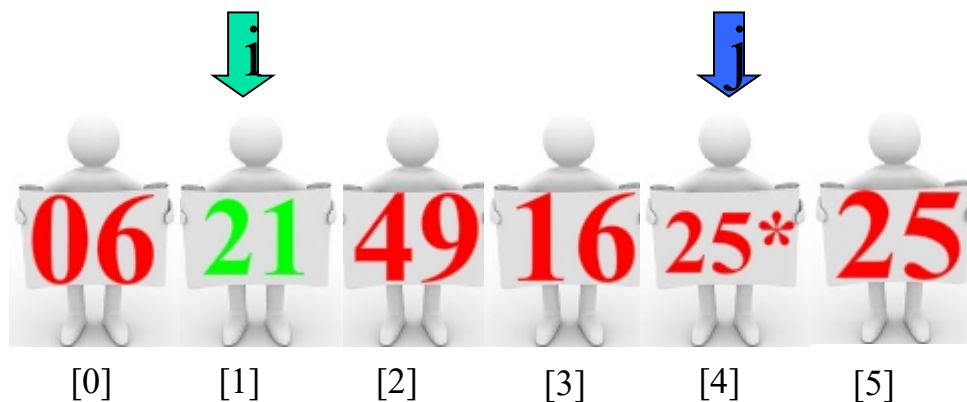
$i=1$ $j=4$





快速排序实例讲解

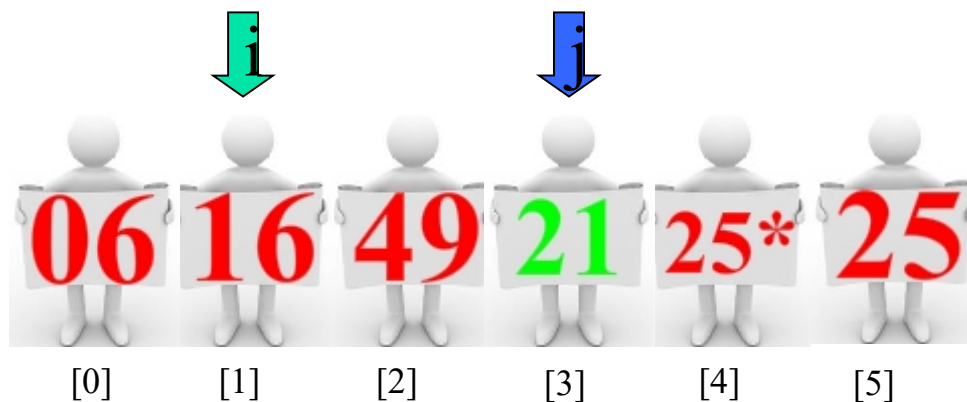
$i=1$ $j=3$





快速排序实例讲解

$i=2$ $j=3$



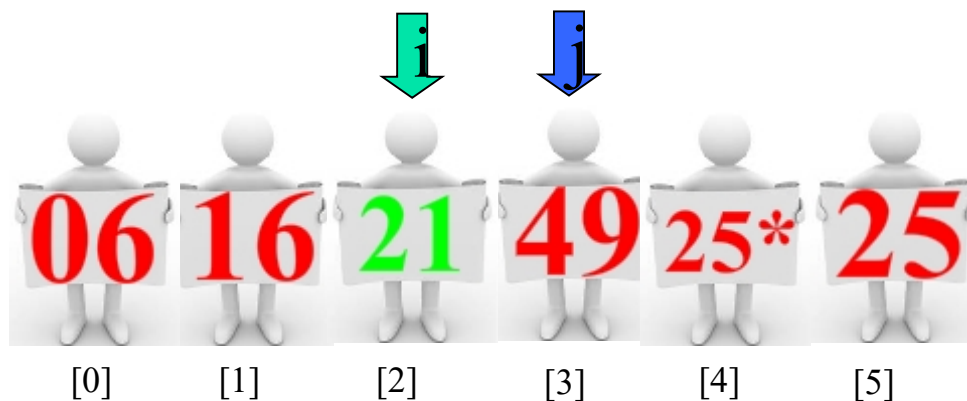


快速排序实例讲解

$i=2$

$j=2$

算法终止





快速排序

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

输入: n 个元素的数组 $A[\text{low} \dots \text{high}]$

输出: 按非降序排列的数组 $A[\text{low} \dots \text{high}]$

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

划分



快速排序

Algorithm: SPLIT($A[\text{low} \dots \text{high}]$)

1. $i \leftarrow \text{low}$
2. $j \leftarrow \text{high}$
3. $x \leftarrow A[\text{low}]$
4. while($i < j$)
 5. while($i < j \ \&\& \ A[i] < x$)
 6. $i = i + 1$
 7. while($i < j \ \&\& \ A[j] > x$)
 8. $j = j - 1$
 9. 互换 $A[i]$ 和 $A[j]$
10. $w \leftarrow i$
11. return A 和 w



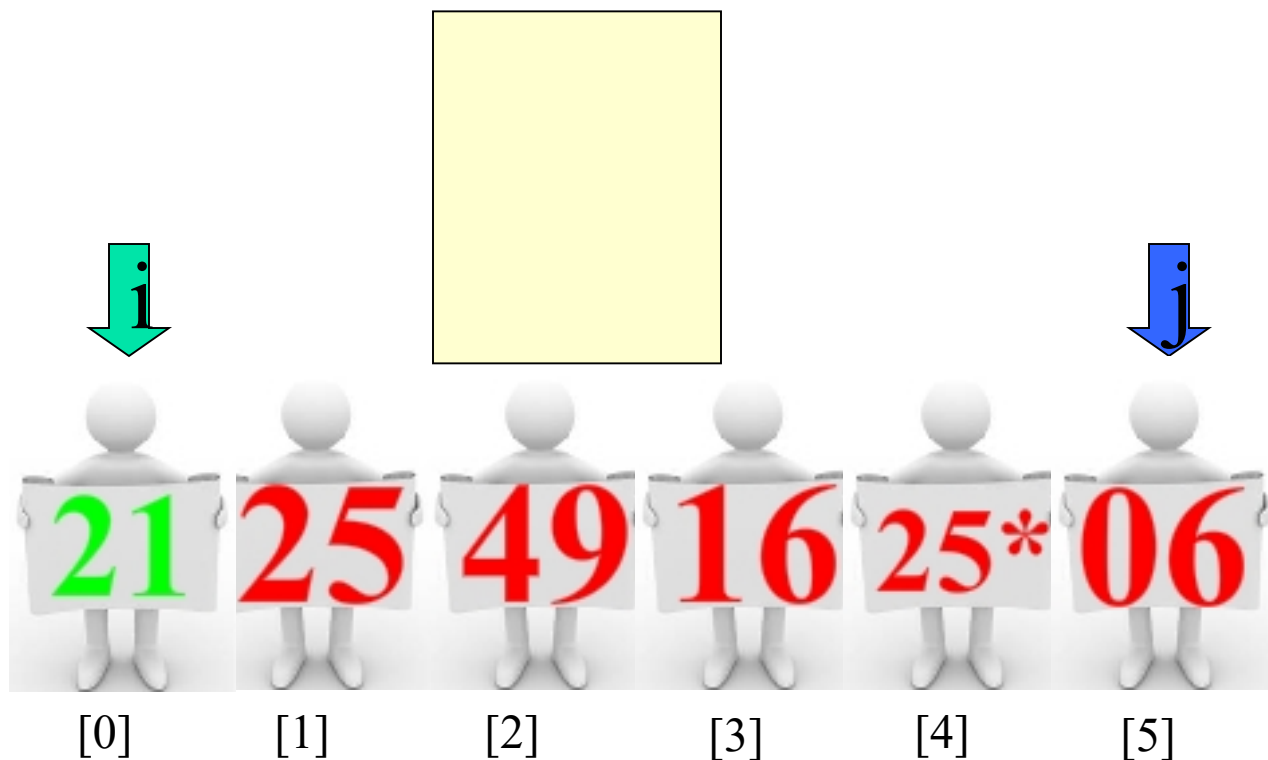
请同学们思考

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

算法终止



程序填空

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个小于中心元素的位置j
```

```
while(  > temp & 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

//从左向右找第1个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;



排序方法对比

- 冒泡排序 $\Theta(n^2)$
- 选择排序 $\Theta(n^2)$
- 插入排序 $\Theta(n^2)$
- 合并排序 $\Theta(n \log n)$
- 堆排序 $\Theta(n \log n)$
- 快速排序?



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：

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

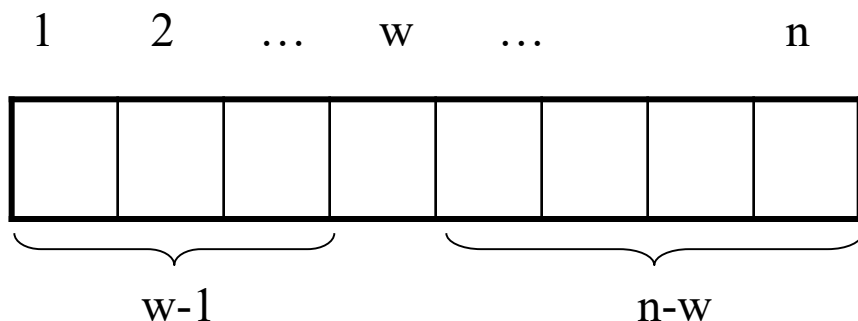
最差情形(已经排好序或是逆序的数组)：每次SPLIT后，只得到左或是右子数组，因此有：

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



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：

$$C(n) = (n-1) + \frac{1}{n} \sum_{w=1}^n (C(w-1) + C(n-w))$$

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

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



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

↓ n-1 替换 n

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

(a)-(b), 并适当变换

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

$$\text{令 } D(n) = \frac{C(n)}{n+1} \quad \downarrow$$

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

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

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

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



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);
    }
}
```



分治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，算法的时间复杂度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：

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

- 回顾Master Theorem



Master Theorem

设 $a \geq 1$, $b > 1$ 为常数。 $s(n)$ 为一给定的函数, $T(n)$ 递归定义如下:

$$T(n) = a \cdot T(n/b) + s(n)$$

并且 $T(n)$ 有适当的初始值。那么, 当 n 充分大时, 有:

- (1) 若存在 $\varepsilon > 0$, 使得 $s(n) = \Omega(n^{\log_b^a + \varepsilon})$ 成立, 并且存在 $c < 1$, 使得 $a \cdot s(n/b) \leq c \cdot s(n)$, 那么有 $T(n) = \Theta(s(n))$
- (2) 若 $s(n) = \Theta(n^{\log_b^a})$, 那么 $T(n) = \Theta(n^{\log_b^a} \cdot \log n)$
- (3) 若存在 $\varepsilon > 0$, 使得 $s(n) = O(n^{\log_b^a - \varepsilon})$ 成立, 那么有 $T(n) = \Theta(n^{\log_b^a})$



Master Theorem

- <https://www.youtube.com/watch?v=2H0GKdrIowU>

Theorem 5.1 *Let a be an integer greater than or equal to 1 and b be a real number greater than 1. Let c be a positive real number and d a nonnegative real number. Given a recurrence of the form*

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

then for n a power of b ,

1. *if $\log_b a < c$, $T(n) = \Theta(n^c)$,*
2. *if $\log_b a = c$, $T(n) = \Theta(n^c \log n)$,*
3. *if $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$.*

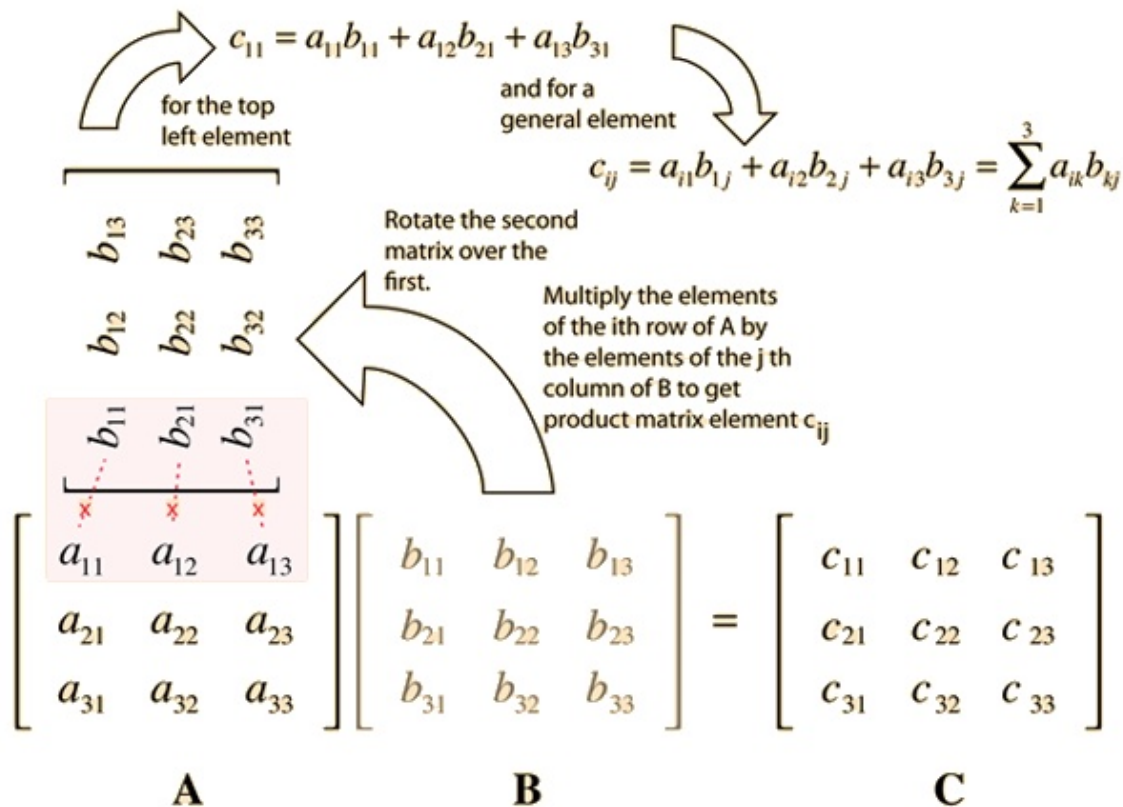


矩阵乘法

- 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





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 m ,每做一次标量加法耗费时间为 a ,那么直接相乘算法的时间复杂度为：

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



矩阵分块

$$A = \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 3 & 3 & 1 & 1 \\ 2 & 1 & 2 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 1 & 1 & 2 \\ 3 & 2 & 2 & 3 \\ 1 & 3 & 1 & 1 \\ 2 & 1 & 2 & 4 \end{pmatrix}$$

$$A \square B = \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 3 & 3 & 1 & 1 \\ 2 & 1 & 2 & 2 \end{pmatrix} \square \begin{pmatrix} 2 & 1 & 1 & 2 \\ 3 & 2 & 2 & 3 \\ 1 & 3 & 1 & 1 \\ 2 & 1 & 2 & 4 \end{pmatrix}$$

$$A \square B = \left(\begin{array}{cc|cc} 1 & 1 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ \hline 3 & 3 & 1 & 1 \\ 2 & 1 & 2 & 2 \end{array} \right) \square \left(\begin{array}{cc|cc} 2 & 1 & 1 & 2 \\ 3 & 2 & 2 & 3 \\ \hline 1 & 3 & 1 & 1 \\ 2 & 1 & 2 & 4 \end{array} \right)$$



方法2: 分块矩阵法 (直接应用分治策略)

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

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

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

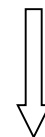
$$\mathbf{C}_{11} = \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21}$$

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

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

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

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



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



Strassen算法



Volker Strassen
giving the Knuth Prize
lecture at SODA 2009.

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



引入下列 $M_i(i=1,2,\dots,7)$:

$$M_1 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$M_2 = (A_{11} + A_{22})(B_{11} + B_{12})$$

$$M_3 = (A_{11} - A_{21})(B_{11} + B_{12})$$

$$M_4 = (A_{11} + A_{12})B_{22}$$

$$M_5 = A_{11}(B_{12} - B_{22})$$

$$M_6 = A_{22}(B_{21} - B_{11})$$

$$M_7 = (A_{21} + A_{22})B_{11}$$

则有: $C_{11} = M_1 + M_2 - M_4 + M_6$, $C_{12} = M_4 + M_5$,

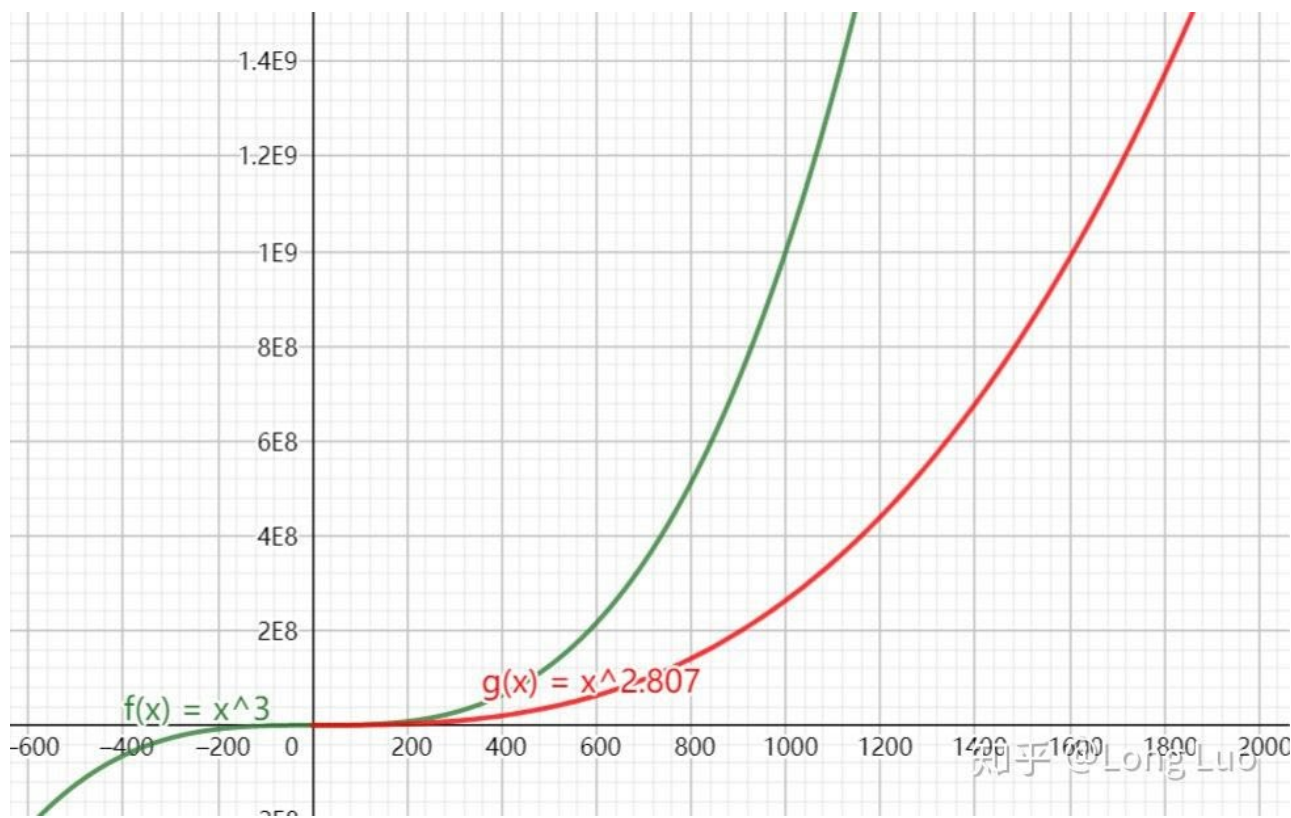
$$C_{21} = M_6 + M_7,$$

$$C_{22} = M_2 - M_3 + M_5 - M_7$$

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



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





算法对比

表 6.2 三种算法所做的算术运算的次数

表 6.3 STRASSEN 算法和传统算法的比较

	n	乘法	加法
传统算法	100	1 000 000	990 000
STRASSEN 算法	100	411 822	2 470 334
传统算法	1000	1 000 000 000	999 000 000
STRASSEN 算法	1000	264 280 285	1 579 681 709
传统算法	10 000	10^{12}	9.99×10^{12}
STRASSEN 算法	10 000	0.169×10^{12}	10^{12}



习题

- 求 x 的 n 次幂
- 复杂度为 $O(\log n)$ 的分治算法



```
1  #include "stdio.h"
2  #include "stdlib.h"
3
4  int power(int x, int n)
5  {
6      int result;
7      if(n == 1)
8          return x;
9      if( n % 2 == 0)
10         result = power(x, n/2) * power(x, n / 2);
11     else
12         result = power(x, (n+1) / 2) * power(x, (n-1) / 2);
13
14     return result;
15 }
16
17 int main()
18 {
19     int x = 5;
20     int n = 3;
21
22     printf("power(%d,%d) = %d \n",x, n, power(x, n));
23 }
```

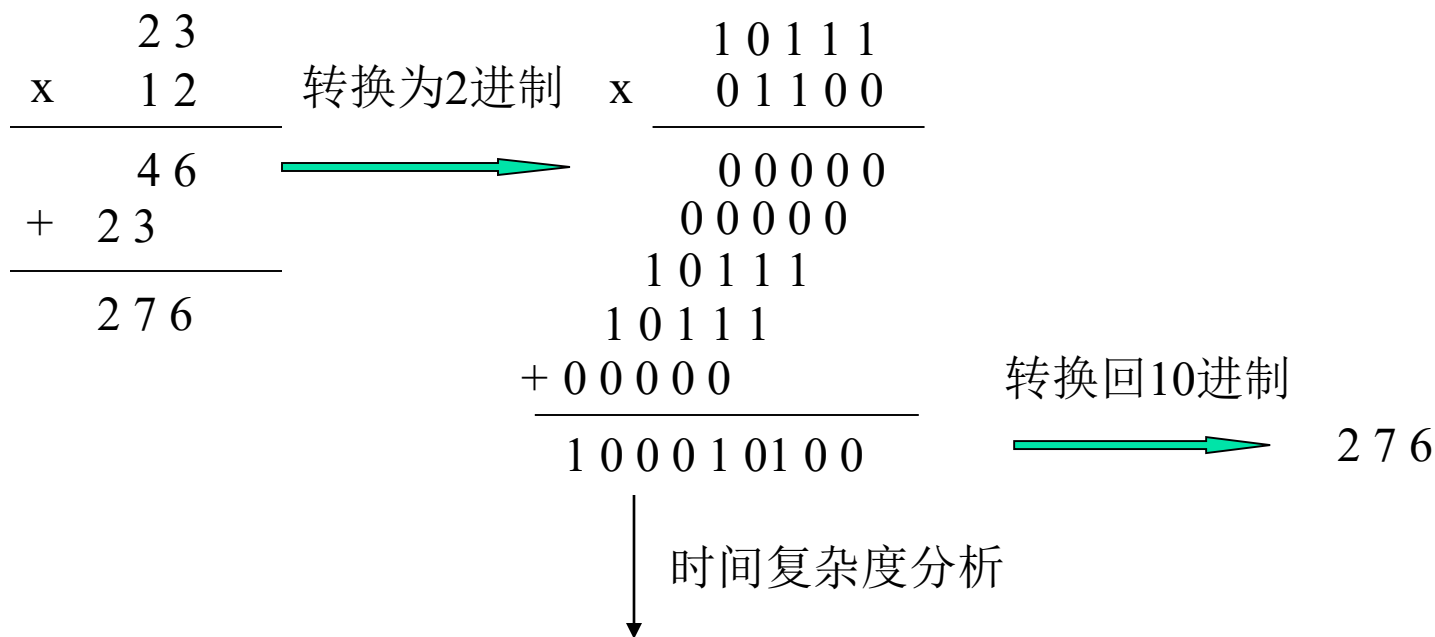


大整数相乘

- 通常，在分析算法的计算复杂度时，都将加法和乘法运算当作基本运算来处理，即将执行一次加法或乘法运算所需要的计算时间当作一个常数，该常数仅仅取决于计算机硬件处理速度。
- 然而，这个假定仅仅在参加运算的整数处于一定范围内时才是合理的。这个整数范围取决于计算机硬件对整数的表示。
- 在某些情况下，要处理很大的整数，它无法在计算机硬件能直接表示的整数范围内进行处理。这时候，就必须使用软件的方法来实现大整数的算术运算。



- 问题：设有两个n bit位的二进制整数X, Y，要计算XY.



将位(bit)的乘法或加法当作基本运算，则逐位相乘算法的时间复杂度为：

$$T(n) = \Theta(n^2)$$



使用分治策略来求解之

$X = \begin{array}{|c|c|} \hline \text{n/2} & \text{n/2} \\ \hline A & B \\ \hline \end{array} \quad Y = \begin{array}{|c|c|} \hline \text{n/2} & \text{n/2} \\ \hline C & D \\ \hline \end{array} \quad \xrightarrow{\text{例}} \quad X = \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 0 & 1 \\ \hline \end{array}$

我们有：

$$A = (11)_2 = 3$$

$$B = (01)_2 = 1$$

$$X = A \cdot 2^{n/2} + B \quad Y = C \cdot 2^{n/2} + D \quad \xrightarrow{\text{例}} \quad X = 3 \cdot 2^{4/2} + 1 = 13$$

下式成立：

$$\begin{aligned} XY &= (A \cdot 2^{n/2} + B)(C \cdot 2^{n/2} + D) \\ &= A \cdot C 2^n + (A \cdot D + C \cdot B) 2^{n/2} + B \cdot D \end{aligned}$$



时间复杂性分析

$$\begin{aligned} XY &= (A \cdot 2^{n/2} + B)(C \cdot 2^{n/2} + D) \\ &= A \cdot C \cdot 2^n + (A \cdot D + C \cdot B) \cdot 2^{n/2} + B \cdot D \end{aligned}$$

(1) 4次 $n/2$ bit位数的乘法($A \cdot C, A \cdot D, C \cdot B, B \cdot D$). // $4T(n/2)$

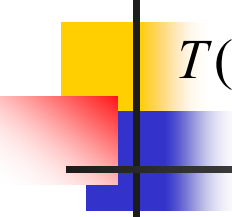
(2) $A \cdot C$ 左移 n 位($A \cdot C \cdot 2^n$). // $\Theta(n)$

(3) 求 $A \cdot D + C \cdot B$ 所作的 n 位加法. // $\Theta(n)$

(4) $A \cdot D + C \cdot B$ 左移 $n/2$ 位($(A \cdot D + C \cdot B) \cdot 2^{n/2}$). // $\Theta(n)$

(5) 两个加法($A \cdot C \cdot 2^n + (A \cdot D + C \cdot B) \cdot 2^{n/2} + B \cdot D$). // $\Theta(n)$





$$T(n) = \begin{cases} a & \text{if } n = 1 \\ 4T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases} \longrightarrow T(n) = \Theta(n^2)$$

如何降低时间复杂性？

观察：

$$A \cdot D + B \cdot C = (A + B)(C + D) - AC - BD$$

所以：

$$\begin{aligned} XY &= (A \cdot 2^{n/2} + B)(C \cdot 2^{n/2} + D) \\ &= A \cdot C \cdot 2^n + (A \cdot D + C \cdot B) \cdot 2^{n/2} + B \cdot D \\ &= A \cdot C \cdot 2^n + ((A + B)(C + D) - AC - BD) \cdot 2^{n/2} + B \cdot D \end{aligned}$$

$$T(n) = \begin{cases} a & , \text{ if } n = 1 \\ 3T(n/2) + \Theta(n) & , \text{ if } n > 1 \end{cases} \longrightarrow T(n) = \Theta(n^{\log 3}) = \Theta(n^{1.59})$$