

# 程序设计方法与实践



## 第5章:分治法

高广宇

guangyugao@bit.edu.cn

北京理工大学，计算机学院

# 提纲

## ① 算法效率分析基础

## ② 算法设计策略

### 第二讲 蛮力法

- 蛮力法 (brute force), 最简单的设计策略, 是一种简单直接地解决问题的方法, 常常直接基于问题的描述和所涉及的概念定义。

### 第三讲 分治法

- 分治法的定义
- 主定理与分治法
- 合并排序
- 大整数乘法
- 快速排序及矩阵乘法

# 分治法定义

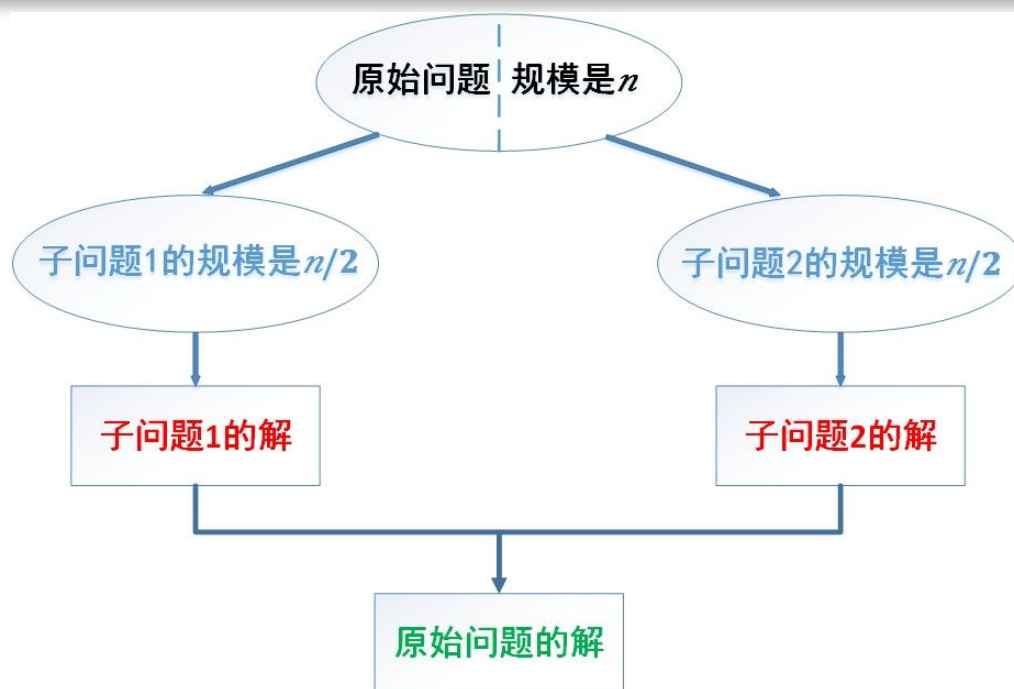
## 分治法：分而治之

1. 将原始问题划分为若干同类型子问题，子问题最好规模相同；
2. 对子问题求解（通常使用递归方法）；
3. 将子问题的求解结果合并，得到原问题的解。

# 分治法定义

## 分治法：分而治之

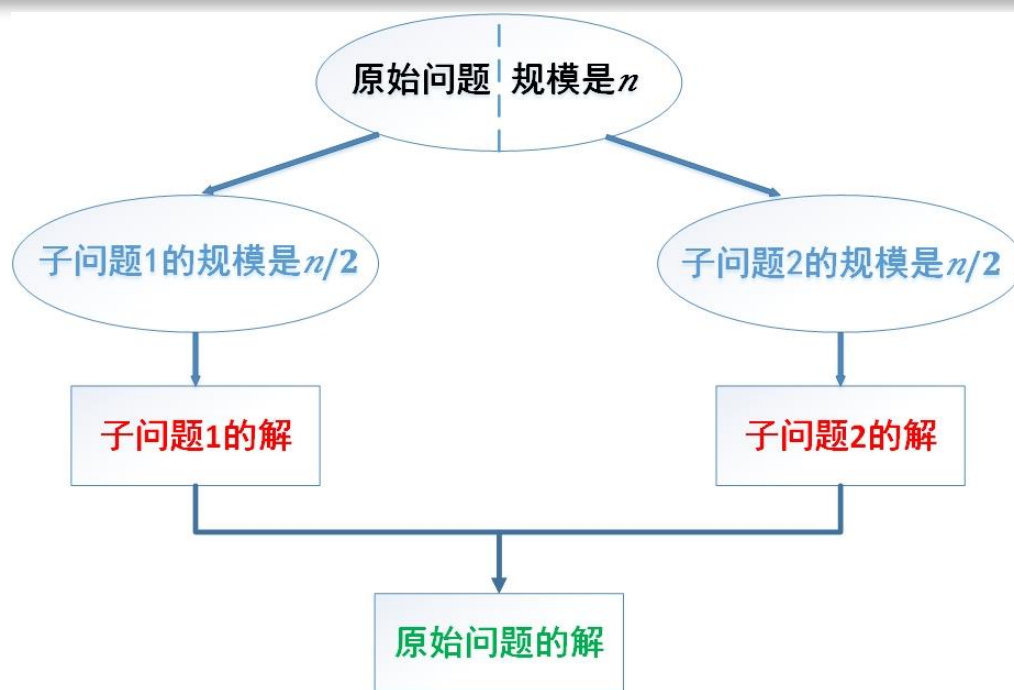
1. 将原始问题划分为若干同类型子问题，子问题最好规模相同；
2. 对子问题求解（通常使用递归方法）；
3. 将子问题的求解结果合并，得到原问题的解。



# 分治法定义

## 分治法：分而治之

1. 将原始问题划分为若干同类型子问题，子问题最好规模相同；
2. 对子问题求解（通常使用递归方法）；
3. 将子问题的求解结果合并，得到原问题的解。



分

治

合

# 主定理与分治法

## 主定理

- 对常数  $a > 0$ 、 $b > 1$  及  $d \geq 0$ ，有  $T(n) = aT(\lceil n/b \rceil) + O(n^d)$  成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

# 主定理与分治法

## 主定理

- 对常数  $a > 0$ 、 $b > 1$  及  $d \geq 0$ ，有  $T(n) = aT(\lceil n/b \rceil) + O(n^d)$  成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

分治算法遵循一种通用模式，即：在解决规模为  $n$  的问题时，总是先递归地分解  $a$  个规模为  $n/b$  的子问题，然后在  $O(n^d)$  时间内将子问题的解合并，其中  $a, b, d > 0$  是一些特定的正数。

# 合并排序

合并排序：需要排序的数组  $A[0, \dots, n - 1]$

# 合并排序

**合并排序**：需要排序的数组  $A[0, \dots n - 1]$

- 一分为二：  $A[0, \dots \lfloor n/2 \rfloor]$  和  $A[\lfloor n/2 \rfloor, \dots n - 1]$
- 对每个子数组以相同方式递归排序
- 将排好序的子数组合并为一个有序数组。

# 合并排序

**合并排序**：需要排序的数组  $A[0, \dots, n-1]$

- 一分为二：  $A[0, \dots, \lfloor n/2 \rfloor]$  和  $A[\lfloor n/2 \rfloor, \dots, n-1]$
- 对每个子数组以相同方式递归排序
- 将排好序的子数组合并为一个有序数组。

- **合并排序算法**

**算法** `Mergesort` ( $A[0, \dots, n-1]$ )

// 递归调用 `mergesort` 来对数组  $A[0, \dots, n-1]$  排序

// 输入：一个可排序数组  $A[0, \dots, n-1]$

// 输出：非降序排列的数组  $A[0, \dots, n-1]$

# 合并排序

**合并排序：**需要排序的数组 $A[0, \dots, n-1]$

- 一分为二： $A[0, \dots, \lfloor n/2 \rfloor]$ 和 $A[\lfloor n/2 \rfloor, \dots, n-1]$
- 对每个子数组以相同方式递归排序
- 将排好序的子数组合并为一个有序数组。

## ● 合并排序算法

**算法** `Mergesort`( $A[0, \dots, n-1]$ )

//递归调用mergesort来对数组 $A[0, \dots, n-1]$ 排序

//输入：一个可排序数组 $A[0, \dots, n-1]$

//输出：非降序排列的数组 $A[0, \dots, n-1]$

if  $n > 1$

    copy  $A[0, \dots, \lfloor n/2 \rfloor - 1]$  to  $B[0, \dots, \lfloor n/2 \rfloor - 1]$

    copy  $A[\lfloor n/2 \rfloor, \dots, n-1]$  to  $C[0, \dots, \lfloor n/2 \rfloor - 1]$

`Mergesort`( $B[0, \dots, \lfloor n/2 \rfloor - 1]$ )

`Mergesort`( $C[0, \dots, \lfloor n/2 \rfloor - 1]$ )

`Merge`( $B, C, A$ ) //参考教材P133

# 合并排序

- 合并排序算法

算法 **Mergesort** ( $A[0, \dots, n-1]$ )

//递归调用mergesort来对数组 $A[0, \dots, n-1]$ 排序

//输入：一个可排序数组 $A[0, \dots, n-1]$

//输出：非降序排列的数组 $A[0, \dots, n-1]$

if  $n > 1$

    copy  $A[0, \dots, \lfloor n/2 \rfloor - 1]$  to  $B[0, \dots, \lfloor n/2 \rfloor - 1]$

    copy  $A[\lfloor n/2 \rfloor, \dots, n-1]$  to  $C[0, \dots, \lfloor n/2 \rfloor - 1]$

**Mergesort** ( $B[0, \dots, \lfloor n/2 \rfloor - 1]$ )

**Mergesort** ( $C[0, \dots, \lfloor n/2 \rfloor - 1]$ )

    Merge ( $B, C, A$ ) //参考教材P133

# 合并排序

- 合并排序算法

算法 `Mergesort` ( $A[0, \dots, n-1]$ )

//递归调用mergesort来对数组 $A[0, \dots, n-1]$ 排序

//输入：一个可排序数组 $A[0, \dots, n-1]$

//输出：非降序排列的数组 $A[0, \dots, n-1]$

if  $n > 1$

    copy  $A[0, \dots, \lfloor n/2 \rfloor - 1]$  to  $B[0, \dots, \lfloor n/2 \rfloor - 1]$

    copy  $A[\lfloor n/2 \rfloor, \dots, n-1]$  to  $C[0, \dots, \lfloor n/2 \rfloor - 1]$

`Mergesort` ( $B[0, \dots, \lfloor n/2 \rfloor - 1]$ )

`Mergesort` ( $C[0, \dots, \lfloor n/2 \rfloor - 1]$ )

    Merge ( $B, C, A$ ) //参考教材P133

- copy操作需要线性时间
- Merge操作需要线性时间
- 基本操作： `Mergesort`( $n$ )，需要需要 $T(n)$ 时间

# 合并排序

## 分治法之合并排序

- 分：把 $A$ 数组一分为二： $B = A[0, \dots, \lfloor n/2 \rfloor]$ 和 $C = A[\lfloor n/2 \rfloor, \dots, n-1]$ .
- 治：求解 $\text{Mergesort}(B)$ 和 $\text{Mergesort}(C)$ .
- 合：在 $O(n)$ 时间内计算 $\text{Merge}(B, C, A)$ .
- 递推式： $T(n) = 2T(n/2) + O(n)$

# 合并排序

## 分治法之合并排序

- **分**：把 $A$ 数组一分为二： $B = A[0, \dots, \lfloor n/2 \rfloor]$ 和 $C = A[\lfloor n/2 \rfloor, \dots, n-1]$ .
- **治**：求解 $\text{Mergesort}(B)$ 和 $\text{Mergesort}(C)$ .
- **合**：在 $O(n)$ 时间内计算 $\text{Merge}(B, C, A)$ .
- 递推式： $T(n) = 2T(n/2) + O(n)$

### 主定理

- 对常数 $a > 0$ 、 $b > 1$ 及 $d \geq 0$ ，有 $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ 成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

# 合并排序

## 分治法之合并排序

- **分**：把 $A$ 数组一分为二： $B = A[0, \dots, \lfloor n/2 \rfloor]$ 和 $C = A[\lfloor n/2 \rfloor, \dots, n-1]$ .
- **治**：求解 $\text{Mergesort}(B)$ 和 $\text{Mergesort}(C)$ .
- **合**：在 $O(n)$ 时间内计算 $\text{Merge}(B, C, A)$ .
- 递推式： $T(n) = 2T(n/2) + O(n)$

### 主定理

- 对常数 $a > 0$ 、 $b > 1$ 及 $d \geq 0$ ，有 $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ 成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

因为 $a = 2, b = 2, d = 1$ ,  
 $\log_b a = 1, d = 1$ , 所以,

# 合并排序

## 分治法之合并排序

- **分**：把 $A$ 数组一分为二： $B = A[0, \dots, \lfloor n/2 \rfloor]$ 和 $C = A[\lfloor n/2 \rfloor, \dots, n-1]$ .
- **治**：求解 $\text{Mergesort}(B)$ 和 $\text{Mergesort}(C)$ .
- **合**：在 $O(n)$ 时间内计算 $\text{Merge}(B, C, A)$ .
- 递推式： $T(n) = 2T(n/2) + O(n)$

### 主定理

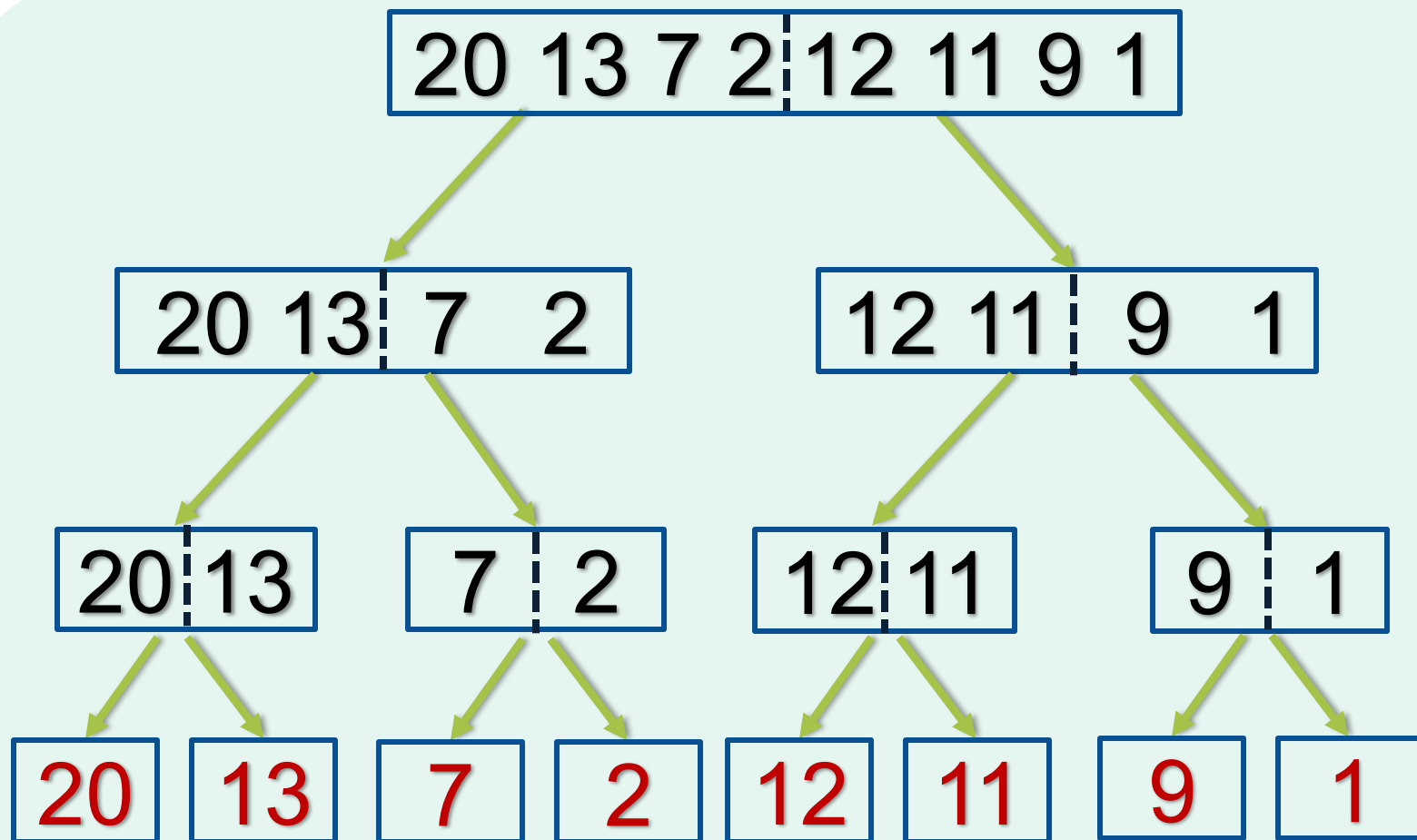
- 对常数 $a > 0$ 、 $b > 1$ 及 $d \geq 0$ ，有 $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ 成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

因为 $a = 2, b = 2, d = 1$ ,  
 $\log_b a = 1, d = 1$ , 所以,  
 $T(n) = O(n \log n)$

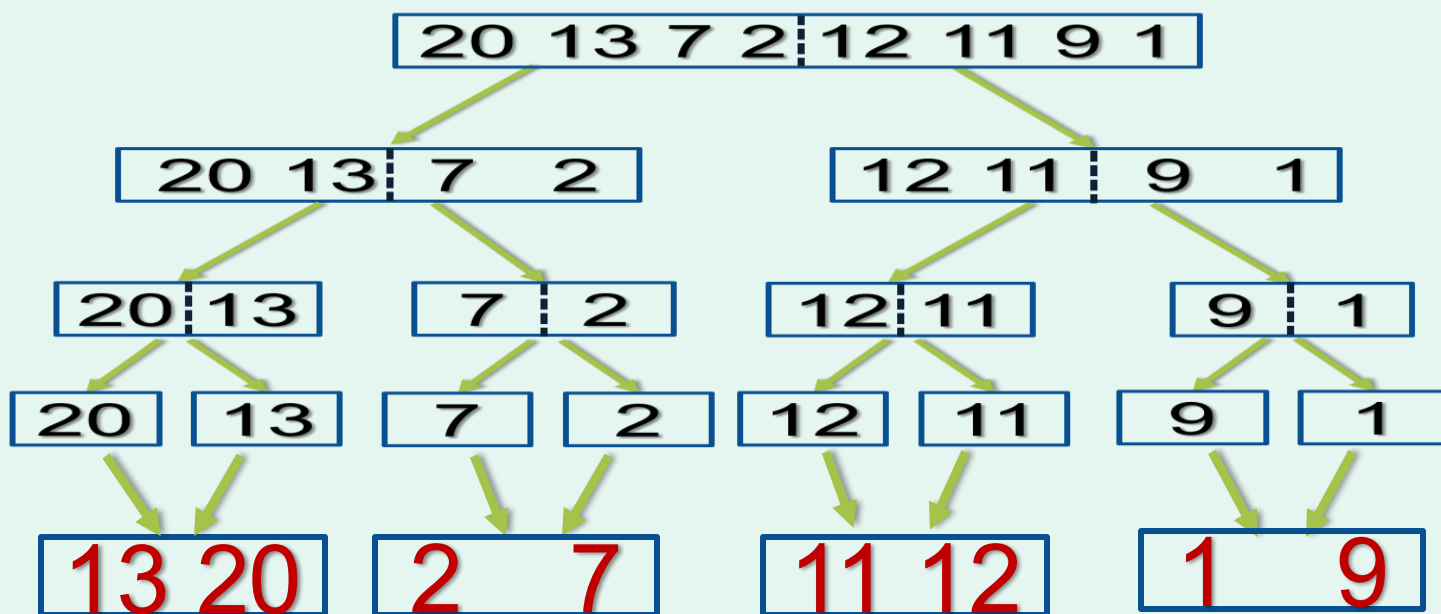
# 合并排序

**示例：**演示对数列 $A[0, \dots, 7] = \{20, 13, 7, 2, 12, 11, 9, 1\}$ 合并排序的过程。



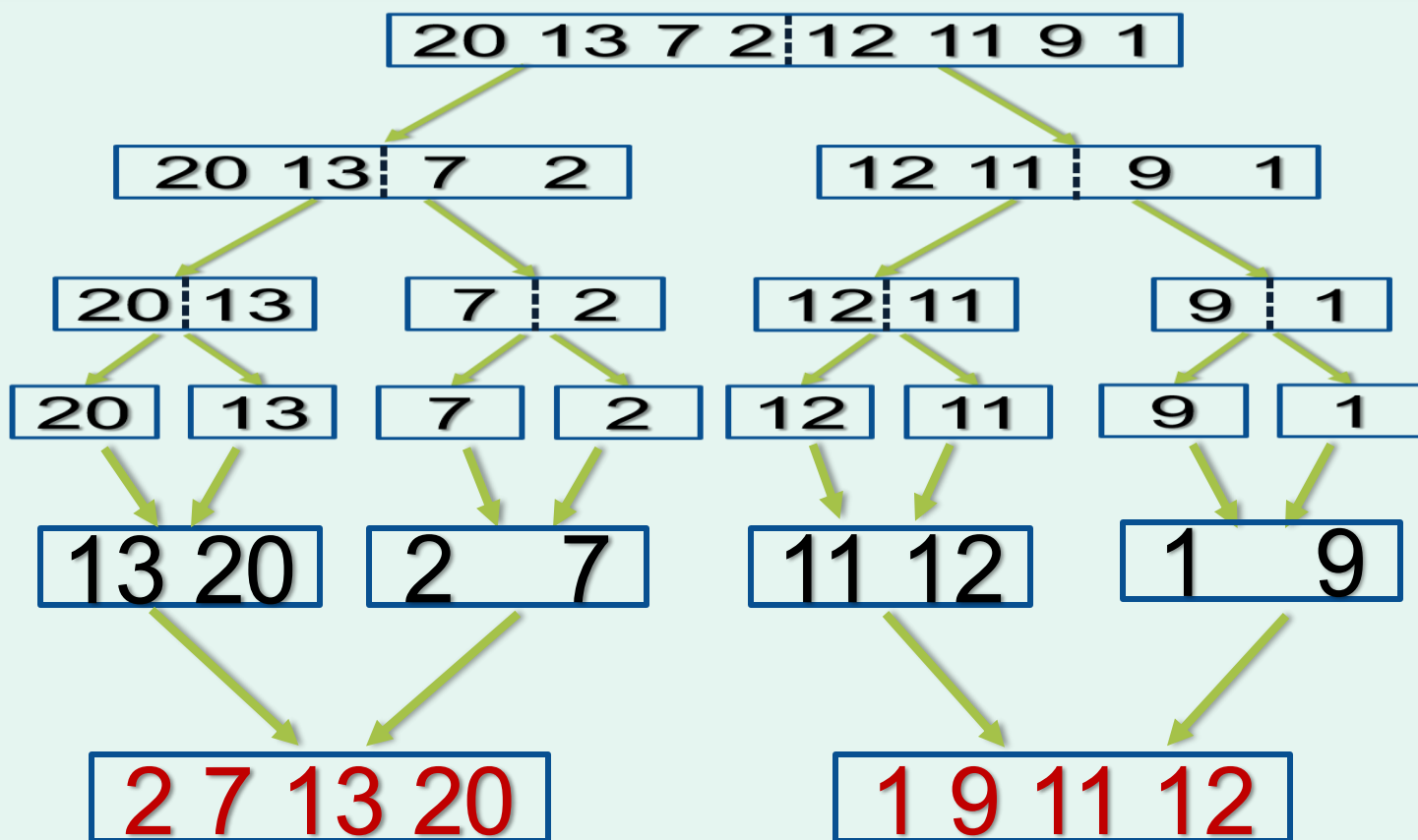
# 合并排序

**示例：**演示对数列 $A[0, \dots, 7] = \{20, 13, 7, 2, 12, 11, 9, 1\}$ 合并排序的过程。



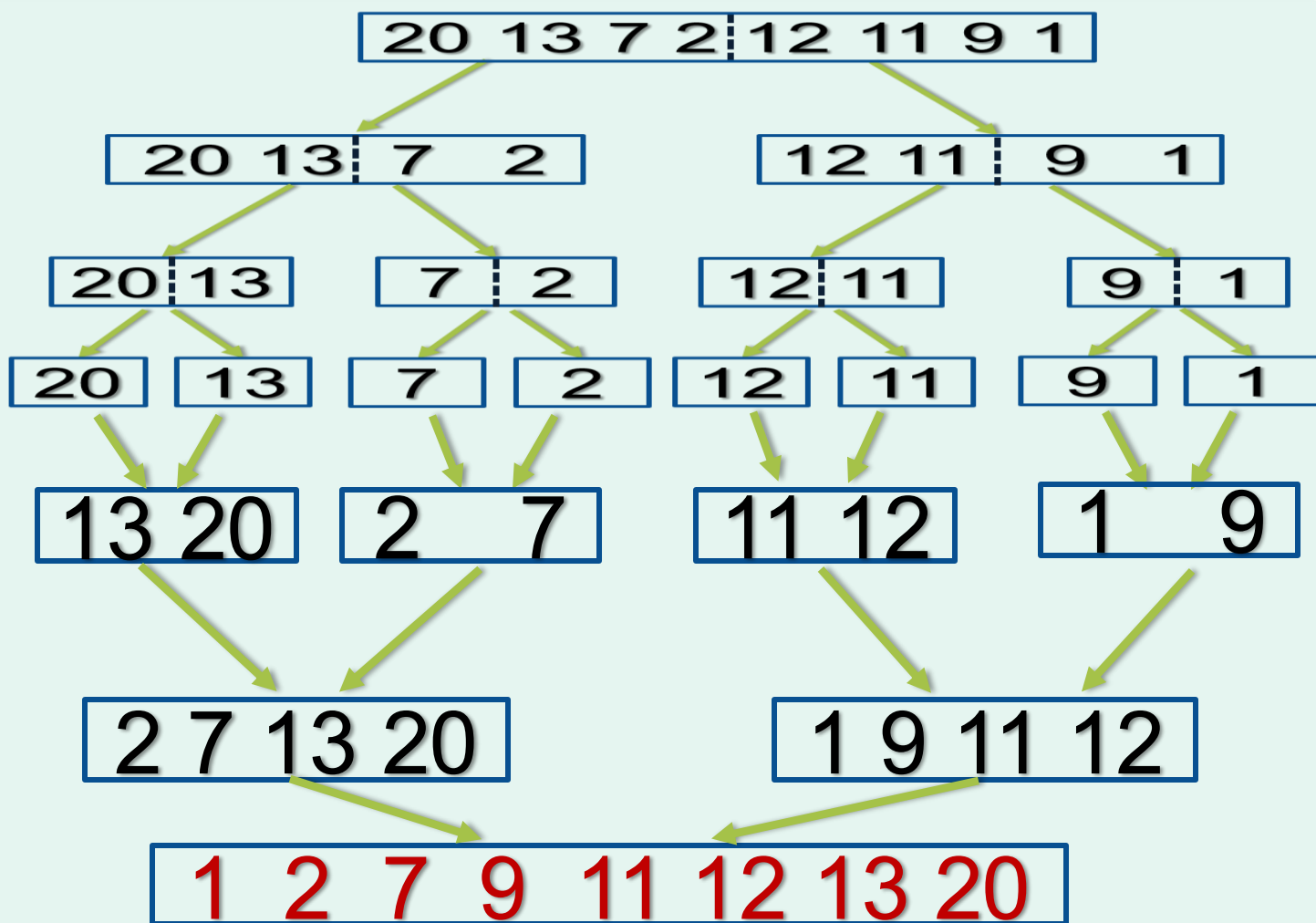
# 合并排序

**示例：**演示对数列 $A[0, \dots, 7] = \{20, 13, 7, 2, 12, 11, 9, 1\}$ 合并排序的过程。



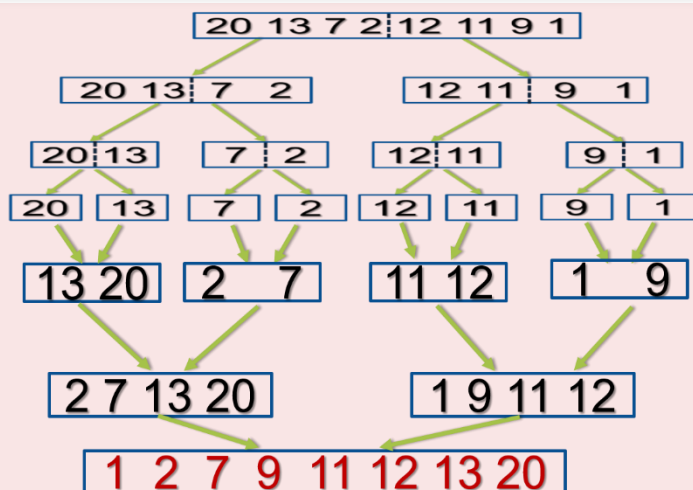
# 合并排序

示例：演示对数列  $A[0, \dots, 7] = \{20, 13, 7, 2, 12, 11, 9, 1\}$  合并排序的过程。



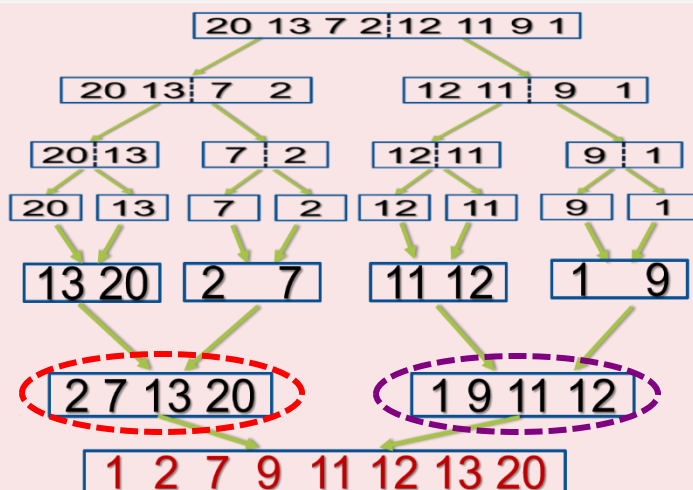
# 合并排序

**示例：**演示对数列 $A[0, \dots, 7] = \{20, 13, 7, 2, 12, 11, 9, 1\}$ 合并排序的过程。



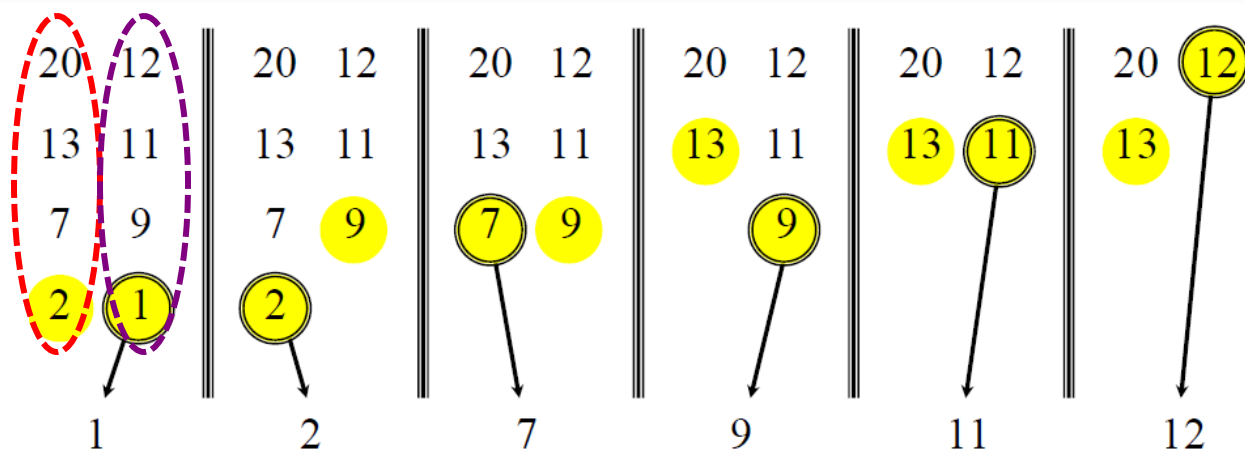
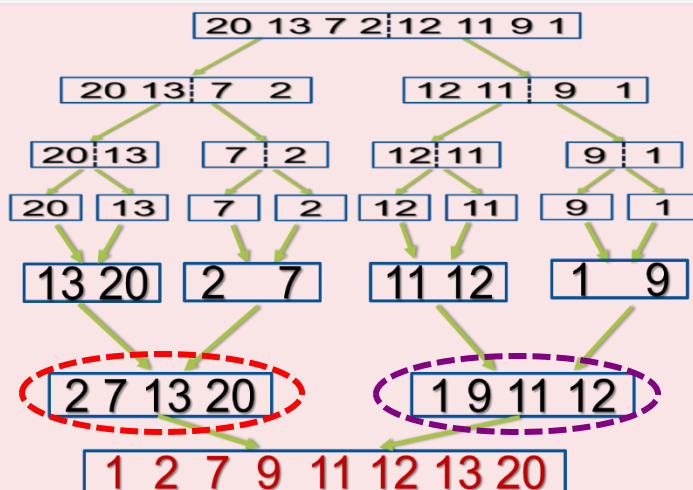
# 合并排序

**示例：**演示对数列 $A[0, \dots, 7] = \{20, 13, 7, 2, 12, 11, 9, 1\}$ 合并排序的过程。



# 合并排序

**示例：**演示对数列 $A[0, \dots, 7] = \{20, 13, 7, 2, 12, 11, 9, 1\}$ 合并排序的过程。



- ①  $A[0..n-1]$  是  $n$  个不同实数构成的数组。如果  $i < j$ ，但是  $A[i] > A[j]$ ，则  $(A[i], A[j])$  被称为一个倒置 inversion。请设计一个  $O(n \log n)$  的算法计算数组中的倒置数量。

- ①  $A[0..n-1]$  是  $n$  个不同实数构成的数组。如果  $i < j$ , 但是  $A[i] > A[j]$ , 则  $(A[i], A[j])$  被称为一个倒置 *inversion*。请设计一个  $O(n \log n)$  的算法计算数组中的倒置数量。

Let *ModifiedMergesort* be a mergesort modified to return the number of inversions in its input array  $A[0..n-1]$  in addition to sorting it. Obviously, for an array of size 1, *ModifiedMergesort*( $A[0]$ ) should return 0. Let  $i_{left}$  and  $i_{right}$  be the number of inversions returned by *ModifiedMergesort*( $A[0..mid-1]$ ) and *ModifiedMergesort*( $A[mid..n-1]$ ), respectively, where  $mid$  is the index of the middle element in the input array  $A[0..n-1]$ . The total number of inversions in  $A[0..n-1]$  can then be computed as  $i_{left} + i_{right} + i_{merge}$ , where  $i_{merge}$ , the number of inversions involving elements from both halves of  $A[0..n-1]$ , is computed during the merging as follows. Let  $A[i]$  and  $A[j]$  be two elements from the left and right half of  $A[0..n-1]$ , respectively, that are compared during the merging. If  $A[i] < A[j]$ , we output  $A[i]$  to the sorted list without incrementing  $i_{merge}$  because  $A[i]$  cannot be a part of an inversion with any of the remaining elements in the second half, which are greater than  $A[j]$ . If, on the other hand,  $A[i] > A[j]$ , we output  $A[j]$  and increment  $i_{merge}$  by  $mid - i$ , the number of remaining elements in the first half, because all those elements (and only they) form an inversion with  $A[j]$ .

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如，1536和2487，计算如下：

$$1536 = 15 \times 10^2 + 36 = 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0,$$

$$2487 = 24 \times 10^2 + 87 = 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如，1536和2487，计算如下：

$$1536 = 15 \times 10^2 + 36 = 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0,$$

$$2487 = 24 \times 10^2 + 87 = 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 3820032$ ,

共使用 $4 \times 4 = 16$ 次位乘。

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如，1536和2487，计算如下：

$$1536 = 15 \times 10^2 + 36 = 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0,$$

$$2487 = 24 \times 10^2 + 87 = 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ,

共使用 $4 \times 4 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [\text{左半部分}][\text{右半部分}] = [x_L][x_R]$$

$$y = [\text{左半部分}][\text{右半部分}] = [y_L][y_R]$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n=4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R]$$

$$y = [y_L][y_R]$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n=4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R$$

$$y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n = 4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R$$

$$y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R)$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n = 4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R$$

$$y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$\begin{aligned}x \times y &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\ &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n=4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R, \quad y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n = 4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R, \quad y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2} (x_L y_R + x_R y_L) + x_R y_R$$

$$x_L y_R + x_R y_L =$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n = 4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R, \quad y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2} (x_L y_R + x_R y_L) + x_R y_R$$

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n=4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R, \quad y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2} (x_L y_R + x_R y_L) + x_R y_R$$

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2} (x_L y_R + x_R y_L) + x_R y_R$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n=4$ ) 位数1536和2487，计算如下：

$$\begin{aligned}1536 &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 \\ &= 15 \times 10^2 + 36,\end{aligned}$$

$$\begin{aligned}2487 &= 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 \\ &= 24 \times 10^2 + 87\end{aligned}$$

所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，共 $n^2 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R, \quad y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2} (x_L y_R + x_R y_L) + x_R y_R$$

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2} ((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R) + x_R y_R$$

# 大整数乘法

大整数乘法，如对>100位的十进制/二进制整数做乘法。

- 常规整数相乘，如， $n$  ( $n = 4$ ) 位数1536和2487，计算如下：  
 $1536 = 15 \times 10^2 + 36 = 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0$ ，  
 $2487 = 24 \times 10^2 + 87 = 2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0$   
所以， $1536 \times 2487 = (1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 6 \times 10^0) \times (2 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 7 \times 10^0) = 210$ ，  
共使用 $n^2 = 4 \times 4 = 16$ 次位乘。

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R, \quad y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(\boxed{x_L y_R + x_R y_L}) + x_R y_R$$

$$\boxed{x_L y_R + x_R y_L} = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

# 大整数乘法

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

# 大整数乘法

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

- 加法操作需要线性时间

# 大整数乘法

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

- 加法操作需要线性时间
- 乘以2的幂次方的操作需要线性时间（相当于移位）

# 大整数乘法

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

- 加法操作需要线性时间
- 乘以2的幂次方的操作需要线性时间（相当于移位）
- 基本操作： $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$

# 大整数乘法

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

- 加法操作需要线性时间
- 乘以2的幂次方的操作需要线性时间（相当于移位）
- 基本操作： $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$

$x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$ 是什么？

# 大整数乘法

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

- 加法操作需要线性时间
- 乘以2的幂次方的操作需要线性时间（相当于移位）
- 基本操作： $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$

$x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$ 是什么？

长度为 $n/2$ 位的二进制大整数乘法！

# 大整数乘法

二进制大整数乘法，长度都为 $n$ 的二进制数 $x, y$ 的乘法运算，

$$x \times y = (2^n - 2^{n/2})x_L y_L + 2^{n/2}(x_L + x_R)(y_L + y_R) + (1 - 2^{n/2})x_R y_R$$

- 加法操作需要线性时间
- 乘以2的幂次方的操作需要线性时间（相当于移位）
- 基本操作： $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$

$x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$ 是什么？

长度为 $n/2$ 位的二进制大整数乘法！

二进制大整数乘法，长度为 $n$ 的二进制数 $x, y$ 的乘法运算，

1. 调用3个 $n/2$ 位二进制乘法子问题，
2. 在 $O(n)$ 时间内汇总计算最终结果。

分治法！

# 大整数乘法

## 分治法之大整数乘法

- 分：长度为 $n$ 的二进制数 $x, y$ 划分成左右部分 $x_L, x_R, y_L, y_R$ .
- 治：求解 $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$
- 合：在 $O(n)$ 时间内加法计算最终结果：

$$2^n x_L y_L + 2^{n/2} \left( (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R \right) + x_R y_R$$

- 递推式： $T(n) = 3T(n/2) + O(n)$

# 大整数乘法

## 分治法之大整数乘法

- 分：长度为 $n$ 的二进制数 $x, y$ 划分成左右部分 $x_L, x_R, y_L, y_R$ .
- 治：求解 $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$
- 合：在 $O(n)$ 时间内加法计算最终结果：

$$2^n x_L y_L + 2^{n/2} \left( (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R \right) + x_R y_R$$

- 递推式： $T(n) = 3T(n/2) + O(n)$

### 主定理

- 对常数 $a > 0$ 、 $b > 1$ 及 $d \geq 0$ ，有 $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ 成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^{d \log b}), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

# 大整数乘法

## 分治法之大整数乘法

- 分：长度为 $n$ 的二进制数 $x, y$ 划分成左右部分 $x_L, x_R, y_L, y_R$ .
- 治：求解 $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$
- 合：在 $O(n)$ 时间内加法计算最终结果：

$$2^n x_L y_L + 2^{n/2}((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R) + x_R y_R$$

- 递推式： $T(n) = 3T(n/2) + O(n)$

### 主定理

- 对常数 $a > 0$ 、 $b > 1$ 及 $d \geq 0$ ，有 $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ 成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

因为 $a = 3, b = 2, d = 1$ ,  
 $\log_b a \approx 1.59, d < 1.59$  所以，

# 大整数乘法

## 分治法之大整数乘法

- 分：长度为 $n$ 的二进制数 $x, y$ 划分成左右部分 $x_L, x_R, y_L, y_R$ .
- 治：求解 $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$
- 合：在 $O(n)$ 时间内加法计算最终结果：

$$2^n x_L y_L + 2^{n/2} ((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R) + x_R y_R$$

- 递推式： $T(n) = 3T(n/2) + O(n)$

### 主定理

- 对常数 $a > 0$ 、 $b > 1$ 及 $d \geq 0$ ，有 $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ 成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

因为 $a = 3, b = 2, d = 1$ ,  
 $\log_b a \approx 1.59, d < 1.59$  所以,  
 $T(n) = O(n^{1.59})$

# 大整数乘法

## 分治法之大整数乘法

- 分：长度为 $n$ 的二进制数 $x, y$ 划分成左右部分 $x_L, x_R, y_L, y_R$ .
- 治：求解 $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$
- 合：在 $O(n)$ 时间内加法计算最终结果：

$$2^n x_L y_L + 2^{n/2} ((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R) + x_R y_R$$

- 递推式： $T(n) = 3T(n/2) + O(n)$

### 主定理

- 对常数 $a > 0$ 、 $b > 1$ 及 $d \geq 0$ ，有 $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ 成立，则，

$$T(n) = \begin{cases} O(n^d), & \text{如果 } d > \log_b a \\ O(n^d \log n), & \text{如果 } d = \log_b a \\ O(n^{\log_b a}), & \text{如果 } d < \log_b a \end{cases}$$

因为 $a = 3, b = 2, d = 1$ ,  
 $\log_b a \approx 1.59, d < 1.59$  所以,

$$T(n) = O(n^{1.59})$$

常规乘法

位乘次数是位数乘积： $n^2$

# 快速排序

算法思路:

对于输入 $A[0..n-1]$ , 按以下三个步骤进行排序:

(1) 分区: 取 $A$ 中的一个元素为中心点(pivot) 将 $A[0..n-1]$ 划分成3段:  $A[0..s-1]$ ,  $A[s]$ ,  $A[s+1..n-1]$ , 使得

$A[0..s-1]$ 中任一元素 $\leq A[s]$ ,

$A[s+1..n-1]$ 中任一元素 $\geq A[s]$ ;

下标 $s$ 在划分过程中确定。

(2) 递归求解: 递归调用快速排序法分别对 $A[0..s-1]$ 和 $A[s+1..n-1]$ 排序。

(3) 合并: 合并 $A[0..s-1]$ ,  $A[s]$ ,  $A[s+1..n-1]$ 为 $A[0..n-1]$

# 快速排序

快速排序算法 QuickSort( $A[l \dots r]$ )

// 使用快速排序法对序列或者子序列排序

// 输入：子序列 $A[l..r]$ 或者序列本身 $A[0..n-1]$

// 输出：非递减序列 $A$

if  $l < r$

$s \leftarrow \text{Partition}(A[l..r])$

    QuickSort( $A[l..s-1]$ )

    QuickSort( $A[s+1..r]$ )

// $s$ 是中轴元素/基准点，是数组分区位置的标志  
中轴元素怎么选？

# 快速排序

**算法** LomutoPartition( $A[l..r]$ )

//采用 Lomuto 算法，用第一个元素作为中轴对子数组进行划分

//输入：数组  $A[0..n-1]$  的一个子数组  $A[l..r]$ ，它由左右两边的索引  $l$  和  $r$  ( $l \leq r$ ) 定义

//输出： $A[l..r]$  的划分和中轴的新位置

$p \leftarrow A[l]$

$s \leftarrow l$

**for**  $i \leftarrow l + 1$  **to**  $r$  **do**

**if**  $A[i] < p$

$s \leftarrow s + 1$ ; swap( $A[s], A[i]$ )

swap( $A[l], A[s]$ )

**return**  $s$

# 分区算法

该算法使用了霍尔(A.R. Hoare)两次扫描方法:

与Lomuto算法不同, 从子数组**两端扫描**与中轴元素比较。

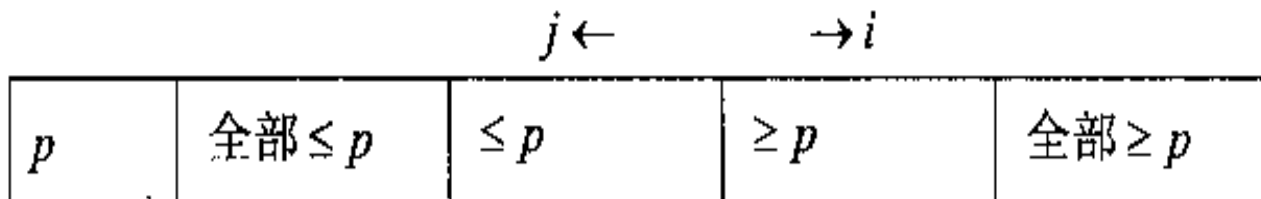
- 指针 $i$ 从数组左边开始扫描, **忽略小于中轴的元素**, 遇到大于等于中轴的元素 $A[i]$ 时停止。
- 指针 $j$ 从数组右边开始扫描, **忽略大于中轴的元素**, 遇到小于等于中轴的元素 $A[j]$ 时停止, 然后交换 $A[i]$ 和 $A[j]$ 。
- 指针不相交则继续。

# 分区算法

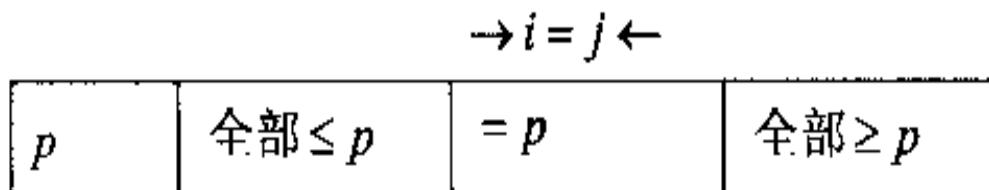
两次扫描全部停止以后，取决于扫描的指针是否相交，会发生 3 种不同的情况。如果扫描指针  $i$  和  $j$  不相交，也就是说  $i < j$ ，我们简单地交换  $A[i]$  和  $A[j]$ ，再分别对  $i$  加 1，对  $j$  减 1，然后继续开始扫描。



如果扫描指针相交，也就是说  $i > j$ ，把中轴和  $A[j]$  交换以后，我们得到了该数组的一个划分。



最后，如果扫描指针停下来时指向的是同一个元素，也就是说  $i = j$ ，被指向元素的值一定等于  $p$ （为什么？）。因此，我们建立了该数组的一个分区：



# 数组的分区算法

算法 HoarePartition( $A[l..r]$ )

//以第一个元素为中轴，对子数组进行划分

//输入：数组 $A[0 \dots n-1]$ 的子数组 $A[l..r]$

//输出： $A[l \dots r]$ 的一个划分，分裂点的位置作为函数的返回值

$p \leftarrow A[l]$

$i \leftarrow l; j \leftarrow r + 1;$

repeat

    repeat  $i \leftarrow i + 1$  until  $A[i] \geq p$

    repeat  $j \leftarrow j - 1$  until  $A[j] \leq p$

    swap( $A[i], A[j]$ )

until  $i \geq j$

swap( $A[i], A[j]$ )     // 当 $i \geq j$ ，撤销最后一次交换

swap( $A[l], A[j]$ )     // 把中轴的值放到对应位置

return  $j$ ;

# 快速排序的例子（双向扫描）

例如：  $n = 8$

初始数组  $A[0..n-1] = [8, 4, 1, 7, 11, 5, 6, 9]$ ,

取元素  $A[0] = 8$  作为分裂点,

位置: 0 1 2 3 4 5 6 7

1. {**8**, 4, 1, 7, 11, 5, 6, 9}

$i \uparrow$

$j \uparrow$

指针  $i$ 、 $j$  分别向中间移动

2. {**8**, 4, 1, 7, 11, 5, 6, 9}

$i \uparrow$

$j \uparrow$

符合条件，指针停止数据交换

3. {**8**, 4, 1, 7, 6, 5, 11, 9}

$i \uparrow$

$j \uparrow$

4. {5, 4, 1, 7, 6, **8**, 11, 9}

# 快速排序的例子（双向扫描）

分解得：

$A[0..s-1]=[5, 4, 1, 7, 6]$ ;  $A[s]=8$ ;  $A[s+1..7]=[11,9]$ ;  $s=5$

排序：

$A[0..s-1]=[1, 4, 5, 6, 7]$ ;  $A[s+1..n-1]=[9, 11]$ 。

合并：

把 $A[0..s-1]$ 中的元素放在分裂点元素8之前,  $A[s+1..n-1]$ 中的元素放在分裂点元素之后, 结果 $[1, 4, 5, 6, 7, 8, 9, 11]$

# 快速排序效率分析

基本操作：比较

最优情况下：所有分裂点均处中部

$$\text{当 } n > 1 \text{ 时, } C_{best}(n) = 2C_{best}(n/2) + n$$

$$C_{best}(1) = 0$$

$$\text{由主定理理解得 } C_{best}(n) \in \Theta(n \log_2 n)$$

# 快速排序效率分析

最坏情况下：所有分裂点均处于极端

在进行 $n + 1$ 次比较( $i, j$ 指针交叉)后建立了分区，还会对数组进行排序，继续到最后一个子数组 $A[n - 2..n - 1]$ 。总比较次数为：

$$\begin{aligned} C(n) &= (n + 1) + n + \cdots + 3 \\ &= (n + 2)(n + 1)/2 - 3 \\ &\in \Theta(n^2) \end{aligned}$$

最坏时间复杂度： $O(n^2)$   
平均时间复杂度： $O(n \log n)$   
稳定性：不稳定

# 快速排序效率分析

算法	平均速度	空间占用	实际表现
快速排序	极快 ( $O(n \log n)$ )	很少 ( $O(\log n)$ )	<b>综合冠军</b> ，在速度和空间上取得了最佳平衡。
归并排序	很快 ( $O(n \log n)$ )	很多 ( $O(n)$ )	因为太占空间，通常不用在内存排序中。
堆排序	较快 ( $O(n \log n)$ )	最少 ( $O(1)$ )	实际速度不如快速排序，但在某些对空间要求极其苛刻的场景下有优势。

# 快速排序效率分析

算法名称	最好时间复杂度	平均时间复杂度	最坏时间复杂度	空间复杂度	稳定性	排序方法
快速排序 (Quick Sort)	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$ (栈空间)	不稳定	交换/分区
归并排序 (Merge Sort)	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$ (额外空间)	稳定	合并
堆排序 (Heap Sort)	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	不稳定	选择/堆化
插入排序 (Insertion Sort)	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定	插入
选择排序 (Selection Sort)	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定	选择
冒泡排序 (Bubble Sort)	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定	交换

# Strassen矩阵乘法

矩阵乘法是线性代数中最常见的运算之一，它在数值计算中有广泛的应用。若A和B是2个  $n \times n$  的矩阵，则它们的乘积  $C = A \times B$  同样是一个  $n \times n$  的矩阵。A和B的乘积矩阵C中的元素  $c[i, j]$  定义为：

$$c[i, j] = \sum_{k=1}^n A[i, k] B[k, j]$$

若依此定义来计算A和B的乘积矩阵C，则每计算C的一个元素  $c[i, j]$ ，需要做  $n$  个乘法和  $n - 1$  次加法。因此，求出矩阵C的  $n^2$  个元素所需的计算时间为  $O(n^3)$ 。

# Strassen矩阵乘法

- Strassen采用了分治技术，将计算2个 $n$ 阶矩阵乘积所需的计算时间改进到

$$O(n\log_2 7) = O(n^{2.18})。$$

- 首先，假设 $n = 2^k$ 。将矩阵A，B和C中每一矩阵都分块成为4个大小相等的子矩阵，每个子矩阵都是 $n/2 \times n/2$ 的方阵。由此可将方程 $C = A \times B$ 重写为：

# Strassen矩阵乘法

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

其中：

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

## Strassen矩阵乘法

则2个2阶方阵的乘积可以直接用上式计算出来，共需8次乘法和4次加法。当子矩阵的阶大于2时，为求2个子矩阵的积，可以继续将子矩阵分块，直到子矩阵的阶降为2。

依此算法，计算2个 $n$ 阶方阵乘积转化为计算8个 $n/2$ 阶方阵的乘积和4个 $n/2$ 阶方阵的加法（可在 $n^2$ 内完成）。因此，上述分治法的计算时间耗费 $T(n)$ 应该满足：

$$\begin{cases} T(n) = 8T(n/2) & n \geq 2 \\ T(1) = 1 \end{cases}$$

这个递归方程的解仍然是 $T(n) = O(n^3)$

# Strassen矩阵乘法

Strassen提出了一种新的算法来计算2个2阶方阵的乘积。他的算法只用了7次乘法运算，但增加了加、减法的运算次数。这7次乘法是：

$$m_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$m_2 = (A_{21} + A_{22}) \times B_{11}$$

$$m_3 = A_{11} \times (B_{12} - B_{22})$$

$$m_4 = A_{22} \times (B_{21} - B_{11})$$

$$m_5 = (A_{11} + A_{12}) \times B_{22}$$

$$m_6 = (A_{21} - A_{11}) \times (B_{11} + B_{10})$$

$$m_7 = (A_{12} - A_{22}) \times (B_{10} + B_{11})$$

# Strassen矩阵乘法

于是可得到：

$$C_{11} = m_1 + m_4 - m_5 + m_7$$

$$C_{12} = m_3 + m_5$$

$$C_{21} = m_2 + m_4$$

$$C_{22} = m_1 + m_3 - m_2 + m_6$$

以上计算的正确性很容易验证。Strassen矩阵乘积分治算法中，用了7次对于 $n/2$ 阶矩阵乘积的递归调用和18次 $n/2$ 阶矩阵的加减运算。由此可知，该算法的所需的计算时间 $T(n)$ 满足如下的递归方程：

# Strassen矩阵乘法

$$\begin{cases} T(n) = 7T(n/2) & n \geq 2 \\ T(1) = 1 \end{cases}$$

其解为  $T(n) \in O(n^{\log_2 7}) \approx O(n^{2.81})$ 。

由此可见，Strassen矩阵乘法的计算时间复杂性比普通矩阵乘法有所改进。

# 分治法解最近点对问题

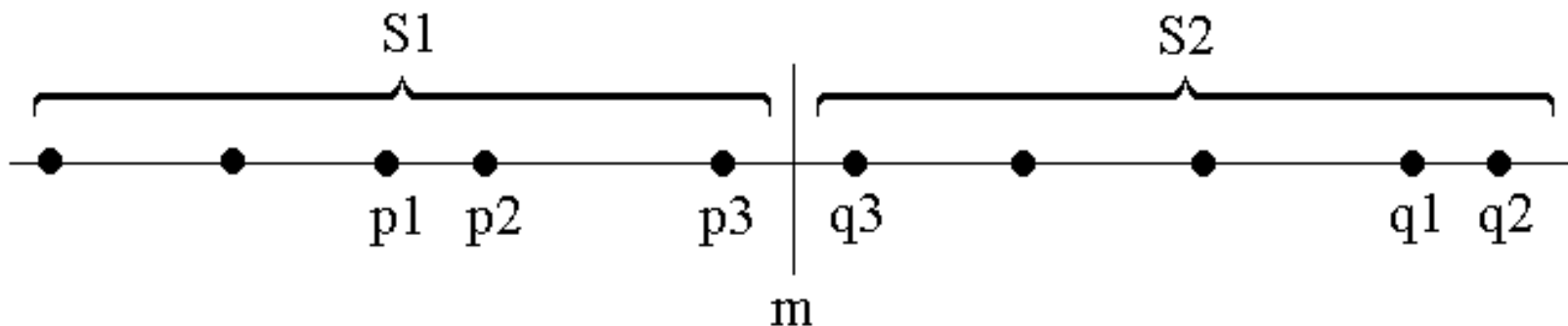
问题：给定平面 $S$ 上 $n$ 个点,找其中的一对点,使得在 $n(n-1)/2$ 个点对中,该点对的距离最小。

算法思路:

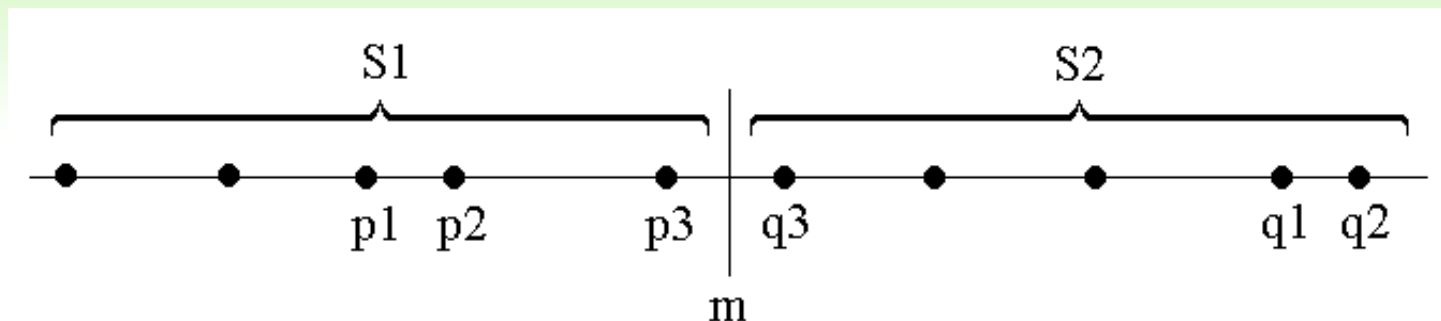
- 1)  $n$ 较小时直接求( $n = 2$ ).
- 2) 将 $S$ 上的 $n$ 个点分成大致相等的2个子集 $S1$ 和 $S2$
- 3) 分别求 $S1$ 和 $S2$ 中的最接近点对
- 4) 求一点在 $S1$ 、另一点在 $S2$ 中的最近点对
- 5) 从上述三对点中找距离最近的一对.

# 最近对问题：共线的情况

- 假设我们用 $x$ 轴上某个点 $m$ 将 $S$ 划分为2个子集 $S_1$ 和 $S_2$ ，基于平衡子问题思想，用 $S$ 中各点坐标的中位数来作分割点。
- 递归地在 $S_1$ 和 $S_2$ 上找出其最接近点对 $\{p_1, p_2\}$ 和 $\{q_1, q_2\}$ ，并设 $d = \min\{|p_1 - p_2|, |q_1 - q_2|\}$ ， $S$ 中的最接近点对或者是 $\{p_1, p_2\}$ ，或者是 $\{q_1, q_2\}$ ，或者是某个 $\{p_3, q_3\}$ ，其中 $p_3 \in S_1$ 且 $q_3 \in S_2$ 。
- 能否在线性时间内找到 $p_3, q_3$ ？



# 最近对问题：共线的情况

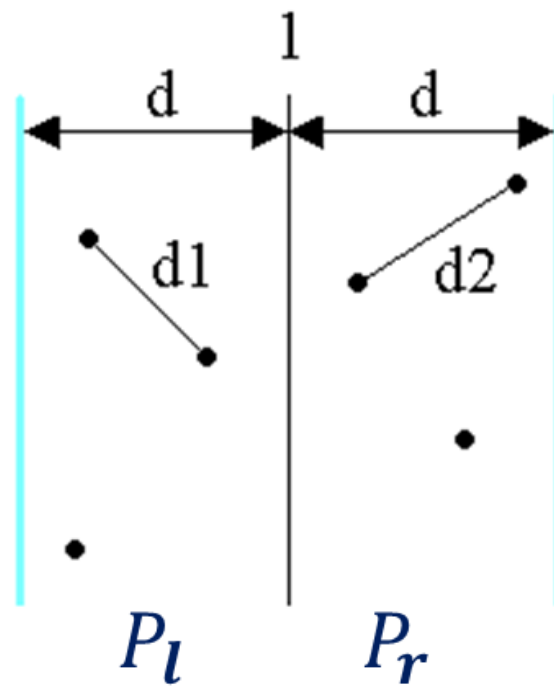


能否在线性时间内找到 $p_3, q_3$ ?

- 如果 $S$ 的最近点对是 $\{p_3, q_3\}$ ，即 $|p_3 - q_3| < d$ ，则 $p_3$ 和 $q_3$ 两者与 $m$ 的距离不超过 $d$ ，即 $p_3 \in (m - d, m]$ ， $q_3 \in (m, m + d]$ 。
- 由于 $S_1$ 中，每个长度为 $d$ 的半闭区间至多包含一个点（否则必有两点距离小于 $d$ ），并且 $m$ 是 $S_1$ 和 $S_2$ 的分割点，因此 $(m - d, m]$ 中至多包含 $S$ 中的一个点。由图可以看出，如果 $(m - d, m]$ 中有 $S$ 中的点，则此点就是 $S_1$ 中值最大的点。
- 因此，用线性时间能找到区间 $(m - d, m]$ 和 $(m, m + d]$ 中所有点，即 $p_3$ 和 $q_3$ 。从而用线性时间可以将 $S_1$ 的解和 $S_2$ 的解合并为 $S$ 的解。

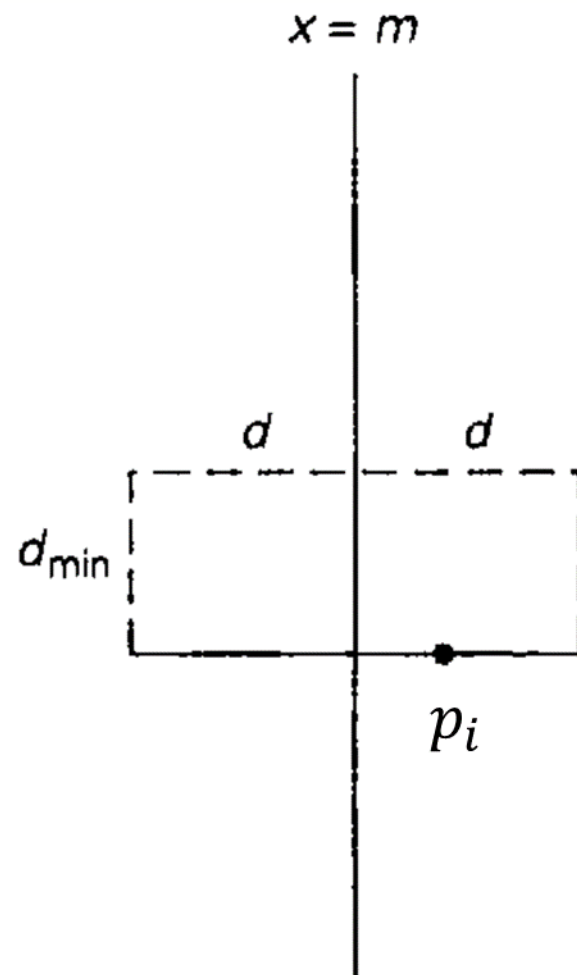
# 最近对问题：二维情形

- 令  $P$  为  $n > 1$  个点构成的集合，且按  $x$  坐标升序排列。 $Q$  是该  $n$  个点按  $y$  坐标升序排列。
- 选取点集在  $x$  轴方向中位数  $m$ ，做垂线  $L: x = m$  作为分割线，将  $P$  分割为  $P_l$  和  $P_r$ 。
- 递归地在  $P_l$  和  $P_r$  上找出其最小距离  $d_1$  和  $d_2$ ，并设  $d = \min\{d_1, d_2\}$ ， $P$  中的最接近点对或者是  $d$ ，或者是某个  $\{p, q\}$ ，其中  $p \in P_l$  且  $q \in P_r$ 。
- 能否在线性时间内找到  $p, q$ ？



# 最近对问题：二维情形

- 考虑 $P_r$ 中任意一点 $p_i$ ，它若与 $P_l$ 中的点 $q$ 构成最接近点对候选，则必有 $distance(p, q) < d$ 。满足该条件的 $P_r$ 中的点一定落在垂线 $L$ 为对称轴、宽度为 $2d$ 的垂直带中。
- 设 $S$ 是按 $y$ 值升序排列的集合 $Q$ 中，位于分割线 $2d$ 宽度范围内的点列表。
- 由 $d$ 的意义可知，任意 $p_i$ ，可能的候选点对 $(p_i, q)$ 中的 $q$ ，只能位于如图所示的 $d \times 2d$ 矩形中，且这样的点最多6个。
- 因此，在分治法合并步骤中最多只需要检查按 $y$ 值升序的后续5个候选点。



# 最近对问题

**算法** EfficientClosestPair( $P, Q$ )

//使用分治算法来求解最近点对问题

//输入：数组  $P$  中存储了平面上的  $n \geq 2$  个点，并且按照这些点的  $x$  轴坐标升序排列

// 数组  $Q$  存储了与  $P$  相同的点，只是它是按照这点的  $y$  轴坐标升序排列

//输出：最近点对之间的欧几里得距离

**if**  $n \leq 3$

    返回由蛮力算法求出的最小距离

**else**

    将  $P$  的前  $\lceil n/2 \rceil$  个点复制到  $P_l$

    将  $Q$  的前  $\lceil n/2 \rceil$  个点复制到  $Q_l$

    将  $P$  中余下的  $\lfloor n/2 \rfloor$  个点复制到  $P_r$

    将  $Q$  中余下的  $\lfloor n/2 \rfloor$  个点复制到  $Q_r$

$d_l \leftarrow \text{EfficientClosestPair}(P_l, Q_l)$

$d_r \leftarrow \text{EfficientClosestPair}(P_r, Q_r)$

$d \leftarrow \min\{d_l, d_r\}$

$m \leftarrow P[\lceil n/2 \rceil - 1].x$

    将  $Q$  中所有  $|x - m| < d$  的点复制到数组  $S[0..num - 1]$

$d_{minsq} \leftarrow d^2$

**for**  $i \leftarrow 0$  **to**  $num - 2$  **do**

$k \leftarrow i + 1$

**while**  $k \leq num - 1$  **and**  $(S[k].y - S[i].y)^2 < d_{minsq}$

$d_{minsq} \leftarrow \min((S[k].x - S[i].x)^2 + (S[k].y - S[i].y)^2, d_{minsq})$

$k \leftarrow k + 1$

**return**  $\text{sqrt}(d_{minsq})$

# 最近对问题

## 复杂度分析

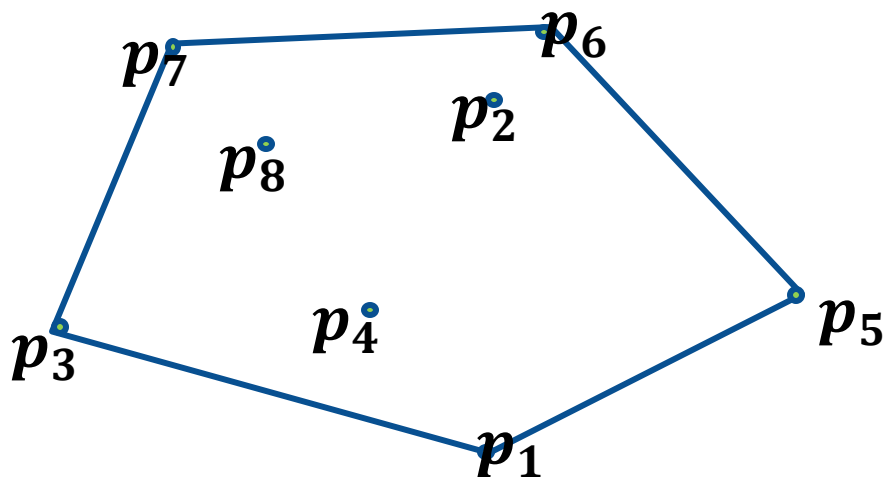
$$T(n) = \begin{cases} O(1) & n < 4 \\ 2T(n/2) + O(n) & n \geq 4 \end{cases}$$

$$\mathbf{T(n) = O(n \log n)}$$

# 分治法解凸包问题

## ► 凸包问题

- 定义：一个点集 $S$ 的凸包(convex hull) 是包含 $S$ 的最小凸集合(“最小”是 $S$ 的凸包一定是所有包含 $S$ 的凸集合的子集)。
- 定理：任意包含 $n > 2$ 个点（不共线）的集合 $S$ 的凸包是以 $S$ 中某些点为顶点的凸多边形。



图中8个点的集合的凸包是以 $P_1, P_5, P_6, P_7$ 和 $P_3$ 为顶点的凸多边形

# 分治法解凸包问题

## ► 凸包问题

- step1: 对于一个 $n$ 个点集合中的两个点 $P_i, P_j$ , 当且仅当该集合中的其他点都位于穿过这两点的直线的同一边时, 它们的连线是该集合凸包边界的一部分。
- step2: 对每一对点都做一遍检查之后, 满足条件的线段构成该凸包的边界。

# 分治法解凸包问题

## ► 凸包问题

1、取集合 $n$ 中的两个点 $i, j$ 点。

2、在坐标平面上穿过两点的直线的构造方程：

$$ax + by = c,$$

其中，

$$a = y_2 - y_1, b = x_2 - x_1, c = x_1 y_2 - y_1 x_2$$

3、 $\geq c$ 或均 $\leq c$ ，说明在直线一侧。

# 分治法解凸包问题

下包当然也可以用同样的方式来构造。如果  $S_1$  为空，上包就是以  $p_1$  和  $p_n$  为端点的线段。如果  $S_1$  不空，该算法找到  $S_1$  中的顶点  $p_{\max}$ ，它是距离直线  $\overline{p_1 p_n}$  最远的点(图 5.9)。如果距离最远的点有多个，就找能使角  $\angle p_{\max} p_1 p_n$  最大的点。(注意，对于以  $p_1$  和  $p_n$  为两个顶点， $S_1$  中的其他点为第三个顶点的三角形， $p_{\max}$  使得这个三角形的面积最大。)然后该算法找出  $S_1$  中所有在直线  $\overline{p_1 p_{\max}}$  左边的点，这些点以及  $p_1$  和  $p_{\max}$ ，构成了集合  $S_{1,1}$ 。 $S_1$  中在直线  $\overline{p_{\max} p_n}$  左边的点以及  $p_{\max}$  和  $p_n$  构成了集合  $S_{1,2}$ 。不难证明：

- $p_{\max}$  是上包的顶点。
- 包含在  $\triangle p_1 p_{\max} p_n$  之中的点不可能是上包的顶点(因此在后面不必考虑)。
- 同时位于  $\overline{p_1 p_{\max}}$  和  $\overline{p_{\max} p_n}$  两条直线左边的点是不存在的。

因此，该算法可以继续递归构造  $p_1 \cup S_{1,1} \cup p_{\max}$  和  $p_{\max} \cup S_{1,2} \cup p_n$  的上包，然后把它们连接起来，以得到整个集合  $p_1 \cup S_1 \cup p_n$  的上包。

# 总结

## ① 算法效率分析基础

## ② 算法设计策略

### 第二讲 蛮力法

### 第三讲 分治法

- 分治法的定义

分而治之思想，分→治→和

- 主定理

$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$  通用分治递推式

- 合并排序

递推式:  $T(n) = 2T(n/2) + O(n)$

- 大整数乘法

递推式:  $T(n) = 3T(n/2) + O(n)$

- 第四讲：减治法（增量法）

## ③ 算法能力的极限