



# 分治算法

东南大学计算机学院 方效林

## 本章内容

- 分治算法的原理
- 大整数乘法
- 矩阵乘法
- 求第k小元素问题
- 寻找最近点对
- 快速傅立叶变换
- 寻找凸包

# 分治算法的原理

## ■ 设计过程

- **Divide:** 整个问题划分为多个子问题
- **Conquer:** 求解各子问题(递归调用子问题的算法)
- **Combine:** 合并子问题的解, 形成原始问题的解

# 分治算法的原理

## ■ 分析过程

### □ 建立递归方程

➤  $T(n) = aT(n/b) + D(n) + C(n)$

□ Divide时间复杂度:  $D(n)$

□ Conquer时间复杂度:  $aT(n/b)$

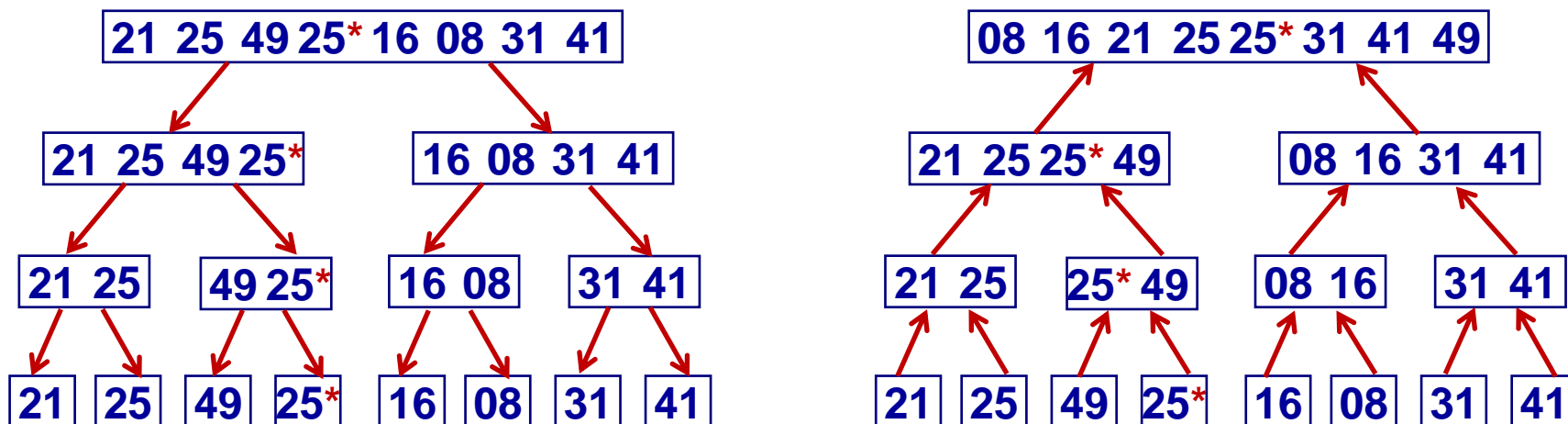
□ Combine:  $C(n)$

➤ 当  $n < c$ ,  $T(n) = \theta(1)$

### □ 递归方程求解

# 分治算法的原理

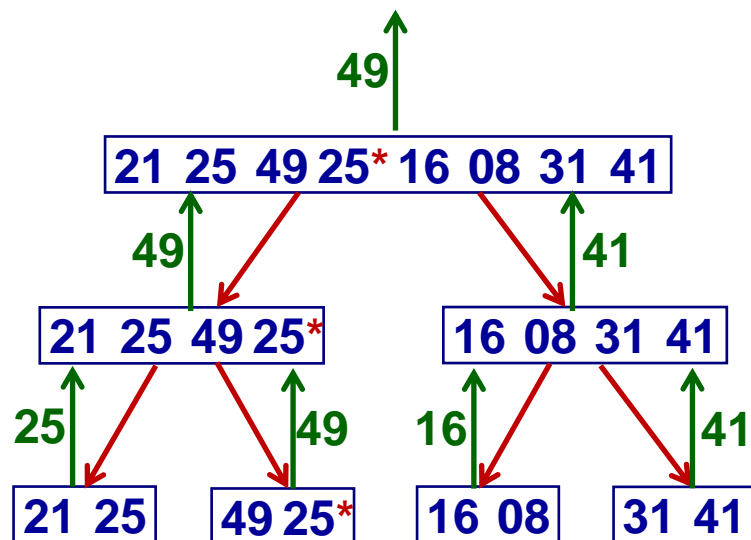
## ■ 例1：归并排序



$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + O(n) \\ &= O(n \log n) \end{aligned}$$

# 分治算法的原理

## ■ 例2：找最大值



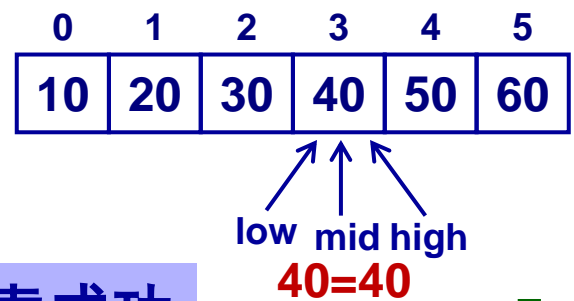
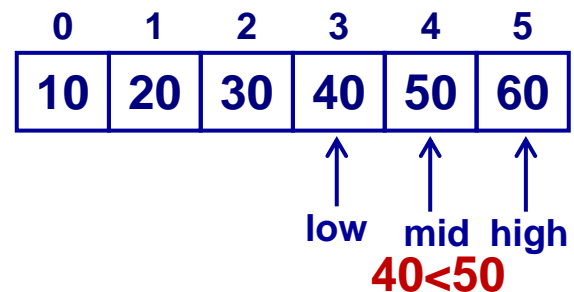
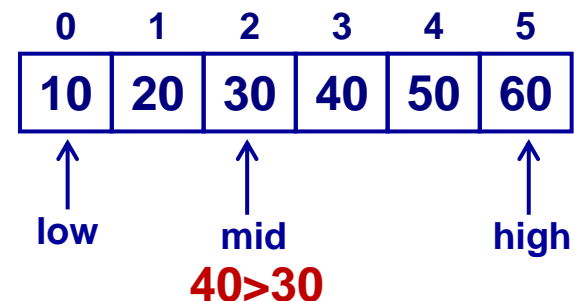
$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + 1 \\ &= n - 1 \end{aligned}$$

# 折半搜索

## 有序顺序表

- $low=0, high=n-1, mid=(low+high)/2$
- **x先和mid元素比较**
  - 若相等，搜索成功，返回下标
  - 若x更小，继续在前半部分搜索
    - $high=mid-1, mid=(low+high)/2$
  - 若x更大，继续在后半部分搜索
    - $low=mid+1, mid=(low+high)/2$

## 搜索40

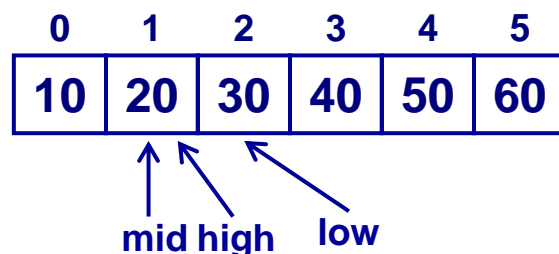


# 搜索成功

# 折半搜索

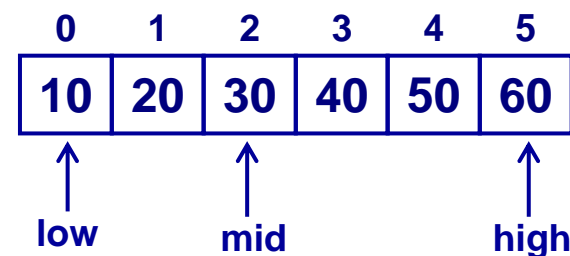
## ■ 有序顺序表

- $low=0, high=n-1, mid=(low+high)/2$
- **x先和mid元素比较**
  - 若相等，搜索成功，返回下标
  - 若x更小，继续在前半部分搜索
    - $high=mid-1, mid=(low+high)/2$
  - 若x更大，继续在后半部分搜索
    - $low=mid+1, mid=(low+high)/2$

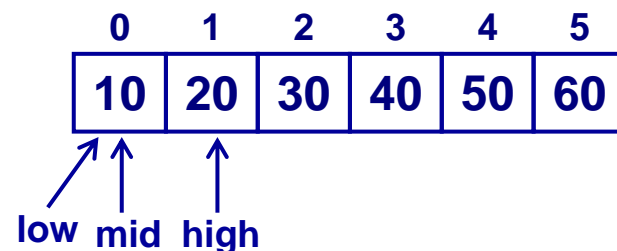


**low>high, 搜索失败**

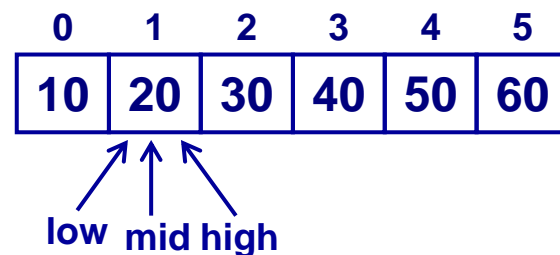
搜索25



**25<30**



**25>10**



**25>20**

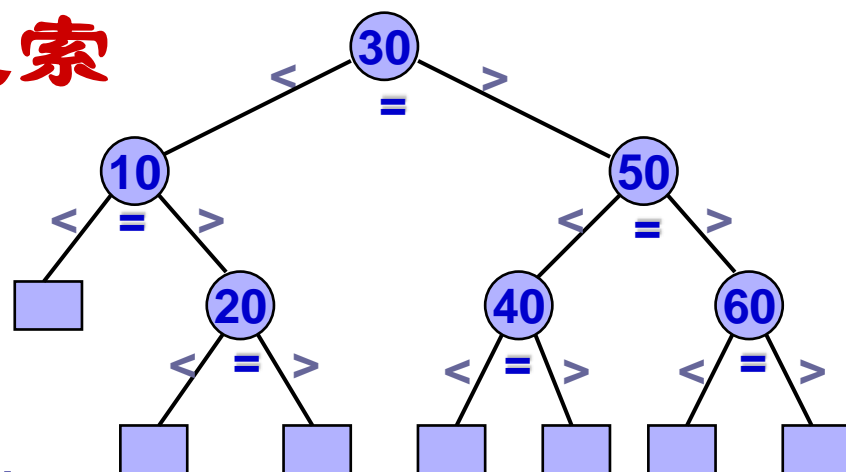


# 折半搜索

## ■ 有序顺序表

### □ 折半搜索构造的判定树

- 设满二叉树  $n=2^h-1$
- 则有  $2^h=n+1$ ,  $h=\log_2(n+1)$
- 平均搜索长度



$$ASL_{succ} = \frac{1}{n} (1 * 2^0 + 2 * 2^1 + 3 * 2^2 + \dots + (h-1) * 2^{h-2} + h * 2^{h-1})$$

$$= \frac{1}{n} ((h-1) \times 2^h + 1)$$

$$= \frac{n+1}{n} \log_2(n+1) - 1 \approx \log_2(n+1) - 1$$

错位相减法

# 大整数乘法

- $n$ 位二进制整数 $X$ 和 $Y$ 相乘
  - 通常算法时间复杂性为  $O(n^2)$
  - 分治算法时间复杂度可降至  $O(n^{1.59})$

# 大整数乘法

$$X = \overset{n/2\text{位}}{\boxed{A}} \overset{n/2\text{位}}{\boxed{B}}$$

$$Y = \overset{n/2\text{位}}{\boxed{C}} \overset{n/2\text{位}}{\boxed{D}}$$

$$XY = (A2^{n/2} + B)(C2^{n/2} + D)$$

← 乘以 $2^n$ 表示左移 $n$ 位

$$= AC2^n + (AD+BC)2^{n/2} + BD$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\because a = 4, b = 2, f(n) = \Theta(n), n^{\log_2 4} = n^2$$

$$f(n) = n^{\log_2 4 - \epsilon}, \epsilon = 1,$$

$$\therefore T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$$

复杂性没有改善

# 大整数乘法

$$X = \overset{n/2\text{位}}{\boxed{A}} \overset{n/2\text{位}}{\boxed{B}}$$

$$Y = \overset{n/2\text{位}}{\boxed{C}} \overset{n/2\text{位}}{\boxed{D}}$$

$$\begin{aligned} XY &= (A2^{n/2} + B)(C2^{n/2} + D) \quad \leftarrow \text{乘以} 2^n \text{表示左移} n \text{位} \\ &= AC2^n + ((A+B)(C+D) - AC - BD)2^{n/2} + BD \end{aligned}$$

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\because a = 3, b = 2, f(n) = \Theta(n), n^{\log_2 3} \approx n^{1.59}$$

$$f(n) = n^{\log_2 3 - \varepsilon}, \varepsilon \approx 0.59,$$

$$\therefore T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59})$$

复杂性降低了

# 矩阵乘法

- 两个  $n \times n$  的矩阵  $A$  和  $B$  相乘
  - 通常算法时间复杂性为  $O(n^3)$
  - 分治算法时间复杂度可降至  $O(n^{2.81})$

# 矩阵乘法

- 两个  $n \times n$  的矩阵  $A$  和  $B$  相乘

- 将  $A$  和  $B$  分成 4 个大小为  $\frac{n}{2} \times \frac{n}{2}$  的子矩阵运算

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

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

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

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$\because a = 8, b = 2, f(n) = \Theta(n^2), n^{\log_2 8} = n^3$$

$$f(n) = n^{\log_2 8 - \varepsilon}, \varepsilon = 1,$$

$$\therefore T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$

复杂性没有改善

# 矩阵乘法

## ■ 两个 $n \times n$ 的矩阵 $A$ 和 $B$ 相乘

□ 将  $A$  和  $B$  分成 4 个大小为  $\frac{n}{2} \times \frac{n}{2}$  的子矩阵运算

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

将 8 个矩阵相乘转变成 7 个矩阵相乘

$$M_1 = A_{11} (B_{12} - B_{22})$$

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

$$M_3 = (A_{21} + A_{22}) B_{11}$$

$$M_4 = A_{22} (B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{22}) (B_{11} + B_{22})$$

$$M_6 = (A_{12} - A_{22}) (B_{21} + B_{22})$$

$$M_7 = (A_{11} - A_{21}) (B_{11} + B_{12})$$



$$C_{11} = M_5 + M_4 - M_2 + M_6$$

$$C_{12} = M_1 + M_2$$

$$C_{21} = M_3 + M_4$$

$$C_{22} = M_5 + M_1 - M_3 - M_7$$

# 矩阵乘法

## ■ 两个 $n \times n$ 的矩阵 $A$ 和 $B$ 相乘

□ 将  $A$  和  $B$  分成 4 个大小为  $\frac{n}{2} \times \frac{n}{2}$  的子矩阵运算

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

将 8 个矩阵相乘转变成 7 个矩阵相乘

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$\because a = 7, b = 2, f(n) = \Theta(n^2), n^{\log_2 7} \approx n^{2.81}$$

$$f(n) = n^{\log_2 7 - \varepsilon}, \varepsilon \approx 0.81,$$

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

$$C_{11} = M_5 + M_4 - M_2 + M_6$$

$$C_{12} = M_1 + M_2$$

$$C_{21} = M_3 + M_4$$

$$C_{22} = M_5 + M_1 - M_3 - M_7$$

复杂性降低



# 矩阵乘法

- 两个  $n \times n$  的矩阵  $A$  和  $B$  相乘
  - 还有更好的算法，复杂度可降至  $\Theta(n^{2.376})$
  - 可否继续降低复杂度？
  - 还没有定论

# 快速排序

## ■ 基本思想

- Partition: 任取一元素x为基准(如选第1个), 小于x的元素放在x左边, 大于等于x的元素放在x右边
- 对左、右部分递归执行上一步骤直至只有一个元素

初始	21	25	49	25*	16	08
第1层	08	16	21	25*	25	49
第2层	08	16	21	25*	25	49
第3层	08	16	21	25*	25	49
第4层	08	16	21	25*	25	49

选21为基准

左部选08, 右部选25\*为基准

左部选16, 右部选25为基准

右部选49为基准

# 快速排序

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \\ = O(n \log n)$$

## ■ Partition(low,high)

- 初始时基准坐标  $p = \text{low}$ ,  $x = a[\text{low}] = 21$
- 从  $i = \text{low} + 1$  位置开始判断, 比  $x$  小的元素与  $p$  下一个位置交换,  $p$  自加1
- 循环直至  $i > \text{high}$ , 最后  $a[\text{low}]$  与  $a[p]$  交换

21	25	49	25*	16	08
----	----	----	-----	----	----

↑  
 $p$   
 $a[i]$ 与 $a[p+1]$ 交换,  $p++$

21	16	49	25*	25	08
----	----	----	-----	----	----

↑  
 $p$   
 $a[i]$ 与 $a[p+1]$ 交换,  $p++$

21	16	08	25*	25	49
----	----	----	-----	----	----

↑  
 $p$   
 $a[\text{low}]$ 与 $a[p]$ 交换

$i > \text{high}$ , 停止

08	16	21	25*	25	49
----	----	----	-----	----	----

↑  
 $p$

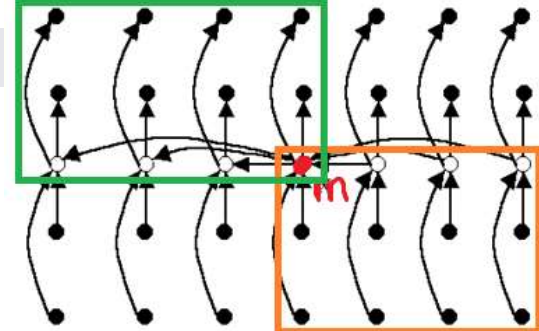
## 快速排序

- 基于比较的排序方法，最好的算法不会好于  $O(n \log n)$ ,
  - 因为比较搜索树中  $2^h \geq n!$ ，从而  $h \geq n \log n$

## 第 $k$ 小元素

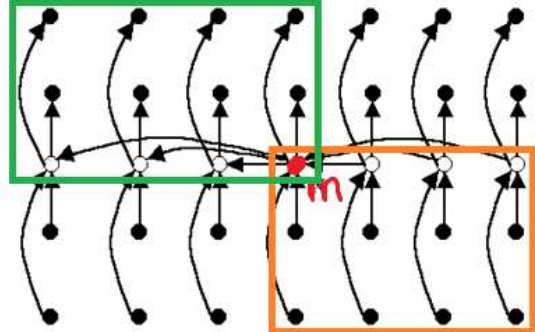
- 在有  $n$  个元素的数组中，找第  $k$  小的元素
  - 可采用堆排序方法找第  $k$  小元素
  - 每获得一个元素，最坏情况下要  $O(\log n)$  次比较
  - 故此方法的时间复杂度在最坏情况下是  $O(k \log n)$
  - 当  $k$  接近  $\frac{n}{2}$  时，此方法需要  $O(n \log n)$  次比较。

## 第 $k$ 小元素



- 在有  $n$  个元素的数组  $S$  中，找第  $k$  小的元素
  - 给出一复杂度为  $O(n)$  的算法  $\text{Select}(S, k)$ 
    - 若  $|S| < 50$ ，采用堆排序的方法找出第  $k$  小元素
    - 否则，
      - 将  $n$  个元素分为  $\lceil n/5 \rceil$  组，每组 5 个元素，每组排序， $O(n)$
      - 将每组第 3 个元素取出，得到大小为  $\lceil n/5 \rceil$  的数组  $M$
      - 最后一组中的元素可能不足 5 个
        - 1 个取出；2 个取较小；3 个取中间；4 个取第 2 小
      - $m = \text{Select}(M, \lceil |M|/2 \rceil)$ ，代价  $T(\lceil n/5 \rceil)$
      - $S$  中至少有  $3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$  个元素大于等于  $m$
      - 类似地， $S$  中至少有  $\frac{3n}{10} - 6$  个元素小于  $m$

## 第 $k$ 小元素



- 在有  $n$  个元素的数组  $S$  中，找第  $k$  小的元素

- 给出一复杂度为  $O(n)$  的算法 **Select( $S, k$ )**

- 若  $|S| < 50$ ，采用堆排序的方法找出第  $k$  小元素

- 否则，

- 依次扫描整个数组  $S$ ，代价  $O(n)$

- $s_i < m$  时放入  $S_1$ ； $s_i = m$  时放入  $S_2$ ； $s_i > m$  时放入  $S_3$ ；

- 当  $k \leq |S_1|$  时，调用 **Select( $S_1, k$ )**,

- $S_1$  大小最多为  $n - (\frac{3n}{10} - 6) = \frac{7n}{10} + 6$ ，代价  $T(\frac{7n}{10} + 6)$

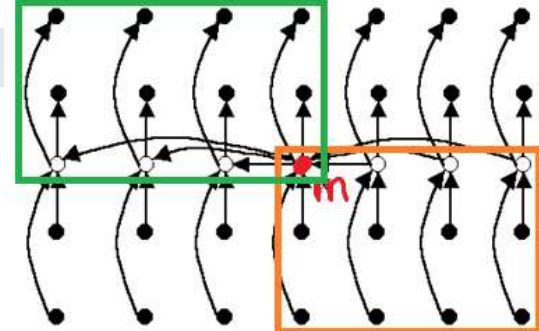
- 当  $|S_1| < k \leq |S_1| + |S_2|$  时，

- 返回  $m$

- 当  $k > |S_1| + |S_2|$  时，调用 **Select( $S_3, k - |S_1| - |S_2|$ )**

- $S_3$  大小最多为  $n - (\frac{3n}{10} - 6) = \frac{7n}{10} + 6$ ，代价  $T(\frac{7n}{10} + 6)$

## 第 $k$ 小元素



- 在有  $n$  个元素的数组  $S$  中，找第  $k$  小的元素
  - 给出一复杂度为  $O(n)$  的算法  $\text{Select}(S, k)$

➤  $T(n) = T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n) \leq O(n)$

- 第二数学归纳法证明：  $n < 50$  时显然成立

□ 假设  $n \leq m$  时，  $T(m) = O(m)$  ,  $T(m) \leq cm$

□  $n = m + 1$  时,  $T(m + 1) = T\left(\left\lceil \frac{m+1}{5} \right\rceil\right) + T\left(\frac{7(m+1)}{10} + 6\right) + O(m + 1)$

□  $\leq c_1 \left\lceil \frac{m+1}{5} \right\rceil + c_2 \left(\frac{7(m+1)}{10} + 6\right) + O(m + 1)$

□  $\leq c \left(\frac{m+1}{5} + 1\right) + c \left(\frac{7(m+1)}{10} + 6\right) + a(m + 1)$

□  $= \left(\frac{9c}{10} + a\right)(m + 1) + 7c$

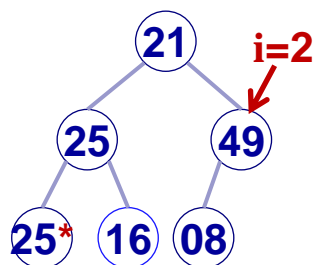
□  $\leq O(m + 1)$



# 堆排序

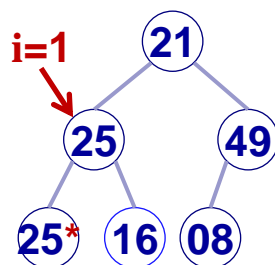
## ■ 算法思想

- 建立最大堆
- 循环执行以下步骤，直至所有元素出堆
  - 每次堆顶元素(即最大元素)与堆中最后一个元素交换
  - 剔除最大元素后调整为最大堆



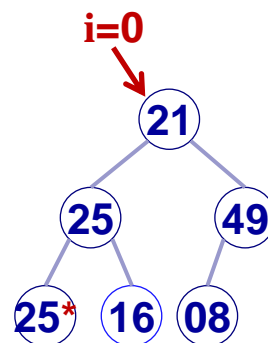
21	25	49	25*	16	08
----	----	----	-----	----	----

最后一元素的父节点  
i=2开始调整



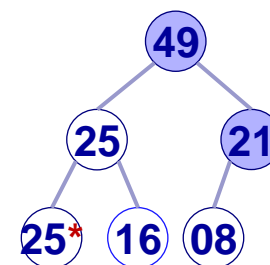
21	25	49	25*	16	08
----	----	----	-----	----	----

调整i=1



21	25	49	25*	16	08
----	----	----	-----	----	----

调整i=0



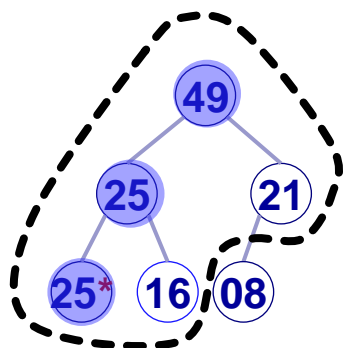
49	25	21	25*	16	08
----	----	----	-----	----	----

形成最大堆

# 堆排序

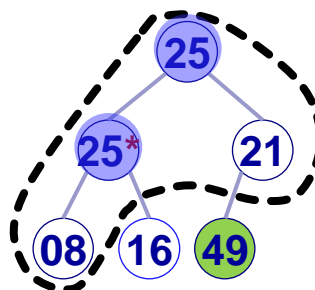
## ■ 算法思想

- 建立最大堆
- 循环执行以下步骤，直至所有元素出堆
  - 每次堆顶元素(即最大元素)与堆中最后一个元素交换
  - 剔除最大元素后调整为最大堆



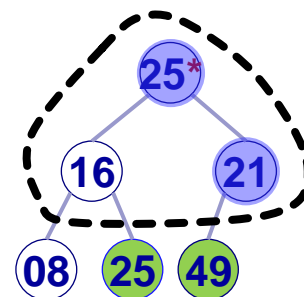
49 25 21 25\* 16 08

堆顶49与堆尾08交换  
虚线内调整为最大堆



25 25\* 21 08 16 49

堆顶25与堆尾16交换  
虚线内调整为最大堆



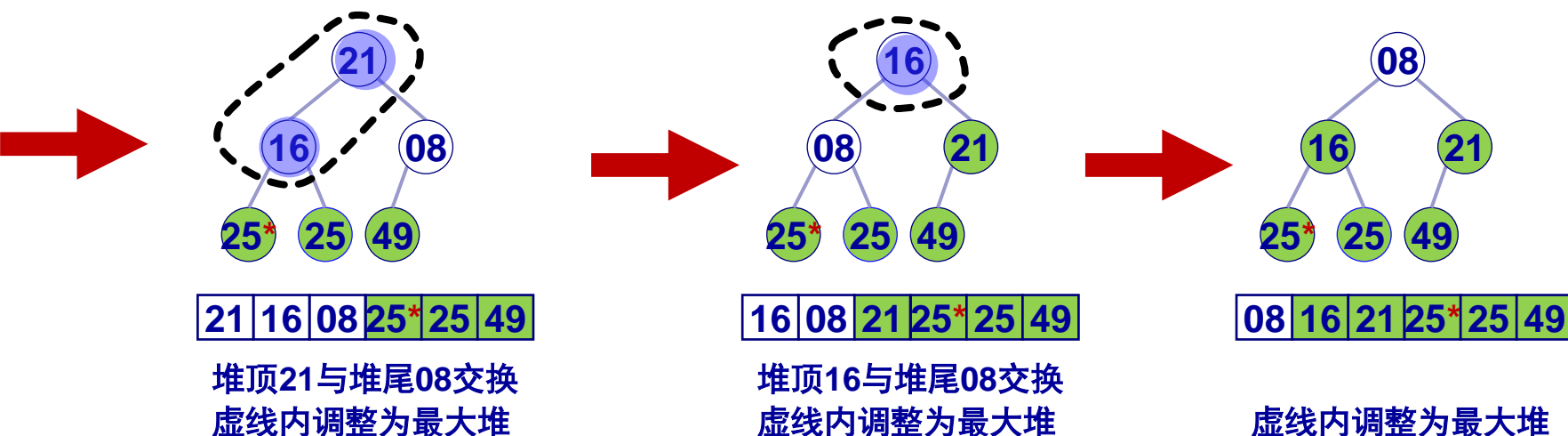
25\* 16 21 08 25 49

堆顶25\*与堆尾08交换  
虚线内调整为最大堆

# 堆排序

## ■ 算法思想

- 建立最大堆
- 循环执行以下步骤，直至所有元素出堆
  - 每次堆顶元素(即最大元素)与堆中最后一个元素交换
  - 剔除最大元素后调整为最大堆



# 堆排序

## ■ 堆排序算法分析

### □ 建立最大堆

- 设堆中有 $n$ 个元素, 对应完全二叉树有 $h$ 层( $2^{h-1} \leq n < 2^h$ )
- 第 $i$ 层向下调整移动距离最大为 $(h-i)$ , 第 $i$ 层节点数为 $2^{i-1}$
- 总移动次数

$$2 \cdot \sum_{i=1}^{h-1} 2^{i-1}(h-i) = O(n)$$

### □ 循环弹出堆顶元素

- 执行 $n-1$ 次向下调整, 每次调整距离 $\lceil \log_2(n+1) \rceil$
- 总调整时间 $O(n \log_2 n)$

## 最近点对

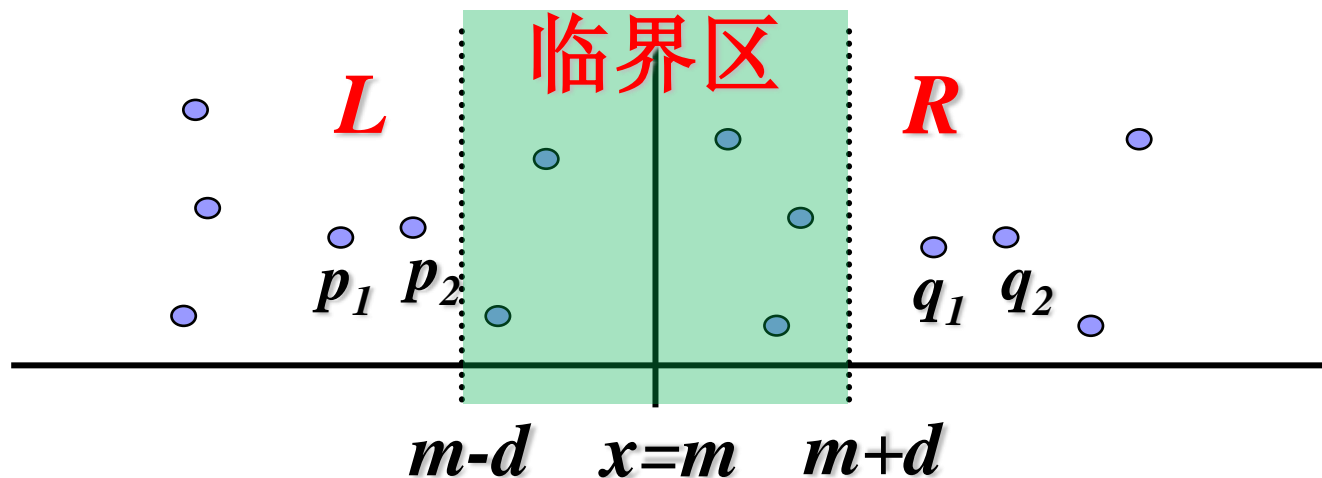
- 在二维平面上n个点中找距离最近的两个点
  - 输入  $Q = \{p_0, p_0, \dots, p_0\}$
  - 输出距离最近的两个点  $(p_r, p_s)$ 
    - $Dis(p_i, p_j)$  为Euclidean距离,
    - 令  $p_i = (x_i, y_i), p_j = (x_j, y_j)$
    - $Dis(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

# 最近点对

- 在二维平面上 $n$ 个点中找距离最近的两个点
  - Preprocessing
    - 如果 $Q$ 中仅包含一个点，则算法结束
    - 把 $Q$ 中的点按  $x$ -坐标值和  $y$ -坐标值排序
  - Divide:
    - 计算 $Q$ 中各点 $x$ -坐标的中位数 $m$
    - 用垂线  $x=m$  把 $P$ 划分成两个大小相等的子集合 $L$ 和 $R$ ,  
 $L$ 中点在  $x=m$ 左边,  $R$  中点在 $x=m$  右边

## 最近点对

- 在二维平面上 $n$ 个点中找距离最近的两个点

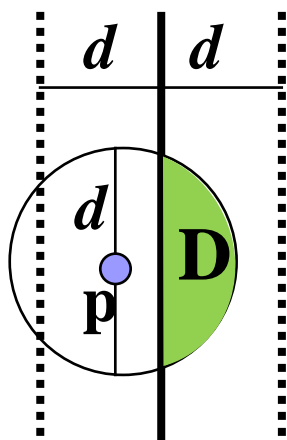


1. 递归地在 $L$ 、 $R$ 中找出最接近点对:  $(p_1, p_2) \in L$ ,  $(q_1, q_2) \in R$
2.  $d = \min\{\text{Dis}(p_1, p_2), \text{Dis}(q_1, q_2)\}$ ;
1. 在临界区查找距离小于 $d$ 的点对  $(p_l, q_r)$ ,  $p_l \in L$ ,  $q_r \in R$ ;
2. 如果找到, 则  $(p_l, q_r)$  是 $Q$ 中最接近点对,
3. 否则 $(p_1, p_2)$ 和 $(q_1, q_2)$  中距离最小者为 $Q$ 中最接近点对.

如何查找  $(p_l, q_r)$

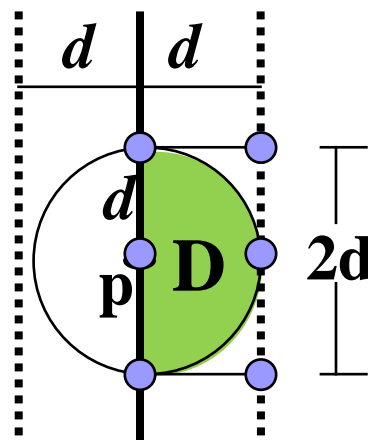
# 最近点对

- 在二维平面上 $n$ 个点中找距离最近的两个点
  - 在临界区查找距离小于 $d$ 的点对  $(p_l, q_r)$ ,  $p_l \in L$ ,  $q_r \in R$



中位线

对于左临界区中任一点  $p$ ,  
在右临界区中找一点  $q$ ,  
若  $\text{Dis}(p, q) < d$ , 则必然有  
 $q$  落于绿色区域  $D$  内



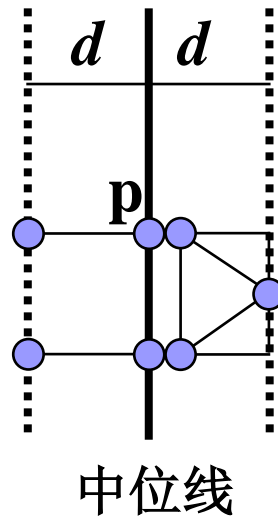
中位线

极端情况下,  $p$  在中位线上,  $q$  在绿色半圆  $D$  中。  
不妨把  $D$  扩大为  $d \times 2d$  的矩形区域, 这样的区域内最多可能存在多少个点, 使每两点距离  $\geq d$ ?  
答案是6个 (鸽巢原理), 即对于任一点  $p$ , 最多只需计算6次距离就可以了



# 最近点对

- 在二维平面上 $n$ 个点中找距离最近的两个点
  - 在临界区查找距离小于 $d$ 的点对  $(p_l, q_r)$ ,  $p_l \in L$ ,  $q_r \in R$



# 最近点对

对Q按x轴从小到大排序  $O(n\log n)$

CPair(i, j, Q) //返回数组Q中i到j的最短距离d以及最近两点p,q, 并将i到j按y轴排序

$n = j - i + 1$

**if**  $n < 3$  **then return** 最短距离d,及其点对(p,q);

$x_{\text{中位线}} = Q_x[(i+j)/2]$

$(d1, pr, ps) = \text{CPair}(i, i + \lceil n/2 \rceil - 1, Q);$   $T(n/2)$

$(d2, pr', ps') = \text{CPair}(i + \lceil n/2 \rceil, j, Q);$   $T(n/2)$

$(d, pr, ps)$ 记录d1和d2更小值

归并 $Q[i, i + \lceil n/2 \rceil - 1]$ 和 $Q[i + \lceil n/2 \rceil, j]$ 按y轴排序  $O(n)$

$Yd =$  落于 $[x_{\text{中位线}} - d, x_{\text{中位线}} + d]$ 带中按y轴从小到大排好的序; //不需重新排序

对于 $Yd$ 任一点, 与后6个比较找最小的 $(d', p, q)$

**if**  $d' < d$  **return**  $(d', p, q)$

**else return**  $(d, pr, ps)$

# 最近点对

- 在二维平面上n个点中找距离最近的两个点
  - 时间复杂性

$$T(n) = O(1) \quad n = 2$$

$$T(n) = 2T(n/2) + O(n) \quad n \geq 3$$

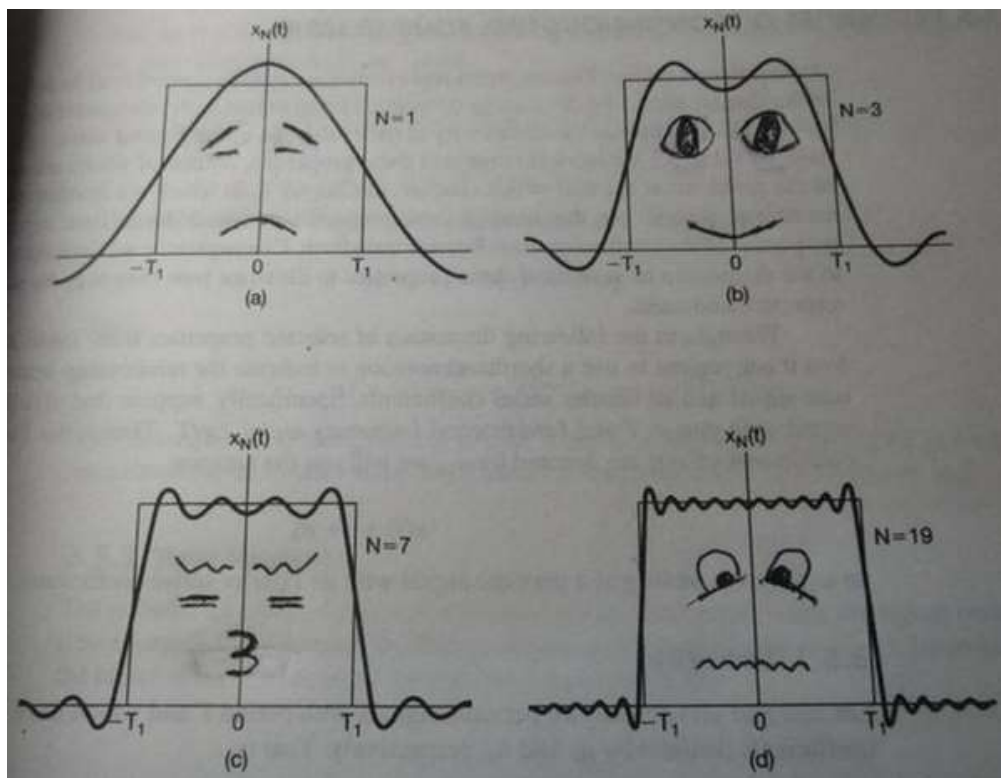
用Master定理求解

$$T(n) = O(n \log n)$$

# 快速傅立叶变换

## ■ 傅立叶变换直观概念

- 任何波形可由多个正弦波叠加近似



- 图1: 1个正弦波  $\cos$
- 图2: 2 正弦波的叠加
- 图3: 4 正弦波的叠加
- 图4: 10个正弦波的叠加

# 快速傅立叶变换

## ■ 傅立叶变换直观概念

- 任何波形可由多个正弦波叠加近似



1个正弦波



2个正弦波的叠加



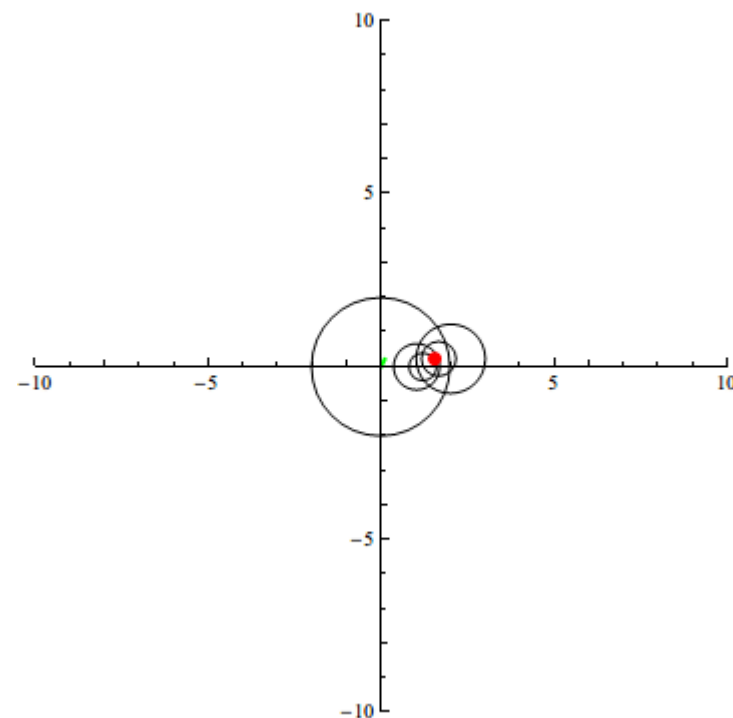
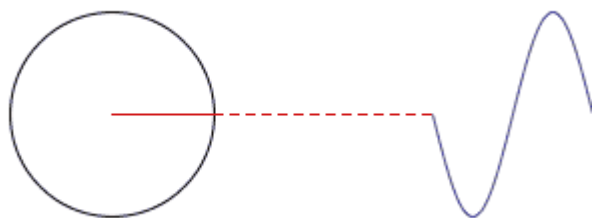
3个正弦波的叠加



4个正弦波的叠加

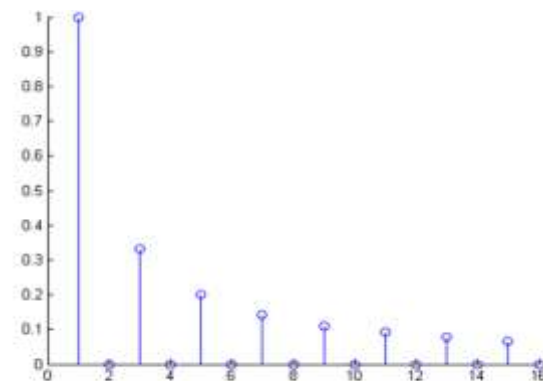
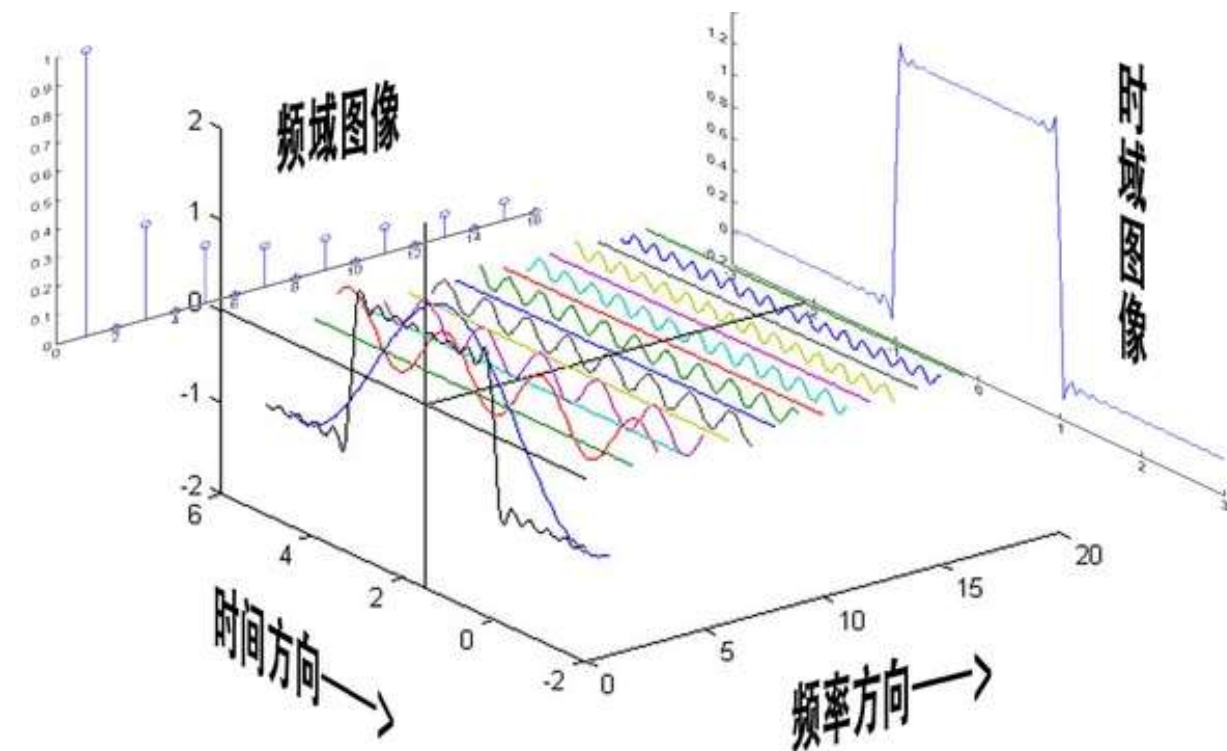
# 快速傅立叶变换

- 傅立叶变换直观概念
  - 任何波形可由多个正弦波叠加近似



# 快速傅立叶变换

- 傅立叶变换直观概念
  - 任何波形可由多个正弦波叠加近似



# 快速傅立叶变换

- 傅立叶变换直观概念
  - 任何波形可由多个正弦波叠加近似
    - 输入为长度为  $n$  的离散信号序列 ( $n$ 一般为 $2^k$ )
    - 输出为一系列频率上的振幅和相位



# 快速傅立叶变换

## ■ 离散傅立叶变换

□ 令  $\omega_n = e^{2\pi i/n}$ , 称为  $n$  阶复根

□  $\omega_n^n = (e^{2\pi i/n})^n = e^{2\pi i} = \cos 2\pi + i \sin 2\pi = 1$

□  $y_k = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$

□  $= a_0 + a_1 \omega_n^k + a_2 \omega_n^{2k} + \cdots + a_{n-1} \omega_n^{(n-1)k}$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

# 快速傅立叶变换

## ■ 离散傅立叶变换

$$\square \omega_n^{2k} = e^{2\pi i \cdot 2k/n} = e^{2\pi i \cdot k/(\frac{n}{2})} = \omega_{n/2}^k$$

$$\triangleright \text{例如: } \omega_8^2 = \omega_4^1$$

$$\square y_k = a_0 + a_1 \omega_n^k + a_2 \omega_n^{2k} + \cdots + a_{n-1} \omega_n^{(n-1)k}$$

$$\triangleright y_k^{\text{偶}} = a_0 + a_2 \omega_n^{2k} + a_4 \omega_n^{4k} + \cdots + a_{n-2} \omega_n^{(n-2)k}$$

$$\triangleright = a_0 + a_2 \omega_{n/2}^k + a_4 \omega_{n/2}^{2k} + \cdots + a_{n-2} \omega_{n/2}^{(n/2-1)k}$$

$$\triangleright y_k^{\text{奇}} = a_1 \omega_n^k + a_3 \omega_n^{3k} + a_5 \omega_n^{5k} + \cdots + a_{n-1} \omega_n^{(n-1)k}$$

$$\triangleright = \omega_n^k (a_1 + a_3 \omega_{n/2}^k + a_5 \omega_{n/2}^{2k} + \cdots + a_{n-1} \omega_{n/2}^{(n/2-1)k})$$

# 快速傅立叶变换

## ■ 离散傅立叶变换

使用性质:  $\omega_n^{2k} = \omega_{n/2}^k$

$$\begin{aligned} y_1 &= a_0 + a_1\omega_8^1 + a_2\omega_8^2 + a_3\omega_8^3 + a_4\omega_8^4 + a_5\omega_8^5 + a_6\omega_8^6 + a_7\omega_8^7 \\ &= a_0 + a_2\omega_8^2 + a_4\omega_8^4 + a_6\omega_8^6 + \omega_8^1(a_1 + a_3\omega_8^2 + a_5\omega_8^4 + a_7\omega_8^6) \\ &= a_0 + a_2\omega_4^1 + a_4\omega_4^2 + a_6\omega_4^3 + \omega_8^1(a_1 + a_3\omega_4^1 + a_5\omega_4^2 + a_7\omega_4^3) \\ &= a_0 + a_4\omega_4^2 + \omega_4^1(a_2 + a_6\omega_4^2) + \omega_8^1(a_1 + a_5\omega_4^2 + \omega_4^1(a_3 + a_7\omega_4^2)) \\ &= a_0 + a_4\omega_2^1 + \omega_4^1(a_2 + a_6\omega_2^1) + \omega_8^1(a_1 + a_5\omega_2^1 + \omega_4^1(a_3 + a_7\omega_2^1)) \end{aligned}$$

# 快速傅立叶变换

## ■ 离散傅立叶变换

使用性质:  $\omega_n^{2k} = \omega_{n/2}^k$   
 $\omega_n^{n/2} = -1$

$$\begin{aligned} y_5 &= a_0 + a_1\omega_8^5 + a_2\omega_8^{10} + a_3\omega_8^{15} + a_4\omega_8^{20} + a_5\omega_8^{25} + a_6\omega_8^{30} + a_7\omega_8^{35} \\ &= a_0 + a_2\omega_8^{10} + a_4\omega_8^{20} + a_6\omega_8^{30} + \omega_8^5(a_1 + a_3\omega_8^2 + a_5\omega_8^4 + a_7\omega_8^6) \\ &= a_0 + a_2\omega_4^1 + a_4\omega_4^2 + a_6\omega_4^3 - \omega_8^1(a_1 + a_3\omega_4^1 + a_5\omega_4^2 + a_7\omega_4^3) \\ &= a_0 + a_4\omega_4^2 + \omega_4^1(a_2 + a_6\omega_4^2) - \omega_8^1(a_1 + a_5\omega_4^2 + \omega_4^1(a_3 + a_7\omega_4^2)) \\ &= a_0 + a_4\omega_2^1 + \omega_4^1(a_2 + a_6\omega_2^1) - \omega_8^1(a_1 + a_5\omega_2^1 + \omega_4^1(a_3 + a_7\omega_2^1)) \end{aligned}$$

# 快速傅立叶变换

## RECURSIVE-FFT(a)

```
1  n = a.length
2  if n == 1
3      return a
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
T(n/2) 8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
T(n/2) 9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
O(n) 10 for k=0 to n/2-1
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+n/2} = y_k^{[0]} - \omega y_k^{[1]}$  //利用了性质:  $\omega_n^{n/2} = \omega_2 = -1$ 
13      $\omega = \omega \omega_n$ 
14 Return y
```

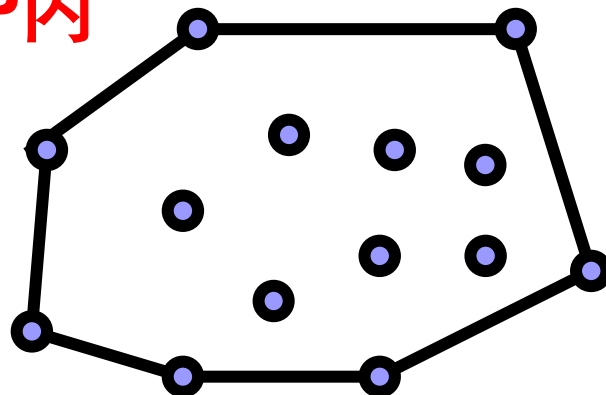
$$T(n) = 2T(n/2) + O(n)$$
$$T(n) = O(n \log n)$$

## 寻找凸包

- 给定平面上 $n$ 个点的集合 $Q$ ，求 $Q$ 的凸包

$Q$ 的convex hull是一个凸多边形 $P$ ，  
 $Q$ 的点或者在 $P$ 上或者在 $P$ 内

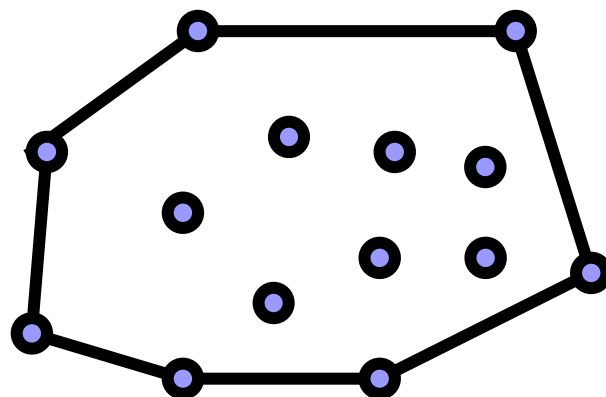
凸多边形 $P$ 是具有如下性质多边形：  
连接 $P$ 内任意两点的边都在 $P$ 内



# 寻找凸包

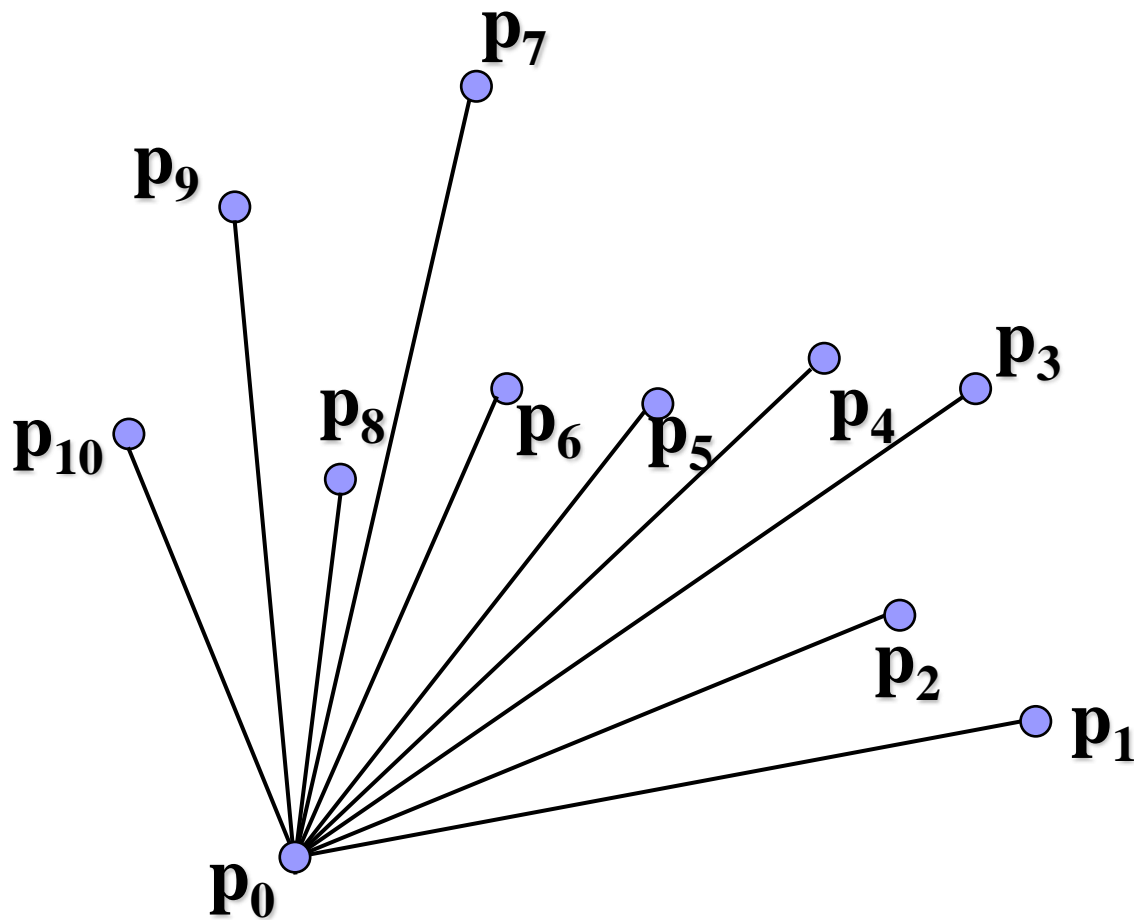
## ■ Graham-scan的基本思想

- 找到最下最左顶点，其他顶点与它连线
- 按夹角从小到大排序
- 夹角最小的开始，寻找凸包点



# 寻找凸包

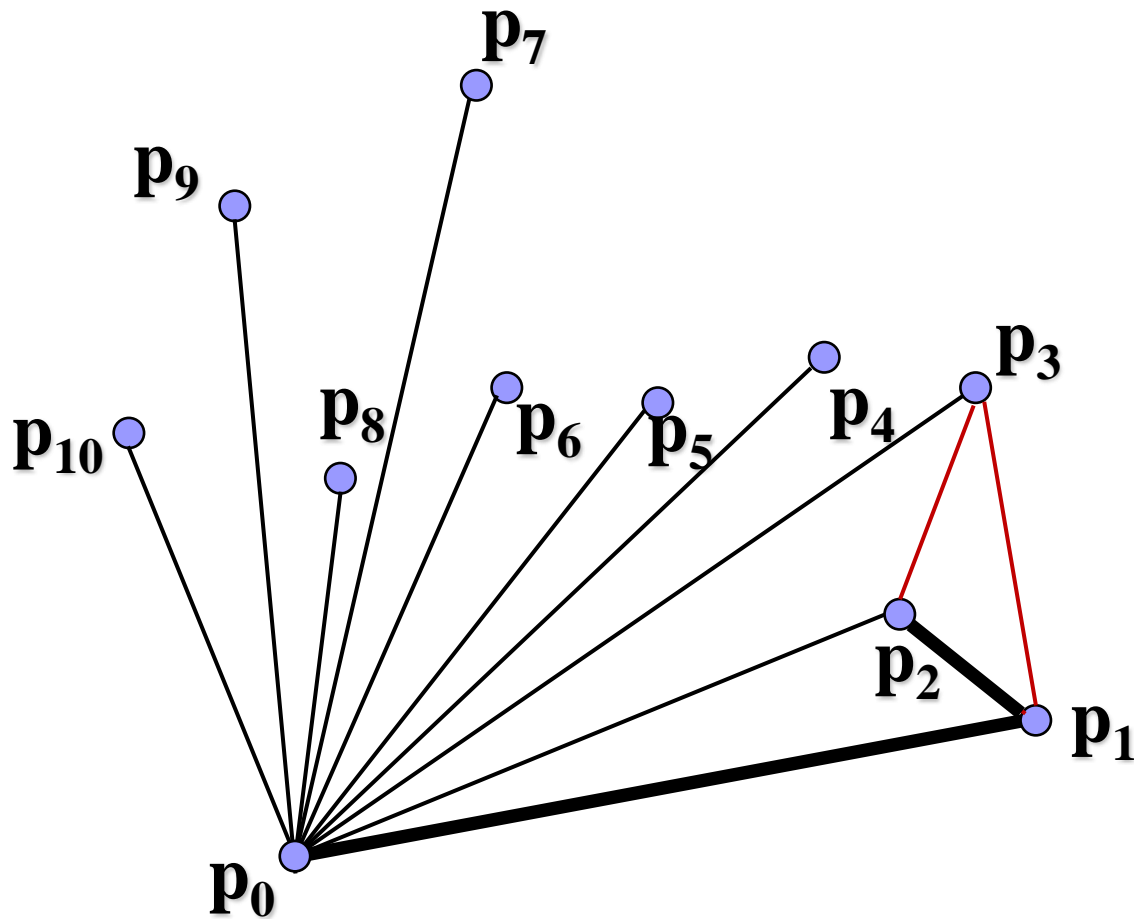
- Graham-scan的基本思想





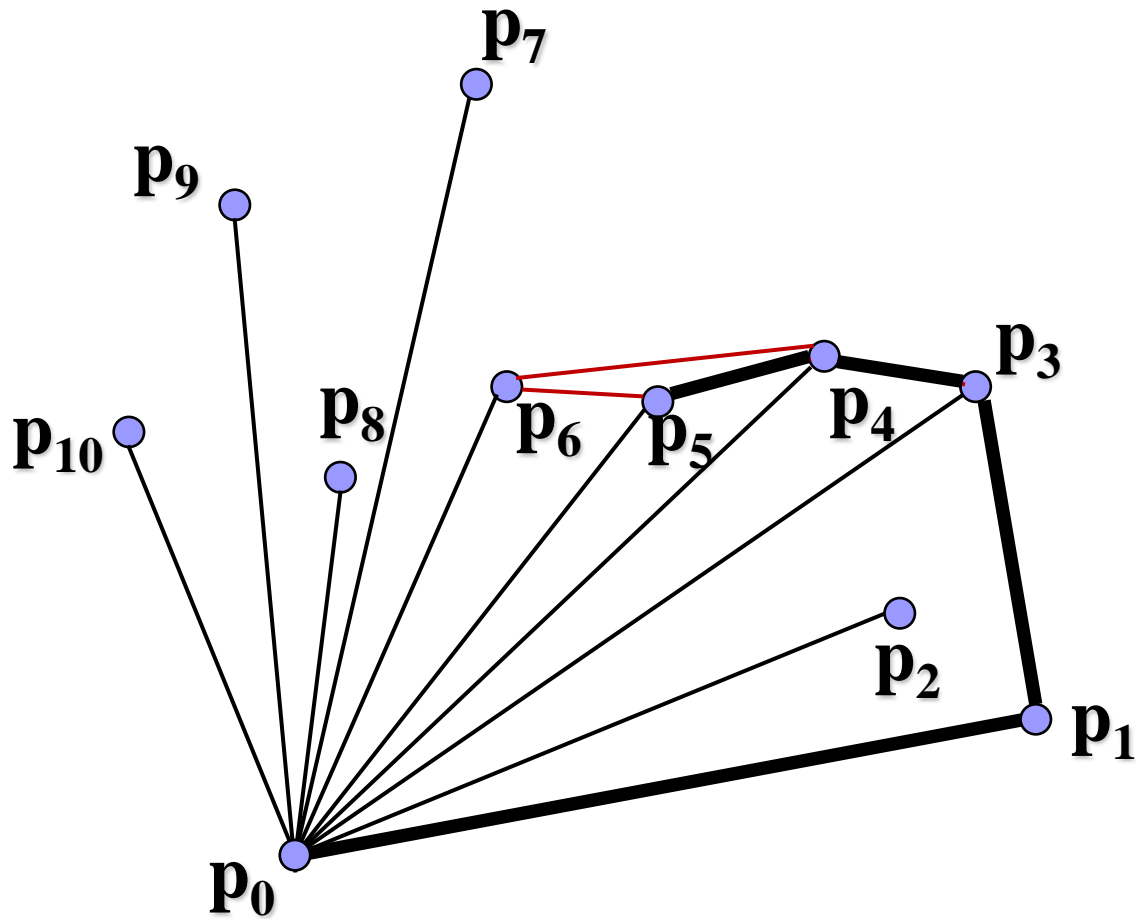
# 寻找凸包

- Graham-scan的基本思想



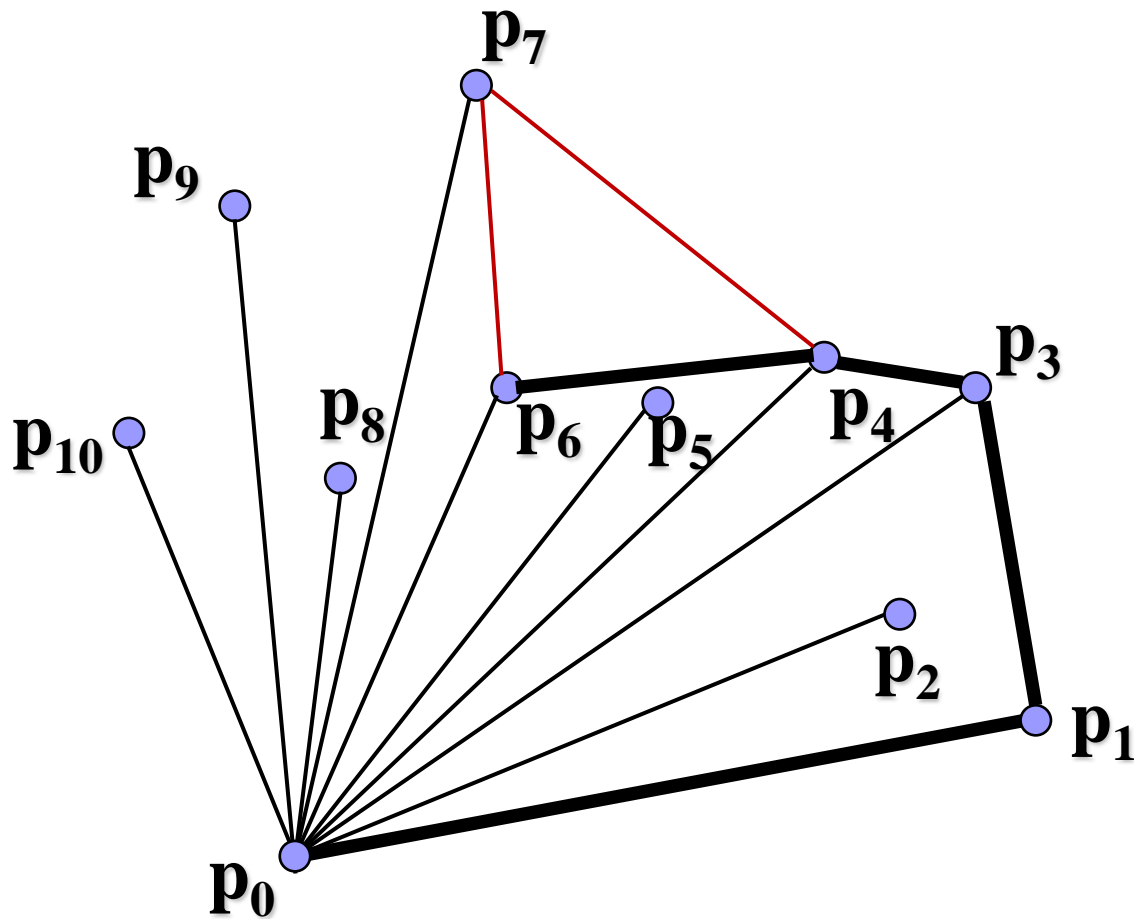
# 寻找凸包

## ■ Graham-scan的基本思想



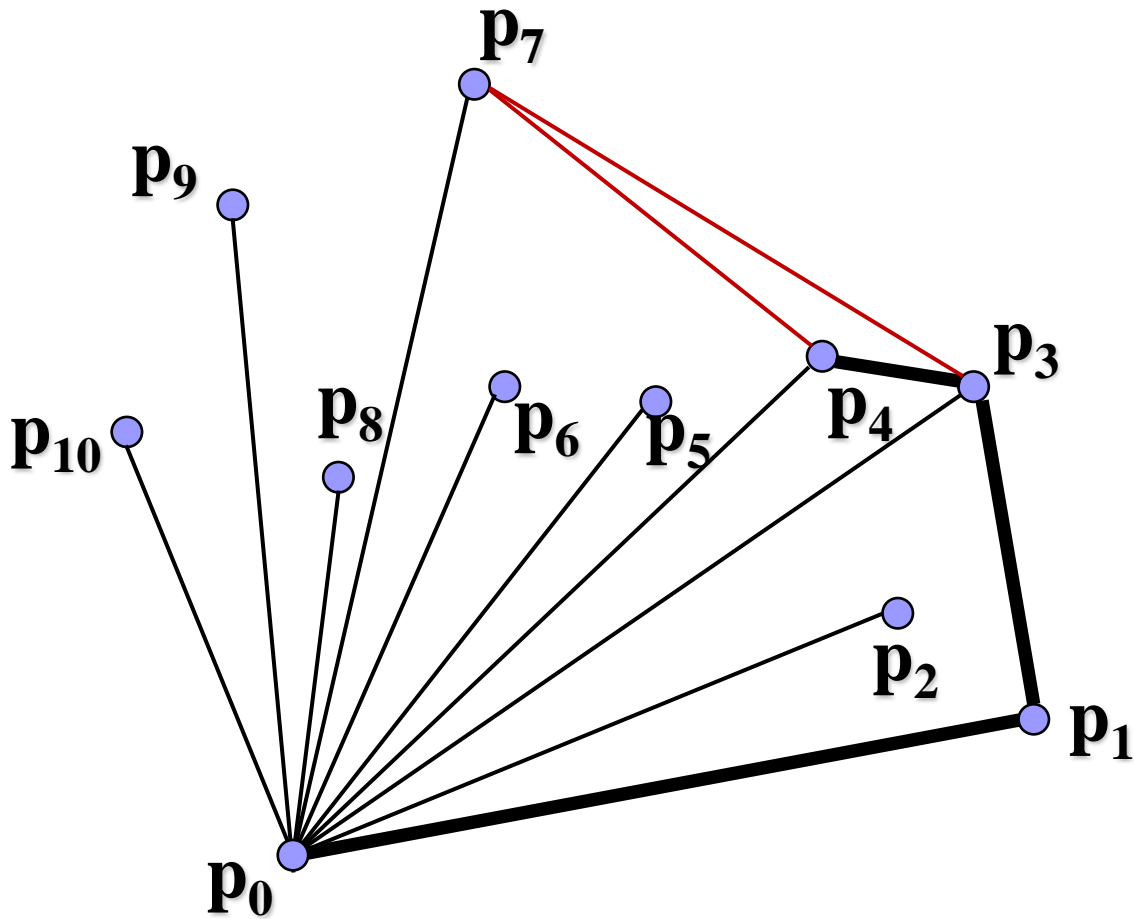
# 寻找凸包

- Graham-scan的基本思想



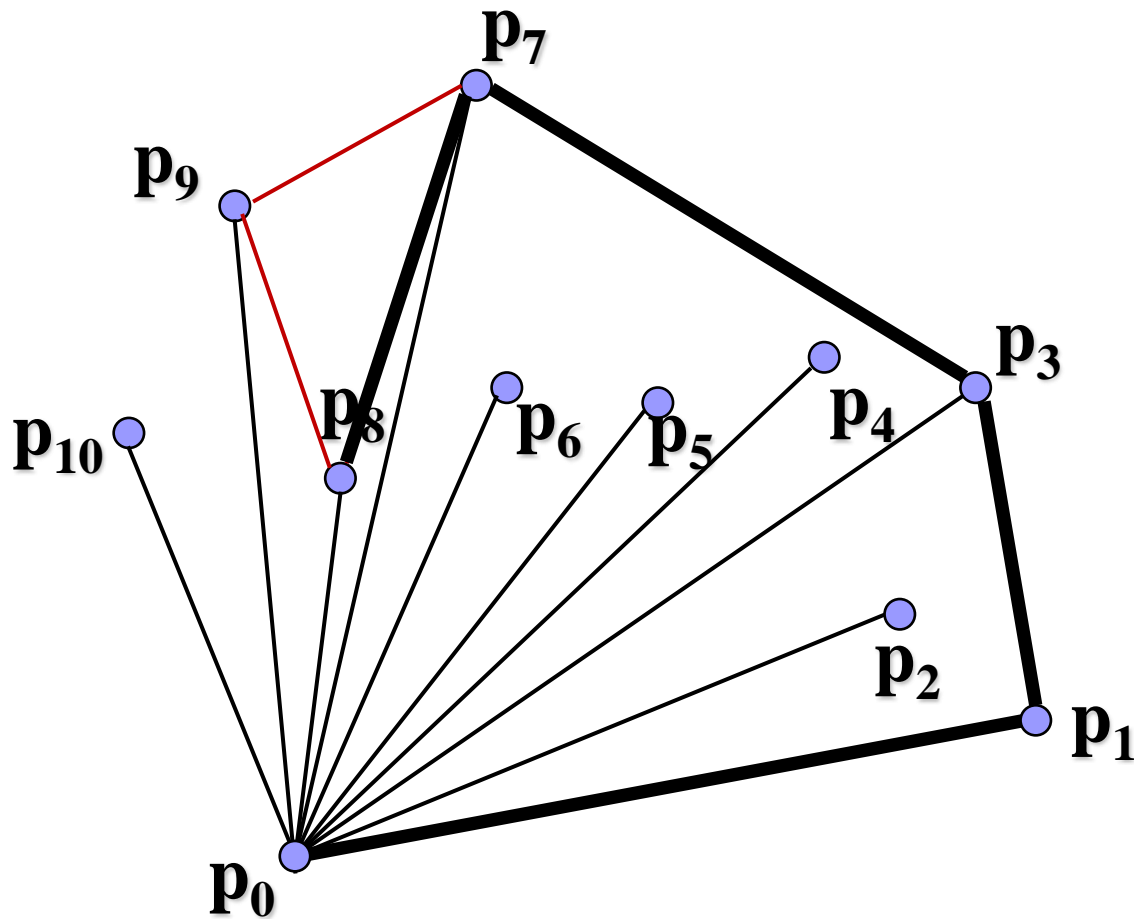
# 寻找凸包

## ■ Graham-scan的基本思想



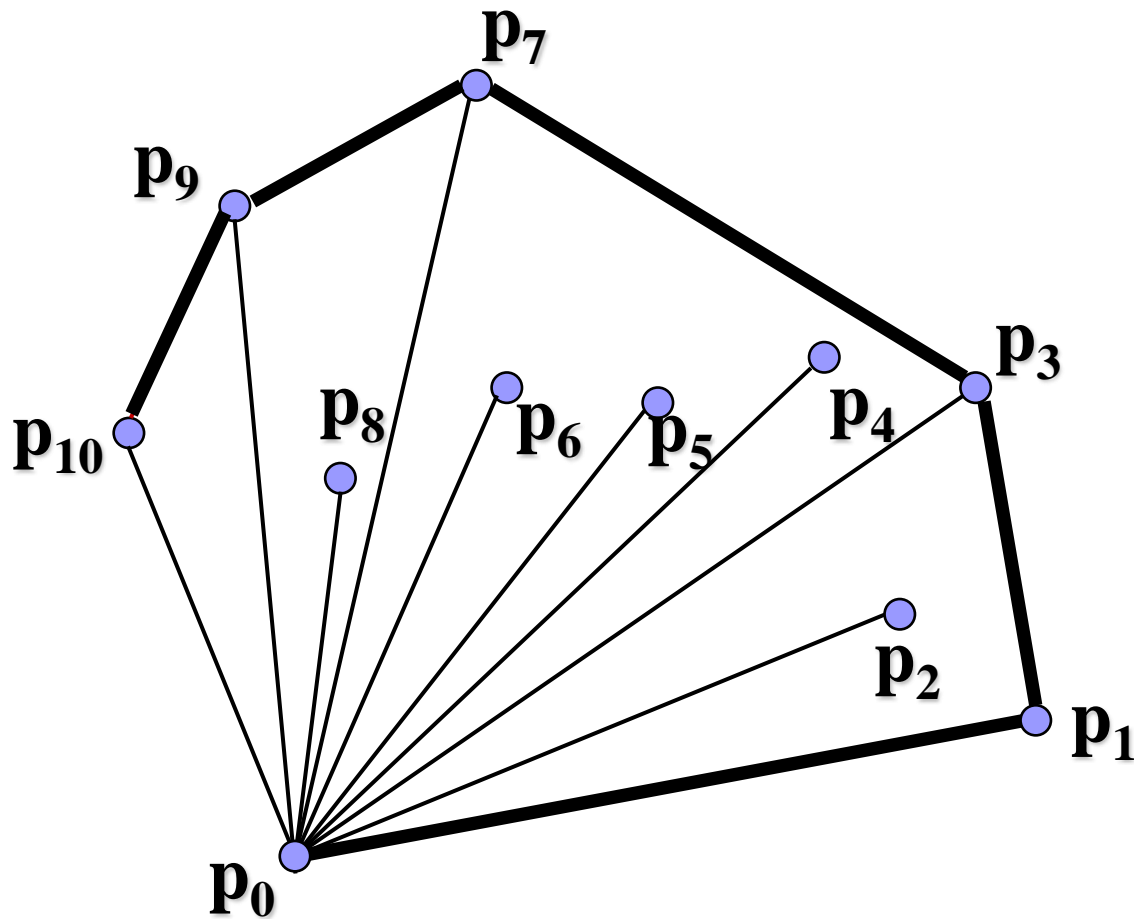
# 寻找凸包

## ■ Graham-scan的基本思想



# 寻找凸包


## ■ Graham-scan的基本思想



# 寻找凸包

## ■ Graham-scan的基本思想

### Graham-scan(Q)

- $O(n)$**  1. 求Q中y-坐标值最小的点 $p_0$ ;
- $O(n\log n)$**  2. 按照与 $p_0$ 极角(逆时针方向)大小排序Q中其余点,  
结果为 $\langle p_1, p_2, \dots, p_m \rangle$ ;
- $O(1)$**  3. Push( $p_0, S$ ); Push( $p_1, S$ ); Push( $p_2, S$ );
- $O(n)$**  4. FOR  $i=3$  TO  $m$  DO 
  - 5. While Next-to-top(S)、Top(S)和 $p_i$ 形成非左移动 Do
  - 6. Pop(S);
  - 7. Push( $p_i, S$ );
  - 8. Rerurn S;

循环为什么是 $O(n)$

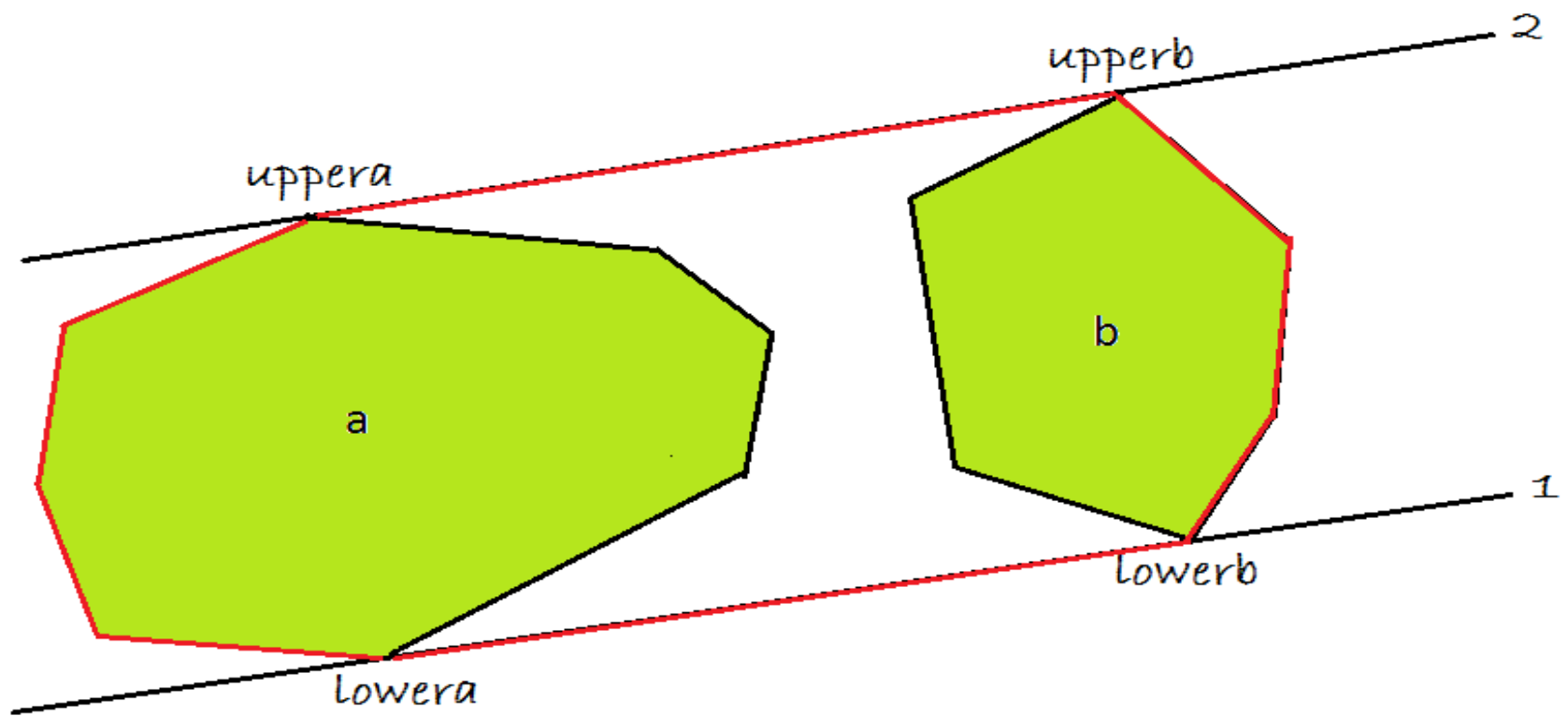
后面会介绍平摊分析

最多 $n$ 次入栈,

那么出栈也是最多 $n$ 次

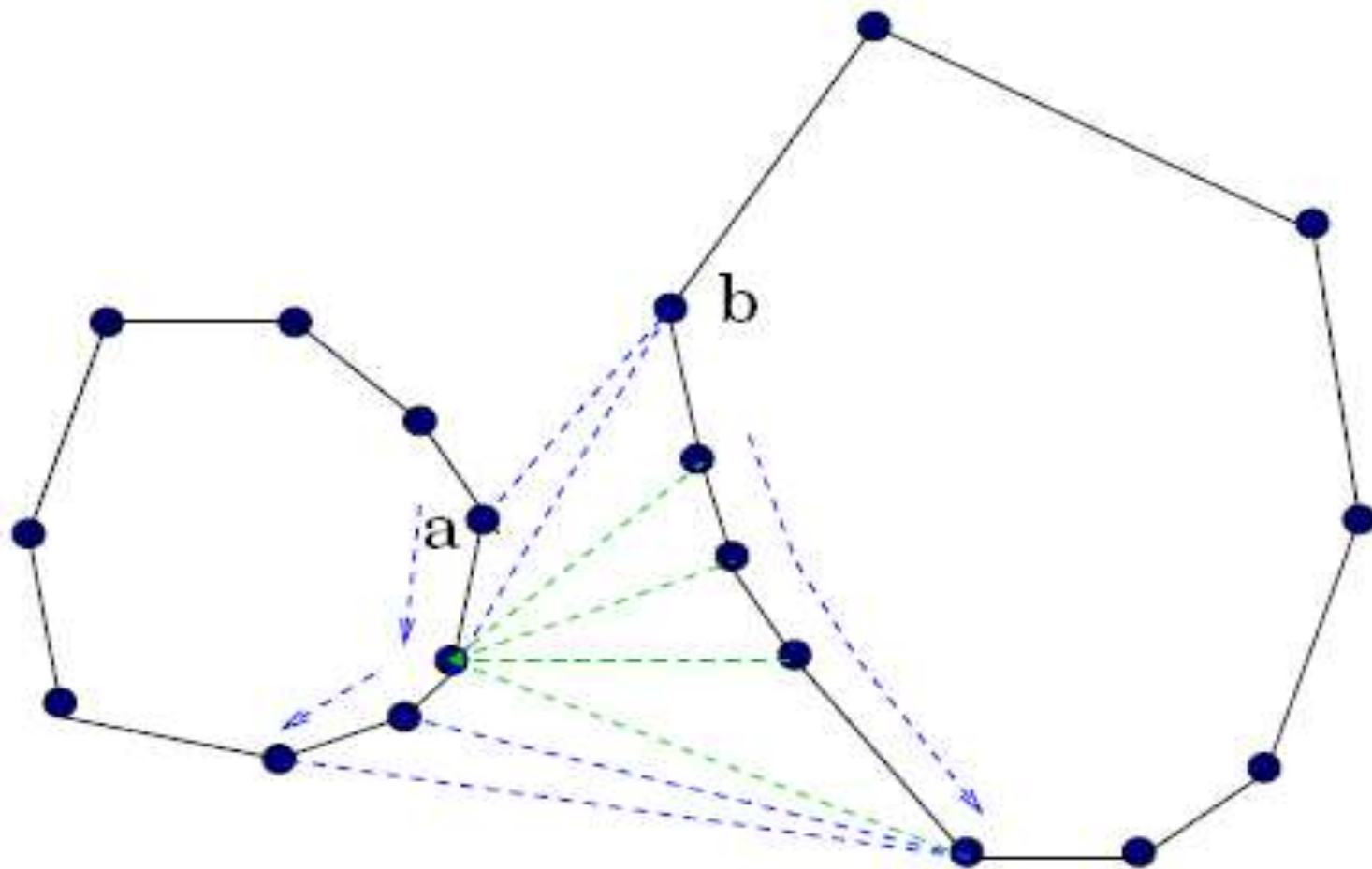
总时间复杂度 $O(n\log n)$

# 寻找凸包





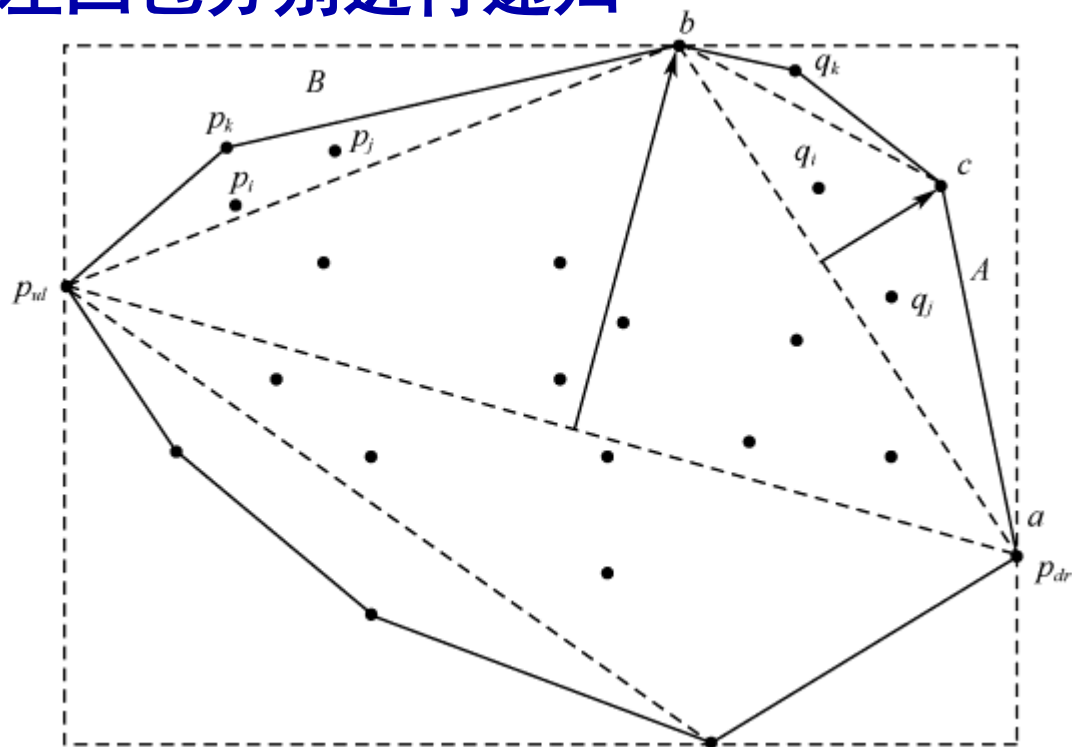
# 寻找凸包



# 寻找凸包

## ■ 分治法

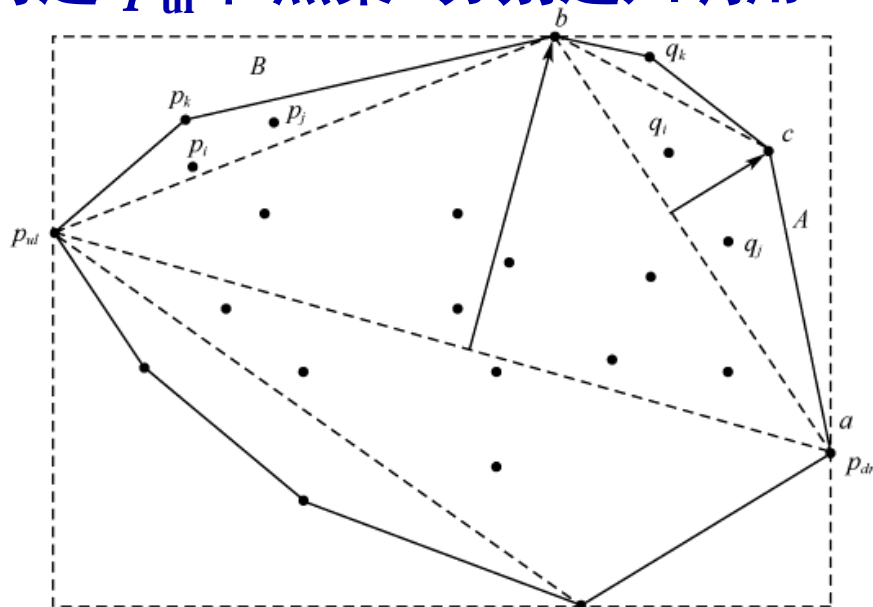
- 取两极端点，最右最下点 $p_{dr}$ 和最左最上点 $p_{ul}$
- 有向线 $p_{dr}p_{ul}$ 将整个凸包被划分为右凸包和左凸包
- 对右凸包和左凸包分别进行递归



# 寻找凸包

## ■ 分治法

- 设 $S_1$ 是严格在直线 $p_{dr}p_{ul}$ 右边的点集（ $S_1$ 可能是空集）
- 在 $S_1$ 中找距离直线 $p_{dr}p_{ul}$ 最远的点 $b$
- 连接 $p_{dr}$ 和 $b$ ，及 $b$ 和 $p_{ul}$
- 把 $p_{dr}b$ 右侧的点集记为A， $bp_{ul}$ 右侧的点集的点记为B
- 对边 $p_{dr}b$ 和点集A、对边 $bp_{ul}$ 和点集B分别递归调用



# 寻找凸包

- 分治法

平均复杂度  $O(n \log n)$   
最坏情况  $O(n^2)$

# 平衡

- 平衡与分治法是密切相关的，甚至是不可分的
  - 把规模为 $n$ 的问题分为两个子问题通常可以分为：
    - 一个规模为1而另一个规模为 $n-1$
    - 一个规模为2而另一个规模为 $n-2$
    - ...
    - 一个规模为 $\lfloor n/2 \rfloor$ 而另一个规模为 $\lceil n/2 \rceil$

哪种更好？

# 平衡

- 平衡与分治法是密切相关的，甚至是不可分的
  - 以排序为例：
    - 冒泡排序找出最小元，再对剩下的 $n-1$ 元排序
    - 一个规模为1而另一个规模为 $n-1$
    - 时间复杂度 $T(n)=T(n-1)+(n-1)$ 且 $T(1)=0$
    - $T(n)=(n-1)+(n-2)+\dots+3+2+1=n(n-1)/2=\Theta(n^2)$

# 平衡

- 平衡与分治法是密切相关的，甚至是不可分的
  - 以排序为例：
    - 归并排序划分成一个规模为 $\lfloor n/2 \rfloor$ ，另一个规模为 $\lceil n/2 \rceil$ 的子表排序，再以 $O(n)$ 时间归并
    - 时间复杂度 $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$
    - $\approx 2T(n/2) + O(n) = \Theta(n \log n)$

**显然归并排序好**

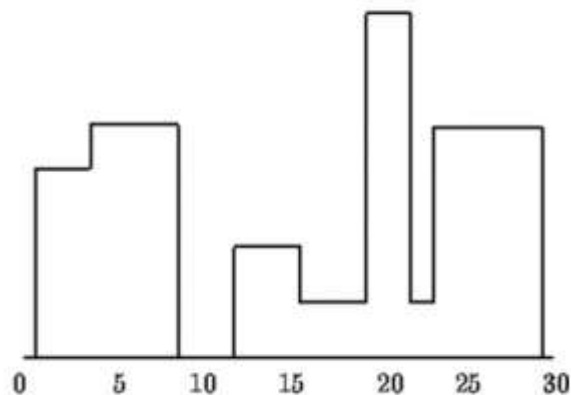
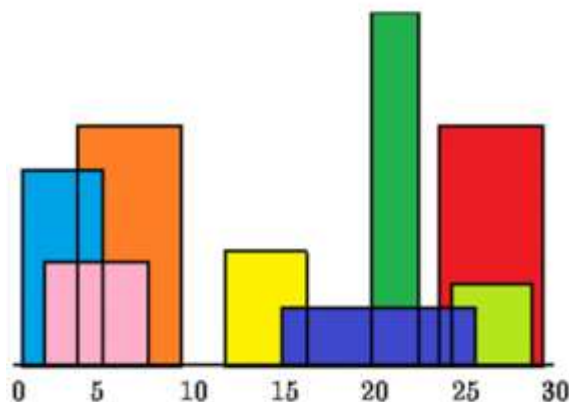
在使用分治法和递归时，  
要尽量把问题分成规模相等，  
或至少是规模相近的子问题，即**平衡**

# POJ 3714

- 设 $X[0:n-1]$ 和 $Y[0:n-1]$ 为两个数组，每个数组中含有 $n$ 个已排好序的数。试设计一个 $O(\log n)$ 时间的分治算法，找出 $X$ 和 $Y$ 的 $2n$ 个数的中位数，并证明算法的时间复杂性为 $O(\log n)$ 
  - 个数为奇数,则处于最中间位置的数
  - 个数为偶数,则中间两个数据的平均数
- 问题同上，只是 $X[0:m-1]$ 和 $Y[0:n-1]$ ，两有序数组长度不同，试设计一个 $O(\log(m+n))$ 时间的分治算法



- 给定 $n$ 座建筑物  $B[1, 2, \dots, n]$ ，每个建筑物  $B[i]$  表示为一个矩形，用三元组  $B[i] = (a_i, b_i, h_i)$  表示，其中  $a_i$  表示建筑左下顶点， $b_i$  表示建筑的右下顶点， $h_i$  表示建筑的高，请设计一个  $O(n \log n)$  的算法求出这  $n$  座建筑物的天际轮廓。例如，左下图所示中8座建筑的表示分别为  $(1, 5, 11)$ ,  $(2, 7, 6)$ ,  $(3, 9, 13)$ ,  $(12, 16, 7)$ ,  $(14, 25, 3)$ ,  $(19, 22, 18)$ ,  $(23, 29, 13)$  和  $(24, 28, 4)$ ，其天际轮廓如右下图所示可用9个高度的变化  $(1, 11)$ ,  $(3, 13)$ ,  $(9, 0)$ ,  $(12, 7)$ ,  $(16, 3)$ ,  $(19, 18)$ ,  $(22, 3)$ ,  $(23, 13)$  和  $(29, 0)$  表示。另举一个例子，假定只有一个建筑物  $(1, 5, 11)$ ，其天际轮廓输出为2个高度的变化  $(1, 11)$ ,  $(5, 0)$ 。



- **最大子数组问题。** 一个包含 $n$ 个整数（有正有负）的数组 $A$ ，设计一 $O(n\log n)$ 算法找出和最大的非空连续子数组。对于此问题你还能设计出 $O(n)$ 的算法吗？
  - 例如： $[0, -2, 3, 5, -1, 2]$ 应返回9，
  - $[-9, -2, -3, -5, -3]$ 应返回-2
- **循环移位问题。** 给定一个数组，数组中元素按从小到大排好序，现将数组中元素循环右移若干位，请设计一算法，计算出循环右移了多少位。

- 两元素和为 $X$ 。给定一个由 $n$ 个实数构成的集合 $S$ 和另一个实数 $x$ ，判断 $S$ 中是否有两个元素的和为 $x$ 。试设计一个分治算法求解上述问题，并分析算法的时间复杂度。
- 有一实数序列 $a_1, a_2, \dots, a_N$ ，若 $i < j$ 且 $a_i > a_j$ ，则 $(a_i, a_j)$ 构成了一个逆序对，请使用分治方法求整个序列中逆序对个数，并分析算法的时间复杂性。
  - 例如：序列 $(4, 3, 2)$ 逆序对有 $(4, 3)$ ， $(4, 2)$ ， $(3, 2)$ 共3个