算法设计与分析 第三部分

递归 分治法及实例分析

主要内容

- 递归
- 递归实例
- 递归式
- 分治法
- 分治法实例

递归的概念

- 递归函数
  - 用函数自身给出定义的函数
- 递归算法
  - 一个算法包含对自身的调用
  - 这种调用可以是直接的，也可以是间接的

递归举例-阶乘函数

- 阶乘函数

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

递归出口

递归方程

```
int factorial(int n)
{
    if(n==0) return 1;
    else return n*factorial(n-1);
}
```

$$n! = 1 \cdot 2 \cdot 3 \cdots\cdots (n-1) \cdot n$$

递归举例-Fibonacci数列

- Fibonacci数列
  - 无穷数列1，1，2，3，5，8，13，21，34，55，…

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

递归出口

递归方程

```
int fibonacci(int n)
{
    if(n<=1) return 1;
    return fibonacci(n-1)+fibonacci(n-2);
}
```

递归举例-Fibonacci数列

- Fibonacci数列 非递归定义

$$F(n)=\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}-\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right)$$

递归举例-整数划分问题

- 整数划分问题
  - 将正整数n表示成一系列正整数之和：$n=n_1+n_2+\ldots+n_k$，其中$n_1\geq n_2\geq\ldots\geq n_k\geq 1$，$k\geq 1$
  - 正整数n的这种表示称为正整数n的划分
  - 正整数n的不同的划分个数称为正整数n的划分数$p(n)$
  - 目标：求正整数n的不同划分个数$p(n)$
  - 例，正整数5的划分,$p(5)=7$

递归举例-整数划分问题

- 整数划分问题
- 引入m,将最大加数n_1不大于m的划分个数记作$q(n,m)$

$$q(n,m)=\begin{cases} 1 & n=1, m=1 \\ q(n,n) & n<m \\ 1+q(n,n-1) & n=m \\ q(n,m-1)+q(n-m,m) & n>m>1 \end{cases}$$

- $p(n)=q(n,n)$

递归举例-整数划分问题

- 整数划分问题

```
int q(int n,int m)
{
   if((n<1)||(m<1)) return 0;
   if((n==1)||(m==1)) return 1;
   if(n<m) return q(n,n);
   if(n==m) return q(n,m-1)+1;
   return q(n,m-1)+q(n-m,m);
}
```

递归举例-汉诺(Hanoi)塔问题

- 汉诺(Hanoi)塔问题
  - 设a,b,c是3个塔座
  - 开始时，在塔座a上有一叠共*n*个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1,2,…,*n*
  - 要求将塔座a上的圆盘移到塔座b上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：
    - 每次只能移动1个圆盘
    - 任何时刻都不允许将较大的圆盘压在较小的圆盘之上
    - 在满足移动规则1和2的前提下，可将圆盘移至a,b,c中任一塔座上

递归举例-汉诺(Hanoi)塔问题

- 汉诺(Hanoi)塔问题分析
  - n=1时，直接a->b即可
  - n>1时，借助c实现移动，可先将n-1个圆盘按照规则a->c，再将大圆盘a->b，最后将n-1个圆盘c->b
  - 可以通过递归实现

伪码：

```
hanoi(int n,int a,int b,int c)
{
    if(n>0){hanoi(n-1,a,c,b);
    move(a,b);
    hanoi(n-1,c,b,a)}
}
```

递归

- 递归的概念非常重要
- 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，为设计算法、调试程序带来很大便利
- 缺点
  - 递归算法的运行效率较低

递归式（Recurrence）

- 当一个算法包含对自身的递归调用时，其运行时间通常可以用递归式来表示
- 例，对于合并排序，其最坏情况时间复杂度

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

- 其解为 $T(n)=\Theta(n\lg n)$

递归式的解法

- 代换法（substitution method）
- 递归树方法（recursion-tree method）
- 主方法（master method）

代换法（substitution method）

- 步骤
  - 猜测解的形式
  - 用数学归纳法证明之
- 只适用于解的形式很容易猜的情形
- 如何猜测则需要经验
- 例 $T(n)=2T(\lfloor n/2 \rfloor)+n \qquad T(n)=O(n\lg n)$

$$T(n)=2T(\lfloor n/2 \rfloor+17)+n \qquad T(n)=O(n\lg n)$$

$$T(n)=T(\lceil n/2 \rceil)+1 \qquad T(n)=O(\lg n)$$

递归树方法（recursion-tree method）

- 每一个节点代表递归函数调用集合中一个子问题的代价，将所有层的代价相加得到总代价
- 当用递归式表示算法的时间复杂度时，可用递归树的方法
- 递归树方法模拟了算法的递归执行，可以由递归树方法产生对算法时间复杂度的较好猜测

递归树方法示例

求解 $T(n) = T(n/4) + T(n/2) + n^2$

主方法（master method）

- $T(n) = aT(n/b) + f(n)$
- $a>=1$, $b>1$, a和b均为常数
- $f(n)$是渐近正函数

主定理（master theorem）

- 对于递归式，比较$f(n)$和$n^{\log_b a}$
- 若对于某常数$\varepsilon > 0$, $f(n) = O(n^{\log_b a-\varepsilon})$, 则$T(n) = O(n^{\log_b a})$
- 若 $f(n) = \Theta(n^{\log_b a})$, 则$T(n) = \Theta(n^{\log_b a} \lg n)$
- 若对于某常数$\varepsilon > 0$,有 $f(n) = \Omega(n^{\log_b a+\varepsilon})$

  且对常数$c<1$与所有足够大的n,有

  $af(n/b) \le cf(n)$, 则$T(n) = \Theta(f(n))$

主方法的应用

- 请注意，上述三种情况没有覆盖所有的$f(n)$
- 在应用时需要注意是否符合这三种情况
- $T(n) = 4T(n/2) + n$
- $T(n) = 4T(n/2) + n^2$
- $T(n) = 4T(n/2) + n^3$
- $T(n) = 4T(n/2) + n^2/\lg n$
- $T(n) = 2T(n/2) + n\lg n$

分治法

- 分治法的基本策略
  - 分解（Divide）：将原问题分解为子问题
  - 解决（Conquer）：求解子问题
  - 合并（Combine）：组合子问题的解得到原问题的解

分治法的适用条件

- 适合分治法求解的问题一般具有以下特征
  - 问题的规模缩小到一定程度就可以容易地解决
  - 问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质
  - 基于子问题的解可以合并为原问题的解
  - 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

平衡

- 使子问题规模尽量接近的做法，就是平衡
- 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以提高算法的效率

分治法实例

- 二分搜索（Binary search）
  - 问题：在已排好序的数组中寻找特定元素
  - 分解：检查中间元素
  - 解决：递归搜索子数组
  - 合并：……
- 例，在数组中寻找9
-

3 5 7	8		9		12	15

二分搜索分析

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

子问题数目

子问题规模

分解和合并代价

- 应用主方法，对应第二种情况
- 即 $a = 1, b = 2, n^{\log_b a} = n^0 = 1$

$f(n) = \Theta(1) = \Theta(n^{\log_b a})$, 则 $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$

分治法实例

- Powering a number，求a^n
  - 直接求解：Θ(n)
  - 分治法求解

$$a^n = \begin{cases} a, & if\ \ n = 1; \\ a^{n/2} * a^{n/2} & if\ \ n > 1\, and\ \ a\ is\ even; \\ a^{(n-1)/2} * a^{(n-1)/2} * a & if\ \ n > 1\, and\ \ a\ is\ odd. \end{cases}$$

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

$$a = 1, b = 2, n^{\log_b a} = n^0 = 1$$

$$f(n) = \Theta(1) = \Theta(n^{\log_b a}),\text{则}T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$$

分治法实例

- 快速排序算法，对数组A[p..r]进行排序
  - 分解：数组A[p..r]被划分为子数组A[p..q-1]和A[q+1..r]， A[p..q-1]中的每个元素都小于等于A[q]，A[q+1..r]中的每个元素都大于等于A[q]，q在划分时确定
  - 解决：通过递归调用快速排序算法，对子数组A[q+1..r]和A[p..q-1]进行排序
  - 合并：由于子数组的排序为原地排序，解的合并不需要操作，整个数组已经排好序

快速排序算法

```
QUICKSORT(A,p,r)
        if p<r
           then   q=PARTITIION(A,p,r)
                  QUICKSORT(A,p,q-1)
                  QUICKSORT(A,q+1,r)
```

QUICKSORT(A,1,length[A])

快速排序算法

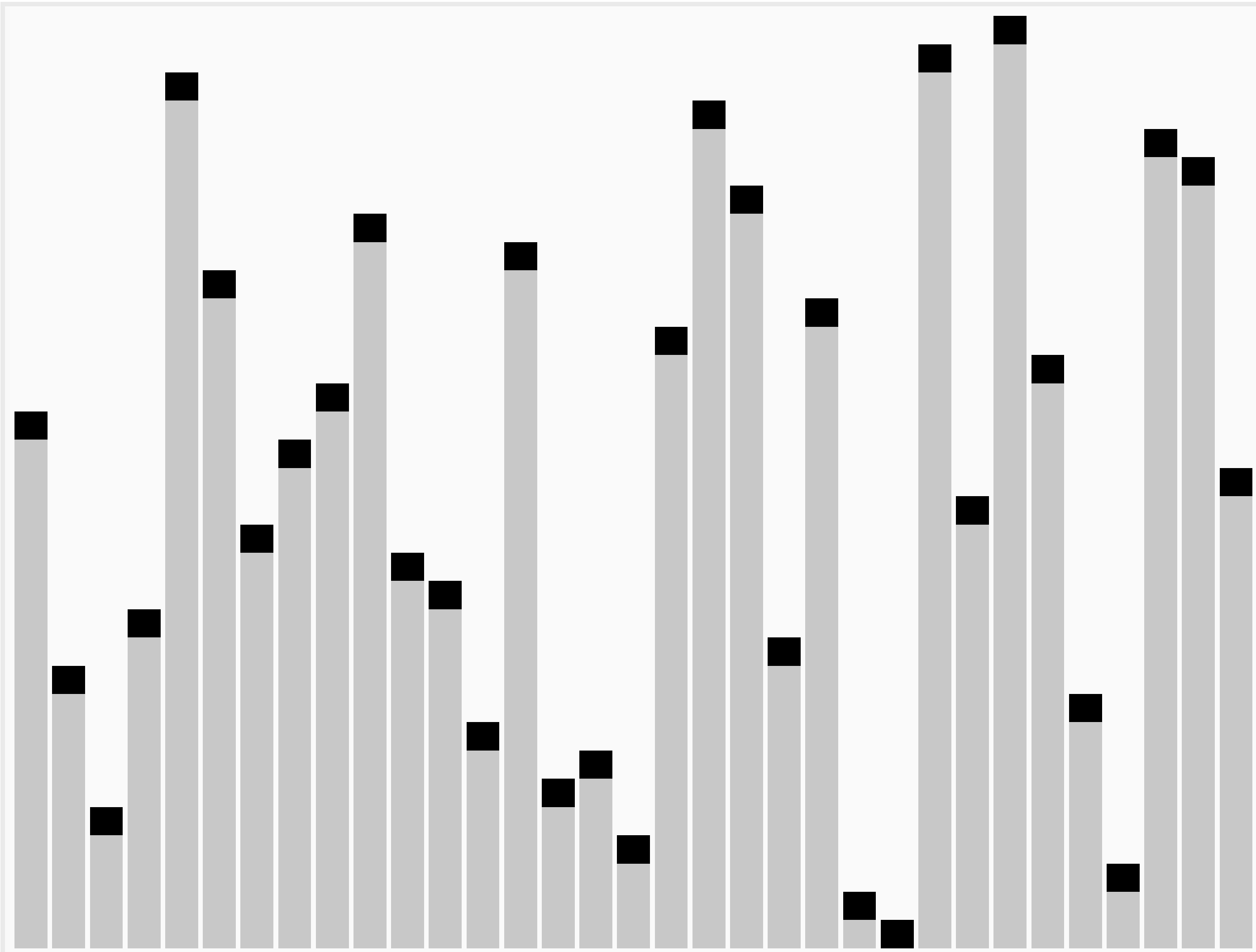
PARTITION(A, p, r)
  $x \leftarrow A[r]$
  $i \leftarrow p$-1
  for $j \leftarrow p$ to r-1
    do if $A[j] \leq x$
      then $i \leftarrow i + 1$
        exchange $A[i] \boxed{\leftrightarrow} A[j]$
  exchange $A[i+1] \boxed{\leftrightarrow} A[r]$
  return i+1

PARTITION过程

- 以 2 8 7 1 3 5 6 4 为例
- 以 6 10 13 5 8 3 2 11为例

PARTITION(A, p, r)
    $x \leftarrow A[r]$
    $i \leftarrow p-1$
    for $j \leftarrow p$ to $r-1$
        do if $A[j] \leq x$
            then $i \leftarrow i+1$
                exchange $A[i] \boxed{\leftrightarrow} A[j]$
    exchange $A[i+1] \boxed{\leftrightarrow} A[r]$
    return $i+1$

快速排序算法



快速排序算法的分析

- 最坏情况时间复杂度$\Theta(n^2)$
- 平均情况时间复杂度$\Theta(n\lg n)$

快速排序的分析

- 最坏情况划分
- 最好情况划分
- 平衡的划分

```
QUICKSORT(A,p,r)
    if p<r
        then   q=PARTITIION(A,p,r)
               QUICKSORT(A,p,q-1)
               QUICKSORT(A,q+1,r)
```

PARTITION(A, p, r)

$x \leftarrow A[r]$

$i \leftarrow p$-1

for $j \leftarrow p$ to r-1

do if $A[j] \le x$

then $i \leftarrow i+1$

exchange $A[i] \leftrightarrow A[j]$

exchange $A[i+1] \leftrightarrow A[r]$

return $i+1$

快速排序的随机化版本

- 主要区别在于主元的选择
  - 不总是选择A[r]作为主元，而是从A[p…r]中随机选择一个元素作为主元
  - 具体操作方法是将A[r]与A[p…r]中的随机选中的一个元素交换

```
RANDOMIZED-PARTITION(A, p, r)
   i ← RANDOM(p, r)
   exchange A[r] ↔ A[i]
   return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
   if p < r
      then q ← RANDOMIZED-PARTITION(A, p, r)
           RANDOMIZED-QUICKSORT(A, p, q - 1)
           RANDOMIZED-QUICKSORT(A, q + 1, r)
```

快速排序的实际应用

- 通常是用于排序的最佳实用选择
- 原地排序，在虚存环境中也可以很好工作

Fibonacci数列

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

- 递归算法：$\Omega(\varphi^n), \varphi=(1+\sqrt{5})/2$
- 自底向上，依次计算
  - $\Theta(n)$
- 有更好的方法吗？

Fibonacci数列

- 矩阵法

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

矩阵乘法

- 矩阵乘法

$$\left.\begin{array}{l} A=\left[a_{ij}\right], B=\left[b_{ij}\right] \\ C=A \cdot B=[c_{ij}] \end{array}\right\} i, j=1,2,\ldots,n$$

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

矩阵乘法

● 直接解法

for $i \leftarrow 1$ **to** n

  do for $j \leftarrow 1$ **to** n

    do $cij \leftarrow 0$

      for $k \leftarrow 1$ **to** n

        do $cij \leftarrow cij + aik \cdot bkj$

● 时间复杂度

$\Theta(n^3)$

矩阵乘法

- 分治策略
  - 假设n为2的幂，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×n/2的方阵

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

$$C = A \cdot B$$

矩阵乘法

- 分治策略分析

$$T(n) = 8T(n/2) + \Theta(n^2)$$
$$T(n) = \Theta(n^3)$$

- 直接分治的时间复杂度并不比直接计算好

Strassen的策略

- 只需要7次子矩阵的乘法
  - 引入M_i(i=1,2,…,7)，如下

$$M_1 = A_{11}(B_{12} - B_{22})$$
$$M_2 = (A_{11} + A_{12})B_{22}$$
$$M_3 = (A_{21} + A_{22})B_{11}$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

$$C_{11} = M_5 + M_4 - M_2 + M_6$$
$$C_{12} = M_1 + M_2$$
$$C_{21} = M_3 + M_4$$
$$C_{22} = M_5 + M_1 - M_3 - M_7$$

Strassen矩阵乘法分析

- $T(n) = 7T(n/2) + \Theta(n^2)$

  $T(n) \approx \Theta(n^{2.81})$

- 更好的算法？

  $T(n) = \Theta(n^{2.376})$

最大元、最小元

- 给定n个数据元素，找出其中的最大元和最小元
  - 直接解法：逐个找，用n-1次比较来找出最大元，再用n-2次比较来找出最小元，比较次数（基本运算）为2n-3次
  - 可以用分治法吗？

最大元、最小元

- 分治法
  - 当n=2时，一次比较就可以找出两个数据元素的最大元和最小元
  - 当n>2时，可以把n个数据元素分为大致相等的两半
  - 求数组最大元、最小元的算法下界

$$\lceil 3n/2-2 \rceil$$

一维的最近点对问题

- n个点退化为n个实数，最近点对即为这n个实数中相差最小的两个实数
- 分治法求解
  - 分解：用各点坐标的中位数m作为分割点，分成两个点集
  - 求解：在两个点集上分别找出其最接近点对{p1,p2}和{q1,q2}
  - 合并：整个点集的最近点对或者是{p1,p2}，或者是{q1,q2}，或者是某个{p3,q3}，其中p3和q3分属两个点集

一维的最近点对问题

- 合并
  - 如果最近点对是{p3,q3}，即 |p3-q3|<d，则p3和q3两者与m的距离不超过d，即p3∈(m-d,m]，q3∈(m,m+d]

最近点对问题

- 有n个点，输入点集记为P
- 分解
  - 将P进行分割，分为2部分求最近点对
  - 选择一条垂线L，将P拆分左右两部分为P_L和P_R

最近点对问题

- 解决
  - 分别寻找P_L和P_R中的最近点对及距离，设其找到的最近点对的距离分别是δ_L和 δ_R
  - 置$\delta=\min(\delta_L, \delta_R)$

最近点对问题

- 合并
  - 对于从P_L和P_R求得的δ_L和δ_R，如何合并？
  - 可能一：最近点对就是某次递归调用找出的距离为δ的点对
  - 可能二：最近点对是由P_L 中的一个点和P_R中的一个点组成的点对

最近点对问题

- 合并子问题
  - 关注以直线L为中心，宽度为2δ的垂直带状区域
  - 看是否可以找到一个点对的距离$\delta' < \delta$

最近点对问题

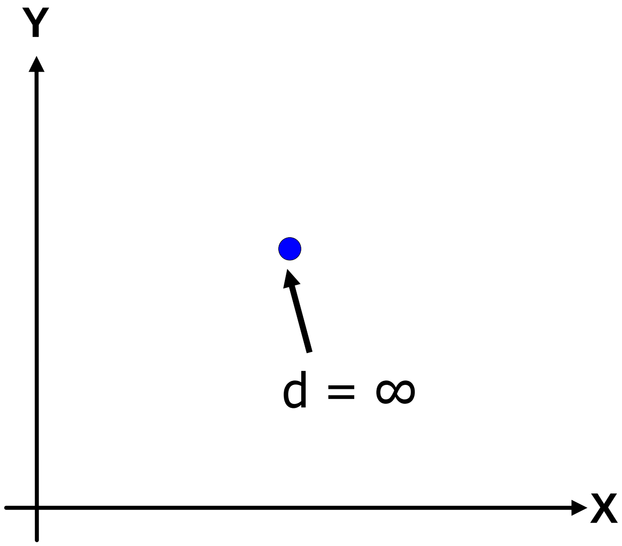
- 合并子问题 考察带状区域中的点
  - 对于带状区域中的每个点p，算法试图找出距离p在δ单位以内的点，如果距离$\delta'<\delta$，则更新当前的最近点对距离
  - 如果带状区域中的点是按y坐标升序排列的，对于每个点p，只需要检查p之后的7个点即可

最近点对问题

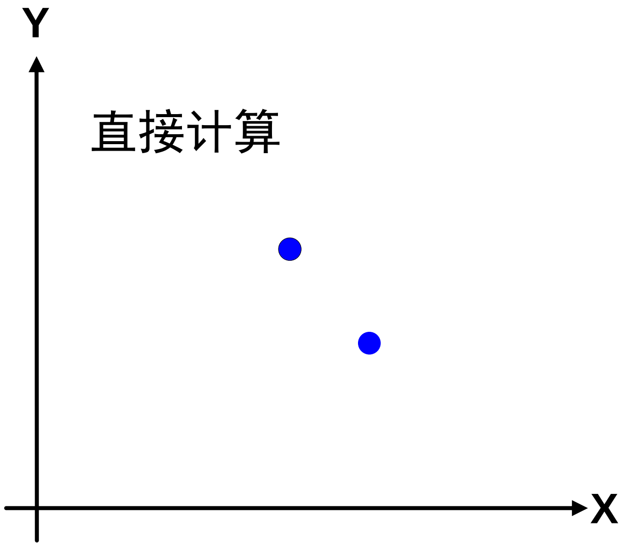
- 合并子问题小结
  - 找出以L为中心线，宽度为2δ的带状区域
  - 获得带状区域中排序后的点集Y'
  - 对Y'中的每个点，检查其后面的7个点，计算距离并更新最近点对的距离

最近点对问题

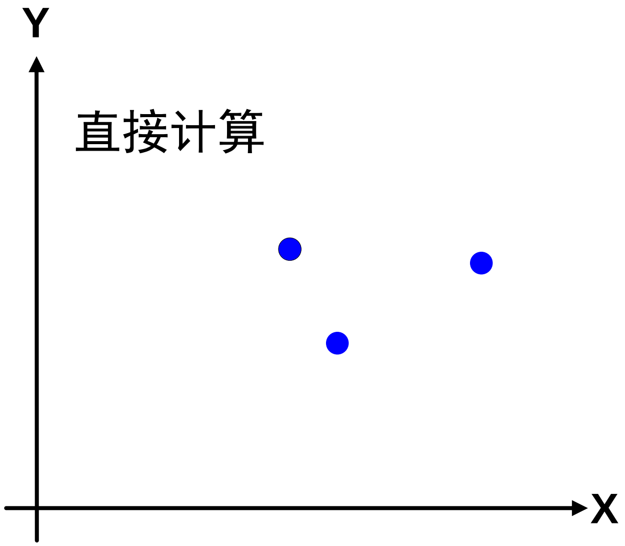
- 递归的出口
- 点数较少时的情形



只有一个点



只有两个点



只有三个点

最近点对问题

- 以L为中心线，宽度为2δ的带状区域中的点，如何筛选，并保证有序（按y坐标升序排列）
  - 筛选出来之后按照y坐标递增的方式进行排序？
  - 对Y进行预排序，将排好序的Y传入递归主过程

最近点对问题

- 难点
  - 如何在线性时间内获得

    Y_L, Y_R, X_L, X_R, Y'
  - 若X,Y已按相应坐标排好序

```
length[Y_L] ← length[Y_R] ← 0
for i ← 1 to length[Y]
    do if Y[i] ∈ P_L
          then length[Y_L] ← length[Y_L] + 1
               Y_L[length[Y_L]] ← Y[i]
          else length[Y_R] ← length[Y_R] + 1
               Y_R[length[Y_R]] ← Y[i]
```

最近点对问题

- 分析
- 递归式为

$$T(n) = 2T(n/2) + f(n)$$

$$f(n) = O(n)$$

$$T'(n) = T(n) + O(n\lg n)$$

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

$$T'(n) = O(n\lg n)$$

寻找顺序统计量问题

- 求第i小元素问题、选择问题
  - 设集合S中共有n个数据元素，要在S中找出第i小元素
- 最小元：第1个顺序统计量
- 最大元：第n个顺序统计量
- 中位数： $i = \lfloor (n+1)/2 \rfloor$

寻找顺序统计量问题

- 如下假设：
  - 集合由n个数值不同的元素组成
  - 在此假设下的结论可以推广到有重复数值的情形
- 问题描述：
  - 输入：一个包含n个不同数的集合A和一个数i
  - 输出：元素x，x大于A中其它的i-1个元素

寻找顺序统计量问题

- 求解方法
  - 排序
  - 期望线性时间
  - 最坏情况线性时间

寻找顺序统计量问题

- 排序
  - 合并排序
  - 堆排序
- 有更好的方法？
- 分治？

寻找顺序统计量问题

- 期望线性时间求解方法 $\Theta(n)$
- 分解：用到Random Partition
- 求解：递归处理
- 合并

寻找顺序统计量问题

- 期望线性时间求解方法 伪码

RAND-SELECT(A, p, q, i) ⊳ ith smallest of $A[p..q]$

if $p = q$ **then** **return** $A[p]$

$r \leftarrow$ RAND-PARTITION(A, p, q)

$k \leftarrow r - p + 1$ ⊳ $k = \mathrm{rank}(A[r])$

if $i = k$ **then** **return** $A[r]$

if $i < k$ **then** **return** RAND-SELECT(A, p, $r - 1$, i)

else **return** RAND-SELECT(A, $r + 1$, q, $i - k$)

寻找顺序统计量问题

● 期望线性时间求解方法 示例

Select the $i = 7$th smallest:

6	10	13	5	8	3	2	11	$i = 7$

pivot

Partition:

2	5	3	6	8	13	10	11	$k = 4$

Select the $7 - 4 = 3$rd smallest recursively.

寻找顺序统计量问题

- 期望线性时间求解方法 直观分析
- 一般情况 $T(n)=T(9n/10)+\Theta(n)=\Theta(n)$
- 最坏情况 $T(n)=T(n-1)+\Theta(n)=\Theta(n^2)$

寻找顺序统计量问题

- 期望线性时间求解方法
- 可以证明，在平均情况下，任何顺序统计量可以在线性时间内得到

寻找顺序统计量问题

- 最坏情况线性时间
- 基本思想：保证对数组的划分是好的划分

寻找顺序统计量问题

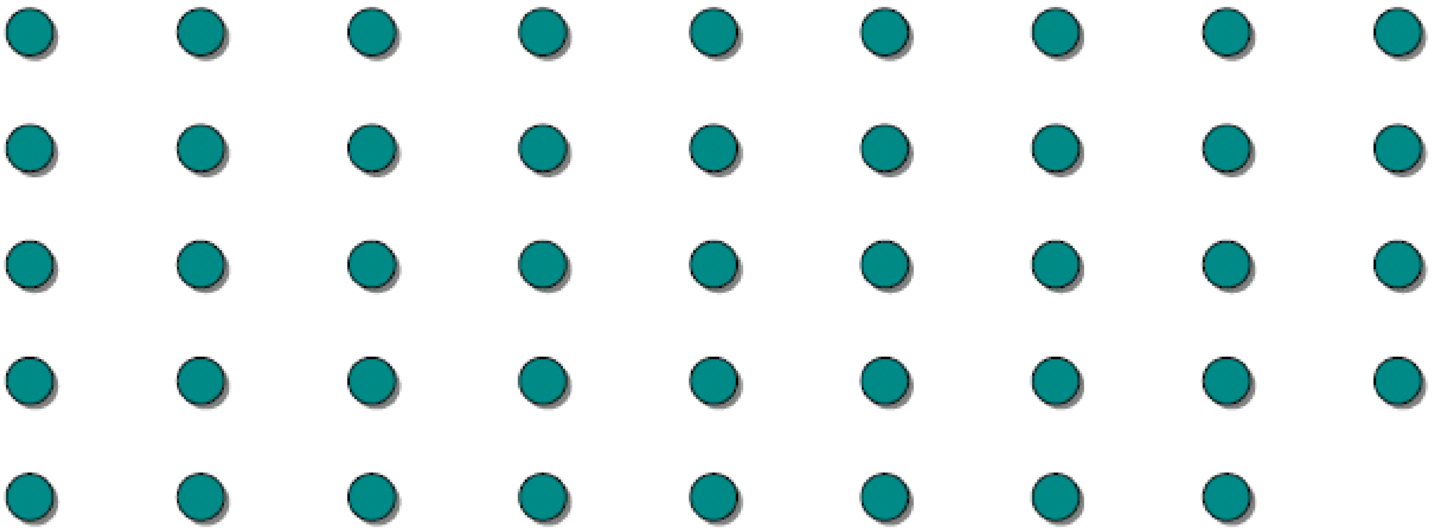
- SELECT的步骤
  - 1.将输入数组的n个元素分为$\lfloor n/5 \rfloor+1$组，其中$\lfloor n/5 \rfloor$组每组5个元素，余下的一组由剩下的n mod 5个元素组成；
  - 2.寻找这$\lfloor n/5 \rfloor+1$组中每一组的中位数（先对每组中的元素进行插入排序，然后从排序后的序列中选出中位数）；
  - 3.对第二步中找出的$\lfloor n/5 \rfloor+1$组中位数，递归调用SELECT以找到其中位数x；

寻找顺序统计量问题

- SELECT的步骤 续
  - 4.PARTITION，按中位数x对输入数组进行划分，x为第k小元素；
  - 5.如果i=k，则返回x；
  - 否则，如果i<k，则在低区递归调用SELECT寻找第i小元素
  - 否则，如果i>k，则 在高区寻找第(i-k)个最小元素

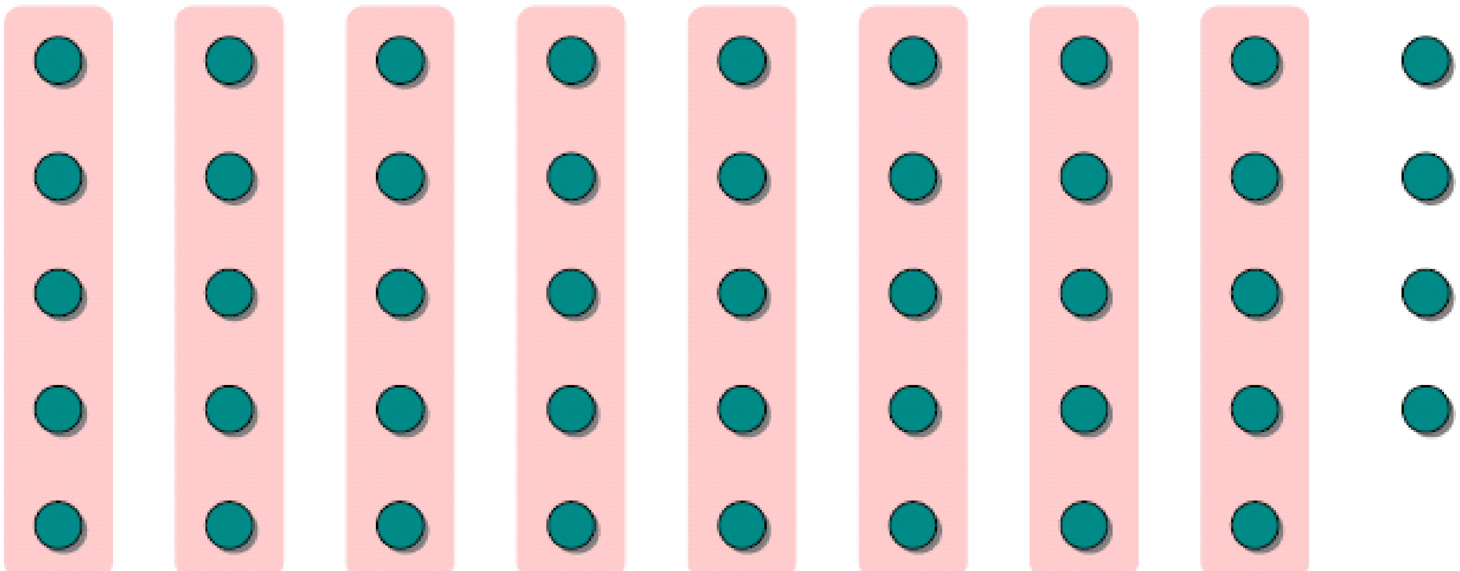
寻找顺序统计量问题

- 示例，n个元素



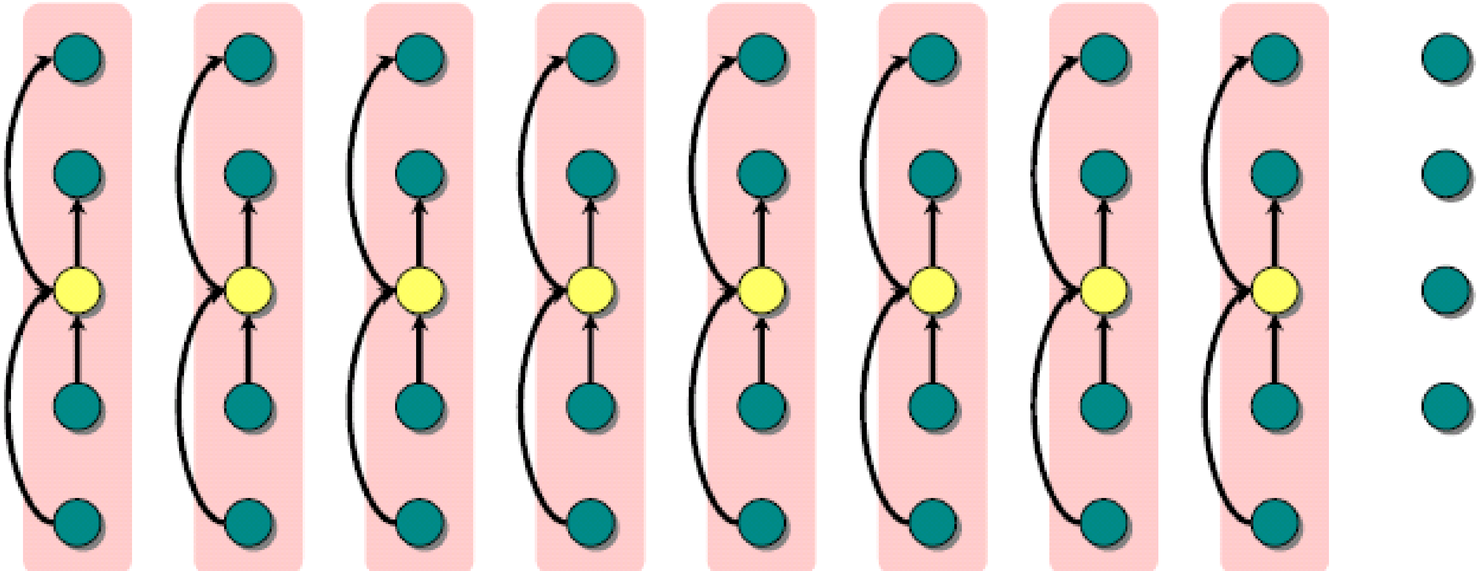
寻找顺序统计量问题

- 示例， n个元素分为$\lfloor n/5 \rfloor$+1组



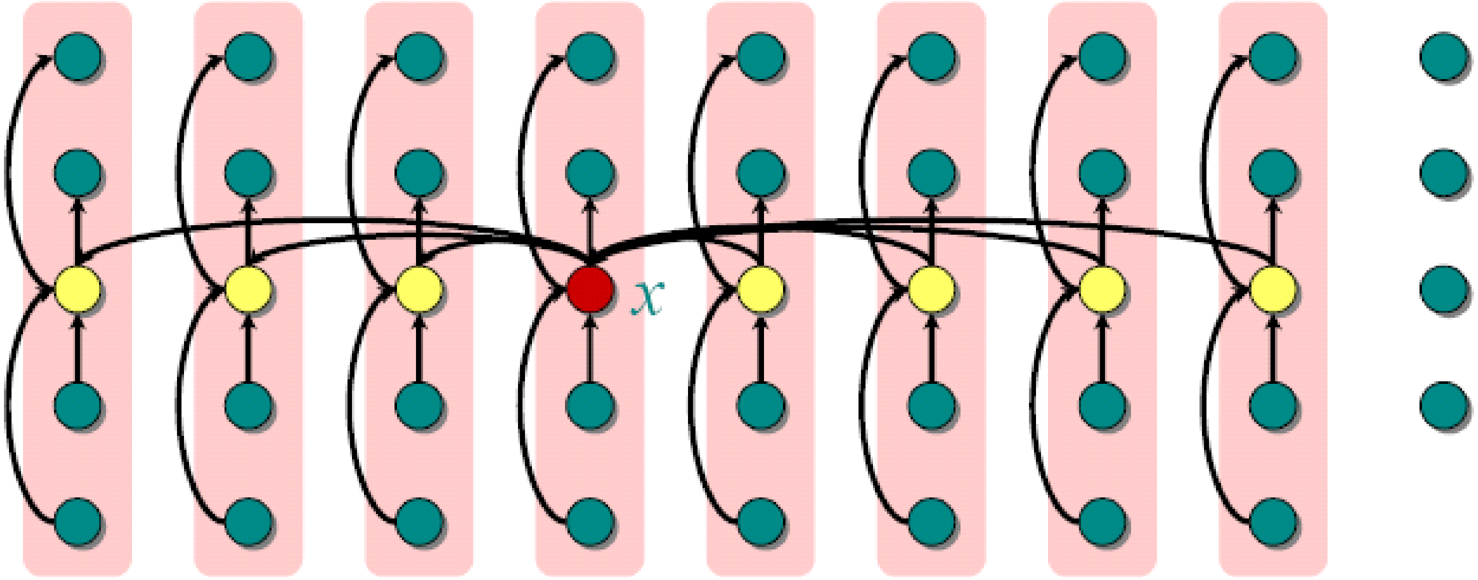
寻找顺序统计量问题

- 示例，寻找$\lfloor n/5 \rfloor+1$组中每一组的中位数



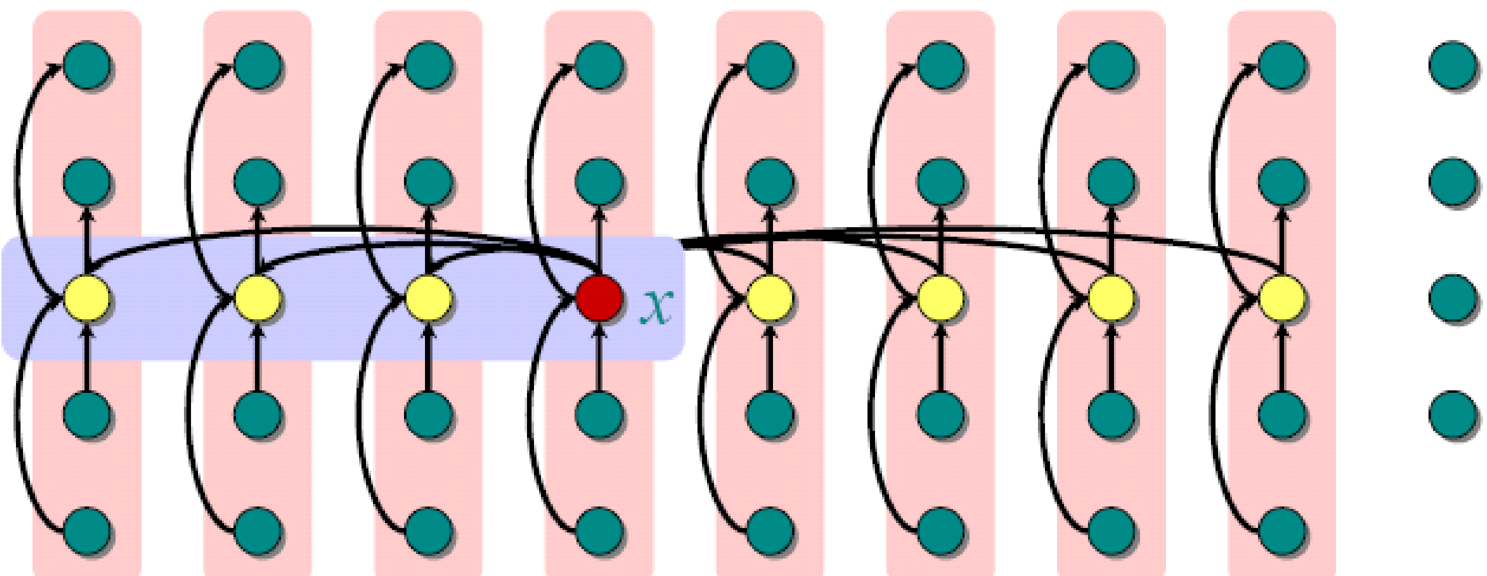
寻找顺序统计量问题

- 示例，递归调用SELECT找出$\lfloor n/5 \rfloor$的中位数x



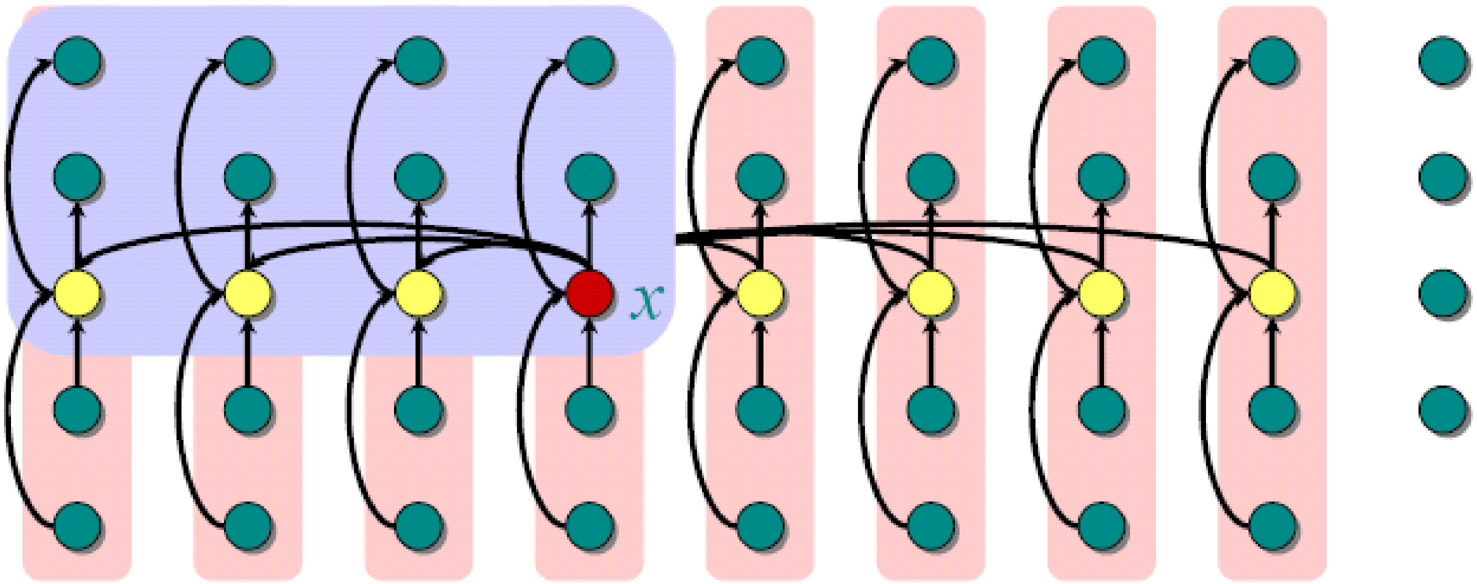
寻找顺序统计量问题

- 至少有一半组（$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$组）的中位数小于或等于x



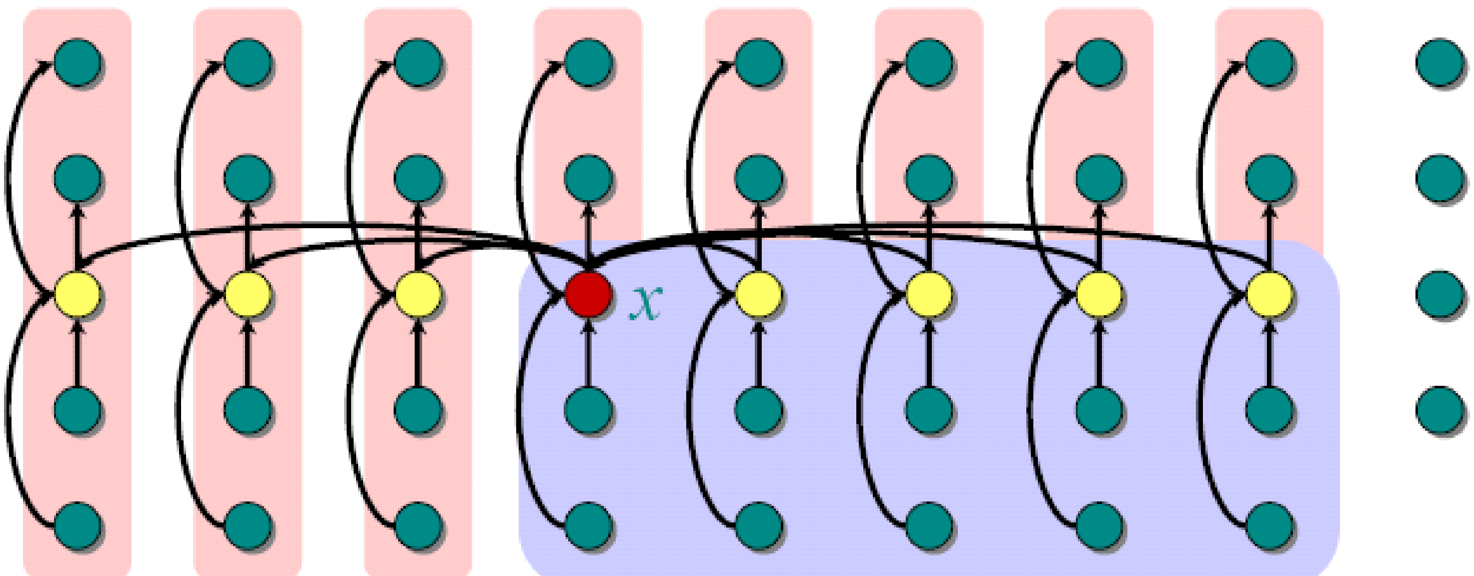
寻找顺序统计量问题

- 至少有一半组（$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$组）的中位数小于或等于x
- 因此，至少$3\lfloor n/10 \rfloor$个元素$\leq$x



寻找顺序统计量问题

- 至少有一半组（$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$组）的中位数小于或等于x
- 因此，至少$3\lfloor n/10 \rfloor$个元素$\leq x$
- 至少$3\lfloor n/10 \rfloor \geq x$



寻找顺序统计量问题

- 分析
  - 1.将输入数组的n个元素分为$\lfloor n/5 \rfloor+1$组；
  - 2.寻找这$\lfloor n/5 \rfloor+1$组中每一组的中位数；
  - 3.对第二步中找出的$\lfloor n/5 \rfloor+1$组中位数，递归调用SELECT以找到其中位数x；
  - 4.PARTITION，按中位数x对输入数组进行划分，x为第k小元素；
  - 5.如果i=k，则返回x；否则，如果i<k，则在低区递归调用SELECT寻找第i小元素；否则，如果i>k，则在高区寻找第(i-k)个最小元素。

分治法小结

- 二分搜索
- 幂乘
- 合并排序
- 快速排序
- Fibonacci数列
- Strassen矩阵乘法
- 最大元、最小元
- 最近点对问题
- 寻找顺序统计量问题