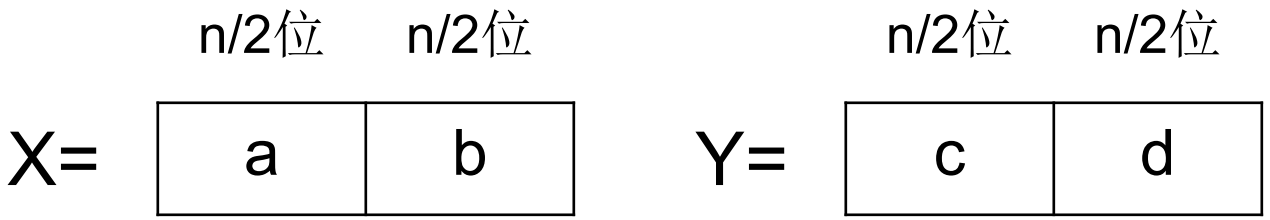
递归与分治

陈长建

计算机科学系

回顾 -大整数乘法

- 请设计一个有效的算法，可以进行两个n位大整数的乘法运算
  - 分治法：



  - $XY = (a*c) \cdot 10^{n} + (a*d+b*c) \cdot 10^{n/2} + b*d$
  - $XY = a*c \cdot 10^{n} + ((a-c)*(b-d)+a*c+b*d) \cdot 10^{n/2} + b*d$

回顾 -大整数乘法

- 更快的方法：
  - 如果将大整数分成更多段，用更复杂的方式把它们组合起来，将有可能得到更优的算法。
  - 最终，这个思想导致了快速傅利叶变换(Fast Fourier Transform)的产生。该方法也可以看作是一个复杂的分治算法，对于大整数乘法，它能在O(nlogn)时间内解决。
- 是否能找到线性时间算法？目前为止还没有结果

回顾-分治法时间复杂度分析

- 递归树方法
  - 二分搜索算法
- 主定理法

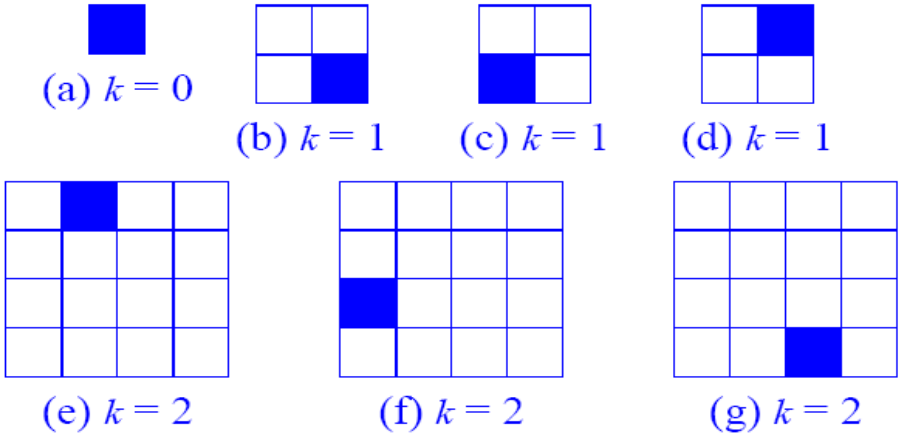
$$T(n)=\begin{cases} c & n=1 \\ aT\left(\frac{n}{b}\right)+cn & n>1 \end{cases}$$

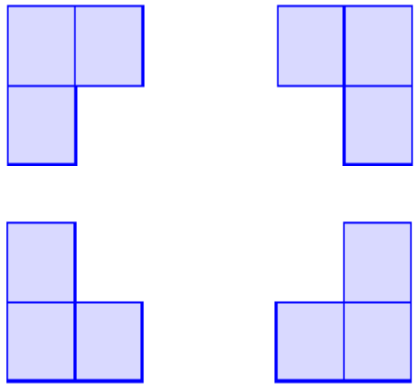
$$T(n)=\begin{cases} \Theta(n) & a<b \\ \Theta(n\log_b n) & a=b \\ \Theta\left(n^{\log_b a}\right) & a>b \end{cases}$$

- 代换法（数学归纳）

回顾 - 棋盘覆盖

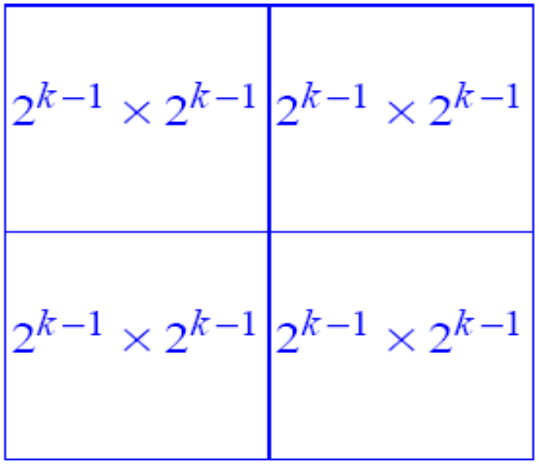
- 在一个$2^k \times 2^k$ 个方格组成的棋盘中，恰有一个方格与其他方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。



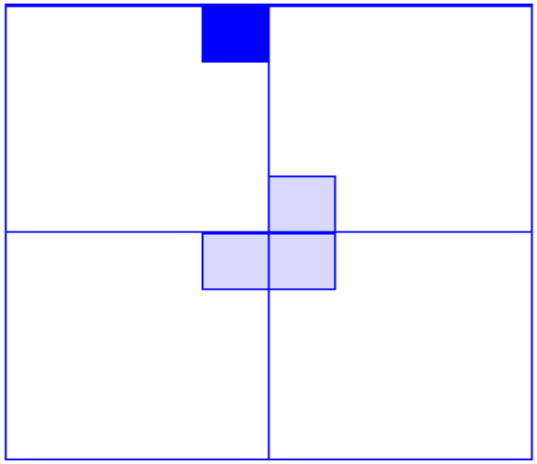


回顾 - 棋盘覆盖

- 当k>0时，将2k×2k棋盘分割为4个2k-1×2k-1 子棋盘(a)所示。 特殊方格必位于4个较小子棋盘之一中，其余3个子棋盘中无特殊方格。为了将这3个无特殊方格的子棋盘转化为特殊棋盘，可以用一个L型骨牌覆盖这3个较小棋盘的会合处，如 (b)所示，从而将原问题转化为4个较小规模的棋盘覆盖问题。递归地使用这种分割，直至棋盘简化为棋盘1×1。



(a) Partitioning


(b) Triomino placed

时间复杂度分析

$$T(k)=\begin{cases}O(1) & k=0\\ 4T(k-1)+O(1) & k>0\end{cases}$$

设M(k)为chessBoard算法在计算覆盖一个2^kx2^k棋盘所需时间：

当k>1时，M(k)=4M(k-1)，　M(0)=1

M(k)=4M(k-1)　　替换M(k-1)=4M(k-2)

=4[4M(k-2)]=4^2 M(k-2)

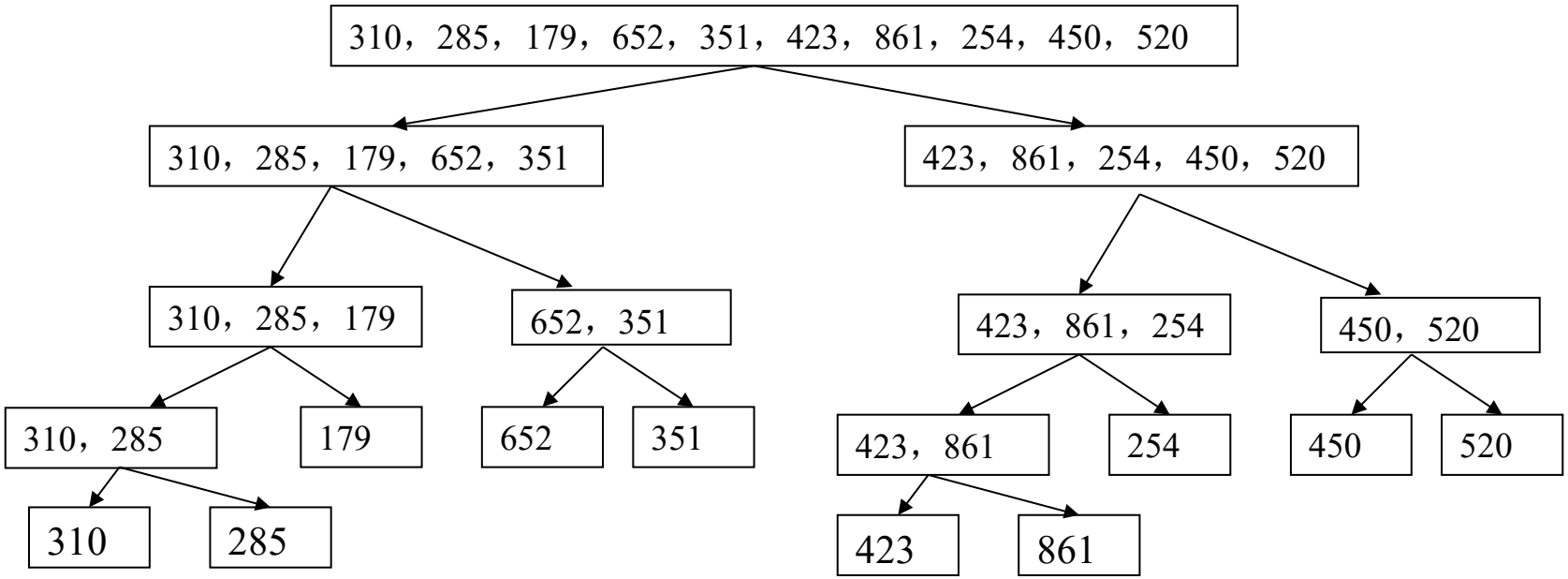
=……

=4^kM(k-k) =4^k

T(K)=O(4^K)

合并排序基本思想

- 将待排序元素分成大小大致相同的2个子集合，分别对2个子集合进行排序，最终将排好序的子集合合并成为所要求的排好序的集合。
  - 例：A=（310，285，179，652，351，423，861，254，450，520）



性能分析

- 1）空间特性
  - 2n+3个空间位置
- 2) 算法**merge**的时间与两数组元素的总数成正
  - (可表示为: cn, n为元素个数, c为某正常数)
- 3) 算法**mergeSort**的时间用递推关系式表示如下:

$$T(n)=\begin{cases} a & n=1,a\text{是常数} \\ 2T(n/2)+cn & n>1,\ c\text{是常数} \end{cases}$$

以比较为基础分类的时间下界

任何以关键字比较为基础的分类算法，其最坏情况下的时间下界都为：$\Omega(n\log n)$

利用二元比较树证明。

假设参加分类的n个关键字A(1),A(2),...,A(n)互异。任意两个关键字的比较必导致A(i)<A(j)或A(i)>A(j)的结果。

以二元比较树描述元素间的比较过程：

- 若A(i)<A(j)，进入下一级的左分支
- 若A(i)>A(j)，进入下一级的右分支

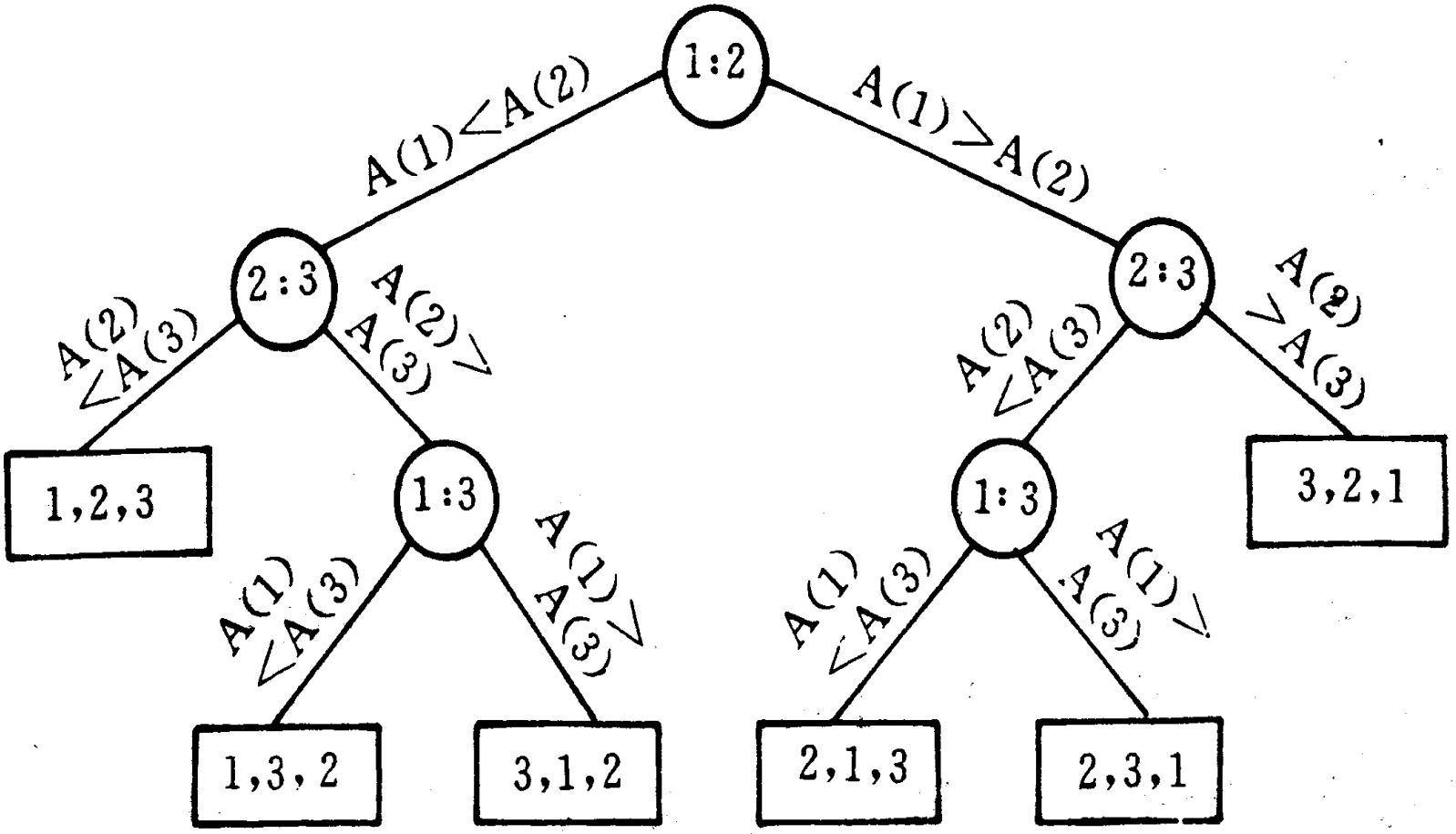


图 2.6　对三个关键字分类的比较树

算法在**外部结点**终止。

从根到某外结点的**路径**代表某个特定输入情况下一种唯一的**分类排序序列**。路径长度表示生成该序列代表的分类表所需要的**比较次数**。而**最长的路径**代表算法在最坏情况下的执行情况，该路径的长度即是算法在最坏情况下所作的比较次数。

故，以比较为基础的分类算法的最坏情况下界等于该算法对应的比较树的**最小高度**。

① 由于n个关键字有n!种可能的排列，所以二元比较树中将有n!个外部结点：每种排列对应于某种特定输入情况下的分类情况，每个外部结点表示一种可能的分类序列。

② 设一棵二元比较树的所有内结点的级数均小于或等于k，则该树中最多有2^k个外结点。

记算法在最坏情况下所作的比较次数为T(n)，则有T(n)=k：生成外结点所代表的分类序列所需的比较次数等于该外结点所在的级数-1；

根据①和②的分析，有：$n! \leqslant 2^{T(n)}$

化简：

当n>1时，有$n! \geqslant n(n-1)(n-2)\cdots(\lceil n/2 \rceil) \geqslant (n/2)^{n/2}$

当n≥4时，有 $T(n) \geqslant (n/2)\log(n/2) \geqslant (n/4)\log n$

故，任何以比较为基础的分类算法的最坏情况的时间下界为：

$\Omega(n\log n)$

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leqslant 2^{T(n)}$$

本章要点

- 算法总体思想
- 递归的概念
- 递归案例分析
- 分治法的概念
- 分治案例分析
  - 二分搜索、大整数法、棋盘覆盖、合并排序、快速排序、线性时间选择 ...

分治-例5 快速排序

- 任何一个基于比较来确定两个元素相对位置的排序算法需要Ω(nlogn)计算时间。如果我们能设计一个需要□ (n1ogn)时间的排序算法，则在渐近的意义上，这个排序算法就是最优的。许多排序算法都是追求这个目标。
- 下面介绍快速排序算法，它在平均情况下需要(nlogn)时间。这个算法是于1962年由C.A.R.Hoare提出的。

基本思想

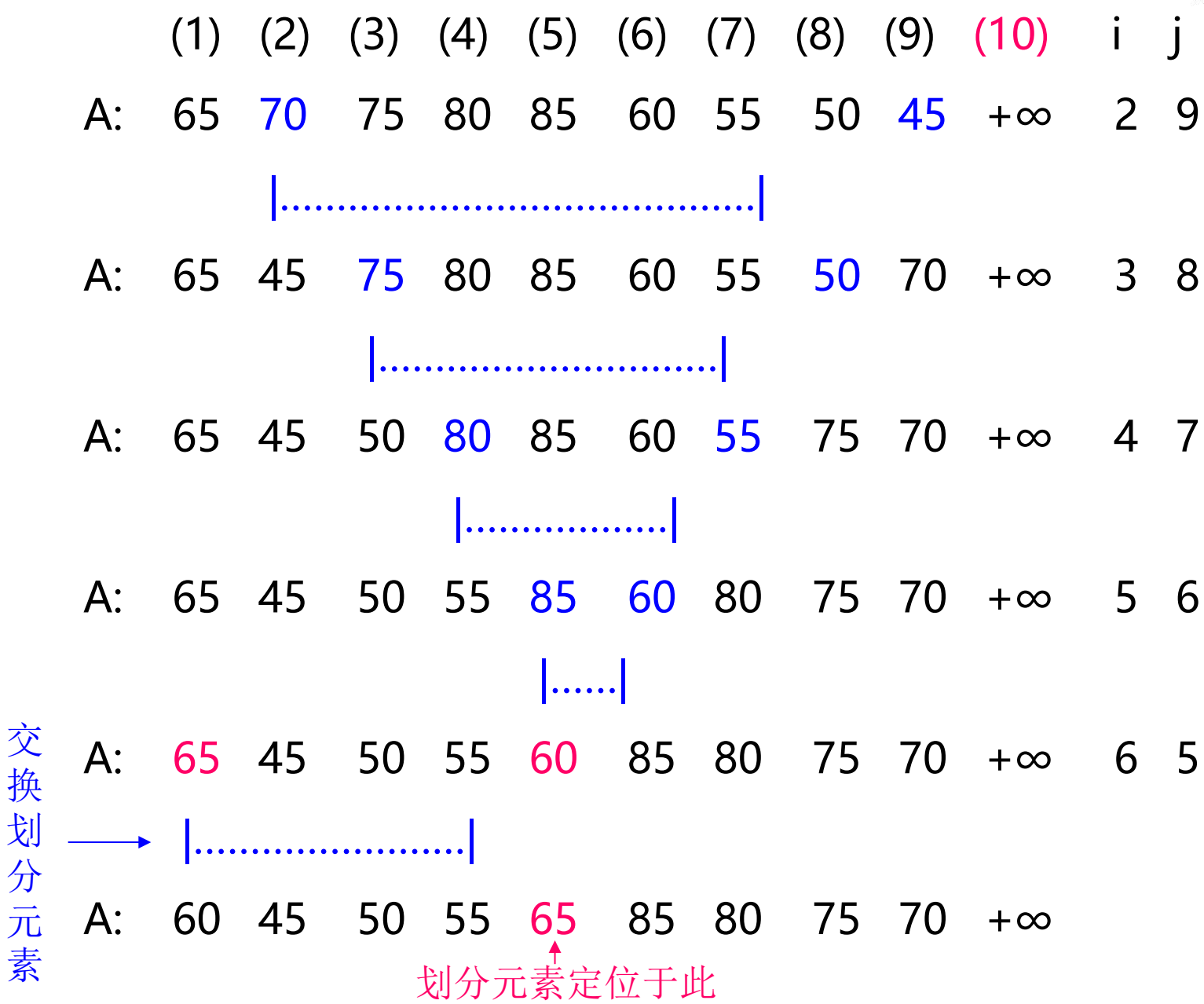
- 快速排序是一种基于划分的排序方法；
- 划分：选取待分类集合A中的某个元素t，按照与t的大小关系重新整理A中元素，使得整理后的序列中所有在t以前出现的元素均小于等于t，而所有出现在t以后的元素均大于等于t。这一元素的整理过程称为划分（Partitioning）。元素t称为划分元素。
- 快速排序：通过反复地对待排序集合进行划分达到排序目的的排序算法。

划分过程的算法描述

用元素 a[p] 划分集合 a[p : r]

```
public static int partition (int p, int r)
{
   int i = p,  j = r + 1;
   Comparable x = a[p];
   // 将>= x的元素交换到左边区域
   // 将<= x的元素交换到右边区域
  while (true) {
      while (a[++i].compareTo(x) < 0); // i由左向右移
      while (a[--j].compareTo(x) > 0);  // j由右向左移
      if (i >= j) break;
      MyMath.swap(a, i, j);
       }
      a[p] = a[j];
      a[j] = x;
    return j;   //划分元素在位置j
  }
```

例子 划分实例


(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) i j
A: 65 70 75 80 85 60 55 50 45 +∞ 2 9
A: 65 45 75 80 85 60 55 50 70 +∞ 3 8
A: 65 45 50 80 85 60 55 75 70 +∞ 4 7
A: 65 45 50 55 85 60 80 75 70 +∞ 5 6
A: 65 45 50 55 60 85 80 75 70 +∞ 6 5
交换划分元素
A: 60 45 50 55 65 85 80 75 70 +∞
划分元素定位于此

分析

- 经过一次“划分”后，实现了对集合元素的调整：其中一个子集合的所有元素均小于等于另外一个子集合的所有元素。
- 按同样的策略对两个子集合进行分类处理。当子集合分类完毕后，整个集合的分类也完成了。这一过程避免了子集合的归并操作。这一分类过程称为快速分类。

快速排序算法实现

通过反复使用划分算法 **partition** 实现对集合元素的排序。

以a[p]为基准元素将a[p:r]划分成3段：

a[p:q-1]，a[q]，a[q+1:r]，

使得：a[p:q-1]中任何元素小于等于a[q]，

a[q+1:r]中任何元素大于等于a[q]，

下标q在划分过程中确定。

```
public  static void qSort(int p, int r)
 {
    if (p<r) {
    int q=partition(p,r);
     qSort (p,q-1); //对左半段排序
     qSort (q+1,r); //对右半段排序
     }
 }
```

快速排序分析

- 记录比较和交换是从两端向中间进行
  - 关键字较大的记录一次就能交换到后面单元
  - 关键字较小的记录一次就能交换到前面单元
  - 记录每次移动的距离较大，因而总比较和移动次数较少。
- 快速排序算法的性能取决于划分的对称性。
  - 可以设计出采用随机选择策略的快速排序算法。
  - 在快速排序算法的每一步中，当数组还没有被划分时，可以在a[p:r]中随机选出一个元素作为划分基准
  - 这样可以使划分基准的选择是随机的，从而可以期望划分是较对称的。

中值快速排序

- 中值快速排序（median-of-three quick sort）是快速排序一种变化，这种算法有更好的平均性能。在快速排序中总是选择a [1]做为支点，而在中值快速排序算法中，取{a[1],a[(1+r)/2],a[r]} 中大小居中的那个元素作为支点。
- 实现中值快速排序算法的一种最简单的方式就是首先选出中值元素并与a[1] 进行交换，然后利用快速排序算法完成排序。

时间复杂度分析

- 统计的对象：元素的比较次数，记为：C(n)
- 两点假设

①参加分类的n个元素各不相同

② **partition**中的划分元素t是随机选取的（针对平均情况的分析）

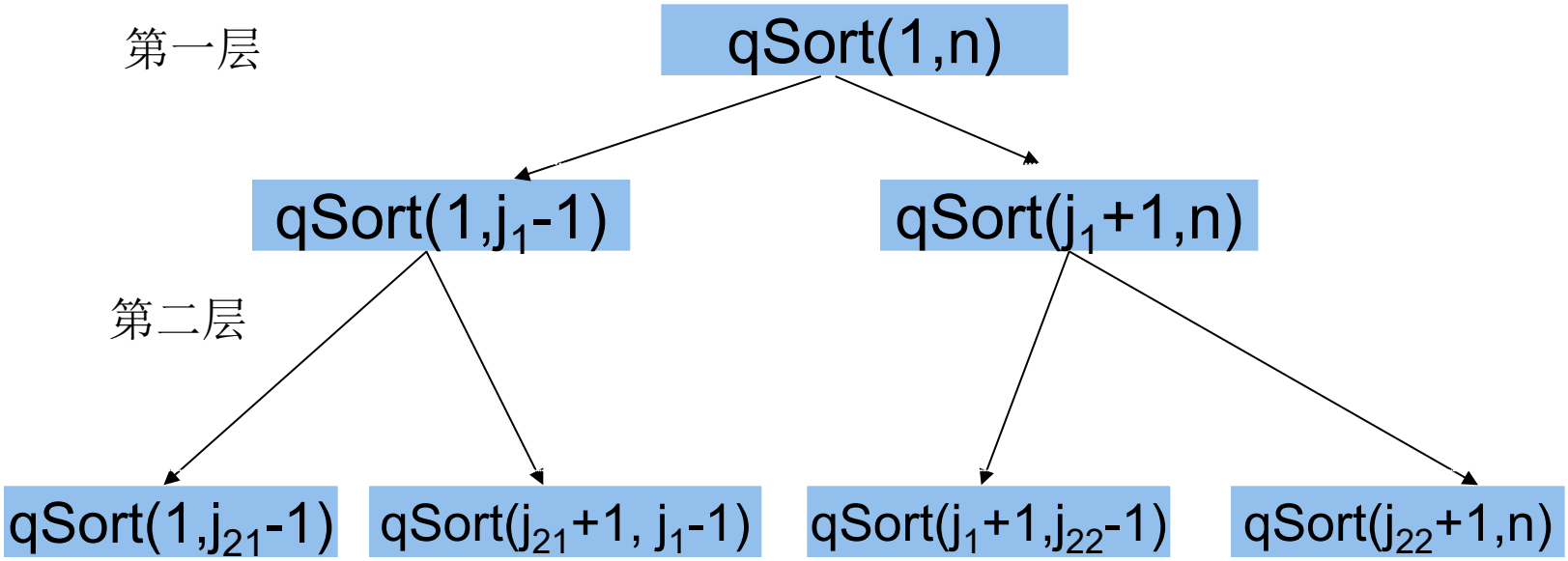
➢ 随机选取划分元素：

在划分区间[p,r]随机生成某一坐标：i← **random**(p,r)；

调换a[p]与a[i]

➢ 作用：将随机指定的划分元素的值依旧调换到a[p]位置。 之后，算法主体不变，仍从a[p]开始执行划分操作。

递归层次



第三层

设在任一级递归调用上，调用partition处理的所有元素总数为r，则，初始时r=n，以后的每级递归上，由于删去了上一级的划分元素，故r比上一级至少1：

理想情况，第一级少1，第二级少2，第三级少4，…；

最坏情况，每次仅减少1（如集合元素已经按照递增或递减顺序排列）

最坏情况分析

- 记最坏情况下的元素比较次数是$C_w(n)$;
- **partition**一次调用中的元素比较数是r – p+1，故每级递归调用上，元素的比较次数等于该级所处理的待分类元素个数。
- 最坏情况下，每级递归调用的元素总数仅比上一级少1，故Cw(n)是r由n到2的累加和。

即：

$$C_w(n) = \sum_{2 \le i \le n} i = O(n^2)$$

最好情况分析

- 记最好情况下的元素比较次数是$C_{best}(n)$；
- 最好情况下，每级递归调用的元素总数仅为上一级的1/2，故$C_{best}(n)$满足递推式：

$$C_{best}(n)=\begin{cases}0 & n=1\\ 2C_{best}(n/2)+n & n>1\end{cases}$$

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

$$=2(2C_{best}(n/2^2)+n/2)+n$$

$$=2^2C_{best}(2^{k-2})+2n=\ldots\ldots$$

$$=2^kC_{best}(1)+kn$$

$$当n=2^k,C_{best}(n)=n\log n$$

平均情况分析

- 记平均情况下的元素比较次数是$C_A(n)$；
  - 平均情况是指集合中的元素以任意一种顺序排列，且任选所有可能的元素作为划分元素进行划分和分类，在这些所有可能的情况下，算法执行性能的平均值。
  - 设调用**partition**(p,r)时，所选取划分元素t 恰好是a[p:r] 中的第i小元素（$1 \leqslant i \leqslant r-p$）的概率相等。则经过一次划分，所留下的待分类的两个子数组恰好是a[p:j-1]和a[j+1:p-1]的概率是：1/(r-p), $p \leqslant j < r$。则有，

$$C_A(n) = n + 1 + \frac{1}{n} \sum_{1 \le k \le n} (C_A(k-1) + C_A(n-k))$$

  - n+1是partition第一次调用时所需的元素比较次数。
  - $C_A(0) = C_A(1) = 0$

$$C_A(n) = n+1+\frac{1}{n}\sum_{1\le k\le n}(C_A(k-1)+C_A(n-k))$$

平均情况分析

- 化简上式可得：

$C_A(n)/(n+1) = C_A(n-1)/n+2/(n+1)$

$= C_A(n-2)/(n-1)+2/n+2/(n+1)$

$= C_A(n-3)/(n-2)+2/(n-1)+2/n+2/(n+1)$

…

$= C_A(1)/2+ 2\sum_{3\le k\le n+1} 1/k$

由于 $\sum_{3\le k\le n+1} 1/k \le \int_2^{n+1} \frac{dx}{x} < \log_e(n+1)$

所以得，$C_A(n)<2(n+1)\log_e(n+1) = O(n\log n)$

各种排序算法的比较

方法	最坏复杂性	平均复杂性
冒泡排序	n^2	n^2
计数排序	n^2	n^2
插入排序	n^2	n^2
选择排序	n^2	n^2
堆排序	nlogn	nlogn
归并排序	nlogn	nlogn
快速排序	n^2	nlogn

分治-例6 线性时间选择

- 问题描述
  - 给出含有n个元素表A[0:n-1]，要求确定其中的第k小元素。
- 设计思路
  - 选择问题可在O(nlogn)时间内解决，方法是首先对这n个元素进行排序(如使用堆排序或归并排序)，然后取出A[k-1]中的元素。若使用快速排序，可以获得更好的平均性能。尽管该算法有一个比较差的渐近复杂性$O(n^2)$。

线性时间选择

- 利用**partition** 函数。如果划分元素v测定在A(j)的位置上，则有j-1个元素小于或等于A(j)，且有n-j个元素大于或等于A(j)。此时，
  - 若k=j，则A(j)即是第k小元素；否则，
  - 若k<j，则第k小元素将出现在A(0:j-1)中；
  - 若k>j，则第k小元素将出现在A(j+1:n-1)中。

算法描述

```
public static void randomizedSelect (int p, int r,int k)
 {
   if( p==r ) return a[p];
   int i = randomizedPartition(p,r),
        j=i-p+1;
   if( k<=j )  return randomizedSelect (p, i, k);//在左半段
   else return randomizedSelect (i+1, r, k-j); //在右半段
}
```

算法分析

- 两点假设

  ① A中的元素互异

  ② 随机选取划分元素，且选择A中任一元素作为划分元素的概率相同

- 分析
  - **randomizedPartition**(p,r),所需的元素比较次数是 O(r-p+1)。
  - 在执行一次**randomizedPartition**(p,r)后，或者找到第k小元素，或者将在缩小的子集 A(p, i)或A(i+1,r)中继续查找。缩小的子集的元素数将至少比上一次划分的元素数少1。

1）最坏情况

randomizedSelect的最坏情况时间是$O(n^2)$

当A中的元素已经按照递增的顺序排列， 且k=n 此时，需要n次调用**randomizedPartition**过程，且每次返回的元素位置是子集中的第一个元素，子集合的元素数一次仅减少 1

则,n次调用的时间总量是

$$O(\sum_{1}^{n}(i+1)) = O(n^2)$$

2) 平均情况

设$T_A^k(n)$是找A(1:n)中第k小元素的平均时间。$T_A(n)$是**randomizedSelect**的平均计算时间，则有

$$T_A(n)=\frac{1}{n}\sum_{1\le k\le n}T_A^k(n)$$

并定义 $R(n)=\max_k\{T_A^k(n)\}$

则有：$T_A(n)\le R(n)$。

定理2.4 ***randomizedSelect*****的平均计算时间**$T_A(n)$**是**$O(n)$

证明：

执行**randomizedPartition时**，比较语句的执行时间是O(n)。在随机等概率选择划分元素时，首次调用**randomizedPartition**中划分元素v刚好是A中第i小元素的概率为1/n，$1\le i\le n$。

则，存在正常数c，$c>0$,有，

$$T_A^k(n)\le cn+\frac{1}{n}\left(\sum_{1\le i<k}T_A^{k-i}(n-i)+\sum_{k<i\le n}T_A^k(i-1)\right)\quad n\ge 2$$

且有，$R(n)\le cn+\frac{1}{n}\max_k\left\{\sum_{1\le i<k}R(n-i)+\sum_{k<i\le n}R(i-1)\right\}$

$$=cn+\frac{1}{n}\max_k\left\{\sum_{n-k+1}^{n-1}R(i)+\sum_{k}^{n-1}R(i)\right\}\quad n\geqslant 2$$

令c≥R(1)。利用数学归纳法证明，对所有n≥2，有R(n)≤4cn.

① 当n=2时，由上式得：
$$R(n) \le 2c + \frac{1}{2}\max\{R(1), R(1)\} \le 2.5c < 4cn$$

② 假设对所有的n,2≤n<m，有R(n)≤4cn

③ 当n = m时，有，
$$R(n) \le cm + \frac{1}{m}\max_{k}\{\sum_{m-k+1}^{m-1} R(i) + \sum_{k}^{m-1} R(i)\}$$

由于R(n)是n的非降函数，故在当m为偶数而k=m/2，或当m为奇数而k=(m+1)/2时，$\sum_{n-k+1}^{n-1} R(i) + \sum_{k}^{n-1} R(i)$ 取得极大值。因此，

若m为偶数，则
$$R(m) \le cm + \frac{2}{m}\sum_{m/2}^{m-1} R(i) \le cm + \frac{8c}{m}\sum_{m/2}^{m-1} i < 4cm$$

若m为奇数，则
$$R(m) \le cm + \frac{2}{m}\sum_{(m+1)/2}^{m-1} R(i) \le cm + \frac{8c}{m}\sum_{(m+1)/2}^{m-1} i < 4cm$$

由于$T_A(n) \le R(n)$，所以$T_A(n) \le 4cn$。故，$T_A(n) = O(n)$

最坏情况是O(n)的选择算法

1）采用两次取中间值的规则精心选取划分元素

首先，将参加划分的n个元素分成 $\lfloor n/r \rfloor$ 组，每组有r个元素($r \geq 1$)。（多余的 $n - r\lfloor n/r \rfloor$ 个元素忽略不计）

然后，对这 $\lfloor n/r \rfloor$ 组每组的r个元素进行分类并找出其中间元素 $m_i, 1 \leq i \leq \lfloor n/r \rfloor$，共得 $\lfloor n/r \rfloor$ 个中间值。

之后，对这 $\lfloor n/r \rfloor$ 个中间值分类，并找出其中间值mm。

将mm作为划分元素执行划分。

2）算法描述

算法　使用二次取中规则选择算法 SELECT2(A,k,n)

//在集合A中找第k小元素，使用两次取中规则//

① 若n≤r，则采用插入法直接对A分类并返回第k小元素

② 把A分成大小为r的 $\lfloor n/r \rfloor$ 个子集合，忽略多余的元素

③ 设M={$m_1,m_2,\ldots m_{\lfloor n/r \rfloor}$}是 $\lfloor n/r \rfloor$子集合的中间值集合

④ v←SELECT2(M,$\lceil \lfloor n/r \rfloor / 2 \rceil$, $\lfloor n/r \rfloor$)

⑤ j←PARTITION(A，v) //v作为划分元素，划分后j等于划分元素所在位置的下标//

⑥ case

:k=j: return(v)

:k<j: 设S是A(1:j-1)中元素的集合

return(SELECT2(S,k,j-1))

:else: 设R是A(j+1:n)中元素的集合

return(SELECT2(R,k-j,n-j))

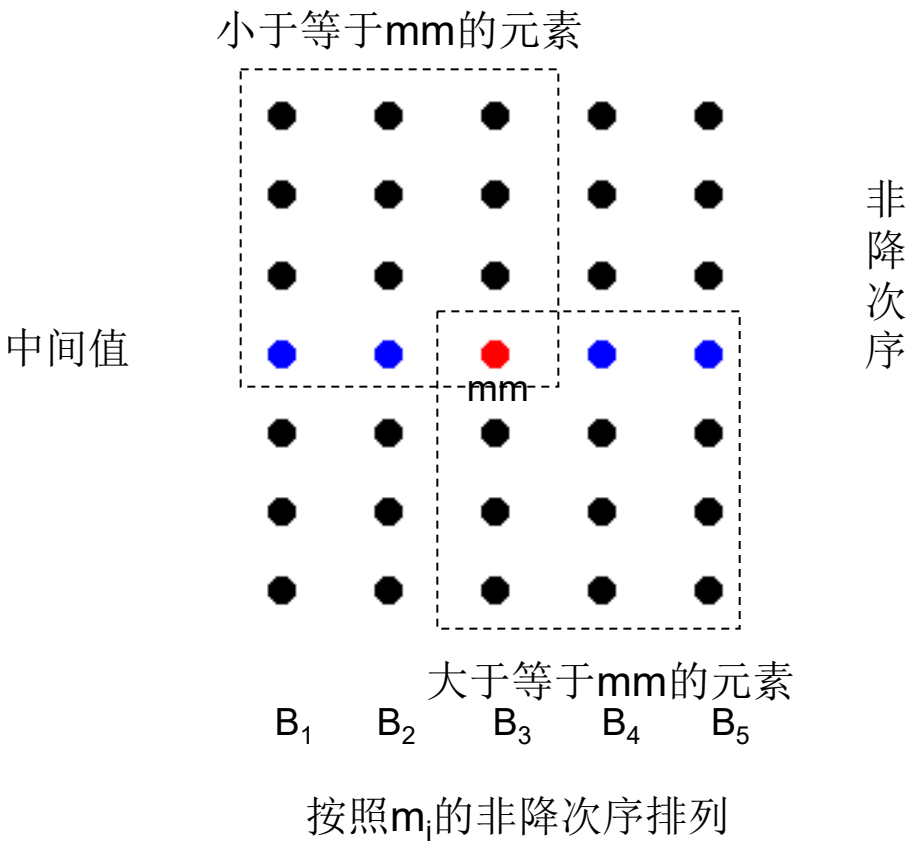
endcase

end SELECT2

- 例：设 n=35，r=7。

- 分为n/r = 5个元素组：B1，B2，B3，B4，B5；每组有7个元素。
- B1-B5按照各组的m_i的非增次序排列。
- mm = m_i的中间值，1≤i≤5

由图所示有：



由于r个元素的中间值是第⌈r/2⌉小元素。则，

- 至少有⌈⌊n/r⌋/2⌉个m_i小于或等于mm；
- 至少有 ⌊n/r⌋-⌈⌊n/r⌋/2⌉+1 ≥⌈⌊n/r⌋/2⌉个mi大于或等于mm。

则，至少有⌈r/2⌉⌈⌊n/r⌋/2⌉个元素小于或等于（或大于或等于）mm。

- 当r=5，则使用两次取中间值规则来选择v=mm，可推出，
- 至少有 1.5⌊n/5⌋个元素小于或等于选择元素v。
- 至多有 n-1.5⌊n/5⌋≤ 0.7n+1.2 个元素大于v。
- 至多有 0.7n+1.2个元素小于v。

故，这样的v可近似平均地划分A中的n个元素。

记T(n)是SELECT2所需的最坏情况时间

对特定的r分析SELECT2:选取r=5。

- 假定A中的元素各不相同，则有

① 若n≤r，则采用插入法直接对A分类并返回第k小元素 →O(1)

② 把A分成大小为r的 $\lfloor n/r \rfloor$ 个子集合，忽略多余的元素 →O(n)

③ 设M={m1,m2,…m $\lfloor n/r \rfloor$ }是 $\lfloor n/r \rfloor$ 个子集合的中间值集合 →O(n)

④ v←SELECT2(M, $\lceil \lfloor n/r \rfloor / 2 \rceil$ $\lfloor n/r \rfloor$) →T(n/5)

⑤ j←PARTITION(A，v) →O(n)

⑥ case →T(3n/4),n≥24

:k=j: return(v)

:k<j: 设S是A(1:j-1)中元素的集合; return(SELECT2(S,k,j-1))

:else: 设R是A(j+1:n)中元素的集合; return(SELECT2(R,k-j,n-j))

endcase

故有，

$$T(n) = \begin{cases} cn & n < 24, \\ T(n/5)+T(3n/4)+cn & n \geq 24 \end{cases}$$

用归纳法可证：

$T(n) \leq 20cn$

即，$T(n) = O(n)$

- 当A中的元素不尽相同

步骤⑤经PARTITION调用所产生的S和R两个子集合中可能存在一些元素等于当前的划分元素v,可能导致|S|或|R|大于0.7n+1.2。

此时上述处理 作一下改进：

方法一：将A集合分成3个子集合U,S和R，其中U是有A中所有与v相同的元素组成，S是由A中所有比v小的元素组成，R则是A中所有比v大的元素组成。

同时步骤⑥更改：

case

:|S|≥k:return(SELECT2(S,k,|S|)

:|S|+|U|≥k:return(v)

:else: return(SELECT2(R,k-|S|-|U|,|R|))

endcase

$T(n) = O(n)$

从而保证 |S|+|R|≤0.7n+1.2成立，故关于T(n)的分析仍然成立。

方法二：选取其他的r值进行计算

特例：当r=5,且A中的元素不尽相同。假设其中有0.7n+1.2个元素比v小而其余的元素都等于v的情况。

则，经过PARTITION，在这些等于v的元素中至多有一半可能在S中，故$|S|\leq 0.7n+1.2+(0.3n-1.2)/2=0.85n+0.6$

同理，$|R|\leq 0.85n+0.6$

可得，步骤④和⑥此时所处理的元素总数将是

$$1.05n+0.6>n$$

不再是线性关系。故有$T(n)\neq O(n)$

改进：

取r=9。经计算可得，此时将有$2.5\lfloor n/9 \rfloor$个元素小于或等于v，同时至少有同样多的元素大于或等于v。

则当n≥90时，|S|和|R|都至多为

$$n-2.5\lfloor n/9\rfloor+\tfrac{1}{2}(2.5\lfloor n/9\rfloor)=n-1.25\lfloor n/9\rfloor\leq 31n/36+1.25\leq 63n/72$$

故有，

$$T(n) = \begin{cases} c_1 n & n < 90 \\ T(n/9)+T(63n/72)+c_1 n & n \geq 90 \end{cases}$$

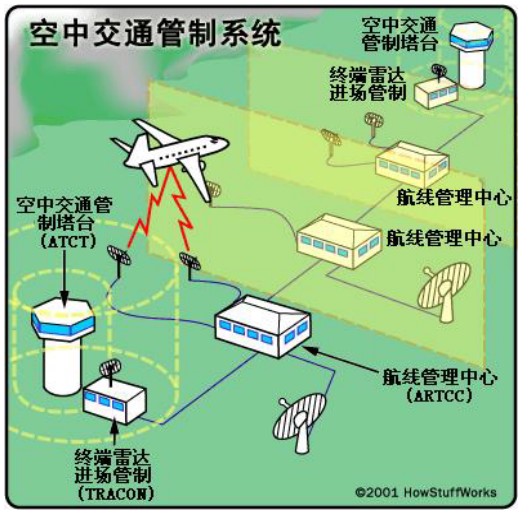
用归纳法可证：

$$T(n) \leq 72 c_1 n$$

即，$T(n) = O(n)$

分治-例7 最接近点对问题

- 给定平面上n个点的集合S，找其中的一对点，使得在n个点组成的所有点对中，该点对间距离最小。
  - 应用：空中交通管制



直接求解方法

- 分别计算每一对点之间的距离，然后找出距离最小的那一对。
- 时间复杂度

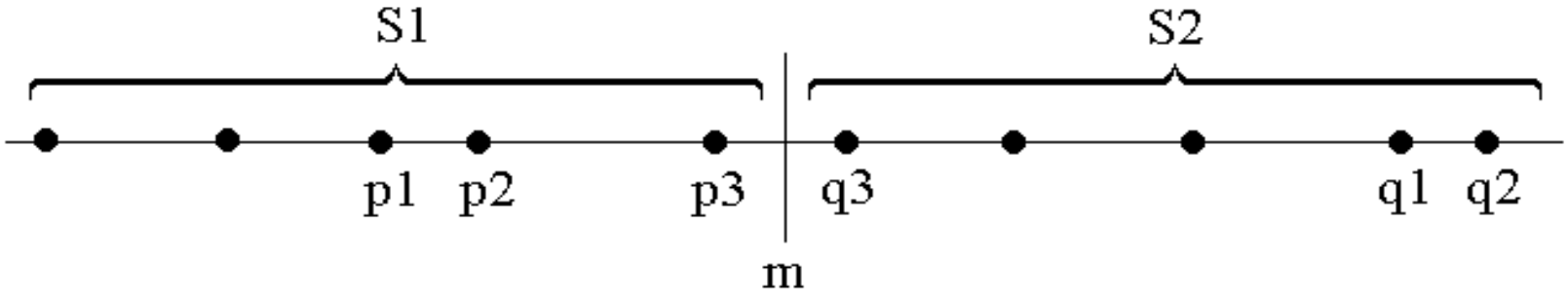
$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2\sum_{i=1}^{n-1} n - i$$

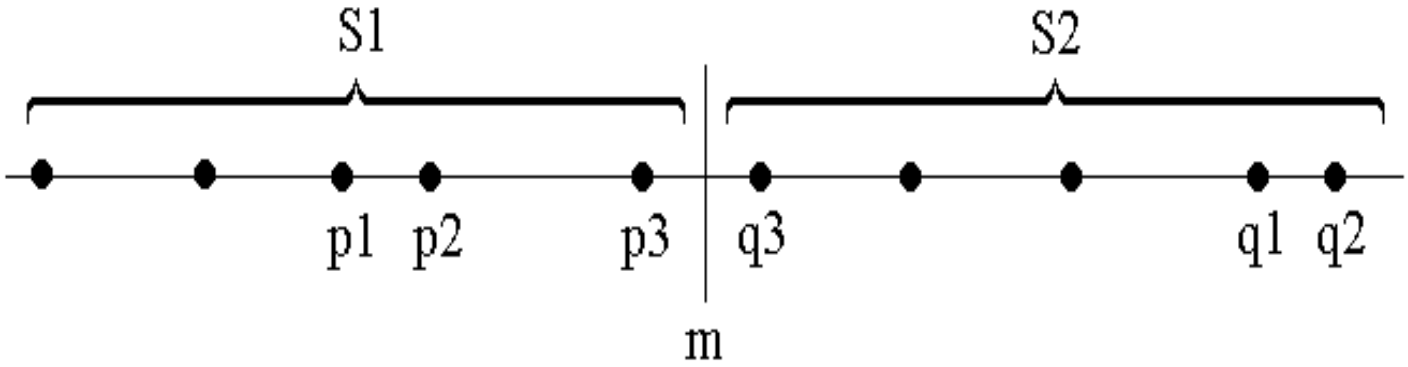
$$= 2[(n-1) + (n-2) + \ldots + 1]$$

$$= (n-1)n \in \Theta(n^2)$$

- 为了使问题易于理解和分析，先来考虑**一维**的情形。此时，S中的n个点退化为x轴上的n个实数 x1,x2,...,xn。最接近点对即为这n个实数中相差最小的2个实数。

- 假设我们用x轴上某个点m将S划分为2个子集S1和S2，基于**平衡子问题**的思想，用S中各点坐标的中位数来作分割点。
- 递归地在S1和S2上找出其最接近点对{p1,p2}和{q1,q2}，并设**d=min{|p1-p2|,|q1-q2|}**，S中的最接近点对或者是{p1,p2}，或者是{q1,q2}，或者是某个{p3,q3}，其中p3∈S1且q3∈S2。
- **能否在线性时间内找到p3,q3?**





能否在线性时间内找到p3,q3?

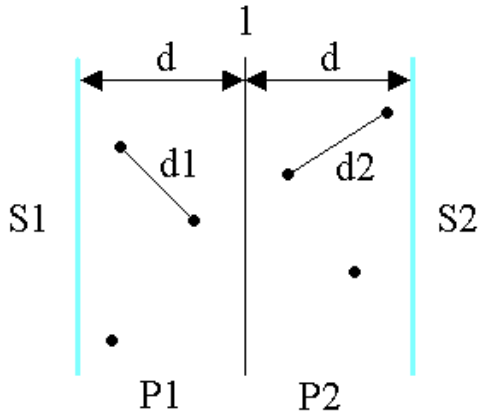
- 如果S的最接近点对是{p3,q3}，即|p3-q3|<d，则p3和q3两者与m的距离不超过d，即p3∈(m-d,m]，q3∈(m,m+d]。
- 由于在S1中，每个长度为d的半闭区间至多包含一个点（否则必有两点距离小于d），并且m是S1和S2的分割点，因此(m-d,m]中至多包含S中的一个点。由图可以看出，如果(m-d,m]中有S中的点，则此点就是S1中最大点。
- 因此，我们用线性时间就能找到区间(m-d,m]和(m,m+d]中所有点，即p3和q3。从而我们用线性时间就可以将S1的解和S2的解合并成为S的解。

```
public static double cpair1(S)
{     n=|S|;
      if (n < 2) return  ∞;
      m=S中各点坐标的中位数；；
     d1=cpair1(S1); // S1={x∈S|x<=m}, 构造S1和S2
     d2=cpair1(S2); // S2={x∈S|x>m}, 构造S2
     p=max(S1);
     q=min(S2);
     d=min(d1,d2,q-p);
```

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

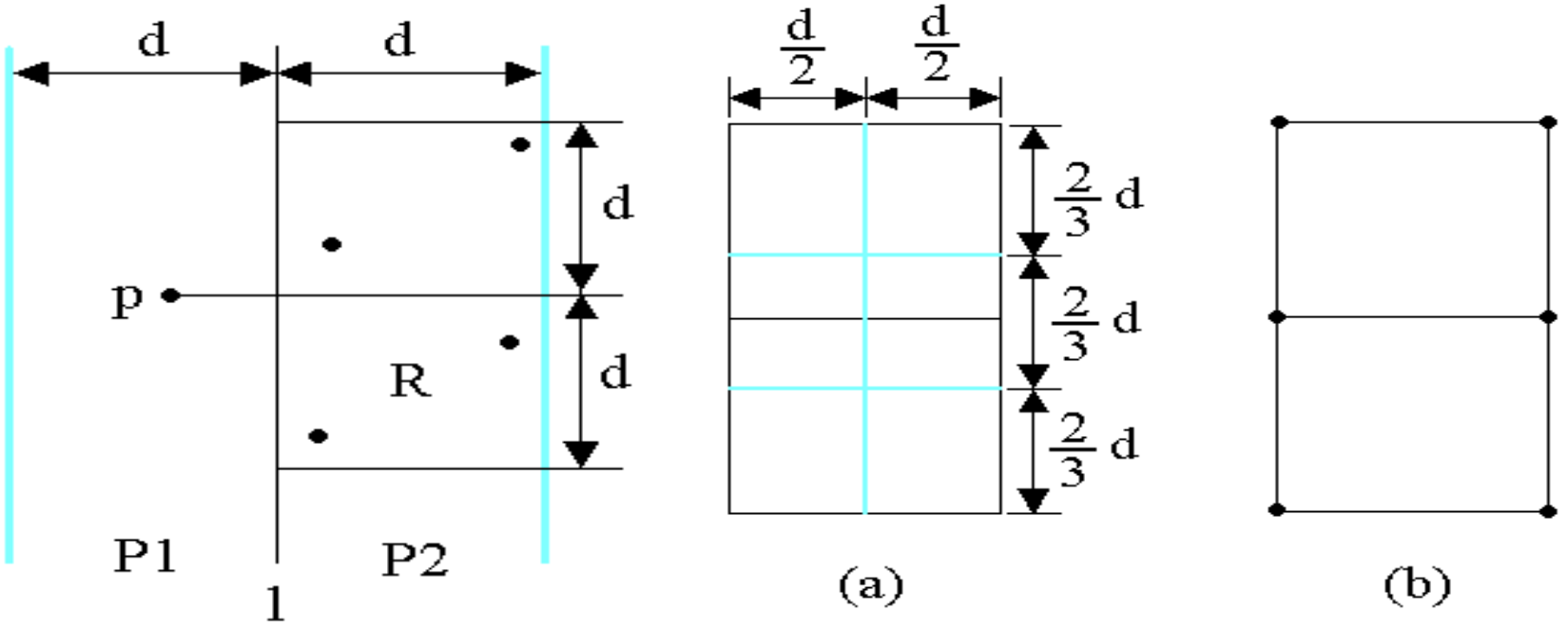
二维情形

- 下面来考虑二维的情形。
- 选取一垂直线 l: x = m 来作为分割直线。其中m为S中各点x坐标的中位数。由此将S分割为S1和S2。
- 递归地在S1和S2上找出其最小距离d1和d2，并设d = min{d1,d2}，S中的最接近点对或者是d，或者是某个{p,q}，其中p∈P1 且 q∈P2。
- 能否在线性时间内找到p,q?



二维情形

- 考虑P1中任意一点p，它若与P2中的点q构成最接近点对的候选者，则必有distance(p，q)＜d。满足这个条件的P2中的点一定落在一个d×2d的矩形R中
- 由d的意义可知，P2 中任何 2 个 S 中的点的距离都不小于d。由此可以推出矩形R中最多只有6个S中的点。
- 因此，在分治法的合并步骤中最多只需要计算 n/2×6 个点对。



二维情形

证明:将矩形R的长为2d的边3等分，将它的长为d的边2等分，由此导出6个(d/2)×(2d/3)的矩形。若矩形R中有多于6个S中的点，则由鸽舍原理易知至少有一个(d/2)×(2d/3)的小矩形中有2个以上S中的点。设u，v是位于同一小矩形中的2个点，则:

$$(x(u)-x(v))^2+(y(u)-y(v))^2 \le (d/2)^2+(2d/3)^2$$

$$\le \frac{25}{36}d^2$$

distance(u,v)<d。这与d的意义相矛盾。

二维情形

- 要检查哪6个点?

  将p和P2中所有S2的点投影到垂直线 l上。由于能与p点一起构成最接近点对候选者的S2中点一定在矩形R中，所以它们在直线l上的投影点距p在l上投影点的距离小于d，这种投影点最多只有6个。

- 因此，若将P1和P2中所有S中点按其y坐标排好序，则对P1中所有点，对排好序的点列作一次扫描，就可以找出所有最接近点对的候选者。对P1中每一点最多只要检查P2中排好序的相继6个点。

```
double cpair2(S)
{
    n=|S|;
    if (n < 2) return ;
1、  m=S中各点x间坐标的
    中位数;构造S1和S2;
    S1={p∈S|x(p)<=m},
    S2={p∈S|x(p)>m}
2、  d1=cpair2(S1);
    d2=cpair2(S2);
3、  dm=min(d1,d2);
```

4、设P1 是S1中距垂直分割线 l 的距离在dm 之内的所有点组成的集合；

P2 是S2 中距分割线 l 的距离在dm 之内所有点组成的集合；

将P1 和P2 中点依其 y 坐标值排序；

并设X 和Y 是相应的已排好序的点列；

5、通过扫描 X 以及对于 X 中每个点检查Y中与其距离在dm 之内的所有点(最多6个)可以完成合并；

当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动；

设dl 是按这种扫描方式找到的点对间的最小距离；

```
6、  d=min(dm,dl);
    return d;
}
```

2.10 最接近点对问题

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

复杂度分析

n≥4,

T(n)=2T(n/2)+O(n)=2[2T(n/4)+O(n/2)]+O(n)

$=2^2T(n/2^2)+2O(n)=\cdots\cdots$

n=2^k $=2^k+kO(n)$

k ≥2 **T(n)=O(nlogn)**

分治-例8 循环赛日程表

- 有n=2k个运动员。设计一个满足以下要求的比赛日程表：
  - (1)每个选手必须与其他n-1个选手各赛一次；
  - (2)每个选手一天只能赛一次；
  - (3)循环赛一共进行n-1天。

思想

- 按分治策略，将所有的选手分为两半
  - n个选手的比赛日程表就可以通过为n/2个选手设计的比赛日程表来决定。
- 递归地对选手进行分割，直到只剩下2个选手时，

只要让这2个选手进行比赛就可以了。

示例

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1