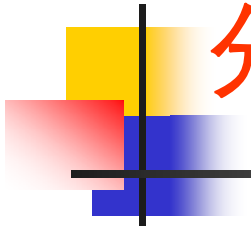
算法设计
——分治法

陈卫东

chenwd@scnu.edu.cn

华南师范大学计算机学院

2021-10



分治法

- 分治法的基本模式

  4.1 几个简单例子

- 分治法的应用

  4.2 归并排序

  4.3 计数逆序

  4.4 快速排序

  4.5 大整数的乘法

  4.6 最近点对问题

分治法的基本模式

- 找假币问题

  8枚硬币中有一枚假币，假币要轻一些。如何在天平上称最少的次数找出这枚假币?

- 方法1

——称3次

- 方法2

——称2次

分治法的基本模式

- 最大最小元素问题
  在序列$A[1..n]$中找最小元素x和最大元素y。

- 直接算法

```
1. x ← A[1]; y ← A[1]
2. for i ← 2 to n
3.     if A[i] < x then x ← A[i]
4.     if A[i] > y then y ← A[i]
5. end for
6. return (x, y)
```

- 算法分析：比较次数为 $2n-2$

思考： **1.** 如何改进算法使其最坏情况比较次数为$2n-2$，平均情况为$3n/2-2$?
2. 能否改进算法使得最坏情况比较次数为$3n/2-2$?

分治法的基本模式

- 分治算法思想

  在序列A[1……n]中找最小和最大元素x,y

  [if (n<=2) 直接比较得结果;

  else

  在序列A[1……mid]中找最小和最大元素x1,y1;

  在序列A[mid+1…..n]中找最小和最大元素x2,y2;

  x=min{x1,x2};

  y=max{y1,y2};]

- **Algorithm** MINMAX
- (最坏情况)元素的比较次数：$3n/2-2$

算法 MINMAX

输入：n 个整数元素的数组 $A[1\cdots n]$，n 为 2 的幂。

输出：(x, y)，A 中的最大元素和最小元素。

1. $minmax(1, n)$

过程 $minmax(low, high)$

1. **if** $high - low = 1$ **then**
2. **if** $A[low] < A[high]$ **then return** $(A[low], A[high])$
3. **else return** $(A[high], A[low])$
4. **end if**
5. **else**
6. $mid \leftarrow \lfloor (low + high)/2 \rfloor$
7. $(x_1, y_1) \leftarrow minmax(low, mid)$
8. $(x_2, y_2) \leftarrow minmax(mid + 1, high)$
9. $x \leftarrow min\{x_1, x_2\}$
10. $y \leftarrow max\{y_1, y_2\}$
11. **return** (x, y)
12. **end if**

- 算法分析
  - 当元素个数n是2的整数次幂时，（最坏情况）算法比较次数为 $3n/2-2$ 。

$$C(n)=\begin{cases}1 & \text{若 } n=2\\ 2C(n/2)+2 & \text{若 } n>2\end{cases}$$

按以下方式用展开来求解这个递推式(令 $k=\log n$)

$$\begin{aligned}C(n) &= 2C(n/2)+2\\ &= 2(2C(n/4)+2)+2\\ &= 4C(n/4)+4+2\\ &= 4(2C(n/8)+2)+4+2\\ &= 8C(n/8)+8+4+2\\ &\vdots\\ &= 2^{k-1}C(n/2^{k-1})+2^{k-1}+2^{k-2}+\cdots+2^2+2\\ &= 2^{k-1}C(2)+\sum_{j=1}^{k-1}2^j\\ &= (n/2)+2^k-2\\ &= 3n/2-2\end{aligned}$$

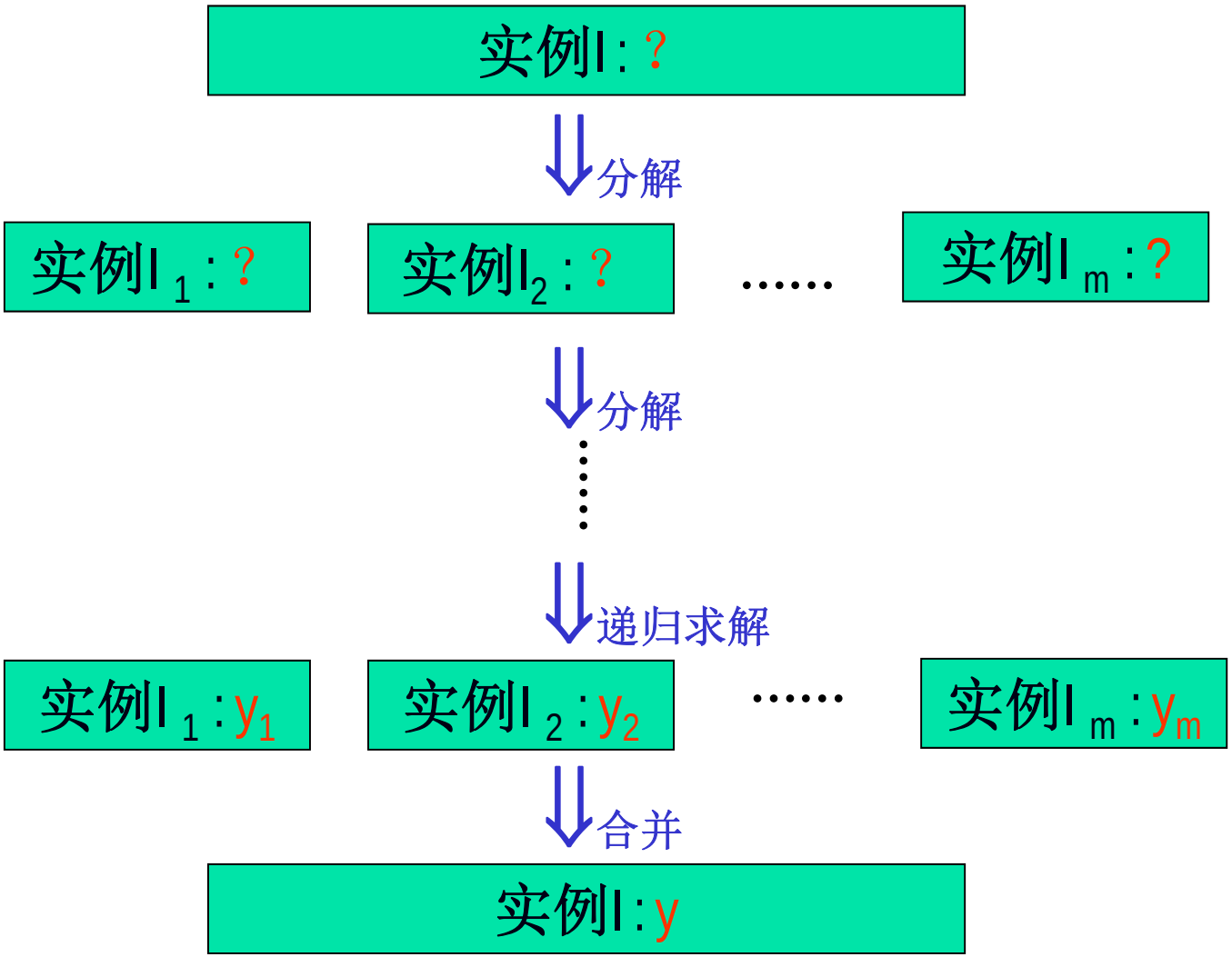
分治法的基本模式

- 关键步骤
  - 划分
  - 处理子问题
  - 合并

分治法的基本模式

- 分而治之法算法的模式
  - 若问题实例I大小“较小”，则直接求解并返回结果；否则进行下一步；
  - （Divide）将I分成m个大小基本相同的子实例$I_1, I_2, \ldots, I_m$；
  - （Conquer）对于每个子实例递归地求解得到它们的解；
  - （Combine）将子实例的解合并得到实例I的解

分治法的直观描述



归并排序

- 问题描述

  将序列A[1..n]中元素按照升序排序。

- 算法思想

  将序列A[1……n]中元素排序

  [if 序列长度n>1

  将序列A[1……mid]中元素排序;

  将序列A[mid+1…..n]中元素排序;

  归并两个有序序列A[1……mid]和A[mid+1…..n];

  ]

- **Algorithm** MERGESORT
- 时间复杂度： $\Theta(n\log n)$

- 算法描述

算法　　MERGESORT

输入： n 个元素的数组 $A[1\cdots n]$。

输出： 按非降序排列的数组 $A[1\cdots n]$。

1. *mergesort*$(A, 1, n)$

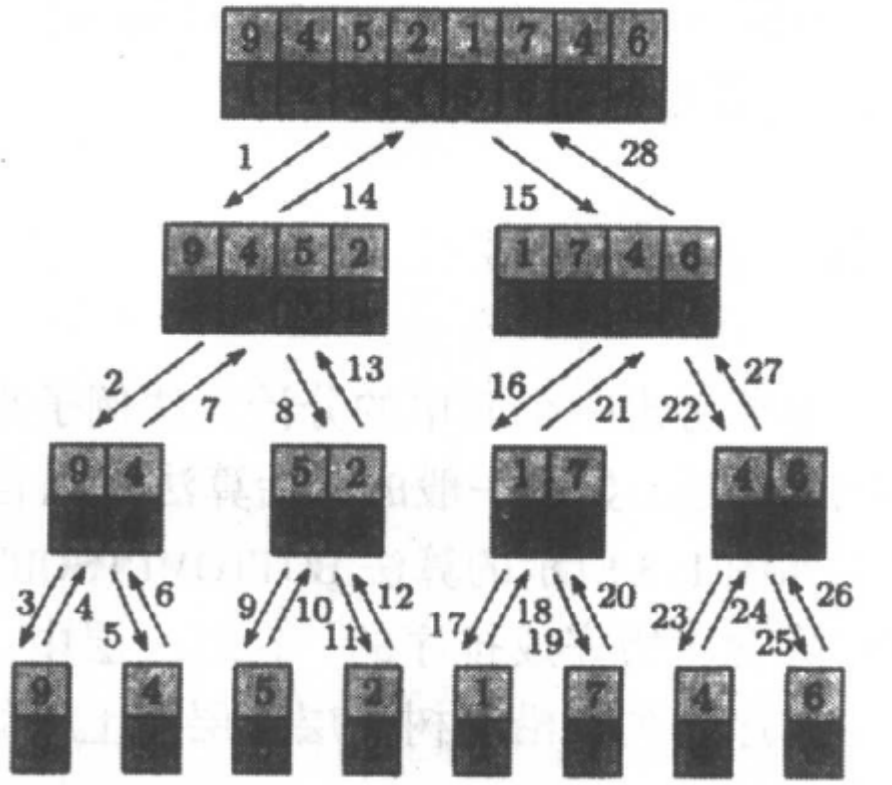
过程　*mergesort*$(low, high)$

1. **if** $low < high$ **then**
2. 　　$mid \leftarrow \lfloor (log + high)/2 \rfloor$
3. 　　*mergesort*(A, low, mid)
4. 　　*mergesort*$(A, mid + 1, high)$
5. 　　MERGE$(A, low, mid, high)$
6. **end if**

- 算法执行过程示例

$A[1\cdots8]=$

9	4	5	2	1	7	4	6



- 算法分析
  - **Case 1:** 元素个数n是2的整数次幂

    - 最好情况时间复杂度:

      $C_B(n)=2C_B(n/2)+n/2\ (\ n\geq 2\ ),\ C_B(1)=0$

      由此得，$C_B(n)=(n\log n)/2=\Theta(n\log n)$

    - 最坏情况时间复杂度:

      $C_W(n)=2C_W(n/2)+n-1\ (\ n\geq 2\ ),\ C_W(1)=0$

      由此得，$C_W(n)=n\log n-n+1=\Theta(n\log n)$

    由上述结果可得:

    - 平均情况时间复杂度: $C_A(n)=\Theta(n\log n)$

- 算法分析
  - **Case 2:** 元素个数n不是2的整数次幂

如果 n 是任意的正整数(不必是 2 的幂)，对于由算法 MERGESORT 执行的元素比较次数 $C(n)$的递推关系式为

$$C(n)=\begin{cases}0 & \text{若 } n=1\\ C(\lfloor n/2\rfloor)+C(\lceil n/2\rceil)+bn & \text{若 } n\geqslant 2\end{cases}$$

b 是某个非负的常数。由定理　　，这个递推式的解是 $C(n)=\Theta(n\log n)$。

定理　　算法 MERGESORT 对一个 n 个元素的数组排序所需的时间是 $\Theta(n\log n)$，空间是 $\Theta(n)$。

计数逆序

让我们考虑把你与一个陌生人对同一组 n 个电影的排名进行比较. 一种自然的方法将是依照你的排名从 1 到 n 标记这些电影，然后依照这个陌生人的排名排序这些标记，并且看看有多少个对“次序出错”. 更具体地说，我们将考虑下面的问题. 给定 n 个数的一个序列 $a_1, a_2, \cdots, a_n$；我们将假设所有的数是不相同的. 我们想定义一个度量，它将告诉我们这个表与处于上升顺序的表相差多远；如果 $a_1 < a_2 < \cdots < a_n$，这个度量的值应该是 0，并且应该随着数变得更加杂乱时增加.

把这个概念量化的一种自然的方式是计数逆序的个数. 我们说两个指标 $i<j$ 构成一个逆序，如果 $a_i > a_j$，即如果两个元素 a_i 与 a_j 是“次序出错”的. 我们想确定在序列 $a_1, a_2, \cdots, a_n$ 中的逆序个数.

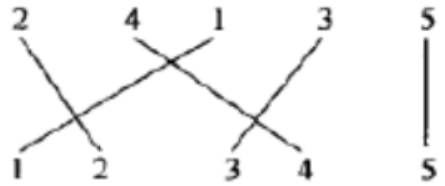


图 5.4 计数在序列 2,4,1,3,5 中的逆序个数. 每对交叉线段与在输入表及上升表中相反次序的一对数——换句话说，即一个逆序对应

先放下这个定义，考虑一个例子，其中序列是 2,4,1,3,5. 在这个序列中存在 3 个逆序：(2,1),(4,1) 与 (4,3). 也存在一个引人注意的几何方法来把逆序形象化，它画在图 5.4 中. 我们把输入数的序列按照它们被提供的次序画出来，而下面的序列处于上升次序. 然后我们在顶部表中的每个数与下面表中相同的数之间画一条线段. 每对交叉线段与在两个表中相反次序的一对数——换句话说，即一个逆序对应.

计数逆序

我们现在显示怎样快得多地用 $O(n\log n)$ 时间计数逆序. 注意因为可能存在平方数个逆序，这样一个算法必须在任何时候不用个别地看每个逆序而能够计算总数. 基本的思想是参照在 5.1 节定义的策略(†). 我们令 $m=\lceil n/2 \rceil$ 并且把这个表分成两部分 $a_1, a_2, \cdots, a_m$ 与 $a_{m+1}, \cdots, a_n$. 我们首先分别计数在这两半中每部分的逆序个数. 然后我们计数逆序 (a_i, a_j)，其中这两个数分属不同的两半. 如果我们想应用定理 5.2，关键是我们必须在 $O(n)$ 时间内做这部分工作. 注意到这些前一半/后一半的逆序有着一种特别好的形式：精确地说，它们是一个对 (a_i, a_j)，其中 a_i 在前一半，a_j 在后一半，并且 $a_i > a_j$.

为了有助于计数在两半边之间的逆序个数，我们将使得这个算法也递归地对两半边的数进行排序. 让递归步多做一点工作(既排序也计数逆序)，将使得这个算法的"组合"部分更容易.

于是在这个处理中的关键程序是 Merge-and-Count. 假设我们已经递归地排序了这个表的前一半与后一半并且计数了每部分的逆序. 我们现在有两个排好序的表 A 与 B，分别包含前一半与后一半. 我们想把它们合并产生一个排好序的表 C，同时也计数 (a,b) 对的个数，其中 $a \in A, b \in B$，且 $a > b$. 根据我们前面的讨论，这恰好就是我们对计算前一边/后一半逆序个数的"组合"步所需要的.

计数逆序

Merge-and-Count(A, B)

维护一个 *Current* 指针指向每个表，初始化指向首元素

维护一个变量 *Count* 用于逆序的个数，初始为 0

While 两个表都不空

令 a_i 与 b_j 是由 *Current* 指针指向的元素

把这两个中较小的元素加到输出表中

If b_j 是较小的元素 then

把 *Count* 加上在 A 中剩余的元素数

Endif

把较小元素选出的表中的 *Current* 指针前移

Endwhile

一旦一个表为空，把另一个表剩余的所有元素加到输出中

返回 *Count* 和合并后的表

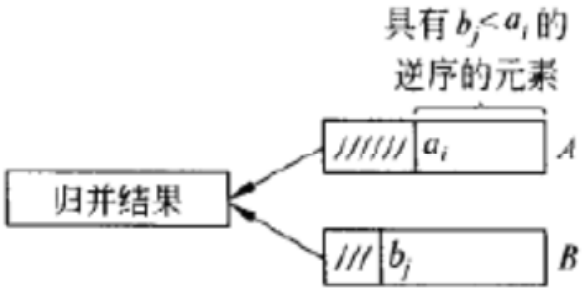


图 5.5 归并两个被排序的数表的同时也计数在它们之间的逆序个数

定理 5.7 Sort-and-Count 算法正确地对输入表排序并且计数逆序个数；它对具有 n 个元素的表运行在 $O(n\log n)$ 时间.

快速排序

算法思想

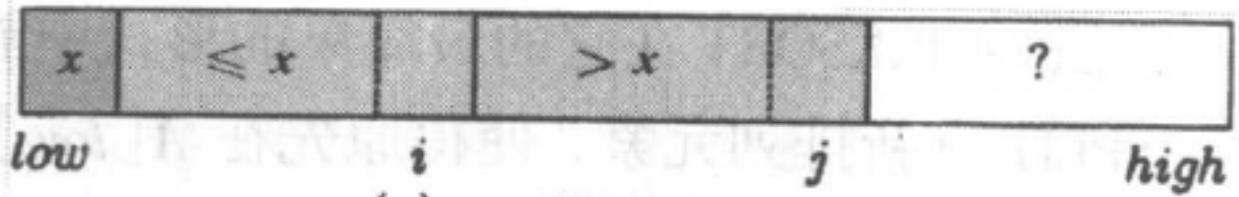
快速分类算法是由著名的计算机科学家霍尔（C.A.R.Hoare）根据分治策略设计的一种高效率的分类算法。基本思想是，对于输入的子序列A[p..q]，快速分类分三步走：

(1) 分解(Divide)：将A[p..q]**划分**成两个非空的子序列A[p..j]和A[j+1..q]，使得A[p..j]中的任一元素小于或等于A[j+1..q]中的任一元素。下标j在划分过程中确定.

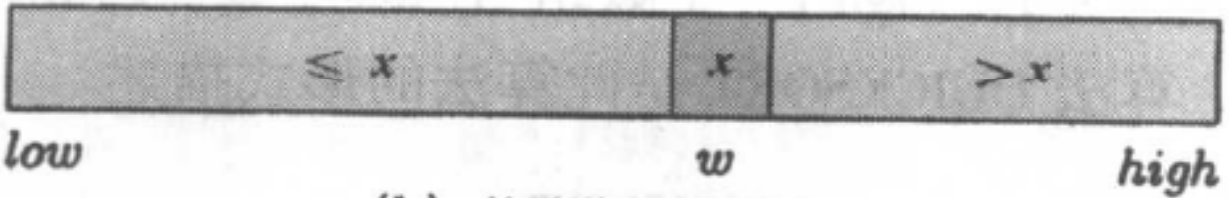
(2) 递归求解(Conquer)：通过递归调用快速分类算法分别对A[p..j-1]和A[j+1..q]进行分类。

(3) 合并(Merge)：由划分的特点知，在A[p..j-1]和A[j+1..q]都分好类后不需任何计算A[p..q]就已分好类。

- 划分算法：SPLIT
  - 如何实现划分？——有多种实现方法



(a) for循环迭代后



(b) 算法终止后

算法 SPLIT 的行为

  - 时间复杂度：*n*-1次比较

- 划分算法：SPLIT

算法　　SPLIT

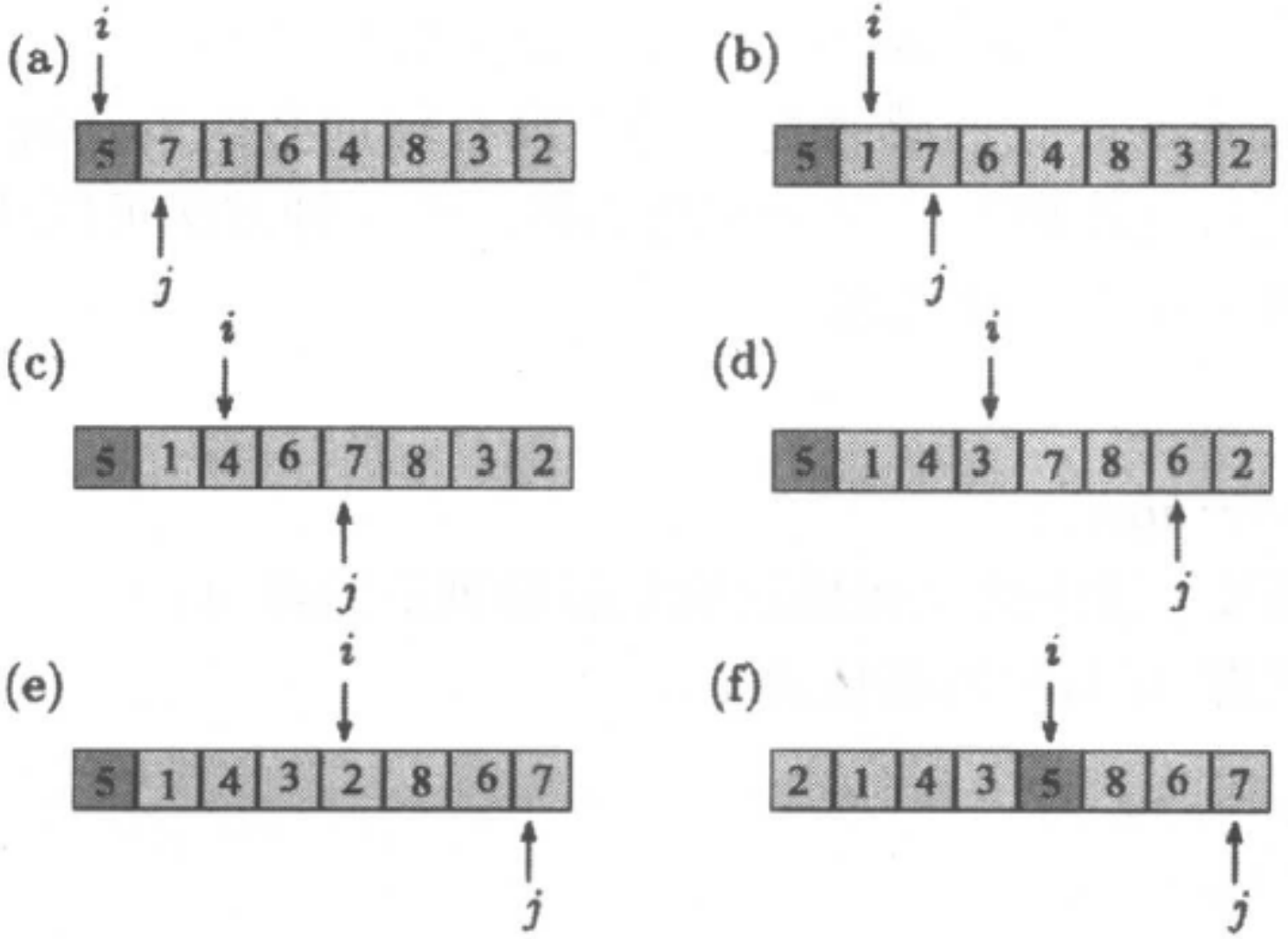
输入：数组 $A[low \cdots high]$。

输出：(1) 如有必要，输出按上述描述的重新排列的数组 A；

(2) 划分元素 $A[low]$的新位置 w。

```
1. i ← low
2. x ← A[low]
3. for j ← low + 1 to high
4.     if A[j] ≤ x then
5.         i ← i + 1
6.         if i ≠ j then 互换 A[i]和 A[j]
7.     end if
8. end for
9. 互换 A[low]和 A[i]
10. w ← i
11. return A 和 w
```

- 划分算法：SPLIT



用 SPLIT 算法划分序列数的例子

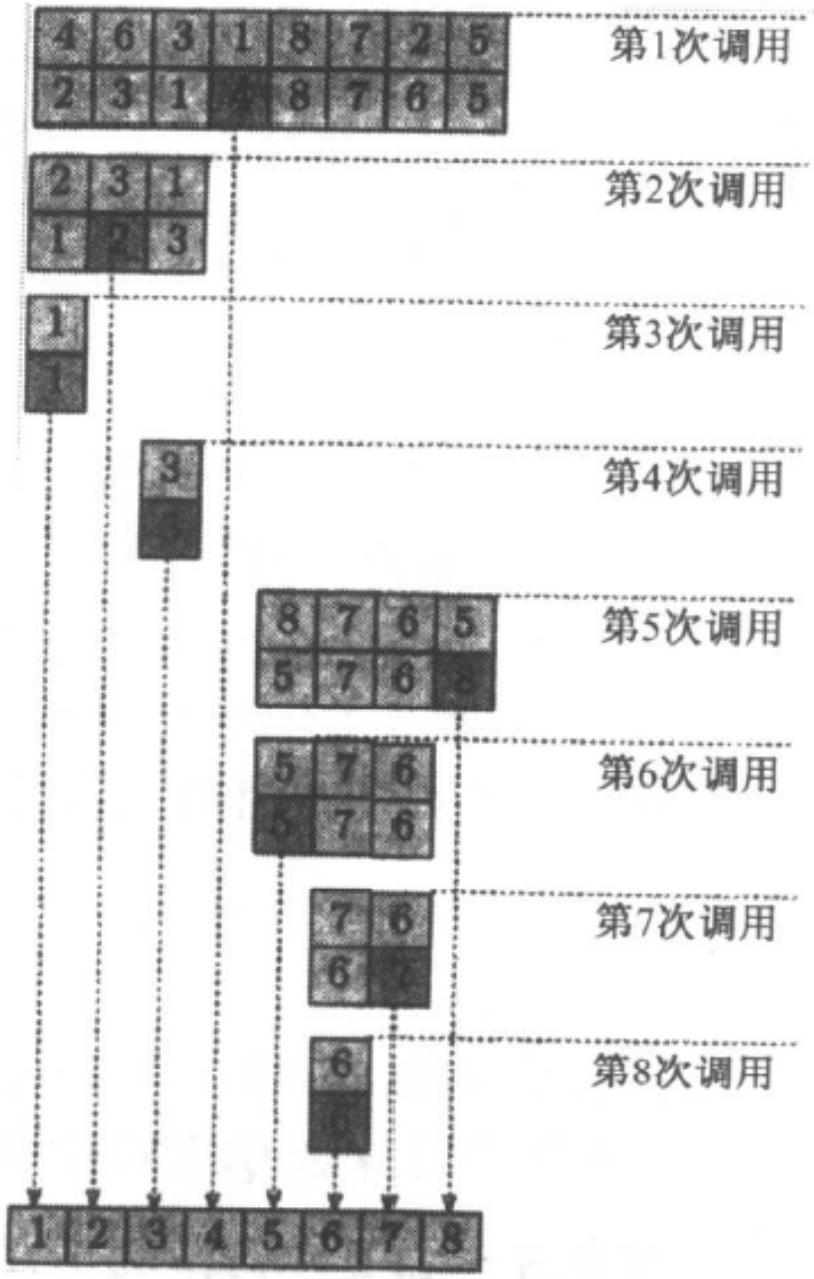
- 快速算法：QUICKSORT

算法　　QUICKSORT
输入： n 个元素的数组 $A[1\cdots n]$。
输出： 按非降序排列的数组 A 中的元素。

```
1. quicksort(A, 1, n)
```

过程　*quicksort*(*A*, *low*, *high*)

```
1. if low < high then
2.     SPLIT(A[low···high], w)    {w 为 A[low]的新位置}
3.     quicksort(A, low, w − 1)
4.     quicksort(A, w + 1, high)
5. end if
```



QUICKSORT 算法运行过程举例

- 算法分析
  - 最坏情况时间复杂度

$$(n-1)+(n-2)+\cdots+1+0=\frac{n(n-1)}{2}=\Theta(n^2)$$

  - 最好情况时间复杂度

$$C(n)=\begin{cases}0 & \text{若 } n=1\\ 2C(n/2)+\Theta(n) & \text{若 } n>1\end{cases}$$

递推式的解是 $C(n)=\Theta(n\log n)$

定理 在最坏的情况下，算法 QUICKSORT 的运行时间是 $\Theta(n^2)$，然而如果总是选择中项作为主元，它的时间复杂性是 $\Theta(n\log n)$。

- 什么实例上快速排序会出现最坏情况时间复杂度?
- ***A*=[1,2,3,…,*n*]上快速排序情况会怎么样?**
- ***A*=[2,2,2,…,2]上快速排序情况会怎么样?**
- **考虑SPLIT算法的其它实现方法，上述结论又如何?**

- 算法分析
  - 平均情况时间复杂度

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

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

$$nC(n) = n(n-1) + 2\sum_{}^{n} C(w-1) \tag{3}$$

$$(n-1)C(n-1) = (n-1)(n-2) + 2\sum_{}^{n-1} C(w-1) \tag{4}$$

（6.3）–（6.4）并整理各项可得:

$$\frac{C(n)}{n+1} = \frac{C(n-1)}{n} + \frac{2(n-1)}{n(n+1)} \tag{5}$$

令 $D(n)=\frac{C(n)}{n+1}$ 由（5）可得:

$$D(n)=D(n-1)+\frac{2(n-1)}{n(n+1)},\ D(1)=0 \tag{6}$$

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

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

$$=4\sum_{j=2}^{n+1}\frac{1}{j}-2\sum_{j=1}^{n}\frac{1}{j}$$

$$=2\sum_{j=1}^{n}\frac{1}{j}-\frac{4n}{n+1}$$

$$=2\ln n-\Theta(1)$$

$$=\frac{2}{\log e}\log n-\Theta(1)$$

$$\approx 1.44\log n$$

$$C(n)=(n+1)D(n)\approx 1.44\ n\log n$$

定理 算法 QUICKSORT 对 n 个元素的数组进行排序执行的平均比较次数是 $\Theta(n\log n)$。

- 快速排序算法时间复杂度
  - 最坏情况: $\Theta(n^2)$
  - 最好情况: $\Theta(n\log n)$
  - 平均情况: $\Theta(n\log n)$

■ 几种排序算法平均性能之比较

给出了当 n 的值在 500 到 5000 之间时，5 种排序算法的平均比较次数的实验结果。

表　　排序算法的比较

n	SELECTIONSORT	INSERTIONSORT	BOTTOMUPSORT	MERGESORT	QUICKSORT
500	124 750	62 747	3852	3852	6291
1000	499 500	261 260	8682	8704	15 693
1500	1 124 250	566 627	14 085	13 984	28 172
2000	1 999 000	1 000 488	19 393	19 426	34 020
2500	3 123 750	1 564 522	25 951	25 111	52 513
3000	4 498 500	2 251 112	31 241	30 930	55 397
3500	6 123 250	3 088 971	37 102	36 762	67 131
4000	7 998 000	4 042 842	42 882	42 859	79 432
4500	10 122 750	5 103 513	51 615	49 071	98 635
5000	12 497 500	6 180 358	56 888	55 280	106 178

每种排序算法下的数据是各个算法执行的比较次数。从表中可知，算法 QUICKSORT 所执行的平均比较次数几乎是算法 MERGESORT 和算法 BOTTOMUPSORT 的两倍。

回顾一个递归方程的解

$f(n)=af(n/c)+bn^x$ $(n>1)$ ，$f(1)=d$，其中 $n=c^k$.

- 引理

引理 设 a 和 c 是非负整数，b,d,x 是非负常数，并且对于某个非负整数 k，令 $n=c^k$，那么，下面递推式

$$f(n)=\begin{cases} d & \text{若 } n=1 \\ af(n/c)+bn^x & \text{若 } n\geqslant 2 \end{cases}$$

的解是

$$f(n)=bn^x\log_c n+dn^x \qquad \text{若 } a=c^x$$

$$f(n)=\left(d+\frac{bc^x}{a-c^x}\right)n^{\log_c a}-\left(\frac{bc^x}{a-c^x}\right)n^x \qquad \text{若 } a\neq c^x$$

$f(n)=af(n/c)+bn^x\ (n>1)$ ，$f(1)=d$，其中 $n=c^k$.

- 定理

定理 设 a 和 c 是非负整数，b,d,x 是非负常数，并且对于某个非负整数 k，令 $n=c^k$，那么，下面递推式

$$f(n)=\begin{cases} d & \text{若 } n=1 \\ af(n/c)+bn^x & \text{若 } n\geqslant 2 \end{cases}$$

的解是

$$f(n)=\begin{cases} \Theta(n^x) & \text{若 } a<c^x \\ \Theta(n^x\log n) & \text{若 } a=c^x \\ \Theta(n^{\log_c a}) & \text{若 } a>c^x \end{cases}$$

特别地，如果 $x=1$，那么

$$f(n)=\begin{cases} \Theta(n) & \text{若 } a<c \\ \Theta(n\log n) & \text{若 } a=c \\ \Theta(n^{\log_c a}) & \text{若 } a>c \end{cases}$$

大整数乘法

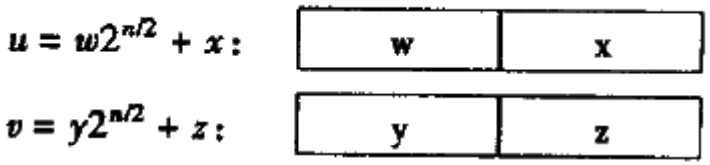
- 问题描述

  设u和v分别两个n-bit的整数（$n=2^k$），求其乘积uv。

- 算法1——传统算法（直接相乘）

  时间复杂度：$\Theta(n^2)$

- **算法2——简单的分治算法**
  - 算法思想



两个大整数的乘法

$$uv = (w2^{n/2} + x)(y2^{n/2} + z) = wy2^n + (wz + xy)2^{n/2} + xz$$

  - 时间复杂度

$T(n)=4T(n/2)+bn$（$n\geq 2$），$T(1)=d$

$\Rightarrow T(n)=\Theta(n^2)$　(定理)

- **算法3——精巧的分治算法**
  - 算法思想（减少子问题个数）

$$wz + xy = (w + x)(y + z) - wy - xz$$

$$uv = wy2^n + ((w + x)(y + z) - wy - xz)2^{n/2} + xz$$

  - 时间复杂度

$T(n)=3T(n/2)+bn$ （$n\geq 2$），$T(1)=d$

$\Rightarrow T(n)=\Theta(n^{\log 3})=O(n^{1.59})$ (定理)

最近点对问题

最近点对问题

在应用中，常用诸如点、圆等简单的几何对象代表现实世界中的实体。在涉及这些几何对象的问题中，常需要了解其邻域中其它几何对象的信息。例如，在空中交通控制问题中，若将飞机作为空中移动的一个点来看待，则具有最大碰撞危险的2架飞机就是空中最接近的一对点。这类问题是计算几何学中研究的基本问题之一。下面我们着重考虑平面上的最接近点对问题。

给定平面上n个点(x_i,y_i)，$1\le i\le n$，找出其中的一对点，使得在n个点的所有点对中，该点对的距离最短。

最近点对问题

- 算法思想
  - 算法一（直接法）：通过检查所有的$n(n-1)/2$对点，并计算每一对点的距离，即可找出距离最近的一对点。这种方法所需的时间为$O(n^2)$。
  - 算法二（分治法）：

1）当n较小时，用直接法着最近点对；

2）当n较大时，将点集分成大致相等的两部分A和B，

（递归地）确定A和B中的最近点对，

确定一点在A中，另一点在B中的最近点对，

从上面得到的三对点中，找出距离最小的一对点。

思考：

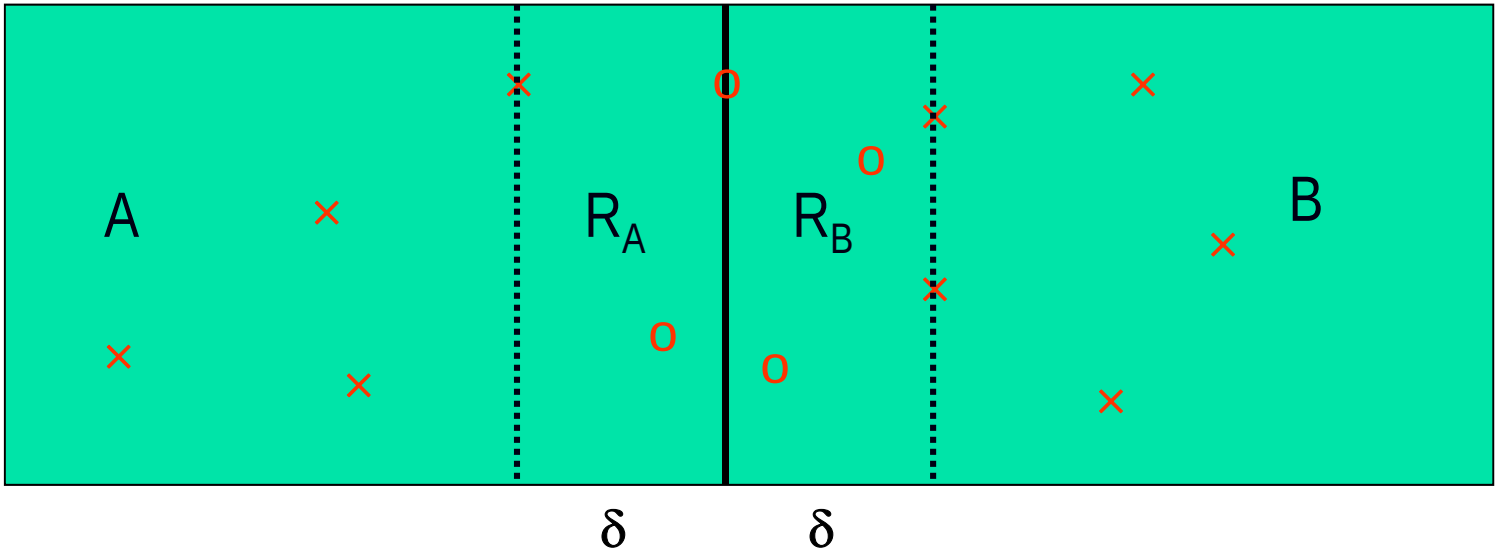
1. 上述分治算法时间复杂度T(n)满足怎样的递归方程?
2. 如何设计划分和合并过程，使得该算法时间复杂度为 $O(n\log n)$?

最近点对问题

在上述分解的步骤中，可在点集中横坐标x的中值处画一条垂线来划分点集。在递归地求解子问题时，关键是确定第三种情况：一点在A中，一点在B中的最近点对。这一步骤能若有效解决，则合并步骤相当简单。如果用一一比较法解它，则花费的时间很多，从而使得整个算法的时间为$O(n^2)$,这与直接法相同。这里我们将它与合并步骤同时考虑。

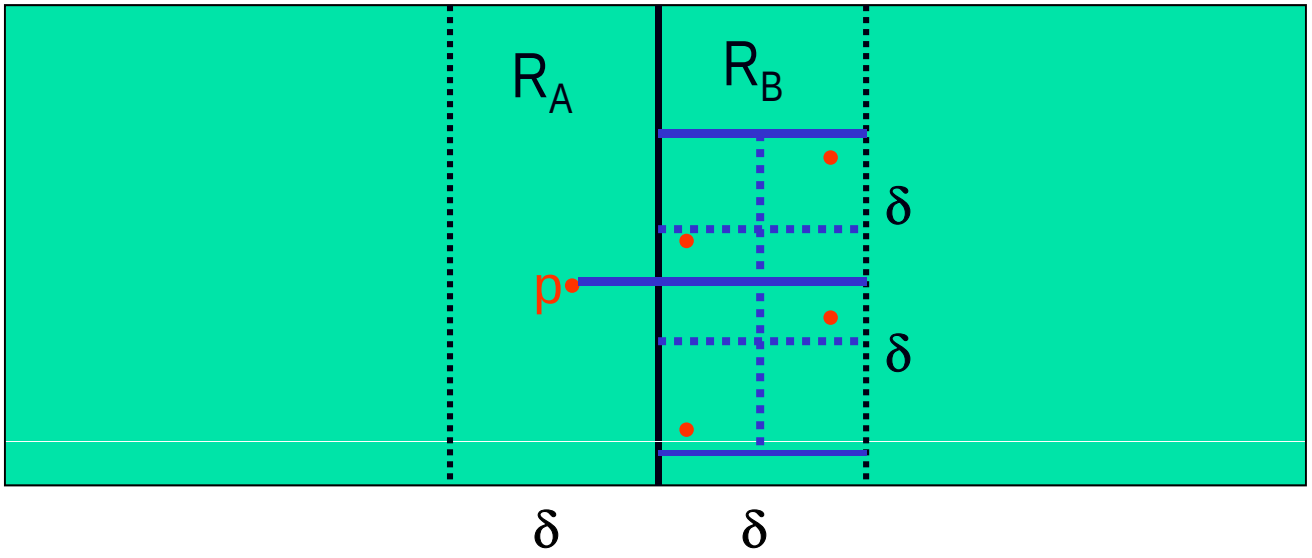
设δ是A的最近点对的距离和B的最近点对的距离中最小者。若第三种情况中的最近点对距离比δ小，则其中每点必然在距离垂线距离<δ的区域内。如下图所示。

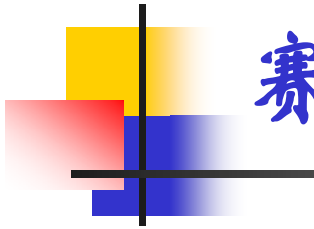
最近点对问题



最近点对问题

对于R_A中的每一点p，只需与位于R_B中的一个大小为$\delta \times 2\delta$的区域中的点相比即可。有鸽笼原理可知，该区域至多有6个点。即R_A中的每一点至多与6个点相比即可求出第三种情况下的最近点对的点距。比一一比较法省时间，它的时间为$O(n)$。如下图所示。





赛程问题

- 问题描述

有n个运动员进行单循环赛，即每个运动员要和所有其它运动员进行一次比赛。试为这n个运动员安排比赛日程。要求每个运动员每天只进行一场比赛，且整个赛程在n-1天内结 束。

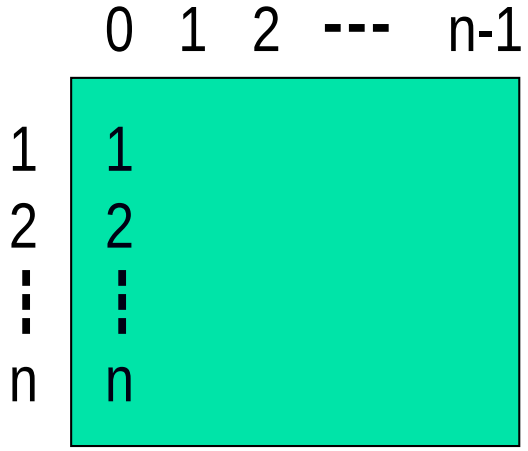
赛程问题

- 算法思想

首先指出，问题可能无解，如n=3。当$n=2^k$时问题有解，当然解可能不唯一，下面用分治法来设计求解该问题一个解的算法。

将运动员从1到n编号，用一个n阶矩阵A[1…n,0…n-1]来表示一种赛程安排，其中第一列是（1,2,…,n），当j>0时，A[i,j]表示第i名运动员在第j天的比赛对手（如下图所示）。它满足 如下条件:

- A每一行不含相同的元素;
- A每一列不含相同的元素;
- 若A[i,j]=k，则A[k,j]=i。



赛程问题

- 求解思路

  将运动员分成两组

  $1, 2, \ldots, n/2$ 和 $n/2+1, n/2+2, \ldots, n$

  为第一组运动员安排赛程，得到n/2阶方阵 A_1；为第二组运动员安排赛程，得到 n/2 阶方阵A_2。由此得到一个n阶方阵A如下左图所示。容易验证A即为原问题的一个解。由此容易写出求解算法来。

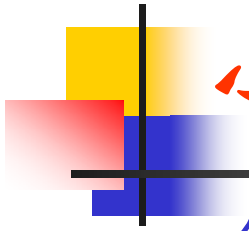
  下面右图是n=8时用我们的算法求得的一个赛程安排。

赛程问题

矩阵A

	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	8
2	2	1	4	3	6	5	8	7
3	3	4	1	2	7	8	5	6
4	4	3	2	1	8	7	6	5
5	5	6	7	8	1	2	3	4
6	6	5	8	7	2	1	4	3
7	7	8	5	6	3	4	1	2
8	8	7	6	5	4	3	2	1

8个选手的比赛日程



小　结

- 分治法的基本模式
  - 基本思想
  - 适用范围
  - 基本步骤
- 应用
  - 掌握每一个分治算法的基本思想

➢ 设计一个好的分治算法有哪些技巧?

——减少子问题个数

——减少合并时间

1. 解释当输入数组 $A[1\cdots n]$ 由 n 个等同的元素组成时，算法 QUICKSORT 的行为。

参考解答：当序列中元素都相同时，每次执行算法**SPLIT**，仅出现一次元素交换，即将序列的第一个元素与最后一个元素交换，且划分元素的新位置为该序列的最后一个位置。因此，在元素均相同的数组$\boldsymbol{A}[1\ldots n]$上，算法**QUICKSORT**的执行特点为：每次划分后只剩下一个非空子序列未处理的。第一　划分后剩下$\boldsymbol{A}[1...n-1]$未处理，第二次划分后剩下$\boldsymbol{A}[1...n-2]$未处理，…，第n-1次划分后剩下$\boldsymbol{A}[1]$（已有序）。在该实例上，算法的执行时间为：$(n-1)+(n-2)+\ldots+2+1=\Theta(n^2)$，属于最坏的情况。最后所得结果中各元素顺序如下：

$$\boldsymbol{A}[2],\boldsymbol{A}[3],\boldsymbol{A}[4],\ldots,\boldsymbol{A}[n],\boldsymbol{A}[1]$$

这里，$\boldsymbol{A}[i]$表示输入时的第i个元素，$i=1,2,\ldots,n$。

思考：如果SPLIT采用另外的实现方法，结论又如何?

2. 设 $x = a + bi$ 和 $y = c + di$ 是两个复数。只要做 4 次乘法就很容易计算乘积 xy，也就是 $xy = (ac - bd) + (ad + bc)i$。设计一个方法，只用 3 次乘法计算乘积 xy。

参考解答：

因为，$(a+b)(c+d)=ac+ad+bc+bd$,

所以有，$xy=(ac-bd)+((a+b)(c+d)-ac-bd)i$.

由此可见，这样计算xy只需要3次乘法。

3. 给出一个分治算法，在一个具有 n 个数的数组中找出第二个最大元素。给出你算法的时间复杂性。

参考解答：

（1）算法思想：当序列$A[1..n]$中元素的个数$n=2$时，通过直接比较即可找出序列的第2大元素。当$n>2$时，先分别求出序列$A[1..n/2]$和$A[n/2+1\ldots n]$中的第1大元素x_1,y_1和第2大元素x_2,y_2；然后，通过2次比较即可在四个元素x_1,y_1,x_2,y_2中找出第2大元素，该元素即为$A[1..n]$中的第2大元素。

（2）当n是2的整数次幂时，分治法求解$A[1..n]$中第2大元素需要比较次数$T(n)$满足递归方程：

$$T(n)=2T(n/2)+2,\ T(2)=1。$$

解得 $T(n)=1.5n-2$ 。