第2章　递归与分治策略2

上节回顾

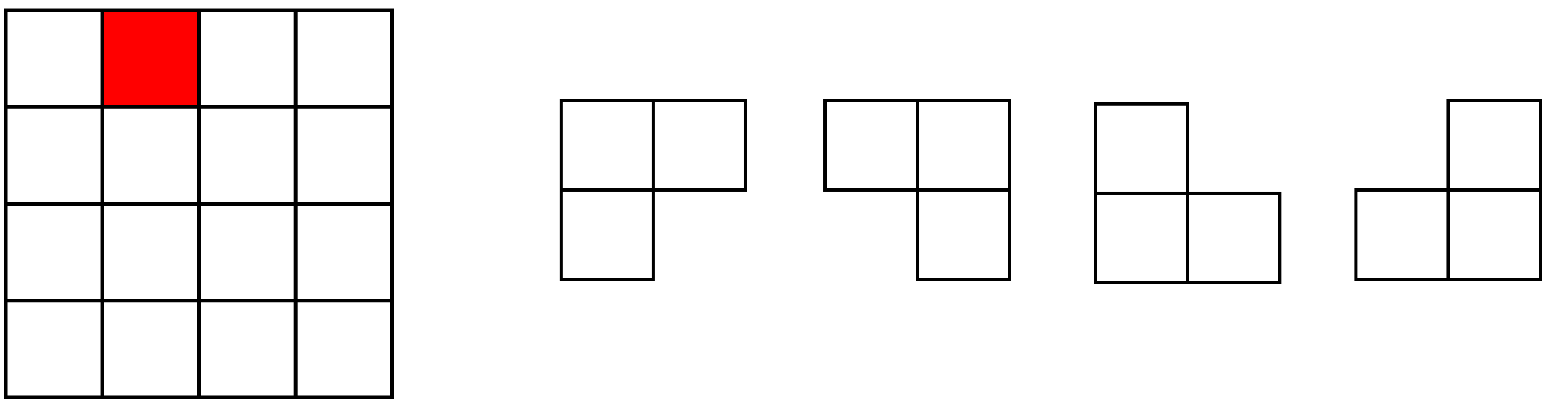
- 理解递归的概念
- 掌握设计有效算法的分治策略
  - 阶乘函数、Fibonacci数列
  - Ackerman函数
- 通过下面的范例学习分治策略设计技巧
  - 排列问题
  - 整数划分问题
  - Hannoi塔问题
  - 二分搜索
  - 大整数乘法

本节内容

- 通过下面的范例学习分治策略设计技巧
  - 棋盘覆盖；
  - 合并排序
  - 快速排序；
  - 线性时间选择；
  - 最接近点对问题；

2.6 棋盘覆盖

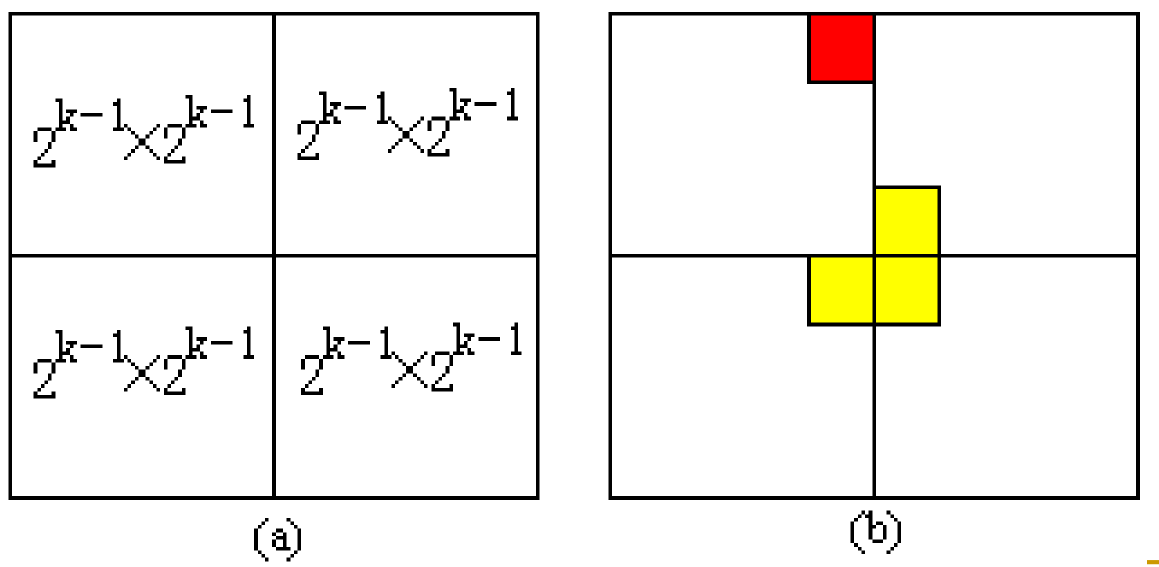
在一个$2^k \times 2^k$ 个方格组成的棋盘中，恰有一个方格与其它方格不同，称该方格为一特殊方格，且称该棋盘为一特殊棋盘。在棋盘覆盖问题中，要用图示的4种不同形态的L型骨牌覆盖给定的特殊棋盘上除特殊方格以外的所有方格，且任何2个L型骨牌不得重叠覆盖。



2.6 棋盘覆盖

当$k>0$时，将$2^k \times 2^k$棋盘分割为4个$2^{k-1} \times 2^{k-1}$ 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。



棋盘覆盖问题中数据结构的设计：

1. 棋盘：可以用一个二维数组board[size][size]表示一个棋盘，其中，size=2^k。为了在递归处理的过程中使用同一个棋盘，将数组board设为全局变量；
2. 子棋盘：整个棋盘用二维数组board[size][size]表示，其中的子棋盘由棋盘左上角的下标tr、tc和棋盘大小s表示；
3. 特殊方格：用board[dr][dc]表示特殊方格，dr和dc是该特殊方格在二维数组board中的下标;
4. L型骨牌：一个2^k×2^k的棋盘中有一个特殊方格，所以，用到L型骨牌的个数为(4^k-1)/3，将所有L型骨牌从1开始连续编号，用一个全局变量t表示。

2.6 棋盘覆盖

```
void chessBoard(int tr, int tc, int dr, int dc, int size)
  {
    if (size == 1) return;
    int t = tile++,  // L型骨牌号
      s = size/2;  // 分割棋盘
    // 覆盖左上角子棋盘
    if (dr < tr + s && dc < tc + s)
      // 特殊方格在此棋盘中
      chessBoard(tr, tc, dr, dc, s);
    else {// 此棋盘中无特殊方格
      // 用 t 号L型骨牌覆盖右下角
      board[tr + s - 1][tc + s - 1] = t;
      // 覆盖其余方格
      chessBoard(tr, tc, tr+s-1, tc+s-1, s);}
    // 覆盖右上角子棋盘
    if (dr < tr + s && dc >= tc + s)
      // 特殊方格在此棋盘中
      chessBoard(tr, tc+s, dr, dc, s);
    else {// 此棋盘中无特殊方格
      // 用 t 号L型骨牌覆盖左下角
      board[tr + s - 1][tc + s] = t;
      // 覆盖其余方格
      chessBoard(tr, tc+s, tr+s-1, tc+s, s);}
    // 覆盖左下角子棋盘
    if (dr >= tr + s && dc < tc + s)
      // 特殊方格在此棋盘中
      chessBoard(tr+s, tc, dr, dc, s);
    else {// 用 t 号L型骨牌覆盖右上角
      board[tr + s][tc + s - 1] = t;
      // 覆盖其余方格
      chessBoard(tr+s, tc, tr+s, tc+s-1, s);}
    // 覆盖右下角子棋盘
    if (dr >= tr + s && dc >= tc + s)
      // 特殊方格在此棋盘中
      chessBoard(tr+s, tc+s, dr, dc, s);
    else {// 用 t 号L型骨牌覆盖左上角
      board[tr + s][tc + s] = t;
      // 覆盖其余方格
      chessBoard(tr+s, tc+s, tr+s, tc+s, s);}
  }
```

2.6 棋盘覆盖

复杂度分析

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

$T(n)=O(4^k)$

2.7 合并排序

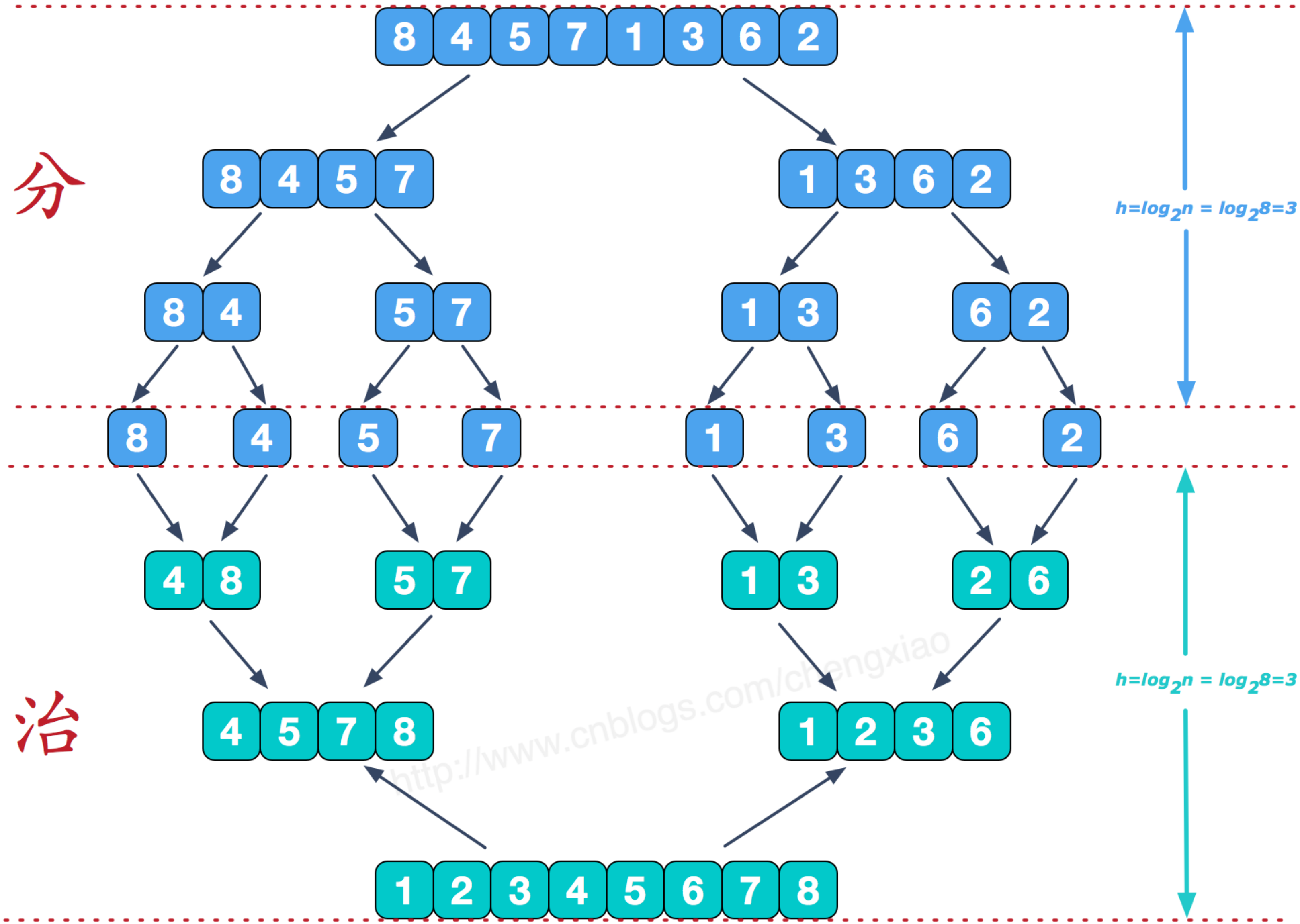
基本思想：将待排序元素分成大小大致相同的2个子集合，分别对2个子集合进行排序，最终将排好序的子集合合并成为所要求的排好序的集合。

6 5 3 1 8 7 2 4

2.7 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。

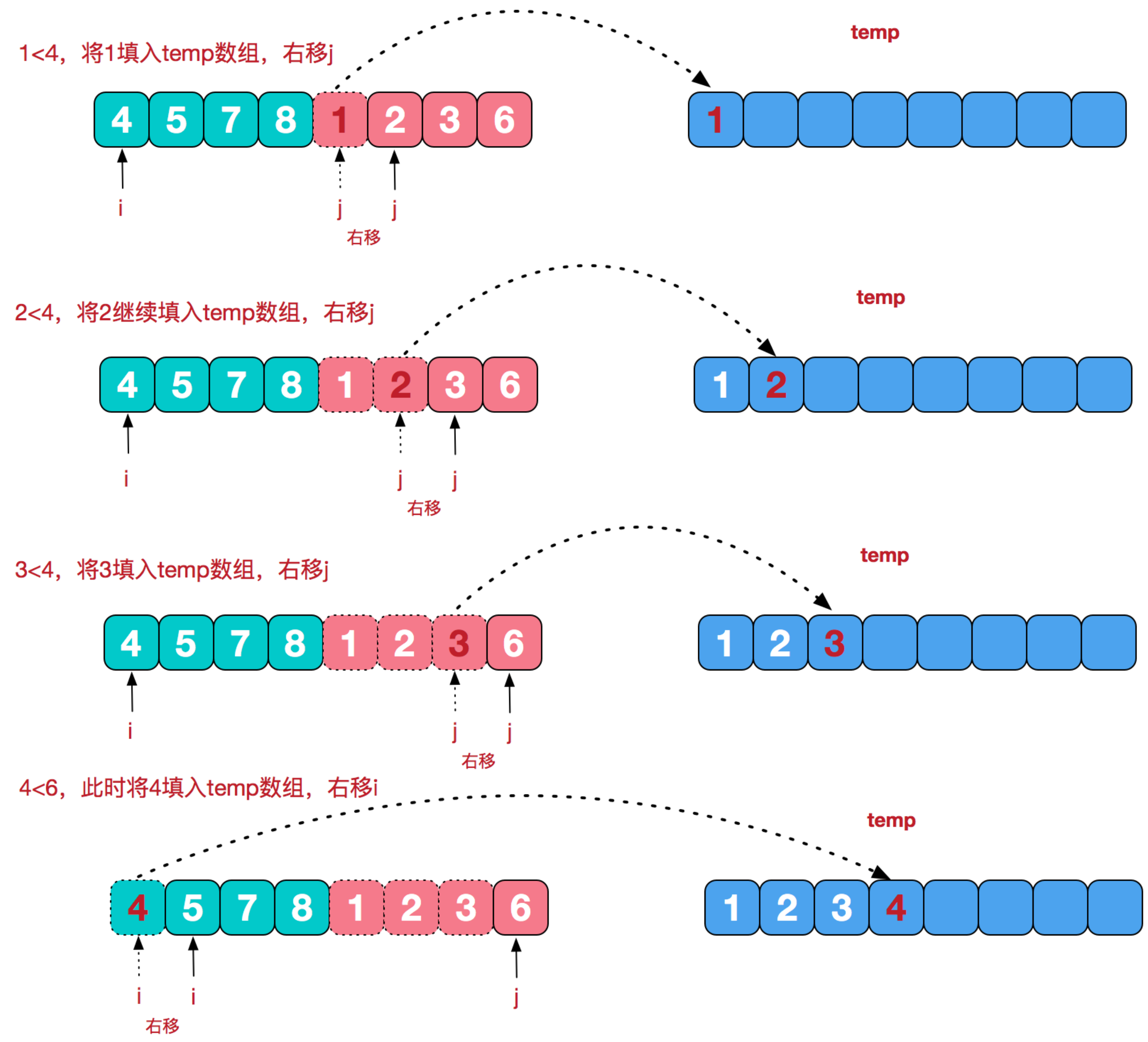
```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```


分
治
8 4 5 7 1 3 6 2
8 4 5 7
1 3 6 2
8 4
5 7
1 3
6 2
8
4
5
7
1
3
6
2
4 8
5 7
1 3
2 6
4 5 7 8
1 2 3 6
1 2 3 4 5 6 7 8
$h=\log_2 n = \log_2 8=3$
$h=\log_2 n = \log_2 8=3$
http://www.cnblogs.com/chengxiao

对两个有序数组的合并可以通过下面的算法完成。

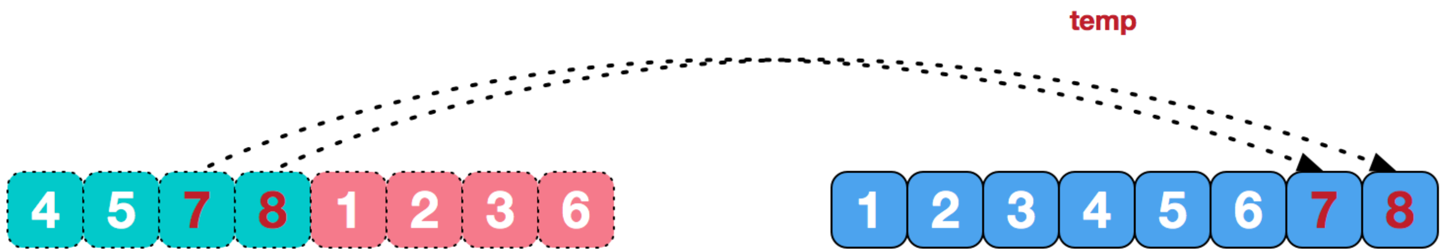
1. 初始状态下，两个指针（数组下标）分别指向两个待合并数组的第一个元素。
2. 然后比较这两个元素的大小，将较小的元素添加到一个新创建的数组中；
3. 接着，被复制数组中的指针后移，指向较小元素的后继元素。
4. 上述操作一直持续到两个数组中的一个被处理完为止。
5. 然后，在未处理完的数组中，剩下的元素被复制到新创建数组的尾部。

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

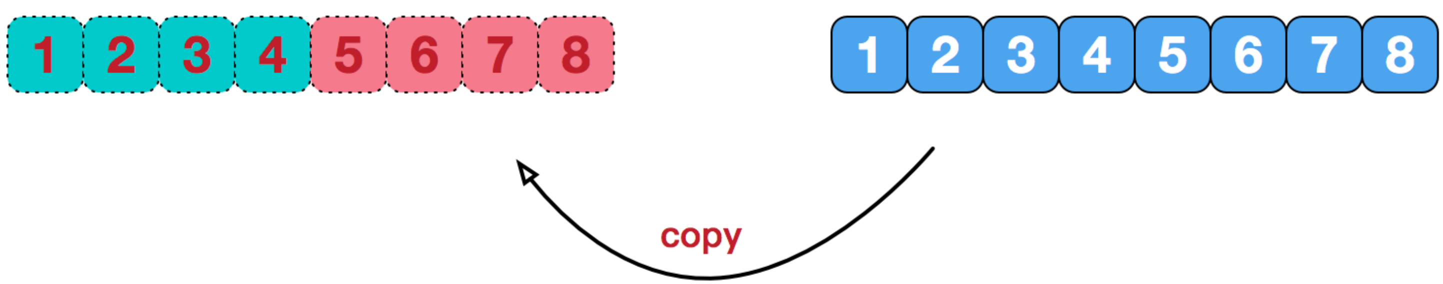


1<4，将1填入temp数组，右移j
temp
4 5 7 8 1 2 3 6
1
i
j
j
右移
2<4，将2继续填入temp数组，右移j
temp
4 5 7 8 1 2 3 6
1 2
i
j
j
右移
3<4，将3填入temp数组，右移j
temp
4 5 7 8 1 2 3 6
1 2 3
i
j
j
右移
4<6，此时将4填入temp数组，右移i
temp
4 5 7 8 1 2 3 6
1 2 3 4
i
i
j
右移

继续重复这种比较+填入的步骤，直到右子序列已经填完，这时将左边剩余的7和8依次填入



最后，将temp中的内容全部拷到原数组中去，排序完成



```
算法 Merge(B[0...p-1]，C[0...q-1]，A[0...p+q-1])
//将两个有序数组合并为一个有序数组
//输入：两个有序数组 B[0...p-1]和 C[0...q-1]
//输出： A[0...p+g-1]中已经有序存放了B和 C中的元素
i ← 0; j ←0; k ←0
while i < p and j< q do
     if B[i]≤C[j]
          A[k]←B[i]; i ← i+1
     else
          A[k] ←C[j]; j ←j+1; k←k + 1
     if i= p
          copy C[j..q-1] to A[k...p+q-1]
     else
          copy B[i..p-1] to A[k...p+q-1]
```

复杂度分析

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

分析C(n)，合并阶段进行键值比较的次数。每做一步都要进行一次比较，比较之后，两个数组中尚需处理的元素总个数减一。在最坏的情况下，无论哪个数组都不会为空。因此，对于最坏情况来说

$$C(n) = n - 1$$

根据主定理有：

$$T(n) \in \Theta(n \log n)$$

2.7 合并排序

- **最坏时间复杂度：O(nlogn)**
- **平均时间复杂度：O(nlogn)**

合并排序

优点：稳定性。在最坏情况下的键值比较次数十分接近于任何基于比较的排序算法在理论上能够达到的最少次数。

缺点：算法需要线性的额外空间

2.8 快速排序

合并排序：按照元素在数组的位置进行划分
快速排序：按照元素的值对他们进行划分

$$\underbrace{A[0] \ldots A[s-1]}_{\text{都} \leq A[s]} \quad A[s] \quad \underbrace{A[s+1] \ldots A[n-1]}_{\text{都} \geq A[s]}$$

$A[s]$已经位于有序数组中的最终位置，对$A[s]$前后子数组进行排序

2.8 快速排序

区别之处：

合并排序：将问题划分成两个子问题很快的，算法主要工作在于合并子问题的解

快速排序：算法主要工作在于划分节点，而不需要再去合并子问题的解

2.8 快速排序

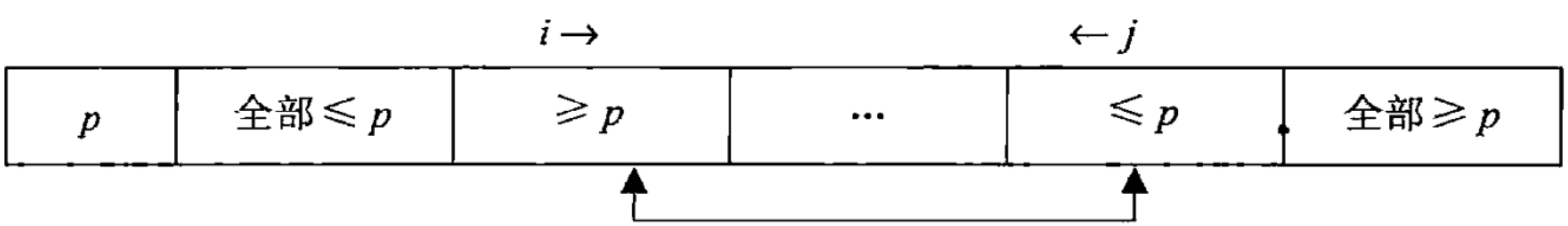
在快速排序中，记录的比较和交换是从两端向中间进行的，关键字较大的记录一次就能交换到后面单元，关键字较小的记录一次就能交换到前面单元，记录每次移动的距离较大，因而总的比较和移动次数较少。

```
template<class Type>
void QuickSort (Type a[], int p, int r)
{
    if (p<r) {
      int q=Partition(a,p,r);
      QuickSort (a,p,q-1); //对左半段排序
      QuickSort (a,q+1,r); //对右半段排序 }}
```

2.8 快速排序

分别从子数组的两端进行扫描，并且将扫描到的元素与中轴相比较

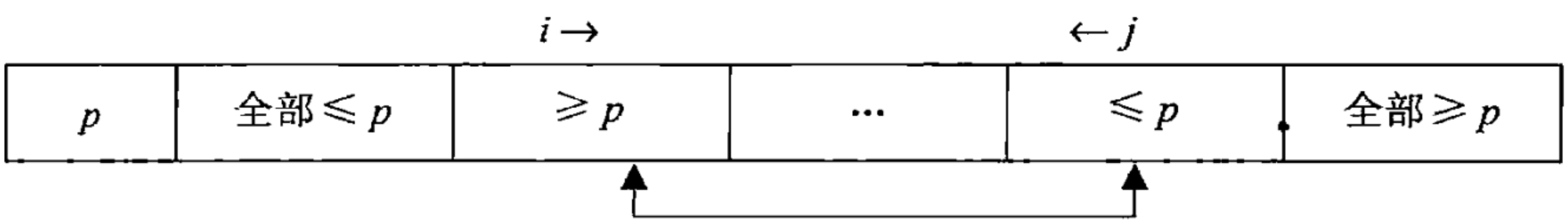
从左到右的扫描，从第二个元素开始。因为我们希望小于中轴的元素位于子数组的左半部分，扫描会忽略小于中轴的元素，直到遇到第一个大于等于中轴的元素才会停止



2.8 快速排序

从右到左的扫描，从最后一个元素开始。因为我们希望大于中轴的元素位于子数组的右半部分，扫描会忽略大于中轴的元素，直到遇到第一个小于等于中轴的元素才会停止。

为什么当遇到与中轴元素相等的元素时值得停止扫描?

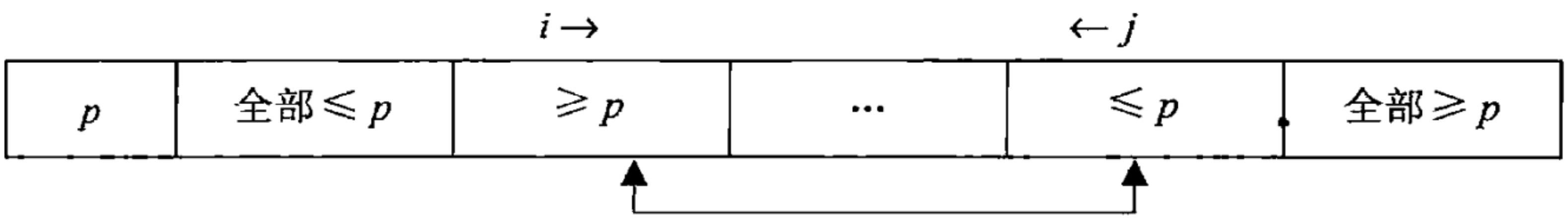


事实上不中止也是可以的，但是当遇到很多相同元素的数组时，用这个方法可以将数组分的更加平均，从而大大提升算法运行速度。

2. 8 快速排序

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。

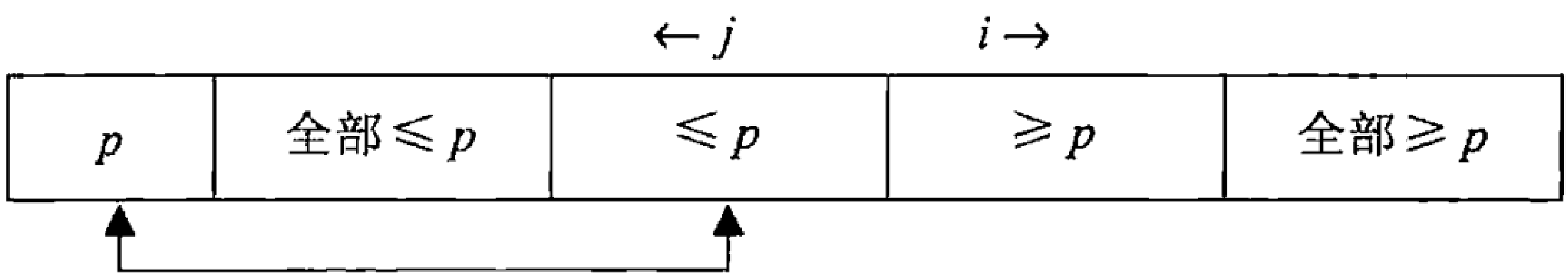
1. 如果扫描指针i和j不相交，也就是说i<j，我们简单地交换A[i],A[j],再分别对i加1，对j减1



2.8 快速排序

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。

2. 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



2.8 快速排序

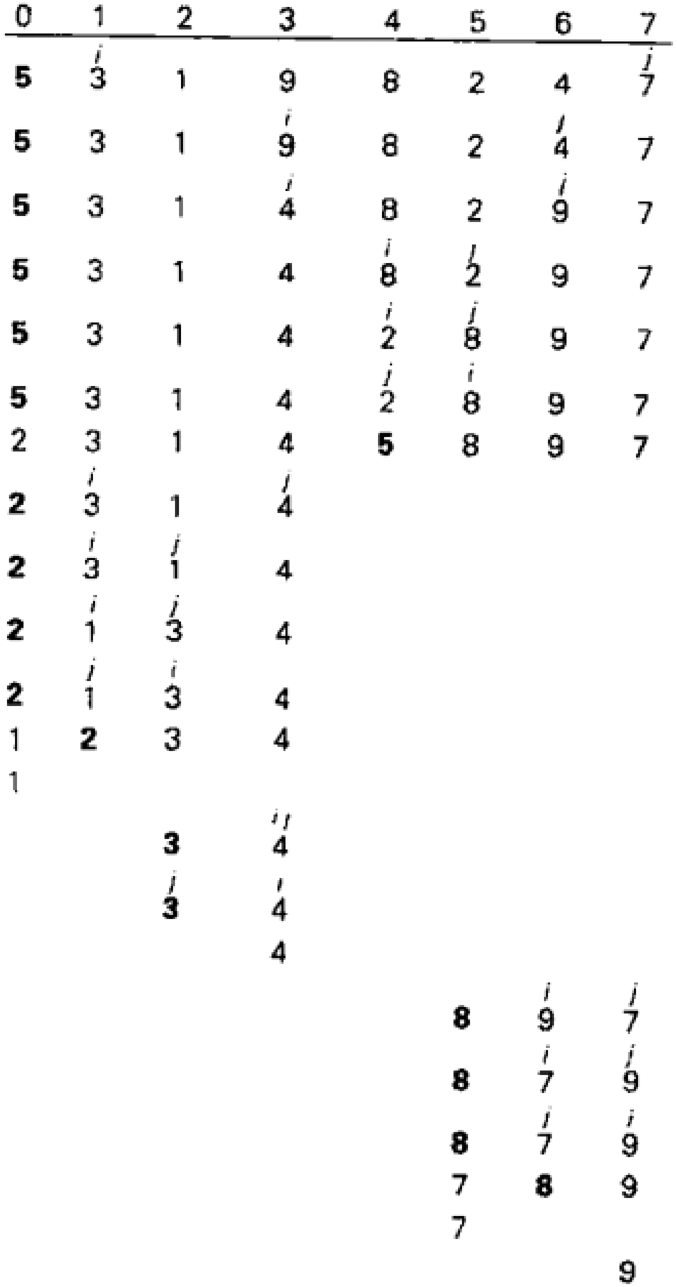
两次扫描全部停止以后，取决于扫描的指针是否相交，会发生3种不同的情况。

3. 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

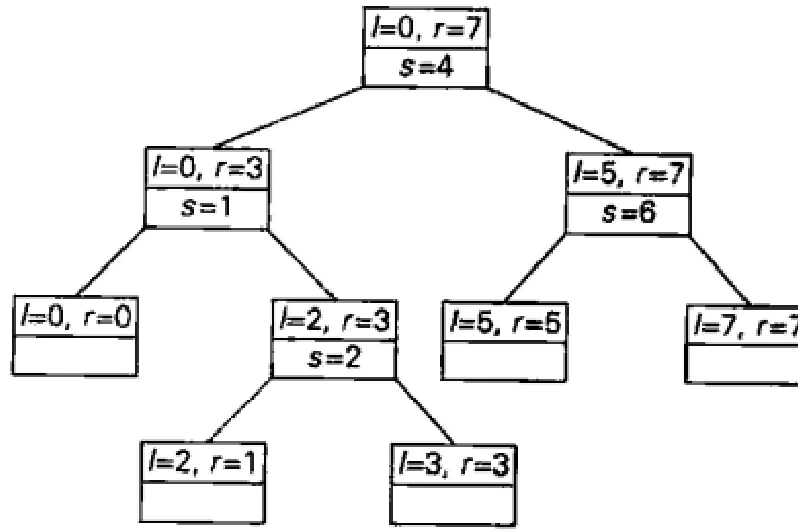
为什么?

$\leftarrow j = i \rightarrow$

p	全部$\leqslant p$	$=p$	全部$\geqslant p$


0 1 2 3 4 5 6 7
5 3 1 9 8 2 4 7
5 3 1 9 8 2 4 7
5 3 1 4 8 2 9 7
5 3 1 4 8 2 9 7
5 3 1 4 2 8 9 7
5 3 1 4 2 8 9 7
2 3 1 4 5 8 9 7
2 3 1 4
2 3 1 4
2 1 3 4
2 1 3 4
1 2 3 4
1
3 4
3 4
4
8 9 7
8 7 9
8 7 9
7 8 9
7
9

(a)


l=0, r=7
s=4
l=0, r=3
s=1
l=5, r=7
s=6
l=0, r=0
l=2, r=3
s=2
l=5, r=5
l=7, r=7
l=2, r=1
l=3, r=3

(b)

2.8 快速排序

如果扫描指针交叉了，划分比较次数是 n+1

如果它们相等，划分比较次数是 n

最好时间复杂度：如果所有的分裂点位于相应子数组的中点，这就是最优的情况

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

最坏时间复杂度：所有的分裂点都趋于极端：两个子数组有一个为空，而另一个子数组仅仅比被划分的数组少一个元素。

$$T(n) = T(n-1) + n + 1 \qquad O(n^2)$$

2.8 快速排序

快速排序算法的性能取决于划分的对称性。通过修改算法**partition**，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。

```
template<class Type>
int RandomizedPartition (Type a[], int p, int r)
{
        int i = Random(p,r);
        Swap(a[i], a[p]);
        return Partition (a, p, r);
}
```

2.8 快速排序

最坏时间复杂度：$O(n^2)$

平均时间复杂度：$O(n\log n)$

辅助空间：$O(n)$或$O(\log n)$

快速排序中分治思想三个要点

1、划分步：把输入的问题划分为子问题

2、治理步：调用处理方法来处理问题

3、组合步：组合步把各个子问题的解组合起来

2.9 线性时间选择

给定线性序集中n个元素和一个整数k，1≤k≤n，要求找出这n个元素中第k小的元素?

- k = 1: 最小值;
- k = n: 最大值;
- k = n/2 : 中位数

最简单的方法: 排序+选择.

$T(n) = \Theta(n \log n) + \Theta(1) = \Theta(n \log n)$

有没有可能更简单?

2.9 线性时间选择

给定线性序集中n个元素和一个整数k，1≤k≤n，要求找出这n个元素中第k小的元素

设列表是以数组实现的，其元素索引从0开始，而s是划分的分割位置，也就是划分后中轴所在元素的索引。

如果s=k-1，中轴p本身显然就是第k小的元素。

如果s>k-1，整个列表的第k小元素就是被划分数组左边部分的第k小的元素。

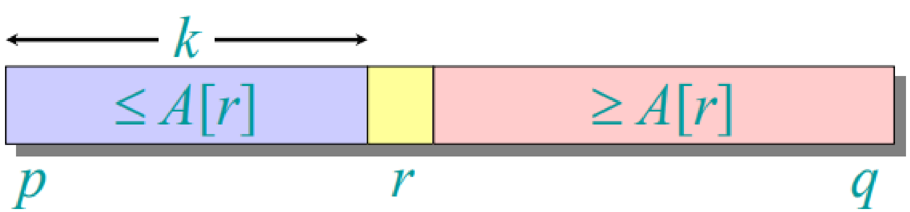
而如果s<k-1，就是数组右边部分的第（k-s）小元素。

它的实例规模变得更小了，这个较小实例可以用同样方法来解决

2.9 线性时间选择

给定线性序集中n个元素和一个整数k，1≤k≤n，要求找出这n个元素中第k小的元素

```
template<class Type>
Type RandomizedSelect(Type a[],int p,int r,int k)
{
    if (p==r) return a[p];
    int i=RandomizedPartition(a,p,r),
    j=i-p+1;
    if (k<=j) return RandomizedSelect(a,p,i,k);
    else return RandomizedSelect(a,i+1,r,k-j);
}
```



2.9 线性时间选择

找到第7个最小的数

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

pivot

划分

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

7 – 4 = 3rd

Lucky:

T(n) =O(n) =O(n)

Unlucky:

$T(n) = T(n-1) + O(n) = O(n^2)$

比合并排序?

2.9 线性时间选择

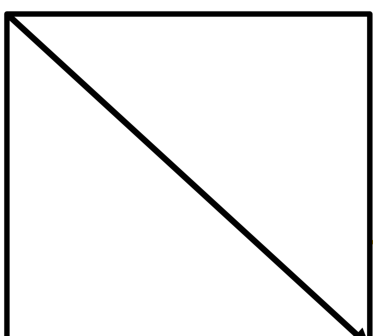
平均时间计算

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

假设第 k 元素均处于最大的划分数组中

$$T(n) = \begin{cases} T(\max\{0, n-1\}) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(\max\{1, n-2\}) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \quad\vdots & \\ T(\max\{n-1, 0\}) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(\max\{k, n-k-1\}) + \Theta(n)\right).$$



2.9 线性时间选择

两边取均值

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(\max\{k, n-k-1\}) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(\max\{k, n-k-1\}) + \Theta(n)\right)\right]$$

均值操作的线性性！

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)]$$

不同选择的独立性

2.9 线性时间选择

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(\max\{k, n-k-1\}) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E[X_k(T(\max\{k, n-k-1\}) + \Theta(n))] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\})] + \frac{1}{n}\sum_{k=0}^{n-1}\Theta(n) \qquad E[X_k] = 1/n. \\
&\le \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)
\end{aligned}
$$

2.9 线性时间选择

证明： 存在足够大的常数 c 满足 $E[T(n)] \le cn$

利用结论：

$$\sum_{k=\lfloor n/2 \rfloor}^{n-1} k \le \frac{3}{8} n^2$$

采用归纳法证：

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n)$$

$$\le \frac{2c}{n}\left(\frac{3}{8} n^2\right) + \Theta(n)$$

2.9 线性时间选择

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} ck + \Theta(n) \\
&\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + \Theta(n) \\
&= cn - \left(\frac{cn}{4} - \Theta(n)\right) \\
&\le cn,
\end{aligned}
$$

总结

优点：线性期望时间，实际应用效果较好

缺点：最坏情况下效果很差

有没有最坏情况下依然为线性的算法?

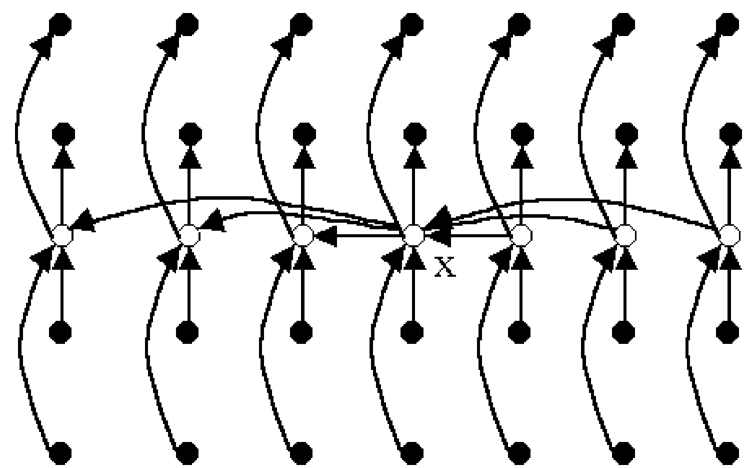
2.9 线性时间选择

改进：通过使划分更均匀，实现在最坏情况下用O(n)时间完成选择任务。

Select算法：找到第i小的元素

1.将n个输入元素划分成⌈n/5⌉个组，每组5个元素，至多有一组由剩下的n mod 5个元素组成。

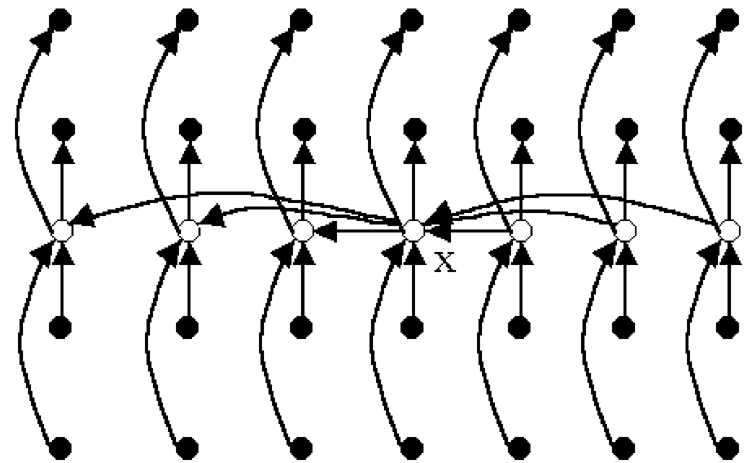
2.用插入排序，将每组中的元素排好序，取出每组的中位数，共⌈n/5⌉个。



2.9 线性时间选择

改进：通过使划分更均匀，实现在**最坏情况下**用O(n)时间完成选择任务。

3.**调用Select**来找出这⌈n/5⌉个中位数的中位数***x***。如果⌈n/5⌉是偶数，就找它的2个中位数中较大的一个。

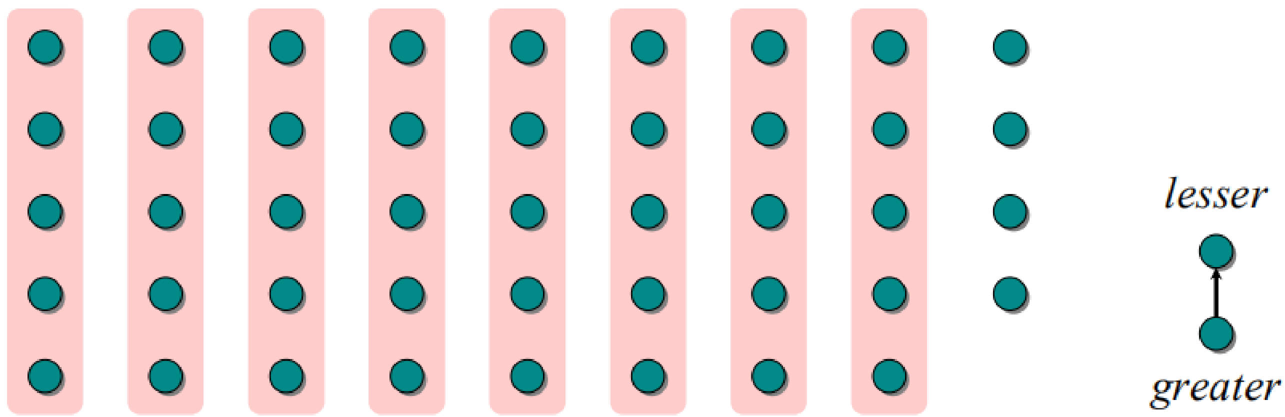


4.按上一步中找到的x作为中轴，对全部 n个元素进行划分，有k-1个数小于x。

5.若i==k，返回x。若i小于k，在低区**调用Select**找出第i小的元素，否则在高区**调用Select**。

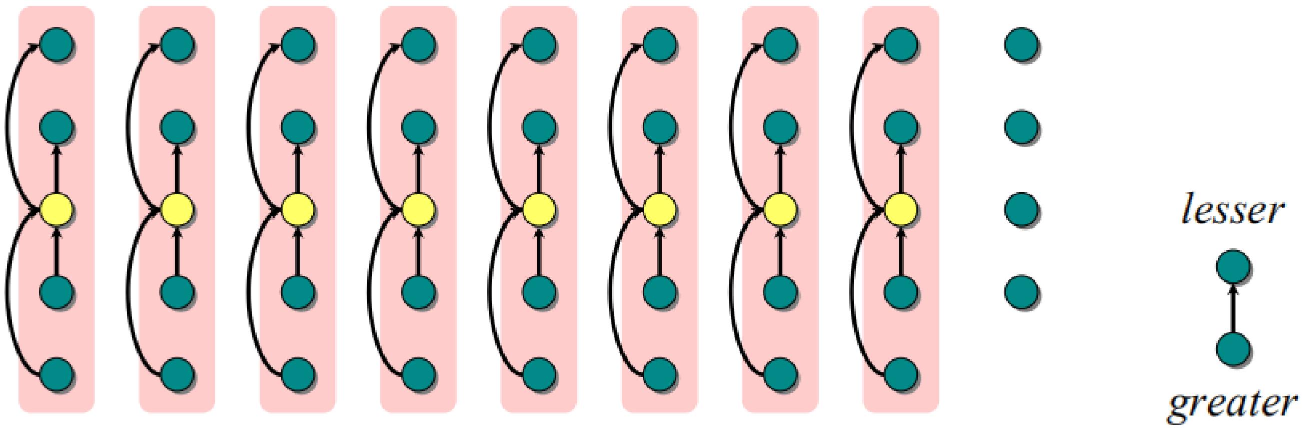
2.9 线性时间选择

Select算法：



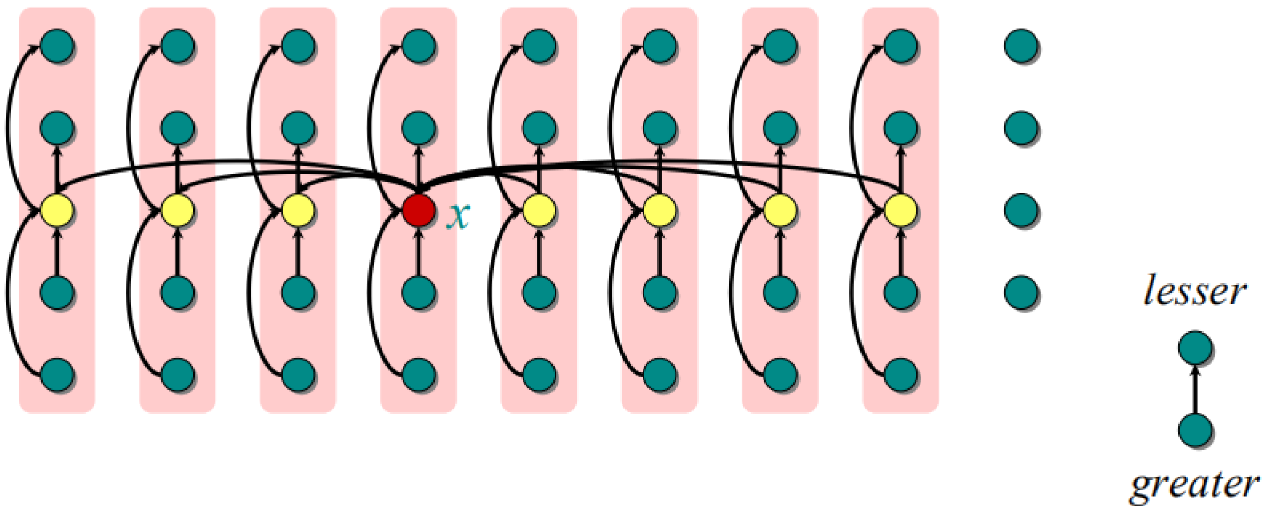
1）将n个输入元素划分成⌈n/5⌉个组，每组5个元素，至多有一组由剩下的n mod 5个元素组成。

2.9 线性时间选择



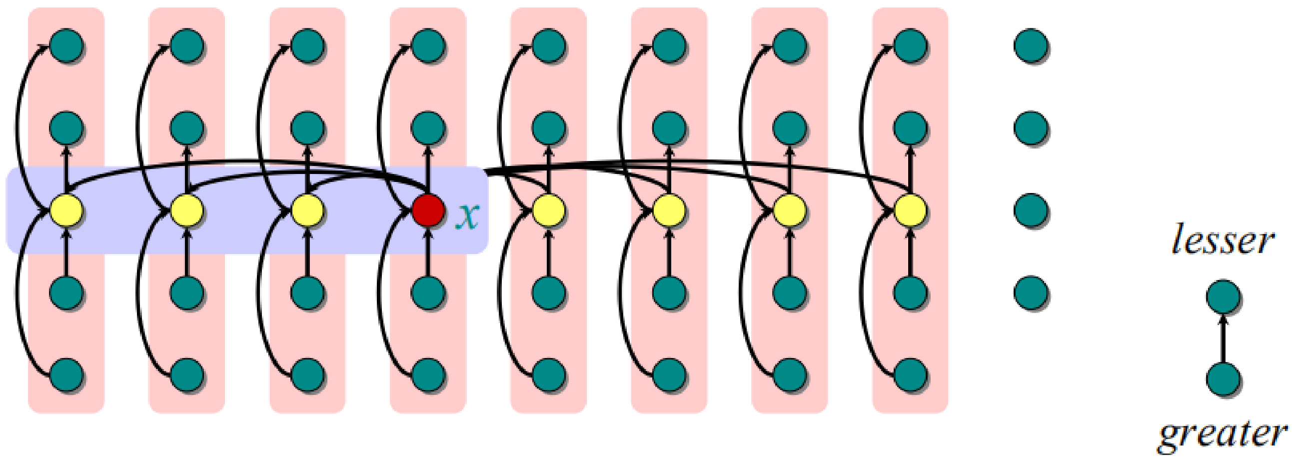
2）用排序算法，将每组中的元素排好序，取出每组的中位数，共⌈n/5⌉个。

2.9 线性时间选择



3）调用**Select**来找出这⌊n/5⌋个中位数的中位数**x**。如果⌊n/5⌋是偶数，就找它的2个中位数中较大的一个。

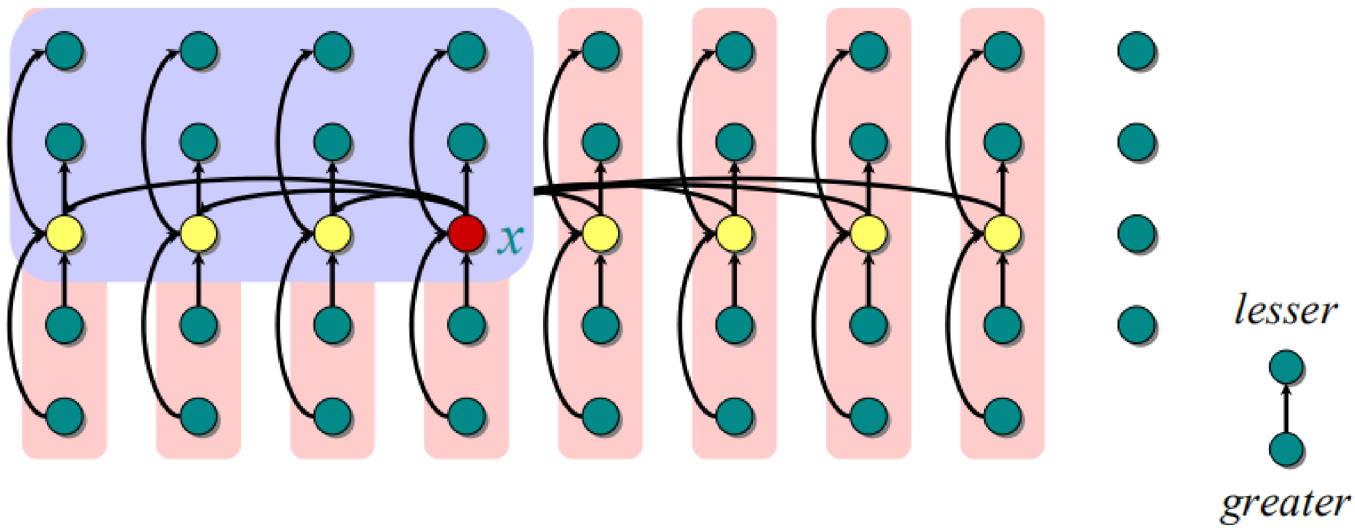
2.9 线性时间选择



最少有一半中位数 ≤ x, 也就是

$$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$$

2.9 线性时间选择

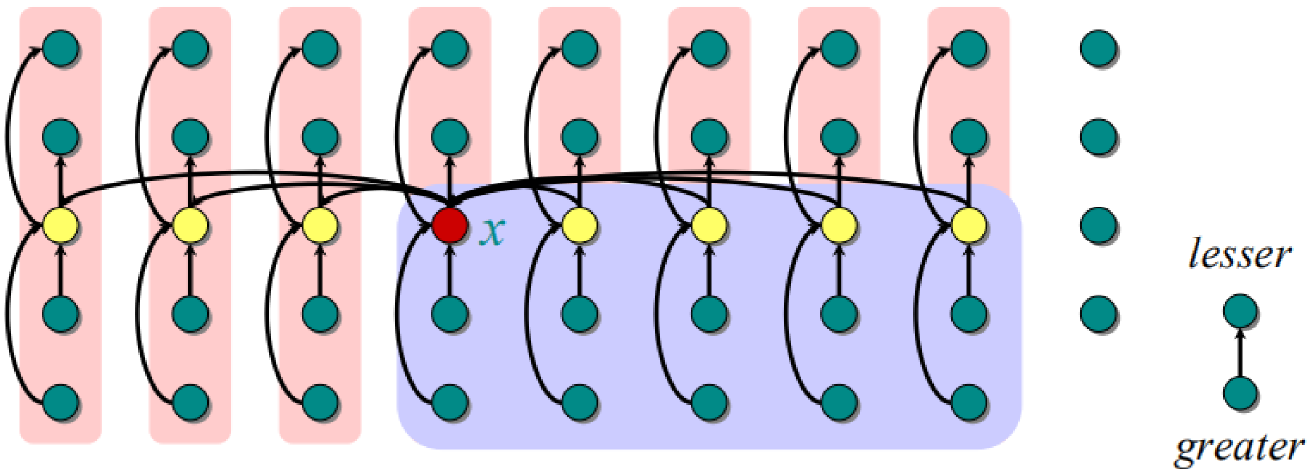


最少有一半中位数 ≤ x, 也就是

$$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$$

因此，3 $\lfloor n/10 \rfloor$ 个数 ≤ x

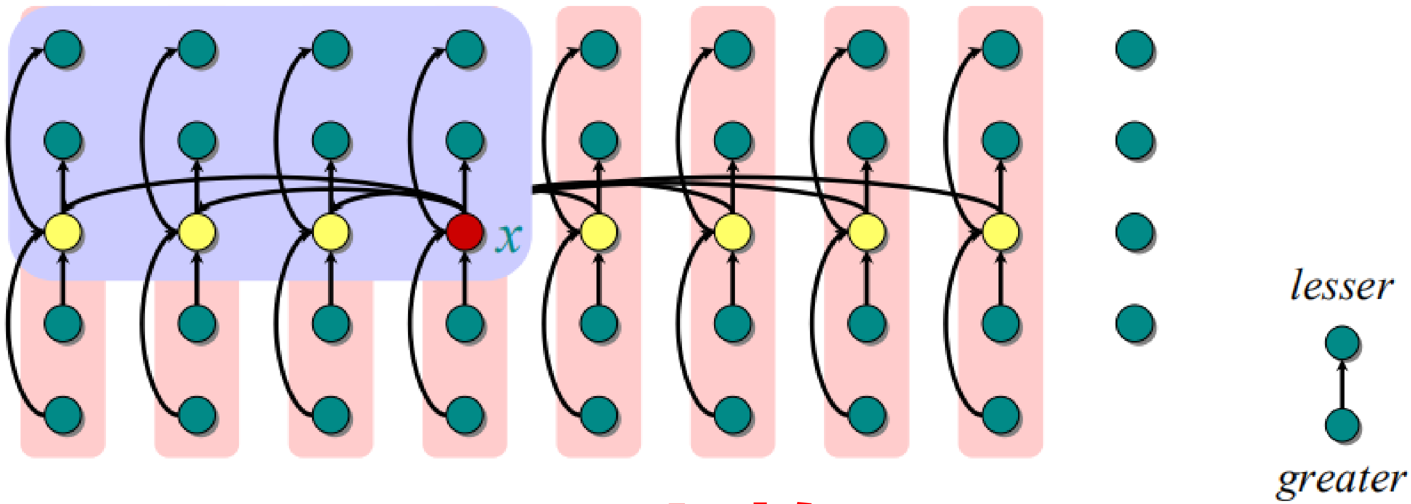
2.9 线性时间选择



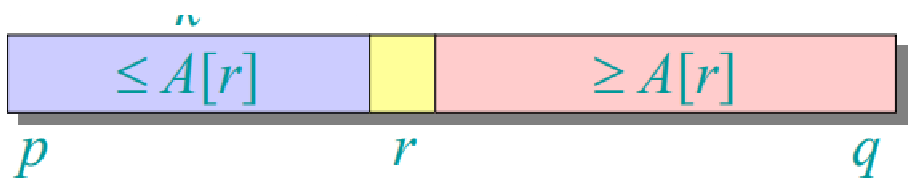
最少有一半中位数 ≤ x, 也就是

$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$

同样地，也有 3 $\lfloor n/10 \rfloor$ 个数 ≥ x



$3\lfloor n/10 \rfloor$ 个数 $\leq x$



4）找到的x对全部 n个元素进行划分，有k-1个数小于x

选取x为主轴这种划分方式的优点?

2.9 线性时间选择

- 对于n≥50，我们有3⌊n/10⌋≥n/4。
  递归调用递归执行步骤4中的SELECT 元素 ≤3n/4

运行时间可以假设步骤4需要时间最坏情况下为T（3n/4）

- 对于n<50，我们知道最坏情况
  时间为 T（n）=Θ（1）

2.9 线性时间选择

计算耗时分析

$T(n)$	SELECT(i, n)
$\Theta(n)$	1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
$T(n/5)$	2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
$\Theta(n)$	3. Partition around the pivot x. Let $k = \text{rank}(x)$.
$T(3n/4)$	4. **if** $i = k$ **then return** x **elseif** $i < k$ **then** recursively SELECT the ith smallest element in the lower part **else** recursively SELECT the $(i-k)$th smallest element in the upper part

复杂度分析

$$T(n) \le \begin{cases} C_1 & n < 75 \\ C_2 n + T(n/5) + T(3n/4) & n \ge 75 \end{cases}$$

T(n)=**O(n)**

归纳法

$$T(n) \le cn$$

$$\begin{aligned} T(n) &\le \frac{1}{5}cn + \frac{3}{4}cn + \Theta(n) \\ &= \frac{19}{20}cn + \Theta(n) \\ &= cn - \left(\frac{1}{20}cn - \Theta(n)\right) \\ &\le cn \end{aligned}$$

,

```
Type Select(Type a[], int p, int r, int k)
{
      if (r-p<75) {
        用某个简单排序算法对数组a[p:r]排序;
        return a[p+k-1];
        }
      for ( int i = 0; i<=(r-p-4)/5; i++ )
         //将a[p+5*i]至a[p+5*i+4]的第3小元素
          与a[p+i]交换位置;
        //找中位数的中位数，r-p-4即上面所说的n-5
        Type x = Select(a, p, p+(r-p-4)/5, (r-p-4)/10);
        int i=Partition(a,p,r, x), j=i-p+1;
        if (k<=j) return Select(a,p,i,k);
        else return Select(a,i+1,r,k-j);
    }
}
```

上述算法将每一组的大小定为5，并选取75作为是否作递归调用的分界点。这2点保证了T(n)的递归式中2个自变量之和n/5+3n/4=19n/20=εn，0<ε<1。这是使T(n)=O(n)的关键之处。当然，除了5和75之外，还有其他选择。

2.9 线性时间选择

如果能在线性时间内找到一个划分基准，使得按这个基准所划分出的2个子数组的长度都至少为原数组长度的ε倍($0<ε<1$是某个正常数)，那么就可以**在最坏情况下**用O(n)时间完成选择任务。

例如，若ε=9/10，算法递归调用所产生的子数组的长度至少缩短1/10。所以，在最坏情况下，算法所需的计算时间T(n)满足递归式$T(n) \le T(9n/10)+O(n)$。由此可得$T(n)=O(n)$。

补充两点：

- 实际上，该算法运行缓慢，因为n前面的常数很大。
- 随机算法比实用。

2.10 最接近点对问题

计算几何学中研究的基本问题之一。

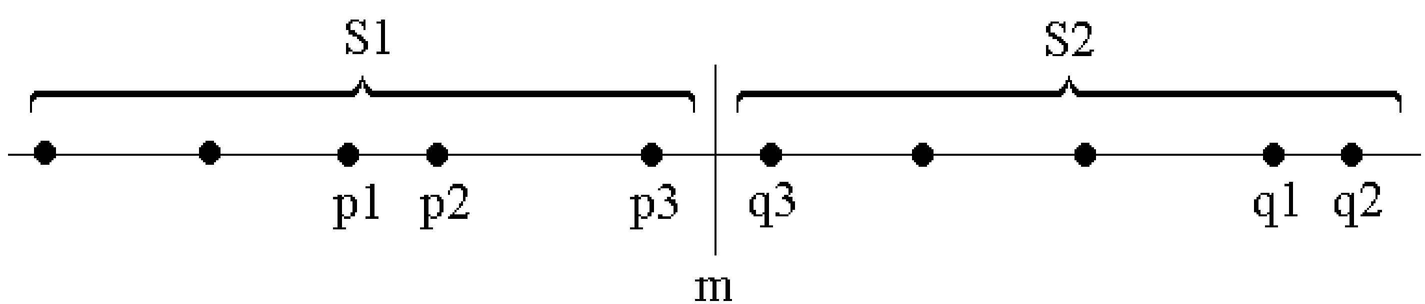
在涉及几何对象的问题中，常需要了解其邻域中其他几何对象的信息。例如，在空中交通控制问题中，若将飞机作为空间中移动的一个点来看待，则具有最大碰撞危险的两架飞机，就是这个空间中最接近的一对点。

最接近点问题：给定平面上的*n*个点，找其中的一对点，使得在*n*个点组成的所有点对中，该点距离最小

2.10 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，该点对间的距离最小。

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。

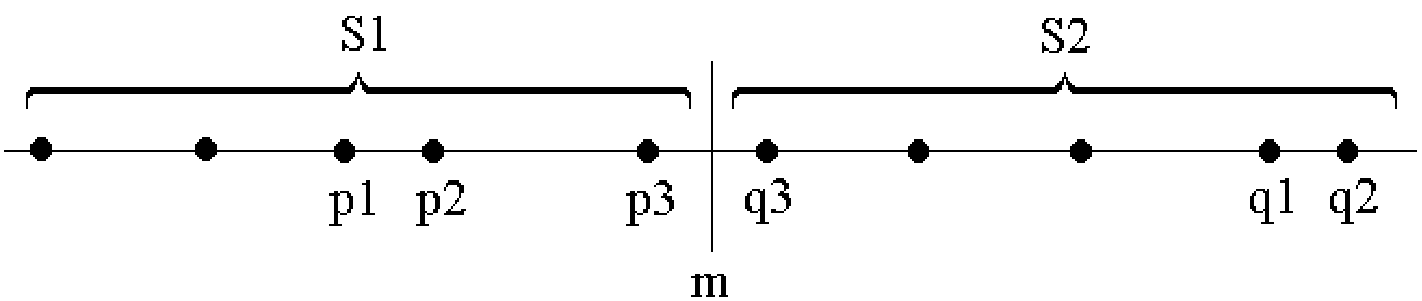


2.10 最接近点对问题

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。

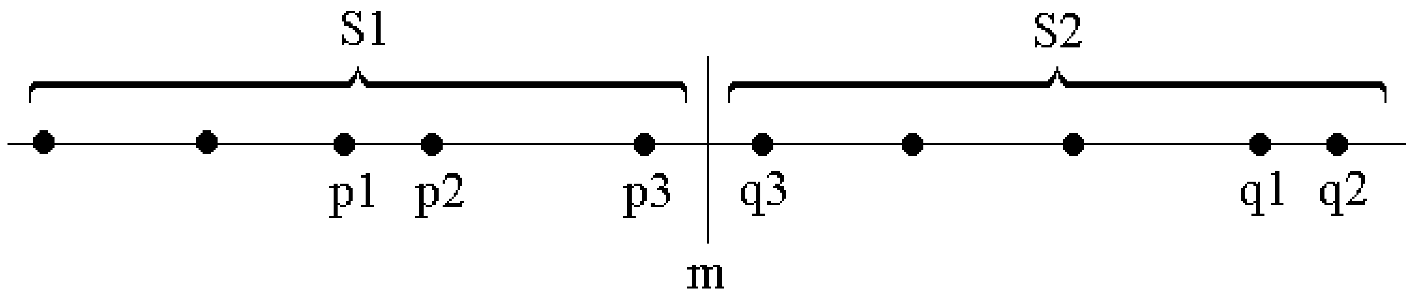
递归地在S1和S2上找出其最接近点对{p1,p2}和{q1,q2}，并设**d=min{|p1-p2|,|q1-q2|}**，S中的最接近点对或者是{p1,p2}，或者是{q1,q2}，或者是某个{p3,q3}，其中p3∈S1且q3∈S2。

能否在线性时间内找到p3,q3?



为什么不用均值

2.10 最接近点对问题

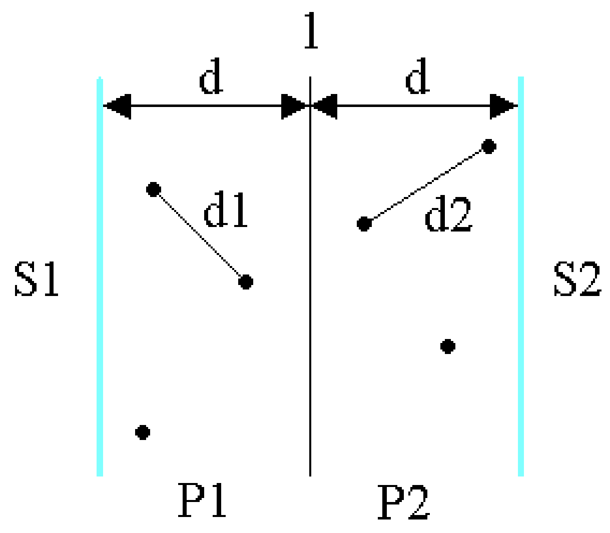


能否在线性时间内找到p3,q3?

- **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。**

2.10 最接近点对问题

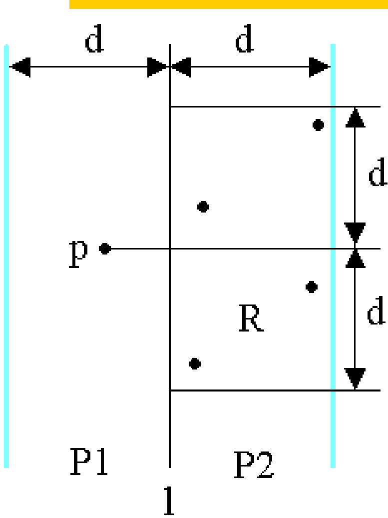
- 下面来考虑二维的情形。
  - 选取一垂直线**l:x=m**来作为分割直线。其中m为S中各点x坐标的中位数。由此将S分割为S1和S2。
  - 递归地在S1和S2上找出其最小距离d1和d2，并设**d=min{d1,d2}**，S中的最接近点对或者是d，或者是某个{p,q}，其中p∈S1且q∈S2。
  - **能否在线性时间内找到p,q?**

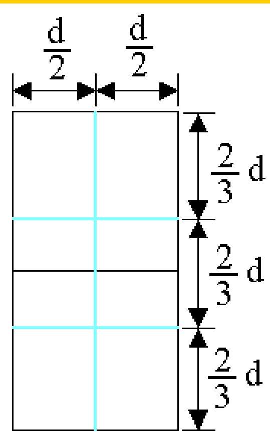


2.10 最接近点对问题

能否在线性时间内找到p,q?

- 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**最多只需要检查6×n/2=3n个候选者**





(a)

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

$$(x(u)-x(v))^2+(y(u)-y(v))^2 \le (d/2)^2+(2d/3)^2=\frac{25}{36}d^2$$

distance(u,v)<d。这与d的意义相矛盾。

2.10 最接近点对问题

- 为了确切地知道要检查哪6个点，可以将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。由上面的分析可知，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。

2.10 最接近点对问题

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

复杂度分析

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

T(n)=O(nlogn)

2.11 循环赛日程表

有n=2^k名选手，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。

Day1	Day2	…	…	…	Day(n-3)	Day(n-2)	Day(n-1)
case1	case1				case1	case1	case1
case2	case2				case2	case2	case2
…	…				…	…	…

日程表直观设计如上，但为了便于计算，可以对数据结构进行简化

2.11 循环赛日程表

有n=2^k名选手，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。

将日程表设计成 n 行 n 列的表，在表中第 i 行、第 j+1 列处填入第 i 个选手在第 j 天所遇到的对手（第一列是运动员编号）。

运动员编号	Day1	Day2	Day3	…	…	Day(n-2)	Day(n-1)
1							
2							
…							
n							

2.11 循环赛日程表

有n=2^k名选手，设计一个满足要求的比赛日程表：

递归关系分析：

- 把n名选手分成两部分：1～n/2 和 (n/2+1)～n；
- 1～n/2日程表A：由数字1～n/2组成的(n/2)×(n/2)表格；
- (n/2+1)～n日程表B：由数字(n/2+1)～n组成的(n/2)×(n/2)表格；
- n名选手日程表是一个n×n的表格，把行和列分别分成两半，可以得到四个(n/2)×(n/2)的表格，记为L1，R1，L2，R2，其中L1与A完全一致，L2与B完全一致；

L1	R1
L2	R2

2.11 循环赛日程表

有n=2^k名选手，设计一个满足要求的比赛日程表：

递归关系分析：

- R1是n/2+1～n组成的(n/2)×(n/2)表格，可以与L2完全一致，因此R2与L1完全一致；

L1	R1
L2	R2

递归终结状态（n=2即k=1）：

n=2时设编号分别为n1，n2，日程表如下：

n1	n2
n2	n1

2.11 循环赛日程表

有n=2^k名选手，设计一个满足要求的比赛日程表：

递归关系分析：

- R1是n/2+1～n组成的(n/2)×(n/2)表格，可以与L2完全一致，因此R2与L1完全一致；

L1	R1
L2	R2

递归终结状态（n=2即k=1）：

n=2时设编号分别为n1，n2，日程表如下：

n1	n2
n2	n1

2.11 循环赛日程表

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。

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。递归地用对选手进行分割，直到只剩下2个选手时，比赛日程表的制定就变得很简单。这时只要让这2个选手进行比赛就可以了。

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

分治算法的分析技术

分治策略的算法分析工具：递推方程

两类递推方程

$$f(n)=\sum_{i=1}^{k}a_i f(n-i)+g(n)$$

$$f(n)=af(\frac{n}{b})+d(n)$$

求解方法

第一类方程：迭代法、换元法、递归树、尝试法

第二类方程：迭代法、递归树、主定理

递推方程的解

方程 $T(n) = aT(n/b) + d(n)$

$d(n)$为常数

$$T(n) = \begin{cases} O(n^{\log_b a}) & a \neq 1 \\ O(\log n) & a = 1 \end{cases}$$

$d(n) = cn$

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

改进分治法的途径

分析问题时候增加预处理

(1) 通过代数变换减少子问题的个数；

例：大整数乘法，Strassen矩阵相乘

$$XY = ac\ 2^n + (ad+bc)\ 2^{n/2} + bd$$

$$XY = ac\ 2^n + ((a-b)(d-c)+ac+bd)\ 2^{n/2} + bd$$

(2) 通过预处理减少递归内部的计算量；

例：最接近点对

$n^2/4$ **$6\times n/2=3n$**

作业

1.一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。答案需要取模 1e9+7(1000000007)，如计算初始结果为：1000000008，请返回 1。

https://leetcode-cn.com/problems/qing-wa-tiao-tai-jie-wen-ti-lcof/

2. 至少有 K 个重复字符的最长子串：

给你一个字符串 s 和一个整数 k ，请你找出 s 中的最长子串，要求该子串中的每一字符出现次数都不少于 k 。返回这一子串的长度。

https://leetcode-cn.com/problems/longest-substring-with-at-least-k-repeating-characters/

作业

3. 最小的K个整数：输入整数数组 arr ，找出其中最小的 k 个数。例如，输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。
备注：使用线性时间选择方法完成

https://leetcode-cn.com/problems/zui-xiao-de-kge-shu-lcof/

4. 教材P36 算法分析2-7；5.算法分析2-15

6. 教材P44 算法实现2-6；7.算法实现2-9

下周三前发送至邮箱 dawei_course@163.com，作业以 “学号_姓名_算法第2次作业”为主题命名邮件，附件名“2010110_张三_算法第二次作业.pdf”所有题目解答合并为一个pdf文件