第三章 分治法

✓目录

- 概述
- 二分查找
- 循环赛日程表
- 合并排序
- 快速排序

教学目标

- 掌握分治法的基本思想和求解步骤
- 理解分治法的精髓，即如何分？如何治？才能使得算法效率更高
- 通过实例学习，掌握运用分治法来解决实际问题的方法

学习分治法的意义

- 任何一个可以用计算机求解的问题所需的计算时间都与其规模有关：问题的规模越小，越容易直接求解。

- 要想直接解决一个规模较大的问题，有时是很困难的。那么，为了更好地解决这些规模较大的问题，分治法应运而生了。

- 在计算机科学中，分治法是一种很重要的算法。它采取各个击破的技巧来解决一个规模较大的问题，该技巧是很多高效算法的基础，如二分查找，排序算法(快速排序，归并排序)等。

分治法的基本思想

- 基本思想
  - 将一个难以直接解决的大问题，分解成一些规模较小的相同问题，以便各个击破，分而治之。
- 何时能、何时应该采用分治法来解决问题呢？
  - 避免：将一个规模n的问题分解为两个或多个规模几乎也为n的问题。
  - 避免：将一个规模为n的问题分解为几乎n个规模为n/c的问题。

分治法的解题步骤

第1步：分解

即将问题分解为若干个规模较小、相互独立、与原问题形式相同的子问题；

第2步：治理

步骤2-1：求解各个子问题

步骤2-2：合并

复杂度分析

- 总结公式

$$T(n)=\begin{cases} c & n=1 \\ aT(n/b)+cn^k & n>1 \end{cases}$$

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

二分查找

- 问题描述
  - 二分查找又称为折半查找，它要求待查找的数据元素必须是按关键字大小有序排列的。问题描述：给定已排好序的n个元素s1,…,sn，现要在这n个元素中找出一特定元素x。
  - 首先较容易想到使用顺序查找方法，逐个比较s1,…,sn，直至找出元素x或搜索遍整个序列后确定x不在其中。显然，该方法没有很好地利用n个元素已排好序这个条件。因此，在最坏情况下，顺序查找方法需要O(n)次比较。

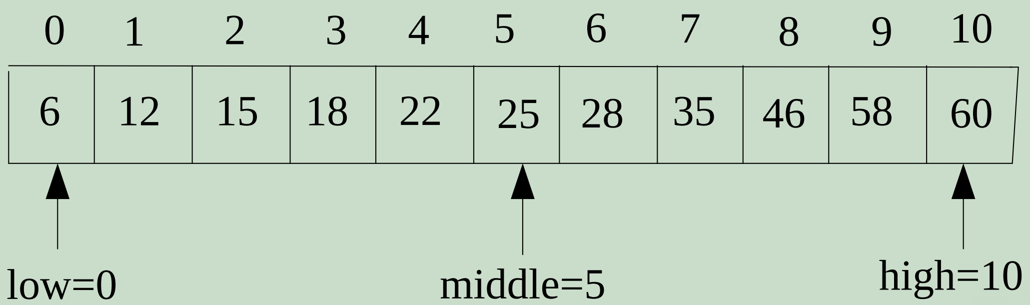
算法思想

- 假定元素序列已经由小到大排好序，将有序序列分成规模大致相等的两部分，然后取中间元素与特定查找元素x进行比较，如果x等于中间元素，则算法终止；如果x小于中间元素，则在序列的左半部继续查找，即在序列的左半部重复分解和治理操作；否则，在序列的右半部继续查找，即在序列的右半部重复分解和治理操作。可见，二分查找算法重复利用了元素间的次序关系。

算法设计

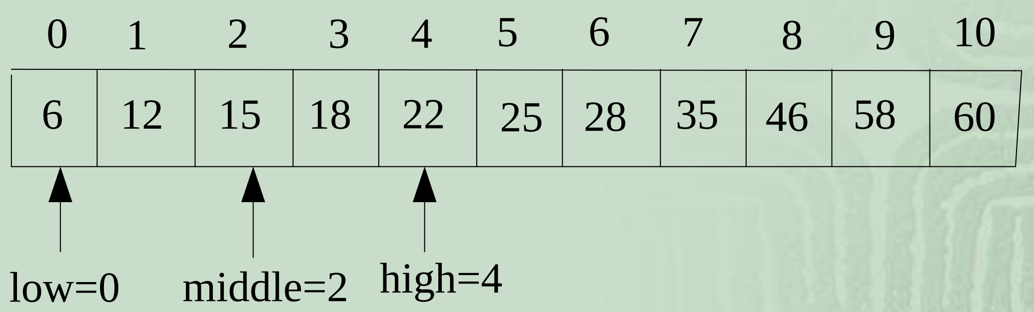
- 步骤1：确定合适的数据结构。设置数组s[n]来存放n个已排好序的元素；变量low和high表示查找范围在数组中的下界和上界；middle表示查找范围的中间位置；x为特定元素；
- 步骤2：初始化。令low=0；high=n-1；
- 步骤3：判定low小于等于high是否成立，如果成立，转步骤5；否则，算法结束；
- 步骤4：middle=(low+high)/2，即指示中间元素；
- 步骤5：判断x与s[middle]的关系。如果x==s[middle]，算法结束；如果x>s[middle]，则令low=middle+1；否则令high=middle-1，转步骤3。

构造实例



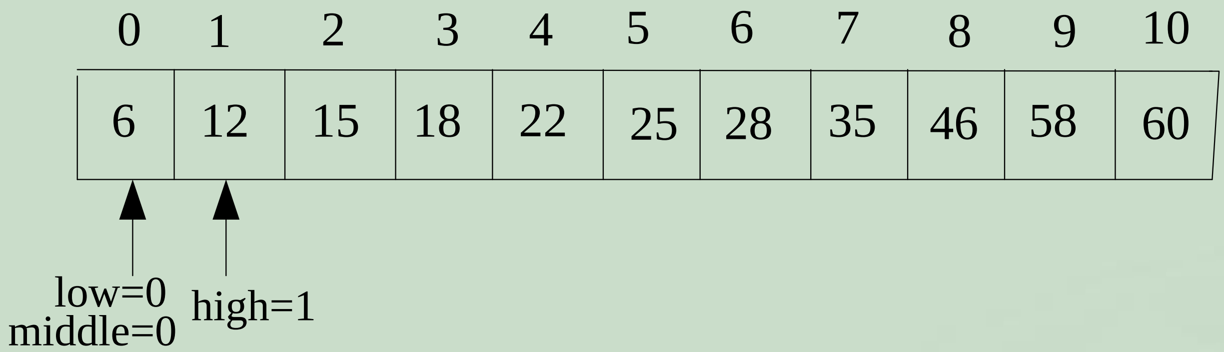
0 1 2 3 4 5 6 7 8 9 10
6 12 15 18 22 25 28 35 46 58 60
low=0
middle=5
high=10

（1）

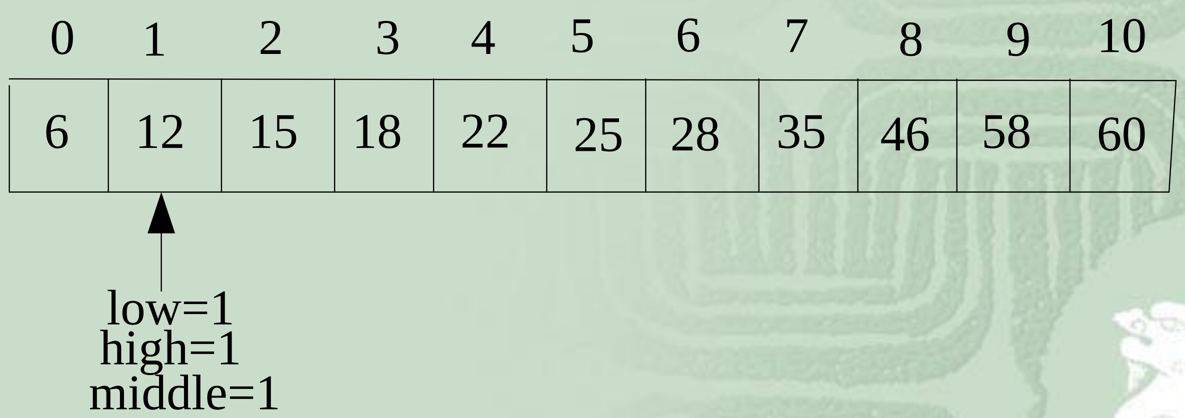


0 1 2 3 4 5 6 7 8 9 10
6 12 15 18 22 25 28 35 46 58 60
low=0
middle=2
high=4

（2）


0 1 2 3 4 5 6 7 8 9 10
6 12 15 18 22 25 28 35 46 58 60
low=0
middle=0
high=1

（3）


0 1 2 3 4 5 6 7 8 9 10
6 12 15 18 22 25 28 35 46 58 60
low=1
high=1
middle=1

（4）

算法描述——非递归形式

```
int NBinarySearch(int n, int s[n], int x)
{     int low=0, high=n-1, middle=0;
      while(low<=high)
      {     middle=(low+high)/2;
            if(x==s[middle])  return middle;
            else if(x>s[middle]) low=middle+1;
            else high=middle-1;
      }
      return -1;
}
```

算法描述——递归形式

```
int BinarySearch(int s[n], int x, int low, int high)
{    if (low>high) return -1;
     int middle=(low+high)/2;
     if(x==s[middle])  return middle;
     else if(x>s[middle])
          return BinarySearch (s, x, middle+1, high);
     else
          return BinarySearch (s, x, low, middle-1);
}
```

算法分析——时间复杂度

- 设给定的有序序列中具有n个元素。
- 显然，当n=1时，查找一个元素需要常量时间，因而T(n)=O(1)。
- 当n>1时，计算序列的中间位置及进行元素的比较，需要常量时间O(1)。递归地求解规模为n/2的子问题，所需时间为T(n/2)。
- 因此，二分查找算法所需的运行时间T(n)的递归形式为：
- 当n>1时，$T(n)=T(n/2)+O(1)$
- $=\ldots\ldots$
- $=T(n/2^x)+xO(1)$
- 简单起见，令$n=2^x$，则$x=\log n$。
- 由此，$T(n)=T(1)+\log n=O(1)+O(\log n)$。因此，二分查找算法的时间复杂性为$O(\log n)$。

算法分析——空间复杂度

```
int BinarySearch(int s[n], int x, int low, int high)
{   if (low>high) return -1;
    int middle=(low+high)/2;
    if(x==s[middle])
        return middle;
    else if(x>s[middle])
        low=middle+1;
    else
        high=middle-1;
    return BinarySearch (s, x, low, high);
}
```

循环赛日程表

问题描述

- 设有$n=2^k$个运动员要进行羽毛球循环赛，现要设计一个满足以下要求的比赛日程表：

（1）每个选手必须与其它n-1个选手各赛一次；

（2）每个选手一天只能比赛一次；

（3）循环赛一共需要进行n-1天。

- 由于$n=2^k$，显然n为偶数。

分治法求解思路

- （1）如何分，即如何合理地进行问题的分解？
  - 一分为二
- （2）如何治，即如何进行问题的求解？
  - 进行合并
- （3）问题的关键——发现循环赛日程表制定过程中存在的规律性。

实例构造

- 【**例3-2**】n=2^1个选手的比赛日程表的制定。

天数 编号	1
1	2
2	1

【例3-3】n=2^2个选手的比赛日程表的制定

表3-2子问题的比赛日程表

天数 / 编号	1
1	2
2	1
3	4
4	3

表3-3 2^2个选手的比赛日程表

天数 / 编号	1	2	3
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

【例3-4】n=2^2个选手的比赛日程表的制定

表3-4 4个子问题的比赛日程表

编号＼天数	1
1	2
2	1
3	4
4	3
5	6
6	5
7	8
8	7

表3-5 子问题解的合并

编号＼天数	1	2	3
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1
5	6	7	8
6	5	8	7
7	8	5	6
8	7	6	5

【例3-4】n=22个选手的比赛日程表的制定

表3-6 2^3个选手的比赛日程表

编号＼天数	1	2	3	4	5	6	7
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

比赛安排的思考

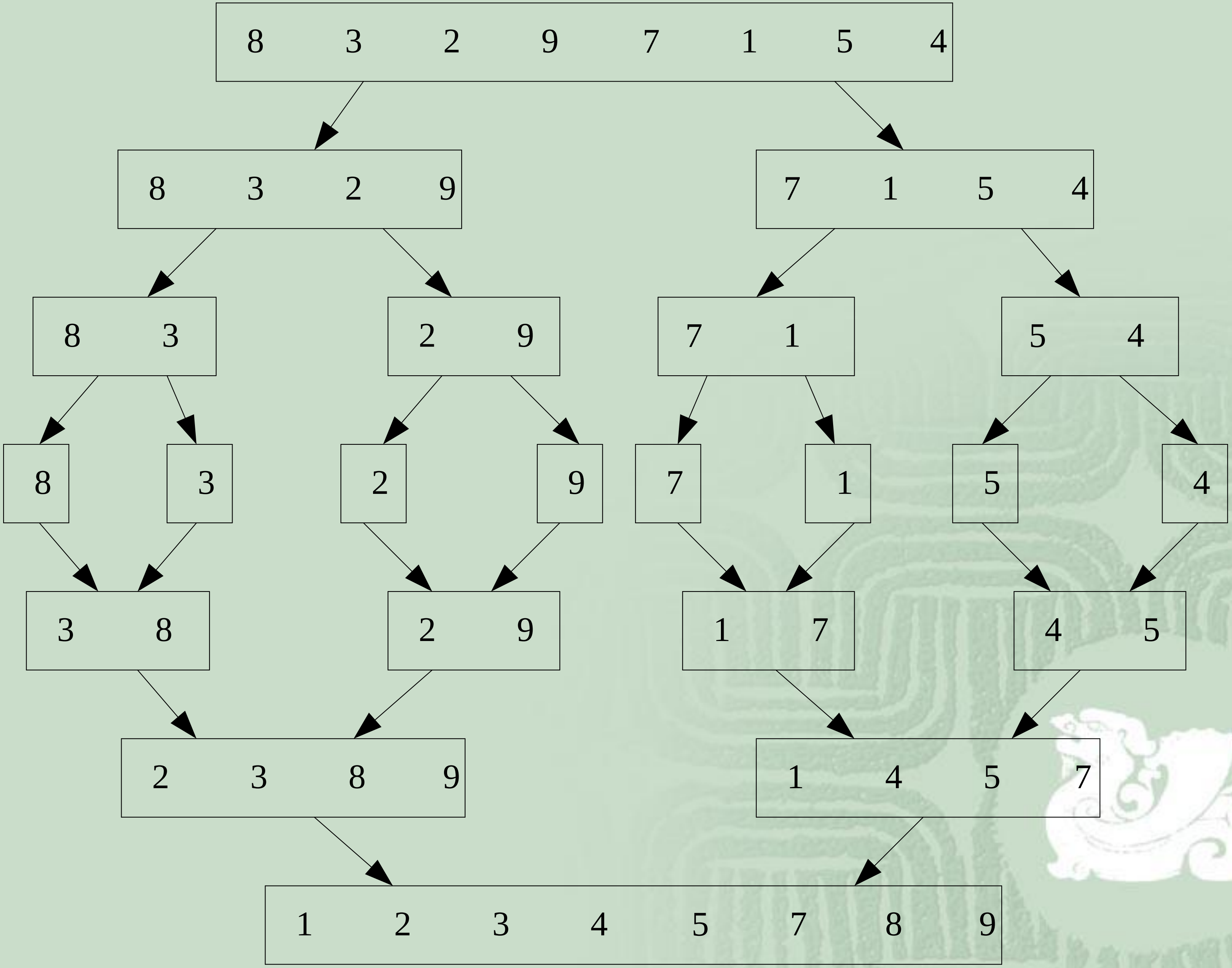
- 比如NBA赛程安排，某些时候故意安排两只队伍比赛，如何实现？

	第1天	第2天	第3天
1	3	4	2
?			
?			
?			

合并(归并)排序

- 算法思想
  - 合并排序是采用分治策略实现对n个元素进行排序的算法，是分治法的一个典型应用和完美体现。它是一种平衡、简单的二分分治策略，其计算过程分为三大步：
  - （1）分解：将待排序元素分成大小大致相同的两个子序列。
  - （2）求解子问题：用合并排序法分别对两个子序列递归地进行排序。
  - （3）合并：将排好序的有序子序列进行合并，得到符合要求的有序序列。

构造实例


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

算法设计

- 合并过程
  - 合并排序的关键步骤在于如何合并两个已排好序的有序子序列。为了进行合并，引入一个辅助过程**Merge(A,low,middle,high)**，该过程将排好序的两个子序列**A[low:middle]**和**A[middle+1:high]**进行合并。其中，**low**、**high**表示待排序范围在数组中的上界和下界，**middle**表示两个序列的分开位置，满足**low≤middle＜high**；由于在合并过程中可能会破坏原来的有序序列，因此，合并最好不要就地进行，本算法采用了辅助数组**B[low:high]**来存放合并后的有序序列。

算法设计

- 合并方法
  - 设置三个工作指针i，j，k。其中，i和j指示两个待排序序列中当前需比较的元素，k指向辅助数组B中待放置元素的位置。比较A[i]和A[j]的大小关系，如果A[i]小于等于A[j]，则B[k]=A[i]，同时将指针i和k分别推进一步；反之，B[k]=A[j]，同时将指针j和k分别推进一步。如此反复，直到其中一个序列为空。最后，将非空序列中的剩余元素按原次序全部放到辅助数组B的尾部。

算法设计

	low			mid				high
A	2	3	8	9	1	4	5	7
	i				j			
B								
	k							

算法描述

```
void Merge(int A[ ]，int low，int middle，int high)
{    int i, j, k; int *B=new int[high-low+1];
     i=low; j=middle+1; k=0;
     while( i<=middle  &&  j<=high )  //两个子序列非空
          if(A[i]<=A[j])  B[k++]=A[i++];
          else  B[k++]=A[j++];
     while (i<=middle) B[k++]=A[i++];
     while (j<=high) B[k++]=A[j++];
     k=0;
     for(i=low;i<=high;i++) A[i++]=B[k++];
}
```

算法描述——递归形式

```
void MergeSort (int A[ ]，int low，int high)
{      int middle;
       if (low<high)
       {     middle=(low+high)/2;   //取中点
             MergeSort(A, low, middle);
             MergeSort(A, middle+1, high);
             Merge(A, low, middle, high); //合并
       }
}
```

算法分析

- 当**n=1**时，**T(n)=O(1)**。
- 当**n>1**时，将时间**T**如下分解：
  - 分解：这一步需要常量时间**O(1)**。
  - 解决子问题：递归求解两个规模为**n/2**的子问题，所需时间为**2T(n/2)**。
  - 合并：**Merge**算法可在**O(n)**时间内完成。
- 得到合并排序算法运行时间**T(n)**的递归形式为：

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

算法分析

- 求得T(n)=nT(1)+nlogn=n+nlogn，即合并排序算法的时间复杂性为O(nlogn)。
- 算法所使用的工作空间取决于Merge算法，每调用一次Merge算法，便分配一个适当大小的缓冲区，退出Merge便释放它。在最后一次调用Merge算法时，所分配的缓冲区最大，需要O(n)个工作单元。所以，合并排序算法的空间复杂性为O(n)。

快速排序

- 算法思想
  - 通过一趟扫描将待排序的元素分割成独立的三个序列：第一个序列中所有元素均不大于基准元素、第二个序列是基准元素、第三个序列中所有元素均不小于基准元素。由于第二个序列已经处于正确位置，因此需要再按此方法对第一个序列和第三个序列分别进行排序，整个排序过程可以递归进行，最终可使整个序列变成有序序列。

∈

划分方法的构造实例

49 38 65 97 76 13 27

i j

图3-6 划分初始状态

27 38 65 97 76 13 49

i j

图3-7 1次交换后的状态

27 38 65 97 76 13 49

i j

图3-8 i后移1位后的状态

27 38 49 97 76 13 65

i j

图3-9 2次交换后的状态

27　38　13　97　76　49　65

i　j

图3-10 3次交换后的状态

27　38　13　49　76　97　65

i　j

图3-11 4次交换后的状态

27　38　13　49　76　97　65

i j

图3-12 j前移1位后的状态

快速排序算法的分治策略体现

- 分解
  - 在**R[low:high]**中选定一个元素作为基准元素**(pivot)**，以此基准元素为标准将待排序序列划分为两个子序列并使序列**R[low : pivotpos-1]**中所有元素的值均小于等于**R[pivotpos]**，序列**R[pivotpos+1 : high]**中所有元素的值均大于等于**R[pivotpos]**。
- 求解子问题
  - 对子序列**R[low : pivotpos-1]**和**R[pivotpos+1 : high],** 分别通过递归调用快速排序算法来进行排序。
- 合并
  - 就地排序。

基准元素的选取

- （a）取第一个元素。
- （b）取最后一个元素。
- （c）取位于中间位置的元素。
- （d）“三者取中的规则”。
- （e）取位于low和high之间的随机数。

划分方法——过程设计

- 假设待排序序列为R[low:high]，该划分过程以第一个元素作为基准元素。
- 步骤1：设置两个参数i和j，i=low，j=high；
- 步骤2：选取R[low]作为基准元素，并将该值赋给变量pivot;
- 步骤3：令j自j位置开始向左扫描，如果j位置所对应的元素的值大于等于pivot，则j前移一个位置(即j--)。重复该过程，直至找到第1个小于pivot的元素R[j]，将R[j]与R[i]进行交换，i++。其实，交换后R[j]所对应的元素就是pivot。
- 步骤4：令i自i位置开始向右扫描，如果i位置所对应的元素的值小于等于pivot，则i后移一个位置(即i++)。重复该过程，直至找到第1个大于pivot的元素R[i]，将R[j]与R[i]进行交换，j--。其实，交换后R[i]所对应的元素就是pivot。
- 步骤5：重复步骤3、4，交替改变扫描方向，从两端各自往中间靠拢直至i=j。此时i和j指向同一个位置，即基准元素pivot的最终位置。

快速排序的算法描述

```
int Partition(int R[ ]， int low， int high)
{     int i=low, j=high, pivot=R[low]；
      while( i<j )
      {     while( i<j && R[j]>=pivot )    j--；
            if( i<j )
                    swap( R[i++], R[j] );
            while( i<j && R[j]<=pivot )   i++；
            if( i<j )
                    swap( R[i], R[j--] );
      }
      return j;
 }
```

快速排序的算法描述

```
void QuickSort(int R[ ]，int low，int high)
{    int pivotpos；
     if(low<high)
     {    pivotpos=Partition(R, low, high)；
          QuickSort(R, low, pivotpos-1)；
          QuickSort(R, pivotpos+1, high)；
      }
 }
```

算法分析

- 最坏情况：**O(n²)**　$T(n)=\begin{cases} O(1) & n=1 \\ T(n-1)+O(n) & n>1 \end{cases}$
- 最好情况：**O(nlogn)**　$T(n)=\begin{cases} O(1) & n=1 \\ 2T(n/2)+O(n) & n>1 \end{cases}$
- 平均情况：**O(nlogn)**　$T(n)=\frac{1}{n}\sum_{k=1}^{n}(T(n-k)+T(k-1))+n$

$$T(n) = \frac{2}{n}\sum_{k=1}^{n} T(k) + n$$

矩阵乘积

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j}$$

$$\begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix} \times \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix} = \begin{bmatrix} 2\times5+3\times6 & 2\times7+3\times8 \\ 4\times5+1\times6 & 4\times7+1\times8 \end{bmatrix} = \begin{bmatrix} 28 & 38 \\ 26 & 36 \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \qquad \text{for } 1 \le i,j \le n$$

矩阵乘积

```
void matrixmult (int n, const number A[ ][ ],
                        const number B[ ][ ],
                        number C[ ][ ])
{     index i, j, k;

      for (i=1; j<= n; j++)
          for (j=1; j<= n; j++) {
              C[ i ] [ j ] = 0;
              for (k=1; k<= n; k++)
                  C[i][j] = C[i][j] + A[i][k] * B[k][j];
          }
}
```

$$T(n) \in \theta(n^3)$$

矩阵乘积

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$m_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$m_2 = (a_{21} + a_{22})\,b_{11}$$
$$m_3 = a_{11}\,(b_{12} - b_{22})$$
$$m_4 = a_{22}\,(b_{21} - b_{11})$$
$$m_5 = (a_{11} + a_{12})\,b_{22}$$
$$m_6 = (a_{21} - a_{11})(b_{11} + b_{12})$$
$$m_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$

8次乘法 & 4次加法

↓

7次乘法 & 18次加/减法

$$C = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

矩阵乘积

$$A_{11} = \begin{bmatrix} a_{11} & a_{12} \cdots a_{1,n/2} \\ a_{21} & a_{22} \cdots a_{2,n/2} \\ & \vdots \\ a_{n/2,1} & \cdots a_{n/2,n/2} \end{bmatrix}$$

$$\begin{array}{cc} & \overleftrightarrow{n/2} \\ n/2 & \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right] \end{array} = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \times \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$$

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$

矩阵乘积

```
void strassen (int n,  matrix A,  matrix B,  matrix& C)
{
      if (n<=threshold)
            compute C = A x B using the standard algorithm;
      else
      {     partition A into four submatrices A11, A12, A21, A22;
            partition B into four submatrices B11, B12, B21, B22;
            strassen (n/2, A11 + A22, B11 + B22, M1);
            strassen ( …… , M2);
            ……
            strassen ( …… , M7);
            compute C = f(M1, M2, M3, M4, M5, M6, M7);
       }
}
```

矩阵乘积

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$
$$T(1) = 0.$$

$$T(n) = 6n^{\lg 7} - 6n^2 \approx 6n^{2.81} - 6n^2 \in \Theta\left(n^{2.81}\right)$$

矩阵乘积的发展历史


ω
3.0
2.9
2.8
2.7
2.6
2.5
2.4
naive
Strassen
Pan
Bini et al.
Schönhage
Romani
Coppersmith, Winograd
Strassen
Coppersmith, Winograd
Stothers
Williams
1950
1960
1970
1980
1990
2000
2010
Year

思考题

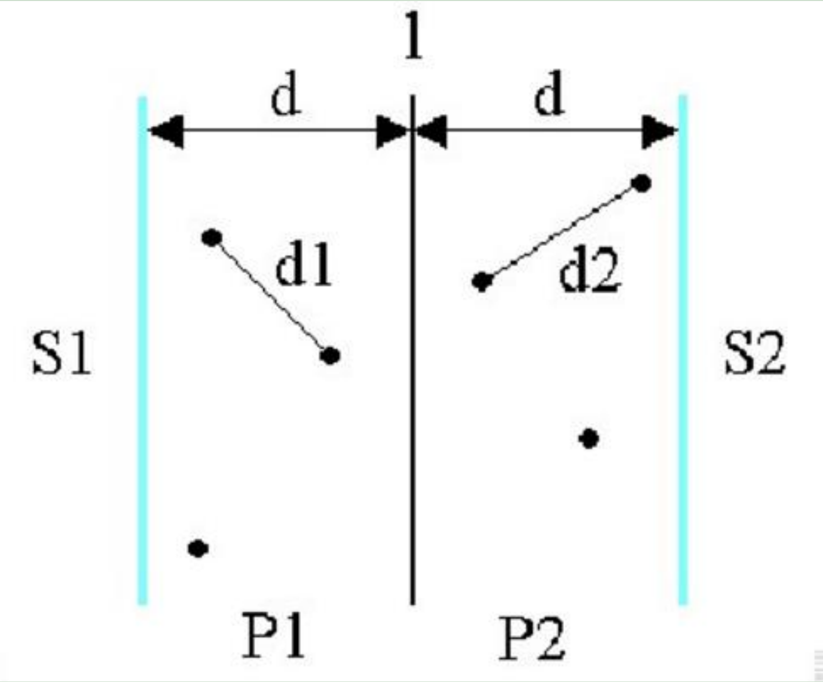
- 电话线被战火炸断了，通讯兵如何快速地找到故障点。
- 如何让卧底在弥留之际告诉探长毒品交易的地点在地图的什么位置。
- 在一堆硬币中存在一个假币，设假币比真币轻。提供一个天平，用什么策略能够最快找出假币，考虑时间复杂度。
- 若二分查找法变为三分查找法、四分查找法，时间复杂度有什么变化？
- 如何用分治法求数组中的最大数，分析时间复杂度。
- 计算求解汉诺塔问题的时间复杂度。

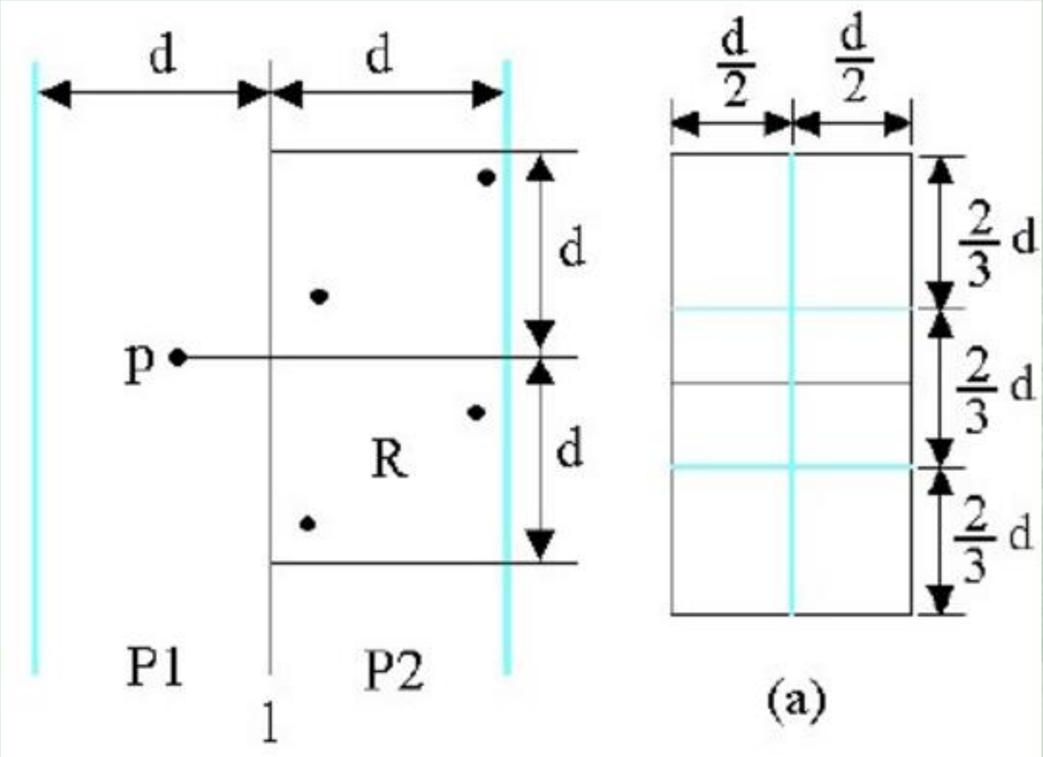
思考题

- 如何求一个数组中第k大的数，分析其时间复杂度。如何用分治法来求，时间复杂度如何？
- 如何求一个数组中的众数，分析其时间复杂度。如何用分治法来求，时间复杂度如何？
- 用分治法求一组数的全排列，分析时间复杂度。
- 用分治法求解一维空间上n个点的最近对问题。

思考题

- 二维最近点对问题
  - 鸽舍原理（抽屉原理）





p最多与R区内的6个点进行比较