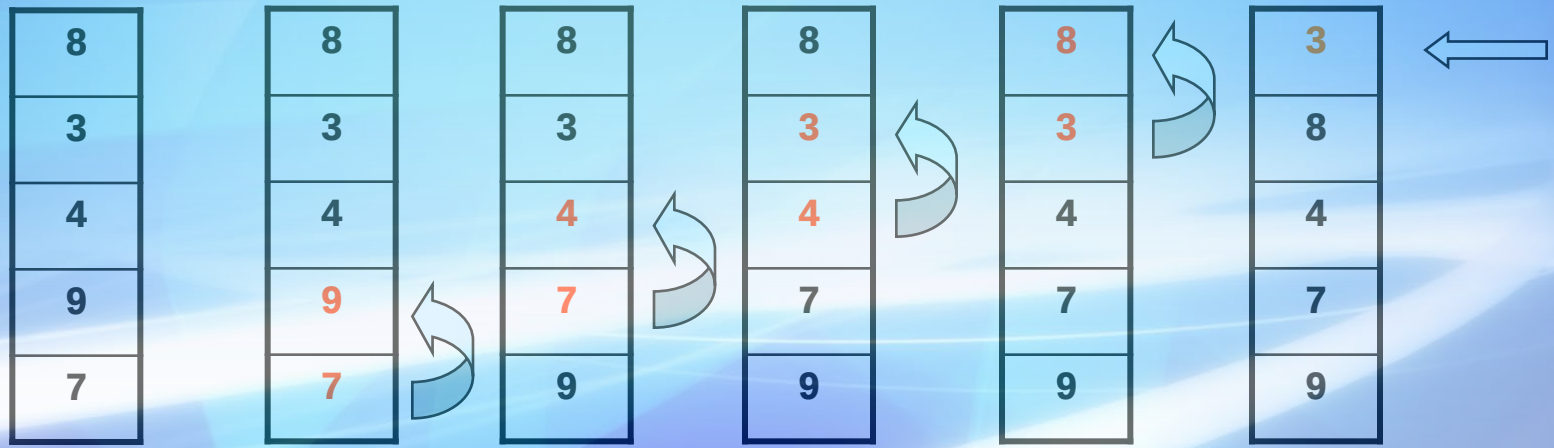
交换排序

《数据结构》
指院网络工程教研中心　陈卫卫



交换排序

学习目标和要求

1、准确复述交换排序的基本思想

2、编写冒泡排序的算法

3、知道算法的时间复杂性和稳定性

交换排序（exchange sort）

逆序： i<j时，a[i]>a[j]

交换排序的基本思想

若i<j，而a[i]>a[j]，则交换a[i]与a[j]。当对任何i<j，a[i]<=a[j]时，排序就完成。

交换排序

冒 泡 排 序

快 速 排 序

冒泡排序

冒泡排序的基本原理：反复扫描待排序列，若相邻元素构成逆序，则交换它们，直至无逆序为止。

下降法

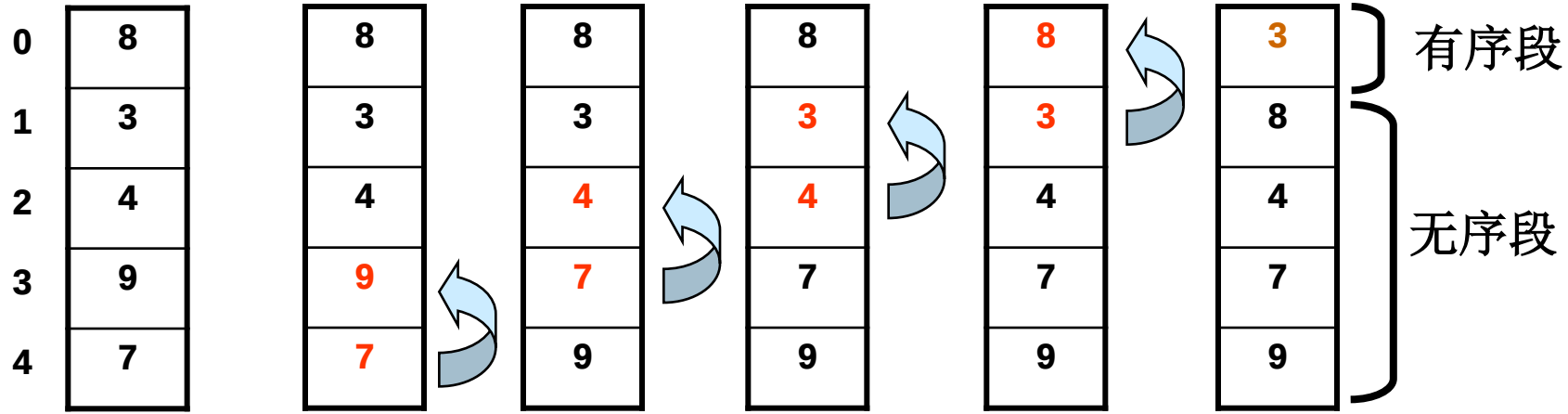
自上而下地扫描，最大元素下降到底部

上升法

自下而上地扫描，最小元素上升到数组顶部

冒泡排序

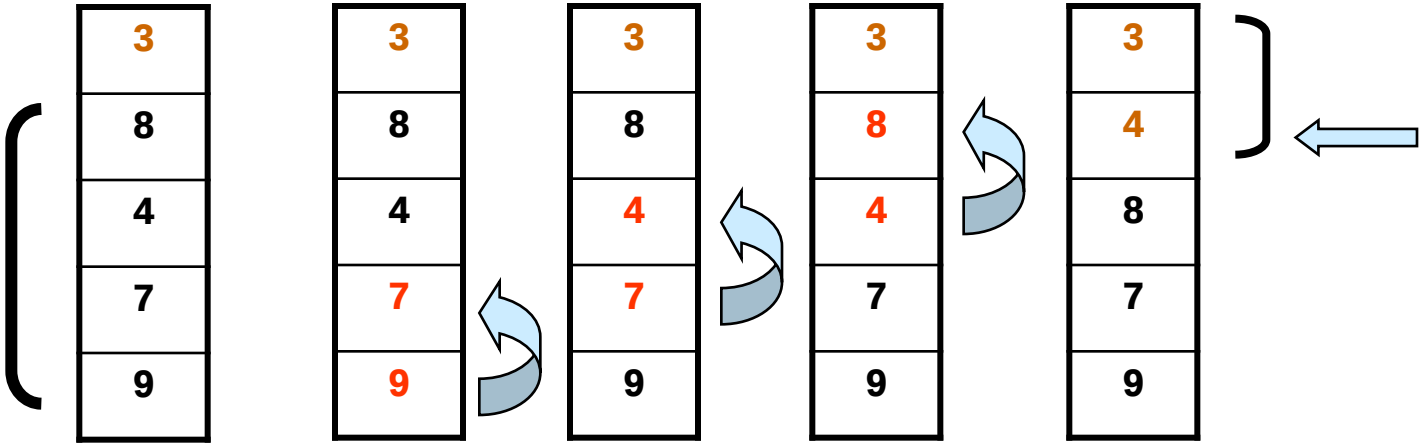
“上升法”冒泡排序示例



一遍扫描使最小元素3成为第一个元素

冒泡排序

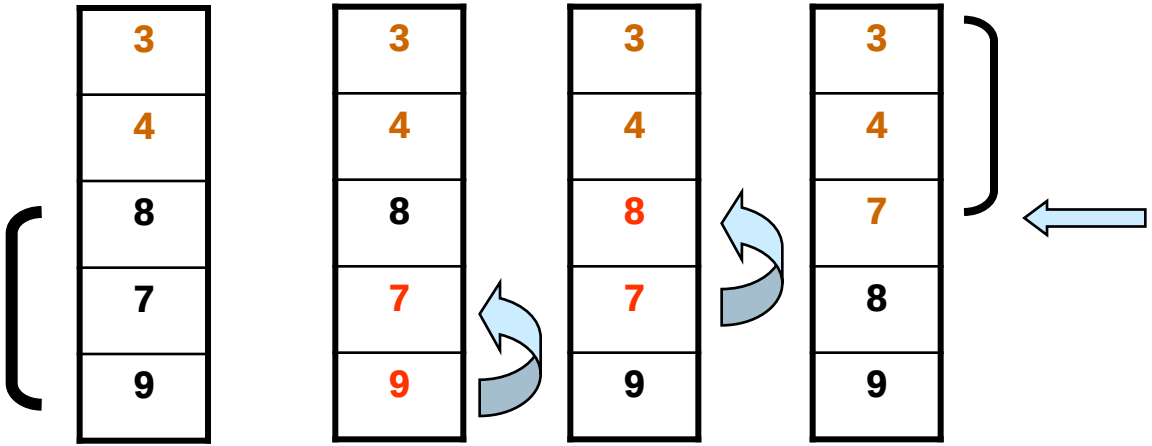
“上升法”冒泡排序示例



第二遍扫描使次小元素4成为第二个元素

冒泡排序

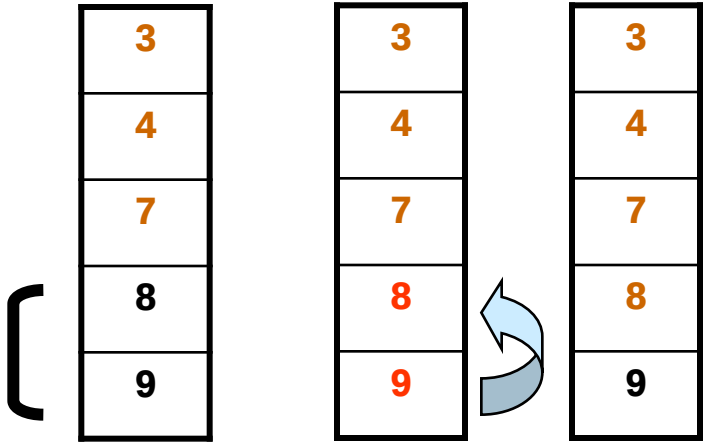
“上升法”冒泡排序示例



第三遍扫描使元素7成为第三个元素

冒泡排序

“上升法”冒泡排序示例



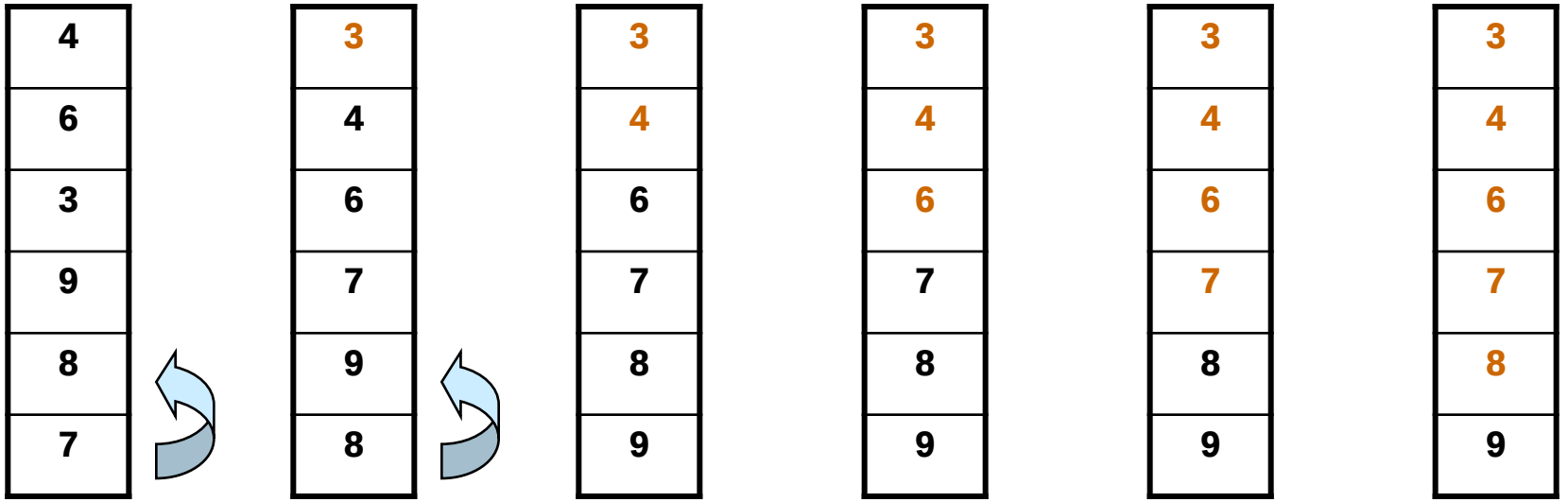
第四遍扫描使元素8成为第四个元素，所有元素都已排序好。

冒泡排序算法

```
//简单冒泡排序算法（上升法）
  void bubble_sort(int a[],int n)
  {  int i,j,x;
1.   for(j=0;j<n-1;j++) //j是本遍扫描终点下标
2.     for(i=n-2;i>=j;i--)
3.       if(a[i]>a[i+1]) //发现逆序就交换
4.       {  x=a[i];
5.          a[i]=a[i+1];
6.          a[i+1]=x;
         }
  }
```

稳定、原地

冒泡排序

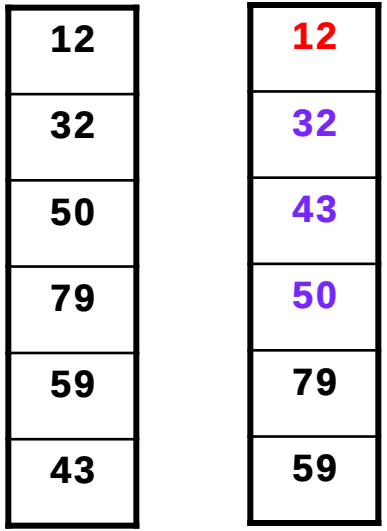

4
6
3
9
8
7
3
4
6
7
9
8
3
4
6
7
8
9
3
4
6
7
8
9
3
4
6
7
8
9
3
4
6
7
8
9
原始数据
第1遍结果
第2遍结果
第3遍结果
第4遍结果
第5遍结果

冒泡排序

```
//改进的简单冒泡排序算法（上升法）
  void bubble_sort_1(int a[],int n)
  {  int i,j,x,flag=1;
1.   j=0;
2.  while  (flag) //存在交换，进入循环
3.   {  flag=0;     //初始化，不存在交换
4.      for(i=n-2;i>=j;i--)
5.        if(a[i]>a[i+1])
6.        {  x=a[i]; a[i]=a[i+1]; a[i+1]=x;
7.           flag=1; //发现交换，说明无序
           }
8.       j++;
      }
   }
```

改进的冒泡排序

“上升法”冒泡排序示例



原始数据

第1遍结果

冒泡排序----时间复杂性分析

- 最坏情况下（输入数据呈倒序形式 ）

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} \approx \frac{1}{2} n^2$$

- 平均情况下（n!种输入序列**等概率** ）

$$\frac{1}{n!} \cdot \frac{n!}{2} \cdot \frac{n(n-1)}{2} = \frac{n(n-1)}{4} \approx \frac{1}{4} n^2$$

各种排序方法比较

排序方法	平均情况	最坏情况	辅助存储	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
希尔排序	与增量序列有关：$O(n^{3/2})$,$O(n(\log n)^2)$			×
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	√

问题

“上升法”冒泡排序示例

一趟冒泡使最小元素19成为第一个元素

快速排序

学习目标和要求

1.准确理解快速排序的基本思想

2.编程实现快速排序算法

3.分析快速排序算法特点

4. 知道算法的时间复杂性和稳定性

分析改进

在冒泡排序中，数据的比较和移动是在相邻单元中进行的，每次交换只能上移或下移一个位置，同时比较过的数据，在下一遍比较时，有可能再次被比较，产生冗余比较。因此，冒泡排序的总比较次数和移动次数较多。

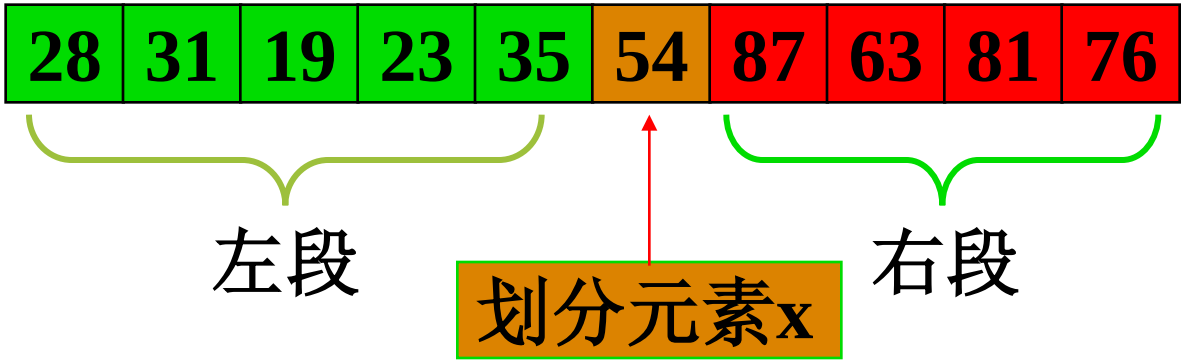
分析改进

改进的着眼点：将数据分组，组内无序，组间有序

分析改进

选x=54进行划分

一次划分结果:



快速排序（quick sort）

也称划分交换排序。因速度非常快而得名

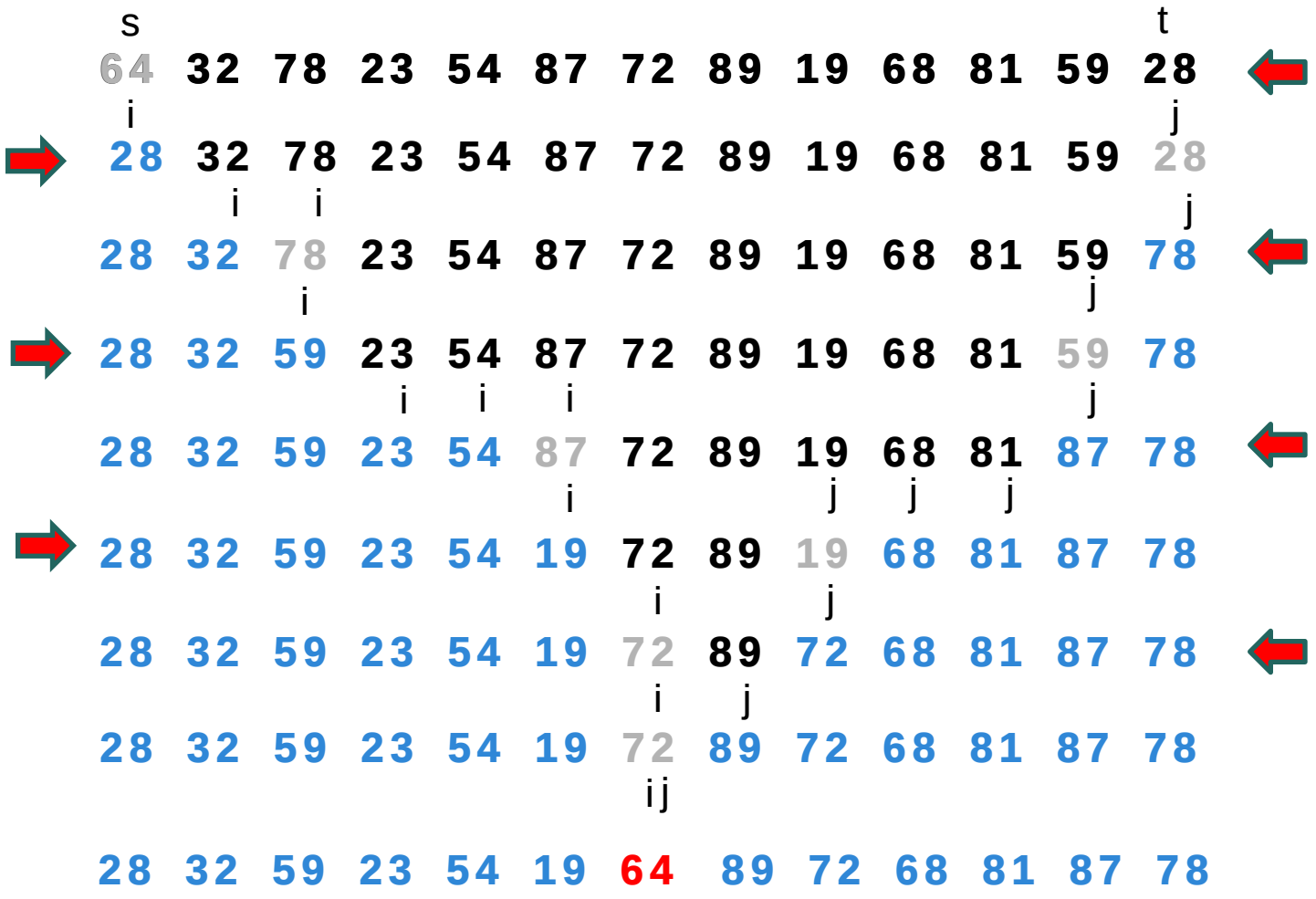
1．基本原理——反复进行有序划分

有序划分方法

- 在数组a中任选一个元素x作为划分元素，通过比较
- 将小于x的元素换到数组的左端（左段）
- 将大于或等于x的元素换到数组右端（右段）
- x本身位于两段之间
- 如果左、右段元素个数多于1，则递归的将左、右段各自划分，直到每段元素个数都不超过1，从而达到排序目的

x = 64

问题：如何进行一次划分？



一次划分的形式化描述：

```
   void  partition(int a[],int s,int t,int &k)   //划分函数
   {   int i,j,x;
1.     x=a[s];                  //取划分元素
2.     i=s; j=t;                //扫描指针初值
3.     do                                  //循环地进行划分
4.     {  while((a[j]>=x)&&(i<j))j--;  //从右向左扫描
5.           if(i<j)a[i++]=a[j];          //小元素向左移
6.        while((a[i]<x)&&(i<j))i++;   //从左向右扫描
7.           if(i<j)a[j--]=a[i];          //大元素向右移
        }
8.     while(i<j);                          //直到指针i和j相等
9.     a[i]=x;                             //划分元素就位
10.    k=i;
   }
```

问题：如何处理划分得到的两个待排子序列？

对划分得到的两个子序列*递归*地执行快速排序。

X=28　　28 32 59 23 54 19 64 89 72 68 81 87 78

X=19　　19 23 28 59 54 32 64 89 72 68 81 87 78

　　19 23 28 59 54 32 64 89 72 68 81 87 78

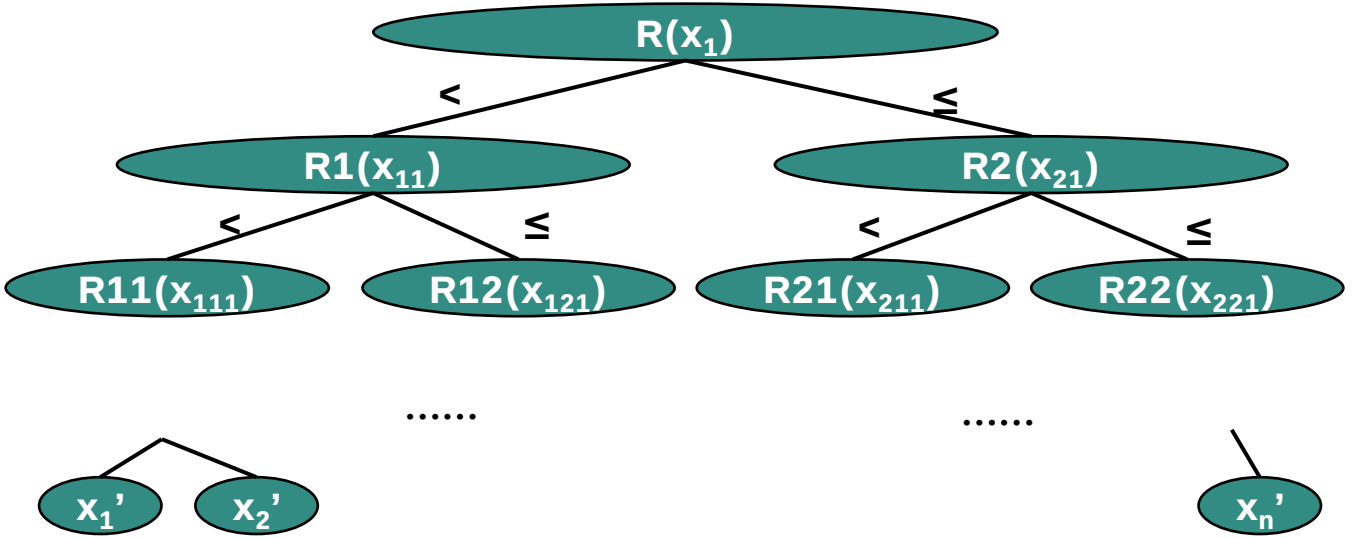
快速排序算法

```
   void quick_sort(int a[],int i,int j)
   {  int k;
1.   if(i<j)     //递归条件
2.   { partition(a,i,j,k);//划分,k为元素a[i]的有序位置
3.     quick_sort(a,i,k-1);//递归,对左段快速排序
4.     quick_sort(a,k+1,j); //递归,对右段快速排序
     }
   }
```

主调语句：quick_sort(a,0,n-1);

算法思想的进一步分析

递归树



分治递归的算法思想

时间复杂性分析

时间复杂性

$$\begin{cases} T(n) = cn & (n = 1) \\ T(n) = T(n_1) + T(n_2) + cn & (n > 1) \end{cases}$$

最好情况：

$$T(n) <= 2T(n/2) + cn = cnlogn + bn = O(nlogn)$$

平均情况：

$$T(n) \le \frac{1}{n} \sum_{n1=0}^{n-1} (T(n1) + T(n-1-n1)) + cn$$

$$= O(nlogn)$$

最坏情况：

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

空间复杂性分析

- 快速排序不属于原地排序
- 由于程序中使用了递归，需要递归调用栈的支持，而栈的长度取决于递归调用的深度。在平均情况下，需要**O(logn)**的栈空间；最坏情况下，栈空间可达**O(n)**。

结论

1）划分元素的选取是影响时间性能的关键。

2）输入数据次序越乱，所选划分元素值的随机性越好，排序速度越快。快速排序不是自然排序方法。

3）改变划分元素的选取方法，至多只能改变算法平均情况下的时间性能，无法改变最坏情况下的时间性能。即最坏情况下，快速排序的时间复杂性总是$O(n^2)$的。

各种排序方法比较

排序方法	平均情况	最坏情况	辅助存储	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
希尔排序	与增量序列有关：$O(n^{3/2})$,$O(n(\log n)^2)$			×
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
快速排序	$O(n\log n)$	$O(n^2)$	$O(\log n)$	×

思考

1、非递归的快速排序算法如何实现？

2、如何选取划分元素，尽量保证子问题的大小平衡？