# 合并排序

《数据结构》

指院网络工程教研中心　陈卫卫

初始　[49] [38] [65] [97] [76] [13] [27]

一趟归并　[38 49] [65 97] [13 76] [27]

二趟归并　[38 49 65 97] [13 27 76]

三趟归并　[13 27 38 49 65 76 97]

# 合并排序

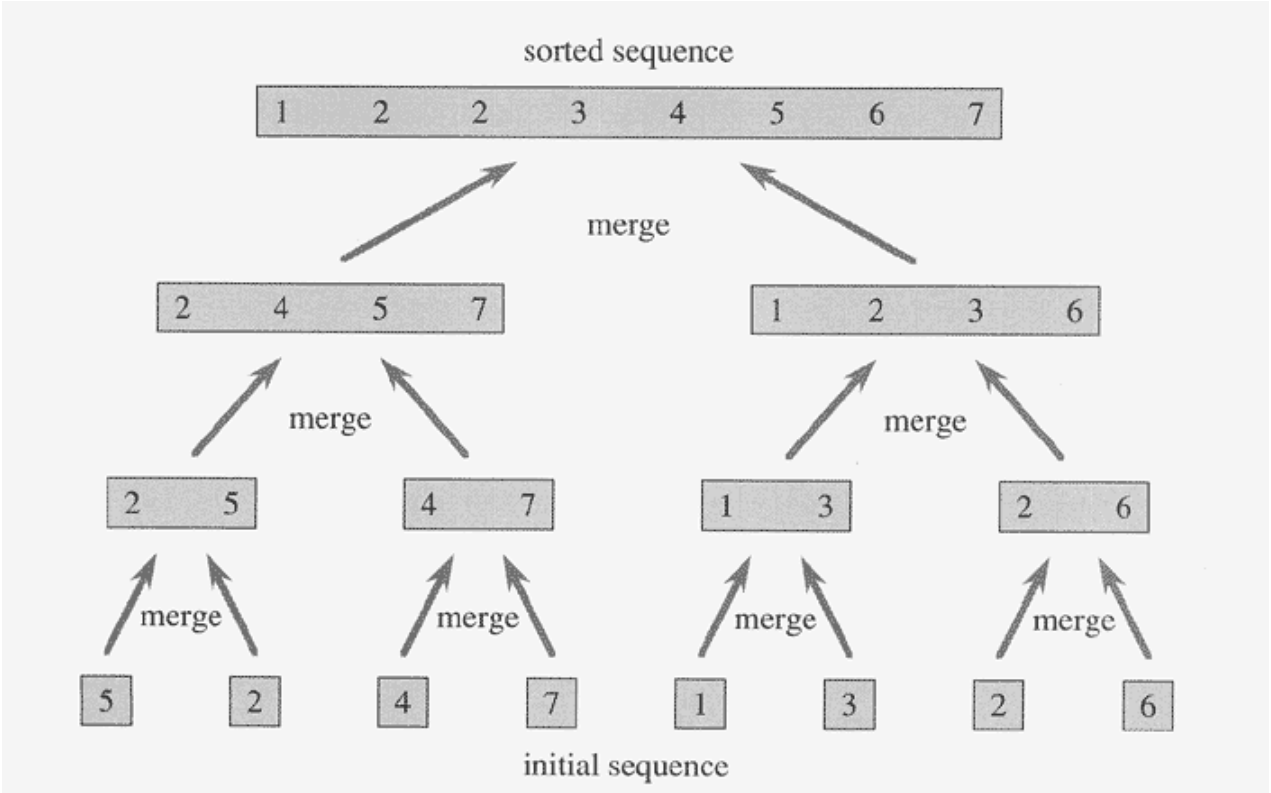
## 学习目标和要求

1、准确复述合并排序的基本思想

2、编写合并排序的算法

3、分析合并排序的特点

4、知道算法的时间复杂性和稳定性


sorted sequence
1 2 2 3 4 5 6 7
merge
2 4 5 7
1 2 3 6
merge
merge
2 5
4 7
1 3
2 6
merge
merge
merge
merge
5
2
4
7
1
3
2
6
initial sequence

# 合并排序

## 合并排序（归并排序）

将两个或两个以上的有序表逐趟进行合并，最终组合成一个新的有序表的排序方法。

# 合并排序

## 2-路归并

将待排的n个元素看成n个有序的子序列，每个子序列的长度为1，两两归并，得到[n/2]个长度为2的子序列；再两两归并，……，如此反复，直至得到一个长度为n的有序序列为止。

# 合并排序

**例：对数据 49 38 65 97 76 13 27， 进行2-路归并排序。**

初始　　[49] [38] [65] [97] [76] [13] [27]

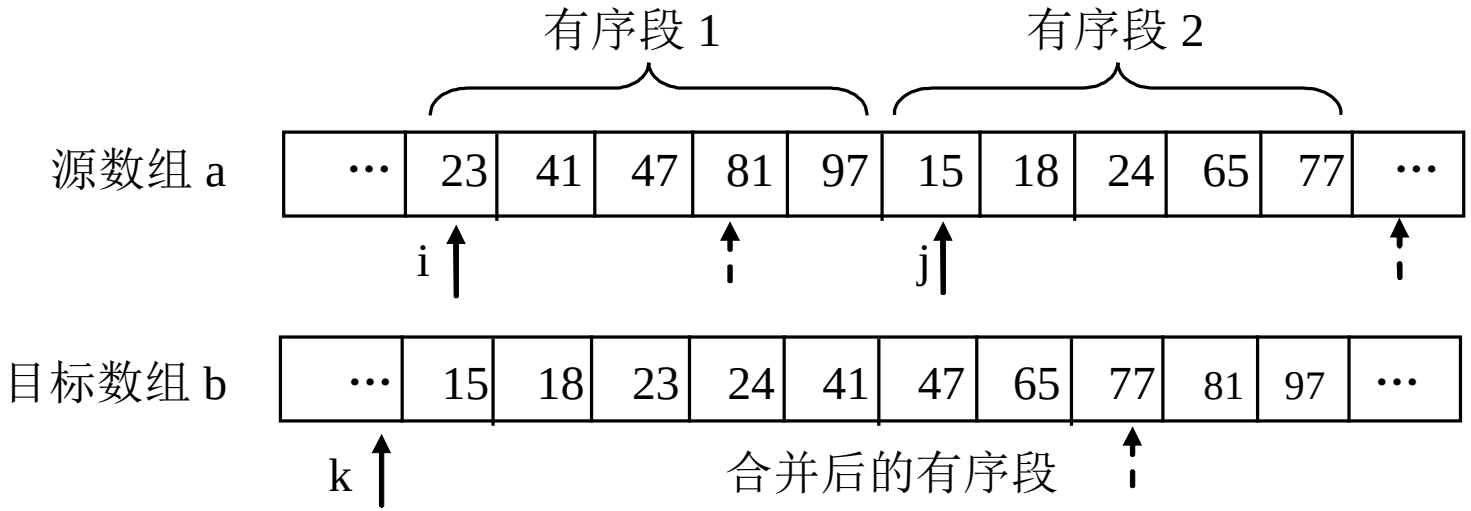
第一趟归并后结果：　[38 49]　[65 97]　[13 76]　[27]

第二趟归并后结果：　[38 49 65 97]　　[13 27 76]

第三趟归并后结果：　[13 27 38 49 65 76 97]

# 递归的合并排序

```
//递归的2-路合并排序算法
  void merge_sort(s)//s表示数组a[n]中的一段元素
 {
1. if(s的元素个数大于1)
2.   {把s分成大小相等（或相差1）的两段s1和s2;
3.     merge_sort(s1); //对s1递归合并排序
4.     merge_sort(s2); //对s2递归合并排序
5.     merge(s1,s2); //合并s1,s2的排序结果
     }
   }
```


有序段 1
有序段 2
源数组 a
… 23 41 47 81 97 15 18 24 65 77 …
i
j
目标数组 b
… 15 18 23 24 41 47 65 77 81 97 …
k
合并后的有序段

# 递归的合并排序

```
void merge (int a[],int p,int q,int s,int t)
//将有序段a[p..q]和有序段a[s..t]合并到b[p..t]
{  int i,j,k,b[n];     //b为合并场地
    i=p;j=s;k=p;  //三个指针i,j,k
   while((i<=q)&&(j<=t))//两个有序段都不空
     if(a[i]<a[j]) b[k++]=a[i++];
     else b[k++]=a[j++];
   while(i<=q) b[k++]=a[i++];//处理有序段1的剩余元素
   while(j<=t) b[k++]=a[j++];//处理有序段2的剩余元素
   for(i=p;i<=t;i++)a[i]=b[i]//将合并后的有序段移回源数组a
}
```

# 递归的合并排序

```
//递归的2-路合并排序算法
void merge_sort(int a[],int i,int j)
{  int k;
1.   if(i<j )
2.   { k=(i+j)/2 ;                    //k为分段中点
3.     merge_sort(a,i,k);             //对左段排序
4.     merge_sort(a,k+1,j);           //对右段排序
5.     merge(a,i,k,k+1,j);            //合并左右段
      }
   }
```

```
merge_sort(a,0,n-1);//主调语句
```

# 合并排序与快速排序的比较

```
//合并排序算法
void merge_sort(int a[],int i,int j)
{  int k;
1.   if(i<j )
     {
2.      k=(i+j)/2 ;
3.      merge_sort(a,i,k);
4.      merge_sort(a,k+1,j);
5.      merge(a,i,k,k+1,j);
     }
   }
```

```
//快速排序算法
void quick_sort(int a[],int i,int j)
{  int k;
   if(i<j)
   {
      partition(a,i,j,k);
      quick_sort(a,i,k-1);
      quick_sort(a,k+1,j);
    }
}
```

# 时间复杂性分析

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

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

# 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 | $O(n\log n)$ | $O(n^2)$ | $O(\log n)$ | × |
| 简单选择排序 | $O(n^2)$ | $O(n^2)$ | $O(1)$ | × |
| 堆排序 | $O(n\log n)$ | $O(n\log n)$ | $O(1)$ | × |
| 合并排序 | $O(n\log n)$ | $O(n\log n)$ | $O(n)$ | √ |
| | | | | |

非递归的合并排序算法的实现方式？