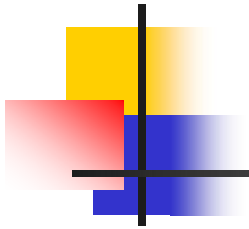
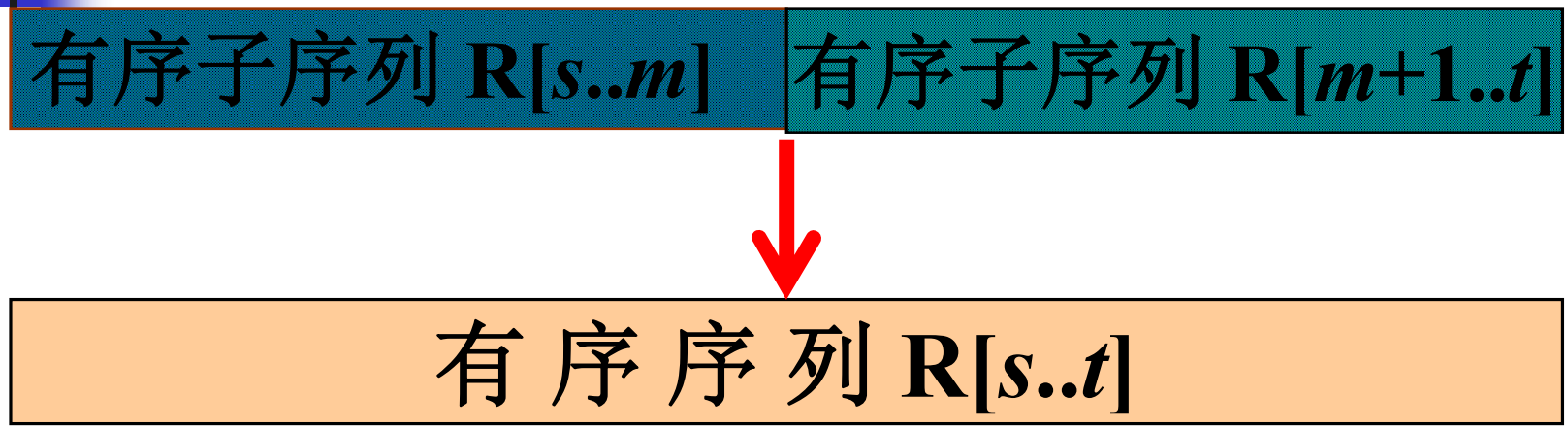


归并排序

- 基本思想：将待排序序列划分成若干有序子序列；将两个或两个以上的有序子序列“合并”为一个有序序列
- 在内部排序中，通常采用的是2-路归并排序。即：将两个位置相邻的有序子序列“合并”为一个有序序列

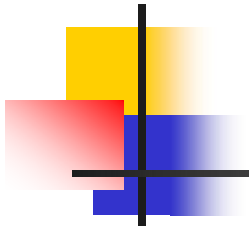
归并排序



这个操作对顺序表而言，是轻而易举的。

```
void Merge (ElemType SR[ ], ElemType TR[ ], int s, int m, int t)
{ // 将有序的序列 SR[s..m] 和 SR[m+1..t]归并为有序的序列 TR[s..t]
  for (i=s, j=m+1, k=s;  i<=m && j<=t;  ++k)
   {  if (SR[i].key<=SR[j].key)  TR[k] = SR[i++];
     else TR[k] = SR[j++];
   }
   if (i<=m) TR[k..t] = SR[i..m];
   if (j<=t) TR[k..t] = SR[j..t];

} // Merge
```



归并排序的算法

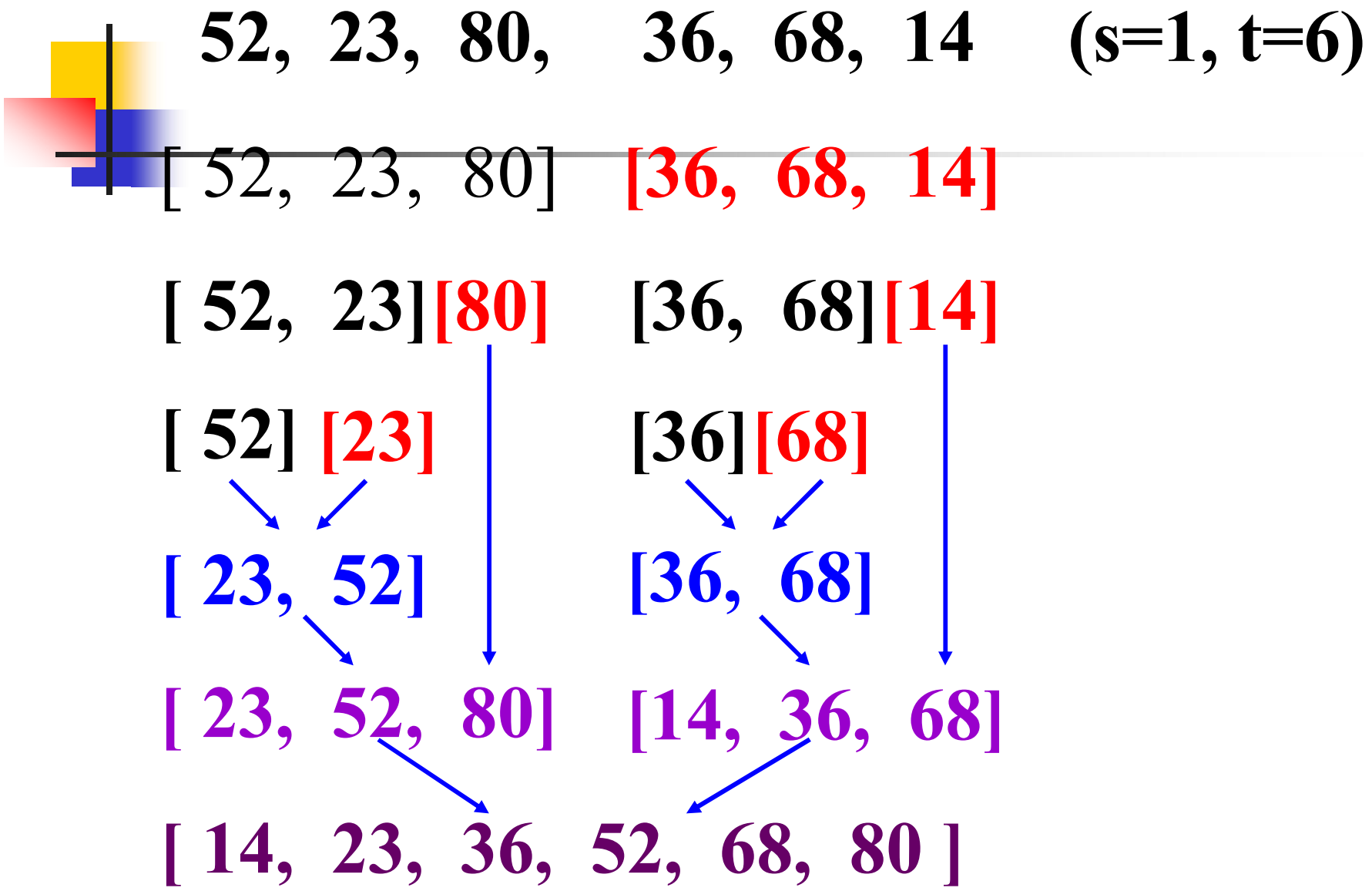
- 如果数据元素无序序列 R[s..t] 的两部分

  R[*s*..⌊(*s*+*t*)/2⌋]和R[⌊(*s*+*t*)/2⌋+1..*t*]

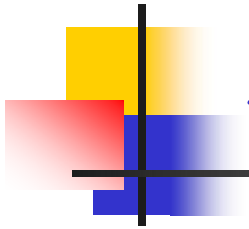
分别按关键字有序，则利用Merge算法很容易将它们合并成一个有序序列。

- 应该先分别对这两部分进行 2-路归并排序。

例如：

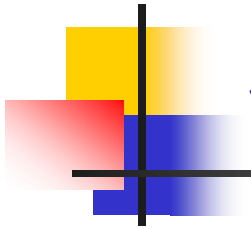

52, 23, 80, 36, 68, 14 (s=1, t=6)
[52, 23, 80] [36, 68, 14]
[52, 23] [80] [36, 68] [14]
[52] [23] [36] [68]
[23, 52] [36, 68]
[23, 52, 80] [14, 36, 68]
[14, 23, 36, 52, 68, 80]

```
void Msort (ElemType SR[],   ElemType TR1[], int s, int t )
{  // 将SR[s..t] 归并排序为 TR1[s..t]
   if (s= =t) TR1[s]=SR[s];
   else
     {  m=(s+t)/2;
        Msort (SR, TR2, s, m);
        Msort (SR, TR2, m+1, t);
        Merge (TR2, TR1, s, m, t);
     }
} // Msort
```



对顺序表L作2-路归并排序

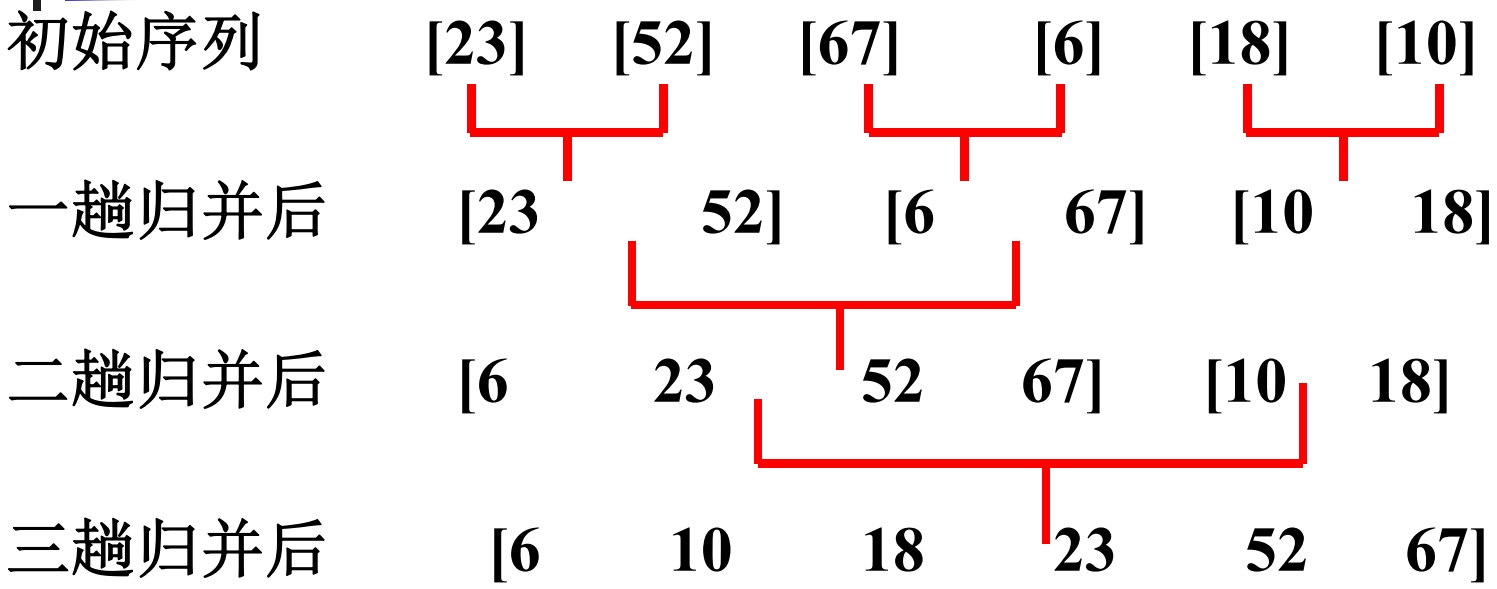
```
void MergeSort (SqList &L)
 {
   MSort(L.r, L.r, 1, L.length);
 }
```



算法分析

- 一趟归并的时间复杂度为 O(n)
- 总共需进行$\lceil \log_2 n \rceil$趟
- n 个记录进行归并排序的时间复杂度为O(nlogn)
- 稳定性?

归并排序的过程

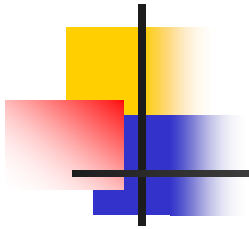


说明：该过程初始时每个数据元素看成是一个有序子序列

每一趟依次将2个相邻子序列合并成一个有序子序列----所谓的**自下而上**的归并排序过程

时间性能

- 平均的时间性能
  - 时间复杂度为 O(nlogn)：快速排序、堆排序和归并排序
  - 时间复杂度为 $O(n^2)$：直接插入、冒泡和简单选择排序
  - 时间复杂度为 O(n)：基数排序
- 当待排记录序列按关键字顺序有序时
  - 直接插入和冒泡排序：0(n)
  - 快速排序：$0(n^2)$ 。
- 简单选择排序、堆排序和归并排序的时间性能不随记录序列中关键字的分布而改变。



空间性能

- **指的是排序过程中所需的辅助空间大小**
  - 所有的简单排序方法(包括：直接插入、起泡和简单选择) 和堆排序的空间复杂度为O(1)；
  - 快速排序为O(logn)，为递归程序执行过程中，栈所需的辅助空间；
  - 归并排序所需辅助空间最多，其空间复杂度为O(n)；
  - 链式基数排序需附设队列首尾指针，则空间复杂度为 O(rd)。