10.5 归并排序

归并排序的过程基于下列**基本思想**进行： 将两个或两个以上的有序子序列 “归并” 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**的记录有序子序列

归并为一个记录的有序序列。

这个操作对顺序表而言，是轻而易举的。

例如图10.13为2-路归并排序的一个例子。

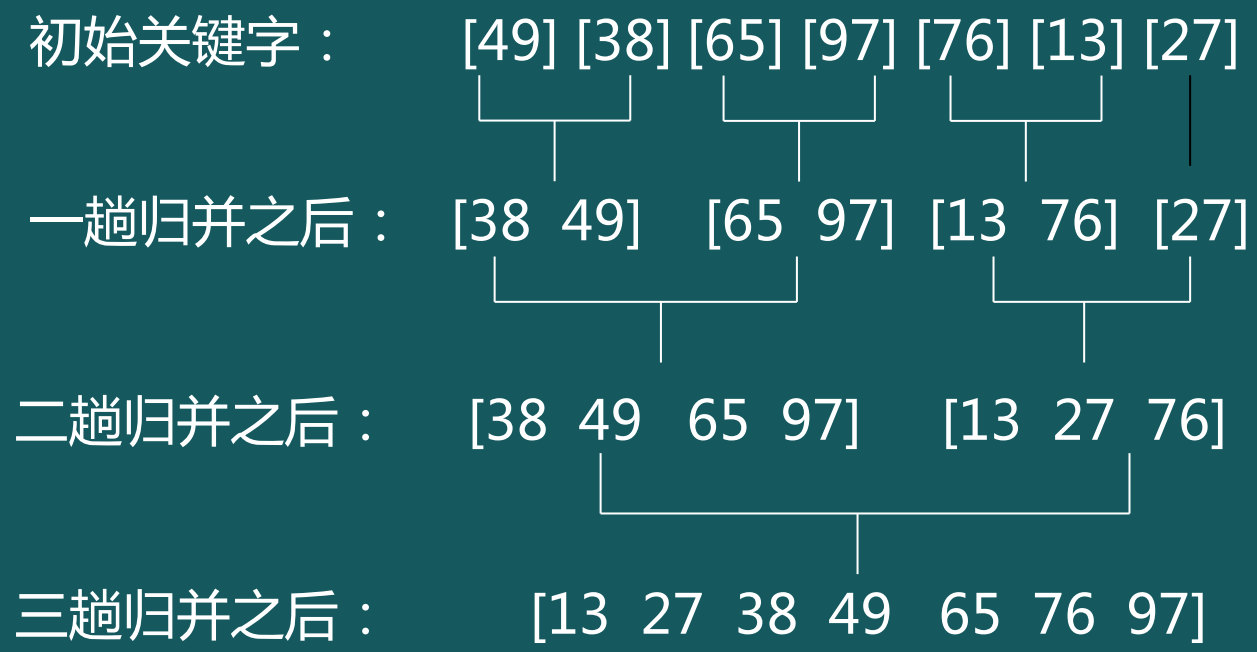


图10.13 2-路归并排序示例

```
void Merge (RcdType SR[], RcdType &TR[],
                                        int i, int m, int n) {
  // 将有序的记录序列 SR[i..m] 和 SR[m+1..n]，归并为有序的//记录序列
TR[i..n]
   for (j=m+1, k=i;  i<=m && j<=n;  ++k)
  {                    // 将SR中记录由小到大地并入TR
    if (SR[i].key<=SR[j].key)  TR[k] = SR[i++];
    else TR[k] = SR[j++];
  }
if (i<=m) TR[k..n] = SR[i..m]; // 将剩余的 SR[i..m] 复制到 TR
if (j<=n) TR[k..n] = SR[j..n];   // 将剩余的 SR[j..n] 复制到 TR
} // Merge
```

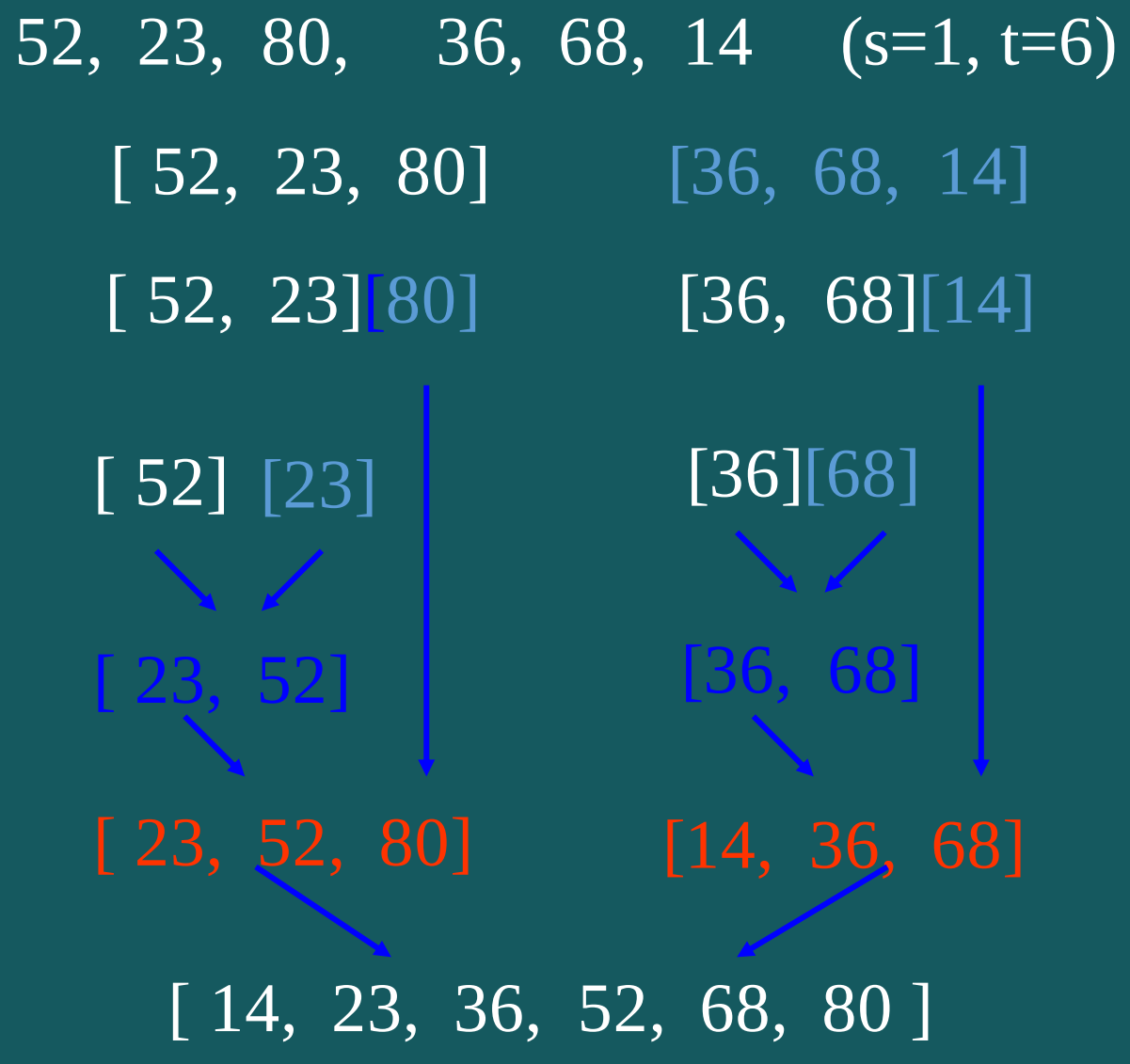
算法 10.12

归并排序的算法

如果记录无序序列 R[s..t] 的两部分 R[$s..\lfloor(s+t)/2\rfloor$] 和 R[$\lfloor(s+t)/2\rfloor+1..t$]，分别按关键字有序，则利用上述归并算法很容易将它们归并成整个记录序列是一个有序序列。

由此，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 ( RcdType SR[],  RcdType &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;   // 将SR[s..t]平分为SR[s..m]和SR[m+1..t]
    Msort (SR, TR2, s, m);
     // 递归地将SR[s..m]归并为有序的TR2[s..m]
    Msort (SR, TR2, m+1, t);
     //递归地SR[m+1..t]归并为有序的TR2[m+1..t]
    Merge (TR2, TR1, s, m, t);
      // 将TR2[s..m]和TR2[m+1..t]归并到TR1[s..t]
      }
} // Msort
```

算法 10.13

```
void MergeSort (SqList &L) {
  // 对顺序表 L 作2-路归并排序
  MSort(L.r, L.r, 1, L.length);
} // MergeSort
```

算法 10.14

容易看出，对 n 个记录进行归并排序的时间复杂度为 $O(n\log n)$。即：

每一趟归并的时间复杂度为 $O(n)$，

总共需进行 $\lceil \log_2 n \rceil$ 趟。

10.5 归并排序

归并排序的过程基于下列**基本思想**进行： 将两个或两个以上的有序子序列 “归并” 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**的记录有序子序列

有序子序列 R[*l*..*m*]	有序子序列 R[*m*+1..*n*]

归并为一个记录的有序序列。

有序序列 R[*l*..*n*]

这个操作对顺序表而言，是轻而易举的。

例如图10.13为2-路归并排序的一个例子。

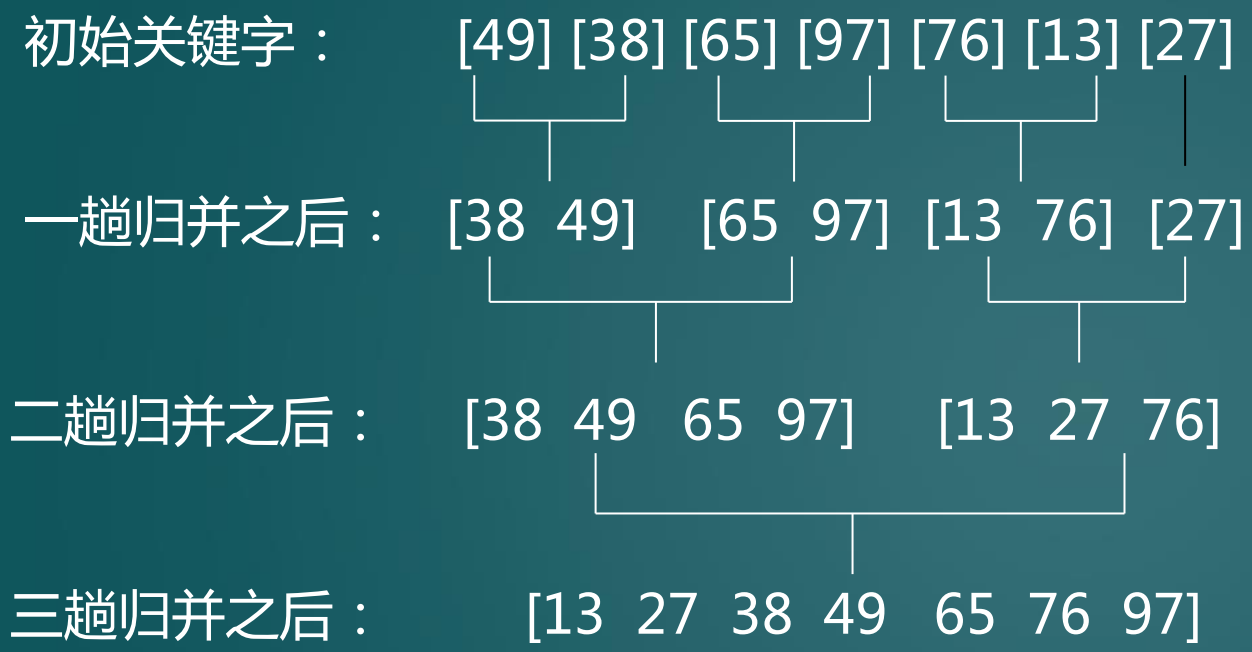


图10.13 2-路归并排序示例

```
void Merge (RcdType SR[], RcdType &TR[],
                                        int i, int m, int n) {
  // 将有序的记录序列 SR[i..m] 和 SR[m+1..n]，归并为有序的//记录序列
TR[i..n]
   for (j=m+1, k=i;  i<=m && j<=n;  ++k)
  {                    // 将SR中记录由小到大地并入TR
    if (SR[i].key<=SR[j].key)  TR[k] = SR[i++];
    else TR[k] = SR[j++];
  }
if (i<=m) TR[k..n] = SR[i..m]; // 将剩余的 SR[i..m] 复制到 TR
if (j<=n) TR[k..n] = SR[j..n];   // 将剩余的 SR[j..n] 复制到 TR
} // Merge
```

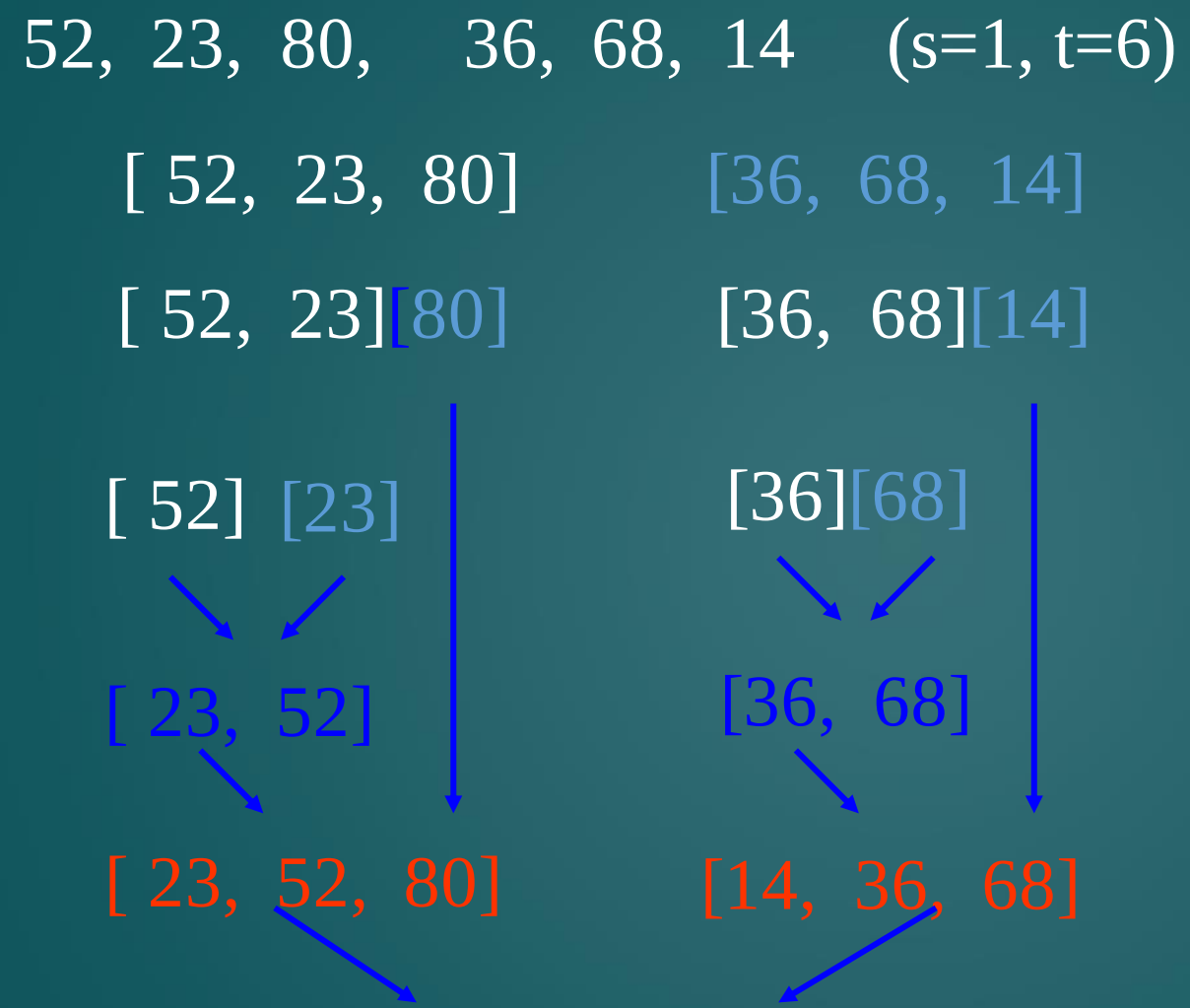
算法 10.12

归并排序的算法

如果记录无序序列 R[s..t] 的两部分 R[$s..\lfloor(s+t)/2\rfloor$] 和 R[$\lfloor(s+t)/2\rfloor+1..t$]，分别按关键字有序，则利用上述归并算法很容易将它们归并成整个记录序列是一个有序序列。

由此，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 ( RcdType SR[],  RcdType &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;   // 将SR[s..t]平分为SR[s..m]和SR[m+1..t]
    Msort (SR, TR2, s, m);
     // 递归地将SR[s..m]归并为有序的TR2[s..m]
    Msort (SR, TR2, m+1, t);
     //递归地SR[m+1..t]归并为有序的TR2[m+1..t]
    Merge (TR2, TR1, s, m, t);
      // 将TR2[s..m]和TR2[m+1..t]归并到TR1[s..t]
      }
} // Msort
```

算法 10.13

```
void MergeSort (SqList &L) {
  // 对顺序表 L 作2-路归并排序
  MSort(L.r, L.r, 1, L.length);
} // MergeSort
```

算法 10.14

容易看出，对 n 个记录进行归并排序的时间复杂度为 $O(n\log n)$。即：

每一趟归并的时间复杂度为 $O(n)$，

总共需进行 $\lceil \log_2 n \rceil$ 趟。