10.5 归并排序

- 通过“归并”两个或两个以上的记录有序子序列，逐步增加记录有序序列的长度。

- 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

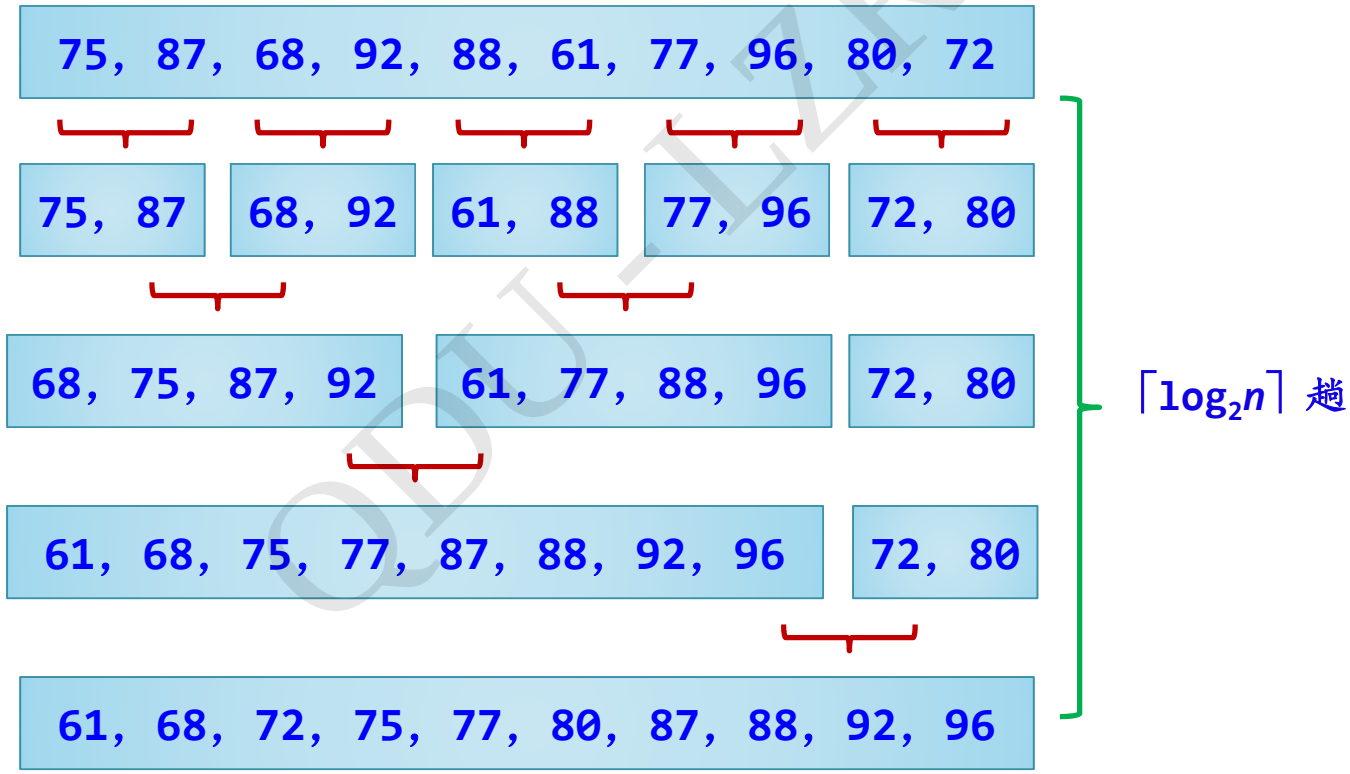
有序子序列 r[*l*..*m*]	有序子序列 r[*m*+1..*n*]

归并为一个记录的有序序列。

有 序 序 列 r[*l*..*n*]

- 归并排序是多次将两个或两个以上的有序表合并成一个新的有序表。

【示例】 已知有10个待排序的记录，它们的关键字序列为（75，87，68，92，88，61，77，96， 80，72），给出用2-路归并排序法进行排序的过程。



将两个有序子表归并为一个有序子表的算法Merge()：

```
void Merge(RedType SR[], RedType TR[], int i, int m, int n)
{
    // 将有序的SR[i..m]和SR[m+1..n]归并为有序的TR[i..n] 算法10.12
    int j, k, l;
    // 将SR中记录由小到大地并入TR
    for(j = m + 1, k = i; i <= m && j <= n; ++k)
        if LQ(SR[i].key, SR[j].key)
            TR[k] = SR[i++];
        else
            TR[k] = SR[j++];
    if(i <= m)
        for(l = 0; l <= m - i; l++)
            TR[k + l] = SR[i + l];     // 将剩余的SR[i..m]复制到TR
    if(j <= n)
        for(l = 0; l <= n - j; l++)
            TR[k + l] = SR[j + l];    // 将剩余的SR[j..n]复制到TR
}
```

将两个有序子表归并为一个有序子表的算法MSort()：

```
void MSort(RedType SR[], RedType TR1[], int s, int t)
{
    // 将SR[s..t]归并排序为TR1[s..t]。算法10.13
    int m;
    RedType TR2[MAX_SIZE + 1];
    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]
        // 递归地将SR[s..m]归并为有序的TR2[s..m]
        MSort(SR, TR2, s, m);
        // 递归地将SR[m+1..t]归并为有序的TR2[m+1..t]
        MSort(SR, TR2, m + 1, t);
        // 将TR2[s..m]和TR2[m+1..t]归并到TR1[s..t]
        Merge(TR2, TR1, s, m, t);
    }
}
```

归并排序算法MergeSort()：

```
void MergeSort(SqList &L)
{
    // 对顺序表L作归并排序。算法10.14
    MSort(L.r, L.r, 1, L.length);
}
```

- 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。
- 所以对n个记录进行归并排序的时间复杂度为 $O(n\log_2 n)$。

归纳起来，二路归并排序算法的性能如表所示。

时间复杂度			空间复杂度	稳定性
最好情况	最坏情况	平均情况		
$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定

END