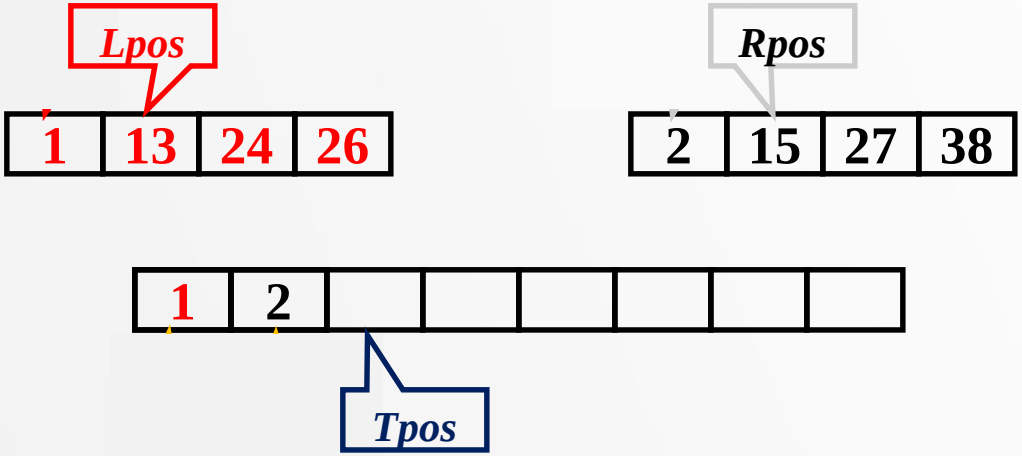
归并排序

1. 合并2个有序表



$$T(N) = O(N)$$

归并排序

归并——将两个或两个以上的有序表组合成一个新的有序表，叫归并排序

2-路归并排序

排序过程

设初始序列含有n个记录，则可看成n个有序的子序列，每个子序列长度为1

两两合并，得到⌊n/2⌋个长度为2或1的有序子序列

再两两合并，……如此重复，直至得到一个长度为n的有序序列为止

例

初始关键字: [49] [38] [65] [97] [76] [13] [27]

一趟归并后: [38 49] [65 97] [13 76] [27]

二趟归并后: [38 49 65 97] [13 27 76]

三趟归并后: [13 27 38 49 65 76 97]

迭代算法：

将序列的每一个数据看成一个长度为1的有序表，

然后，将相邻两组进行归并得到长度为2的有序表（一趟归并）

再对相邻两组长度为2的有序表进行下一趟归并得到长度为4的有序表，

这样一直进行下去，直到整个表归并成有序表。

如果某一趟归并过程中，单出一个表，

该表轮空，等待下一趟归并。

递归思想

- 将无序序列划分成大概均等的2个子序列，然后用同样的方法对2**个子序列进行归并排序得到**2个有序的子序列，再用合并2个有序表的方法合并这2个子序列，得到n个元素的有序序列。

算法描述

```
void MSort( ElementType A[ ], ElementType TmpArray[ ],
                          int Left, int Right )
{  int  Center;
   if ( Left < Right ) {  /* 待排序的数据在数组的下标位置 */
              Center = ( Left + Right ) / 2;
              MSort( A, TmpArray, Left, Center );  /* T( N / 2 ) */
              MSort( A, TmpArray, Center + 1, Right );          /* T( N / 2 ) */
              Merge( A, TmpArray, Left, Center + 1, Right );  /* O( N ) */
   }
}

void Mergesort( ElementType A[ ], int N )
{  ElementType  *TmpArray;  /* need O(N) extra space */
   TmpArray = malloc( N * sizeof( ElementType ) );
   if ( TmpArray != NULL ) {
              MSort( A, TmpArray, 0, N - 1 );
              free( TmpArray );
   }
   else  FatalError( "No space for tmp array!!!" );
}
```

算法描述

```
void MSort( ElementType A[ ], ElementType TmpArray[ ],
                          int Left, int Right )
{  int  Center;
   if ( Left < Right ) {  /* 待排序的数据在数组的下标位置 */
              Center = ( Left + Right ) / 2;
              MSort( A, TmpArray, Left, Center );  /* T( N / 2 ) */
              MSort( A, TmpArray, Center + 1, Right );          /* T( N / 2 ) */
              Merge( A, TmpArra      ft, Center + 1, Right );  /* O( N ) */
   }
}

void Mergeso
{  Elemen
   TmpAr
   if ( Tmp
              M
              free( Tm
   }
   else  FatalError( "No space for tmp array!!!" );
}
```

如果 **TmpArray** 在每次Merge调用的时候都分配空间，那么$S(N) = O(N \log N)$

```
/* Lpos = start of left half, Rpos = start of right half */
void Merge( ElementType A[ ], ElementType TmpArray[ ],
                    int Lpos, int Rpos, int RightEnd )
{  int  i, LeftEnd, NumElements, TmpPos;
   LeftEnd = Rpos - 1;
   TmpPos = Lpos;
   NumElements = RightEnd - Lpos + 1;
   while( Lpos <= LeftEnd && Rpos <= RightEnd ) /* main loop */
      if ( A[ Lpos ] <= A[ Rpos ] )
               TmpArray[ TmpPos++ ] = A[ Lpos++ ];
      else
               TmpArray[ TmpPos++ ] = A[ Rpos++ ];
   while( Lpos <= LeftEnd ) /* Copy rest of first half */
      TmpArray[ TmpPos++ ] = A[ Lpos++ ];
   while( Rpos <= RightEnd ) /* Copy rest of second half */
      TmpArray[ TmpPos++ ] = A[ Rpos++ ];
   for( i = 0; i < NumElements; i++, RightEnd - - )
       /* Copy TmpArray back */
   {
      A[ RightEnd ] = TmpArray[ RightEnd ];
       printf("%d ", A[RightEnd]);
   }
}
```

算法评价

时间复杂度：每一趟归并的时间复杂度为O(n)，　总共需要归并$\log_2 n$趟，因而，总的时间复杂度为$O(n\log_2 n)$。

空间复杂度：　2-路归并排序过程中，需要一个与表等长的存储单元数组空间，因此，空间复杂度为O(n)。

表　　二路归并排序算法的性能

时间复杂度			空间复杂度	稳定性	复杂性
平均情况	最坏情况	最好情况			
O($n$log$n$)	O($n$log$n$)	O($n$log$n$)	O(n)	稳定	较复杂

[例题 1] 将两个各有 n 个元素的有序表归并成一个有序表，其最少的比较次数是（A）。

A. n　　B. $2n-1$　　C. $2n$　　D. $n-1$

分析：假设有两个有序表 A 和 B 都递增有序，当有序表 A 所有元素均小于 B 的元素时，只需将 A 的所有元素与 B 的第一个元素比较即可，共比较 n 次。