第九讲 排序（上）

浙江大学 陈 越

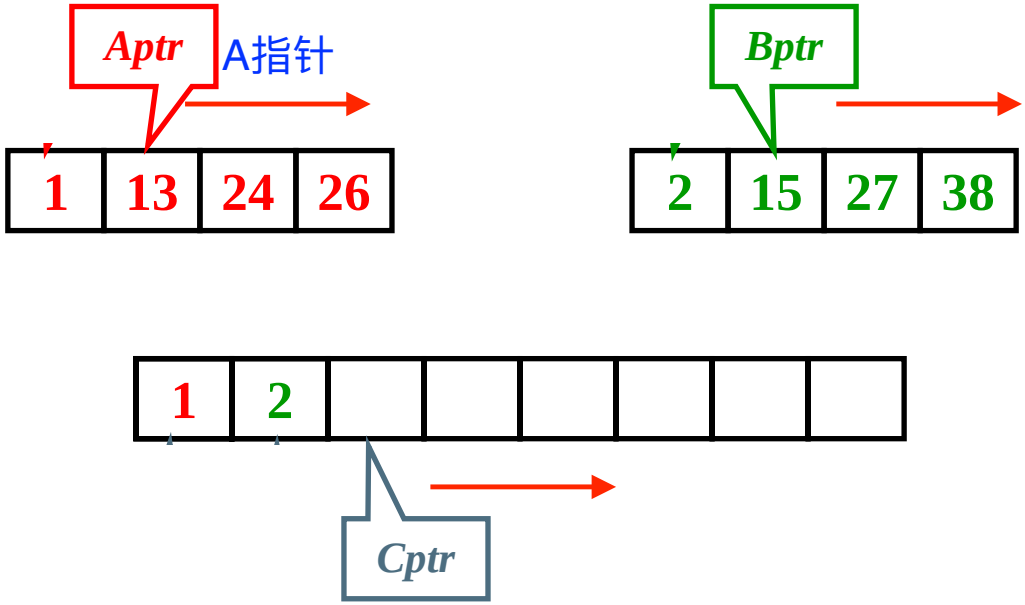
9.4 归并排序

在外排序时非常有效

浙江大学计算机科学与技术学院

核心：有序子列的归并

指针本质:是一个位置(地址)



如果两个子列一共有N个元素，则归并的时间复杂度是？

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

核心：有序子列的归并

```
/* L = 左边起始位置, R = 右边起始位置, RightEnd = 右边终点位置 */
void Merge( ElementType A[], ElementType TmpA[],
            int L, int R, int RightEnd )
{   LeftEnd = R - 1; /* 左边终点位置。假设左右两列挨着 */
    Tmp = L; /* 存放结果的数组的初始位置 */
    NumElements = RightEnd - L + 1;
    while( L <= LeftEnd && R <= RightEnd ) {
        if ( A[L] <= A[R] ) TmpA[Tmp++] = A[L++];
        else                TmpA[Tmp++] = A[R++];
    }
    while( L <= LeftEnd ) /* 直接复制左边剩下的 */
        TmpA[Tmp++] = A[L++];
    while( R <= RightEnd ) /*直接复制右边剩下的 */
        TmpA[Tmp++] = A[R++];
    for( i = 0; i < NumElements; i++, RightEnd -- )
        A[RightEnd] = TmpA[RightEnd];
}
```

右边的终点的位置

Tmp相当于Cptr

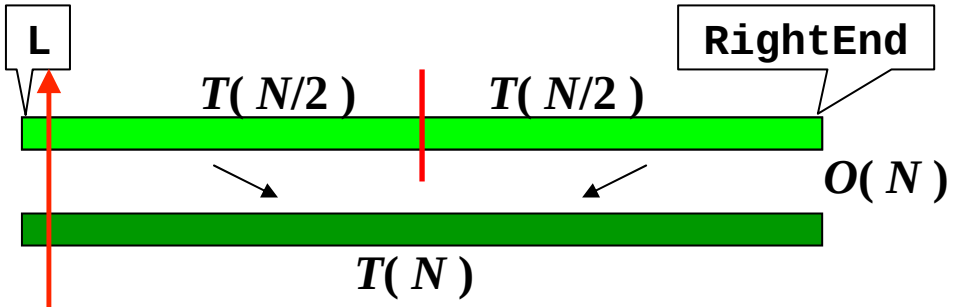
归并完成的数组大小

导回A数组中

递归算法

稳定

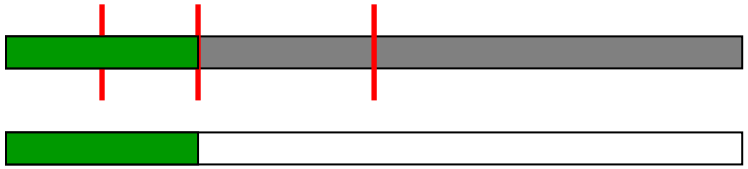
- 分而治之




```
void MSort( ElementType A[], ElementType TmpA[],
            int L, int RightEnd )
{   int  Center;
    if ( L < RightEnd ) { 有元素的时间执行
        Center = ( L + RightEnd ) / 2;
        MSort( A, TmpA, L, Center );
        MSort( A, TmpA, Center+1, RightEnd );
        Merge( A, TmpA, L, Center+1, RightEnd );
    }
}
```

$T(N) = T(N/2) + T(N/2) + O(N)$ ➡ $T(N) = O(N \log N)$ 这个时间复杂度很稳定

递归算法

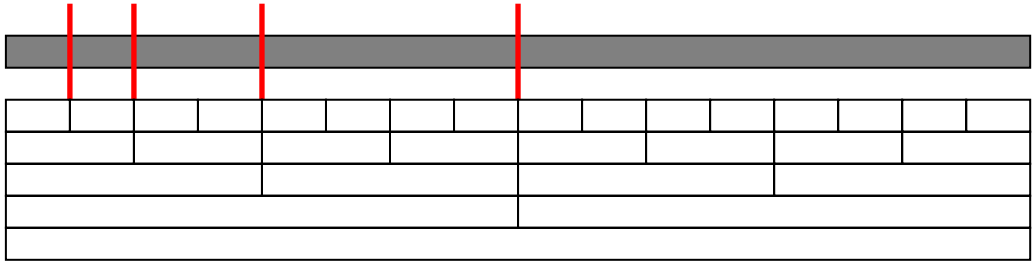


- 统一函数接口 (封装)

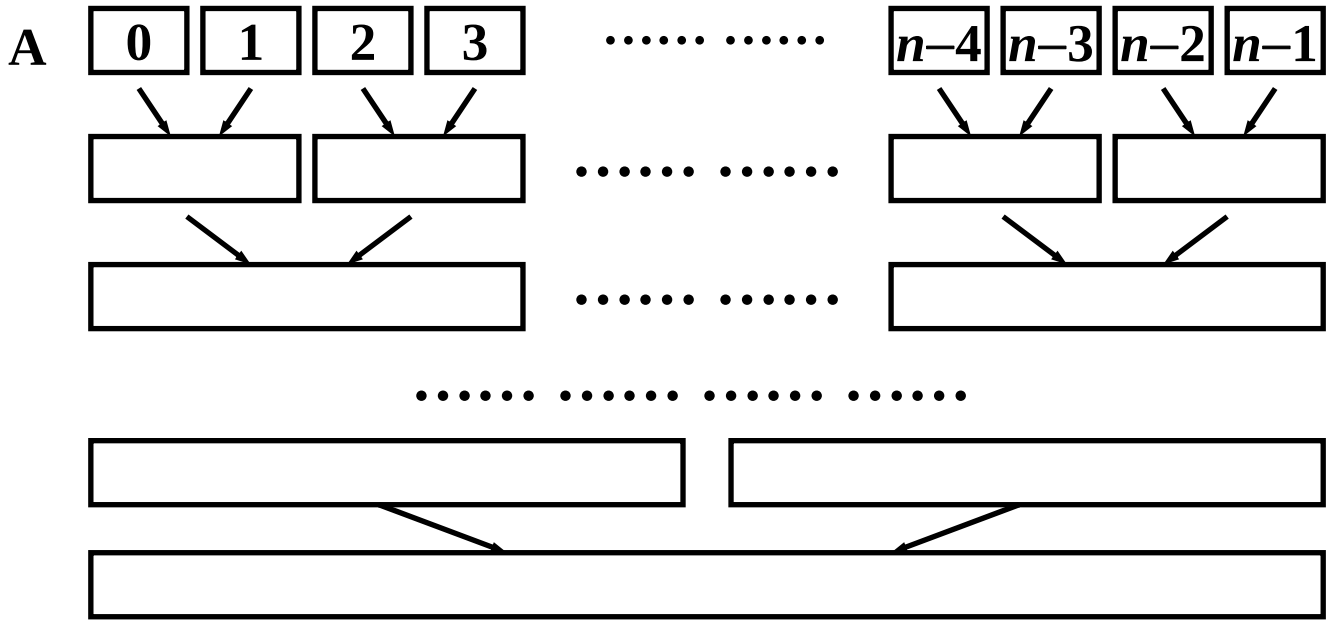
```
void Merge_sort( ElementType A[], int N )
{   ElementType  *TmpA;
    TmpA = malloc( N * sizeof( ElementType ) );
    if ( TmpA != NULL ) {
        MSort( A, TmpA, 0, N-1 );
        free( TmpA );
    }
    else Error( “空间不足" );
}
```

递归算法

- 如果只在**Merge**中声明临时数组 会造成重复申请释放临时数组，不如一开始申明临时数组
  - `void Merge( ElementType A[], int L, int R, int RightEnd )`
  - `void MSort( ElementType A[], int L, int RightEnd )`



非递归算法



额外空间复杂度是???　　$O(N)$

开辟一个临时数组即可，与需要归并的数组A来回导数据

非递归算法

将A中元素归并到TmpA (与传统归并到A)

```
void  Merge_pass( ElementType A[], ElementType TmpA[], int N,
                  int length ) /* length = 当前有序子列的长度 */
{  for ( i=0; i <= N-2*length; i += 2*length )
     Merge1( A, TmpA, i, i+length, i+2*length-1 );
   if ( i+length < N ) /* 归并最后2个子列 */
     Merge1( A, TmpA, i, i+length, N-1);
   else  /* 最后只剩1个子列 */
     for ( j = i; j < N; j++ ) TmpA[j] = A[j];
}
```

length初始为1，假定每一个元素开始为1个有序子序列，之后每次归并完长度加倍

非递归算法

```
void Merge_sort( ElementType A[], int N )
{   int length = 1;
    ElementType  *TmpA;与原始数组等长的临时数组
    TmpA = malloc( N * sizeof( ElementType ) );
    if ( TmpA != NULL ) {
        while( length < N ) {
            Merge_pass( A, TmpA, N, length );
            length *= 2;
            Merge_pass( TmpA, A, N, length );
            length *= 2;
        }
        free( TmpA );
     }
     else Error( “空间不足" );
}
```

稳定