9.4.1有序子列的归并

```
typedef int ElementType;
/* 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)
{
    int LeftEnd=R-1;  /* 左边终点位置，假设左右两列挨着 */
    int Tmp=L;  /* 存放结果的数组的初始位置 */
    int NumElements=RightEnd-L+1;
    int i;
    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];
}
```

9.4.2递归算法

分而治之

```
typedef int ElementType;

void MSort(ElementType A[],ElementType TmpA[],int L,int RightEnd)
{
    int Center;
    if(L<RightEnd)
    {
        Center=(L+RightEnd)/2;
        MSort(A,TmpA,L,Center);
        MSort(A,TmpA,Center+1,RightEnd);
        Merge(A,TmpA,L,Center+1,RightEnd);
    }
}
```

$$T(N)=T(N/2)+T(N/2)+O(N)$$

$$\Rightarrow T(N)=O(NlogN)$$

稳定算法

统一函数接口

```
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;
}
```

这个TmpA一定要在统一函数接口里面去声明，不能在Merge里面去声明，会产生额外的空间复杂度。

9.4.3非递归算法

```
void Merge_Sort(ElementType A[],int N)
{
    int length=1;
    ElementType *TmpA;
    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;
}
```

稳定的

外排序好用