

本节内容

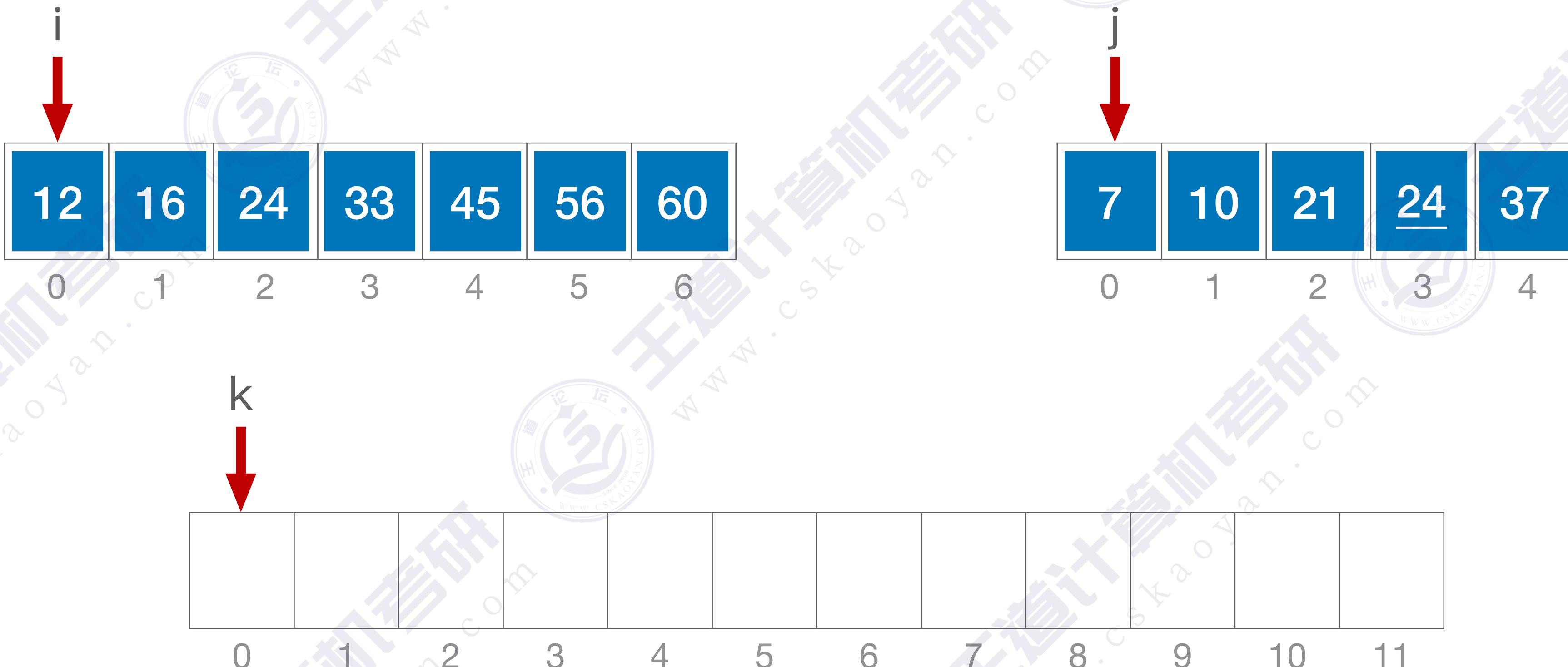
归并排序

(Merge Sort)

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

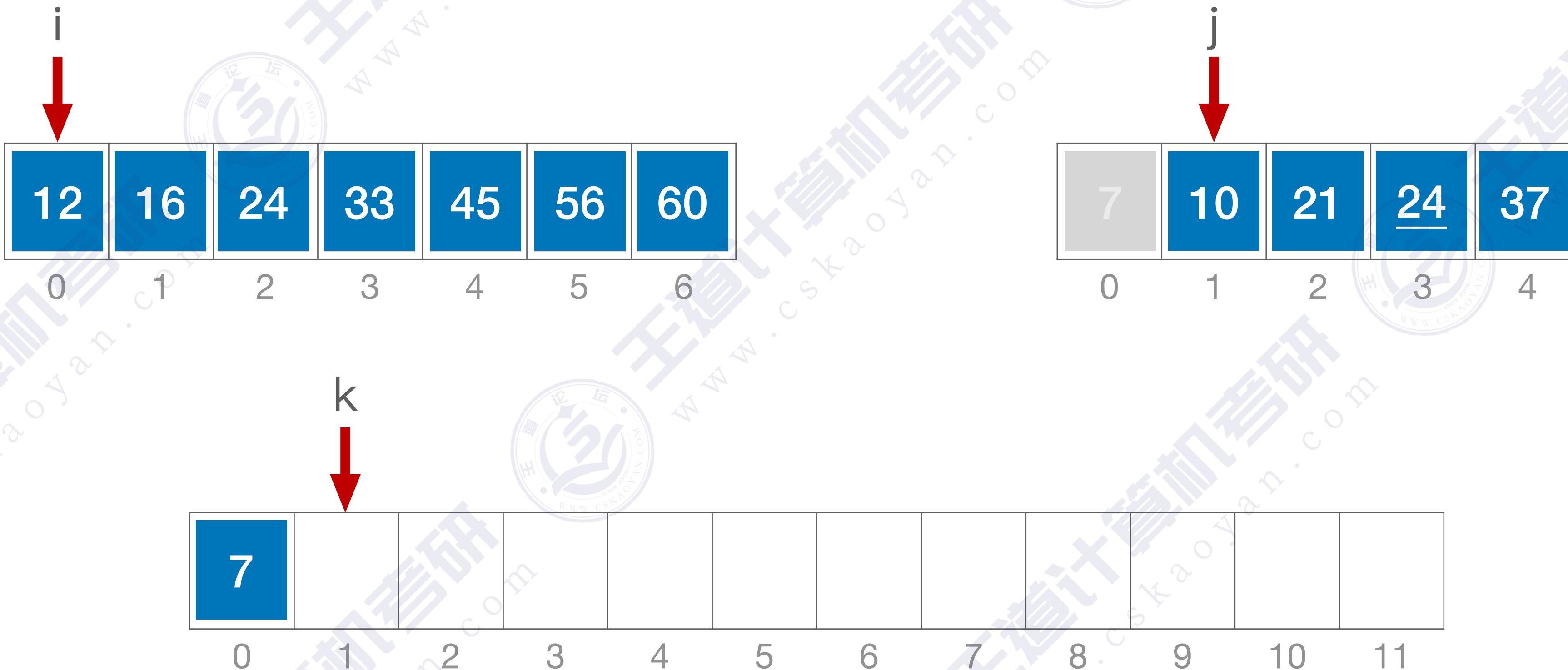


对比 i 、 j 所指元素，选择更小的一个放入 k 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

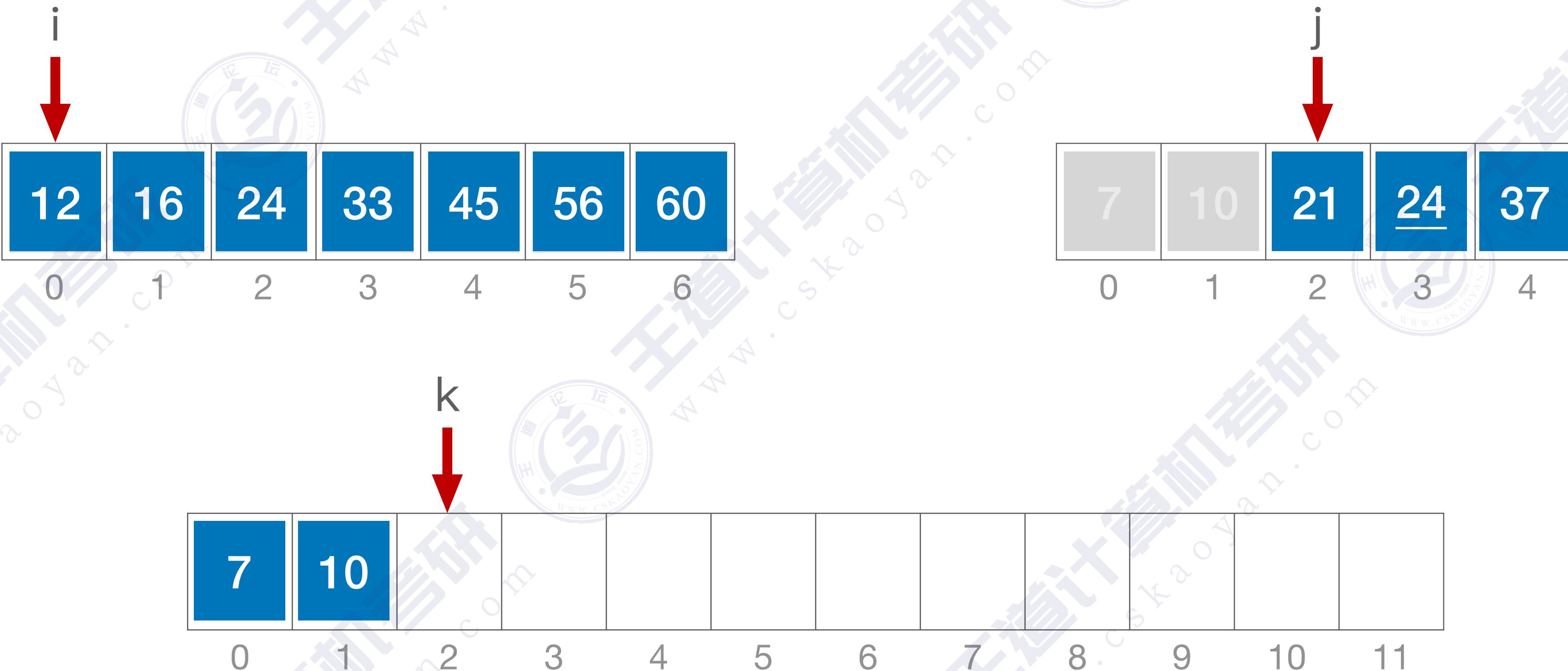


对比 **i, j** 所指元素，选择更小的一个放入 **k** 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

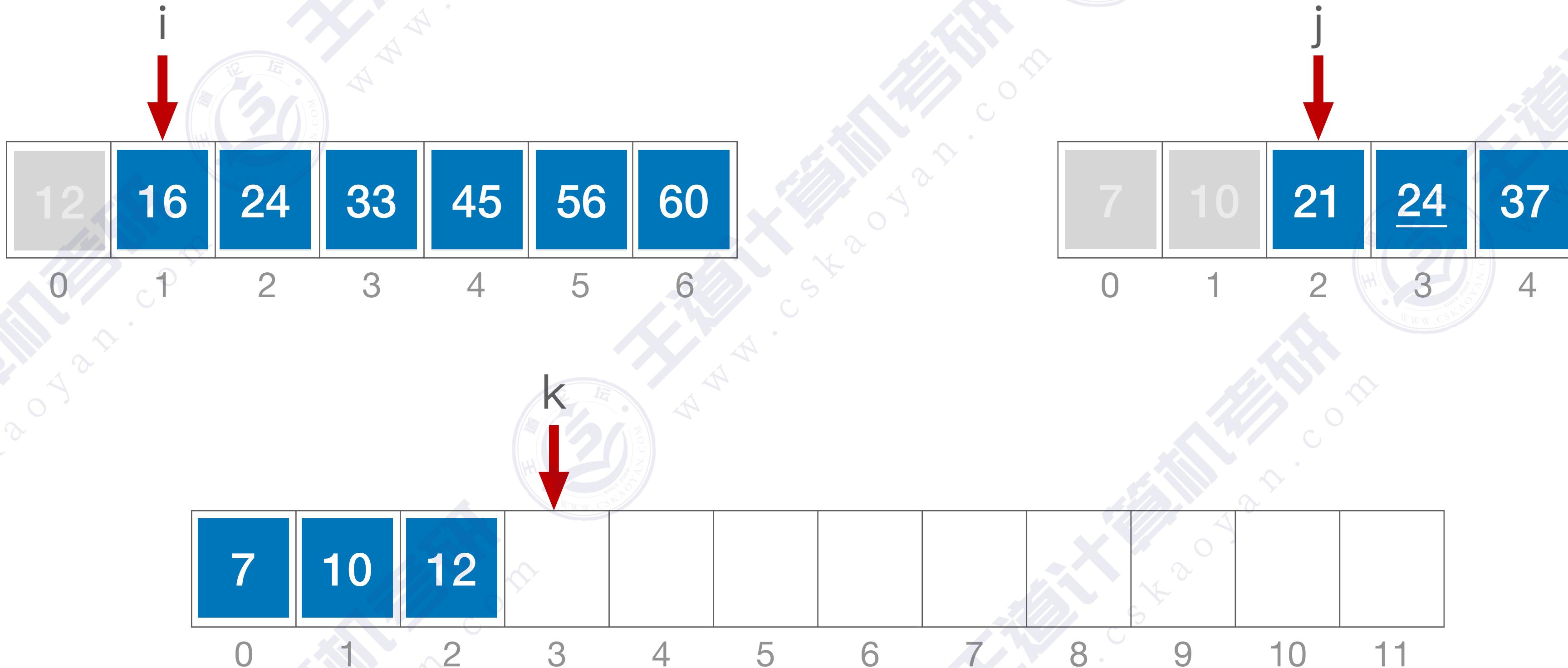


对比 **i, j** 所指元素，选择更小的一个放入 **k** 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

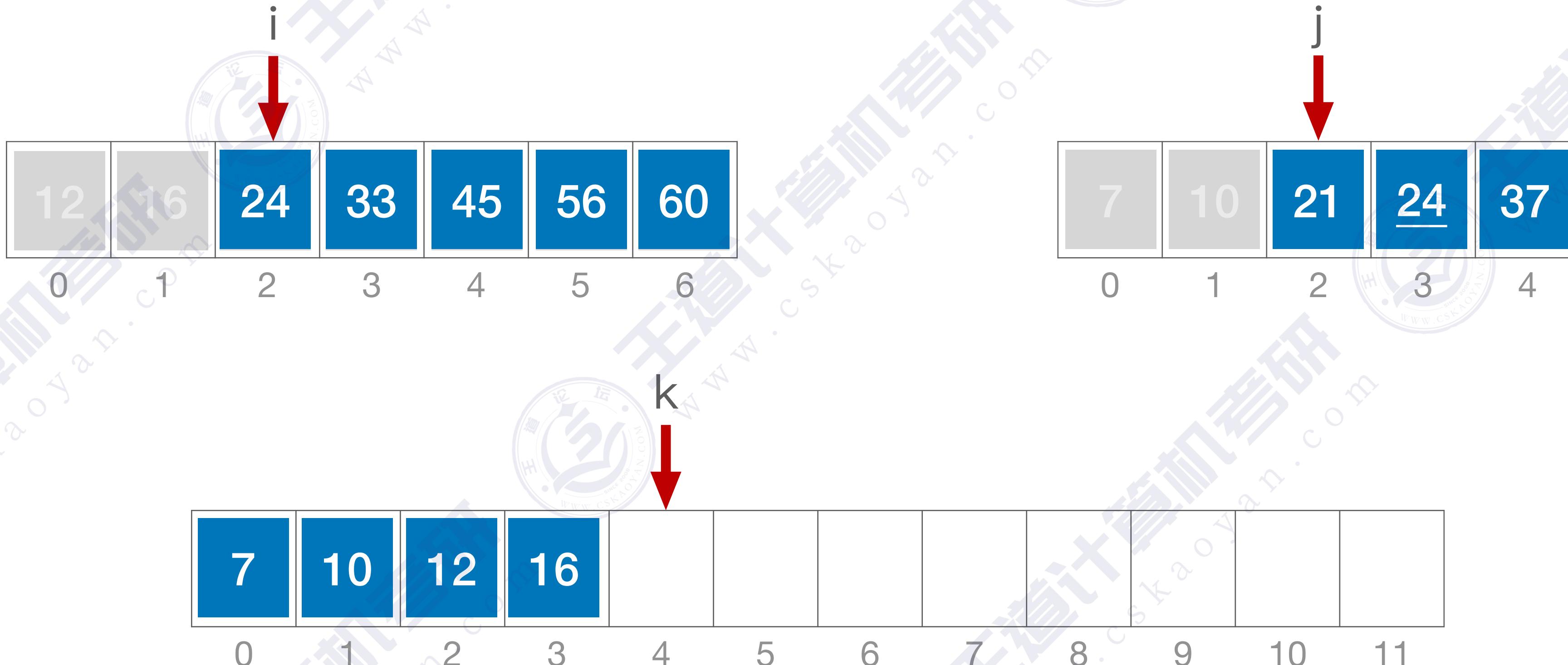


对比 i 、 j 所指元素，选择更小的一个放入 k 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

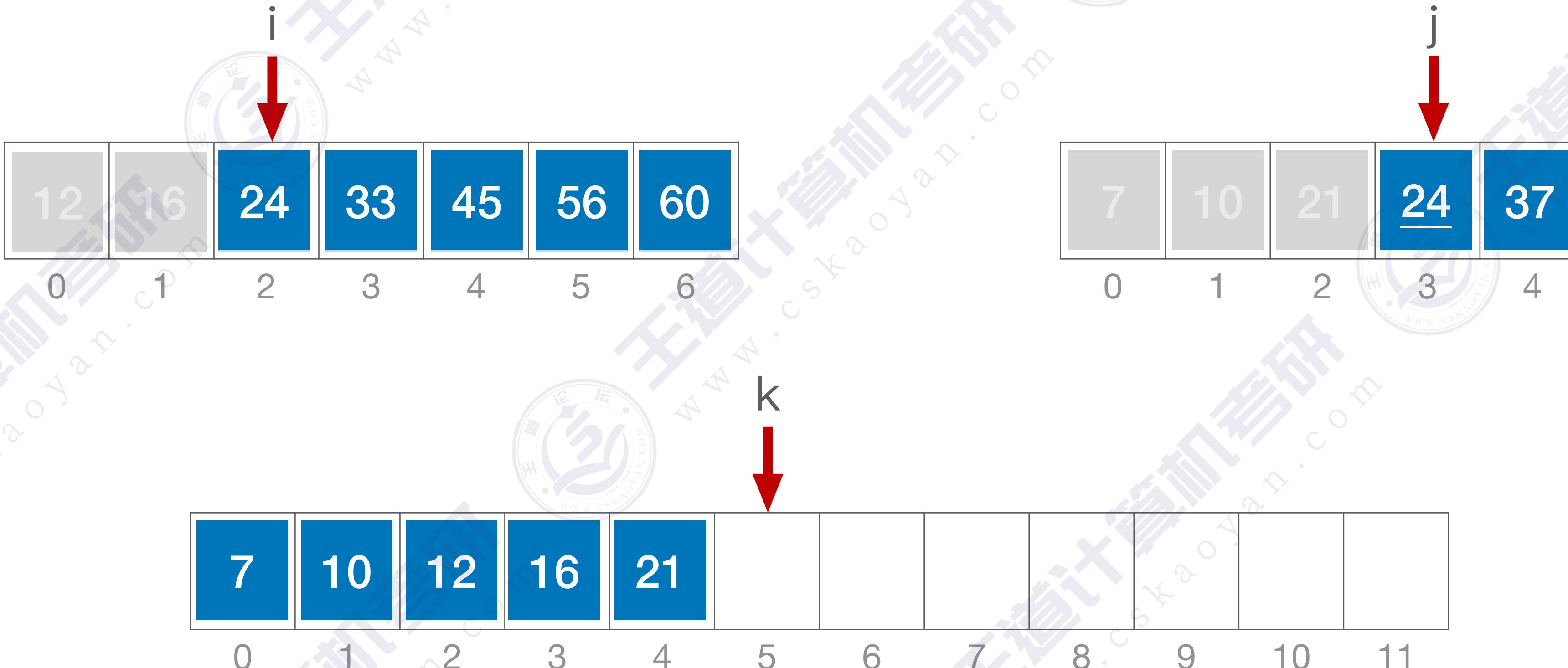


对比 *i*、*j* 所指元素，选择更小的一个放入 *k* 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

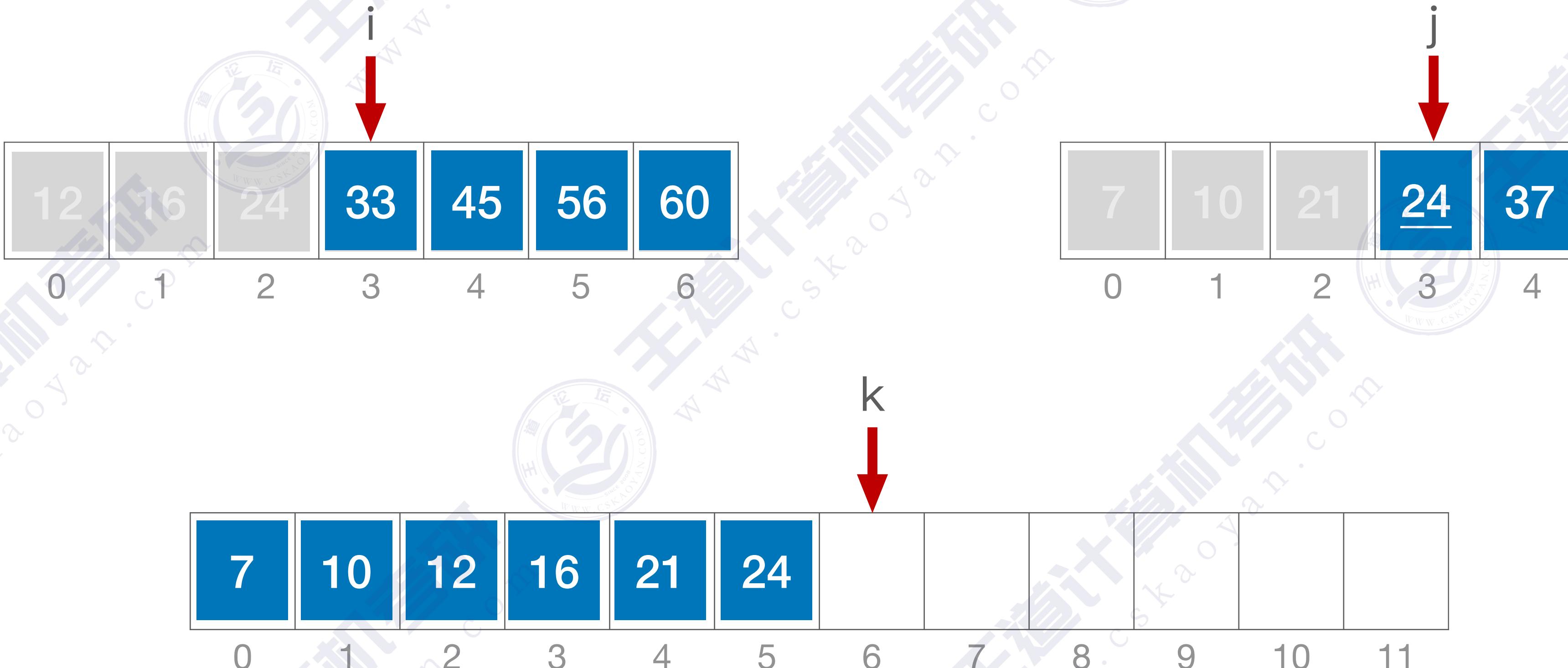


对比 i 、 j 所指元素，选择更小的一个放入 k 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

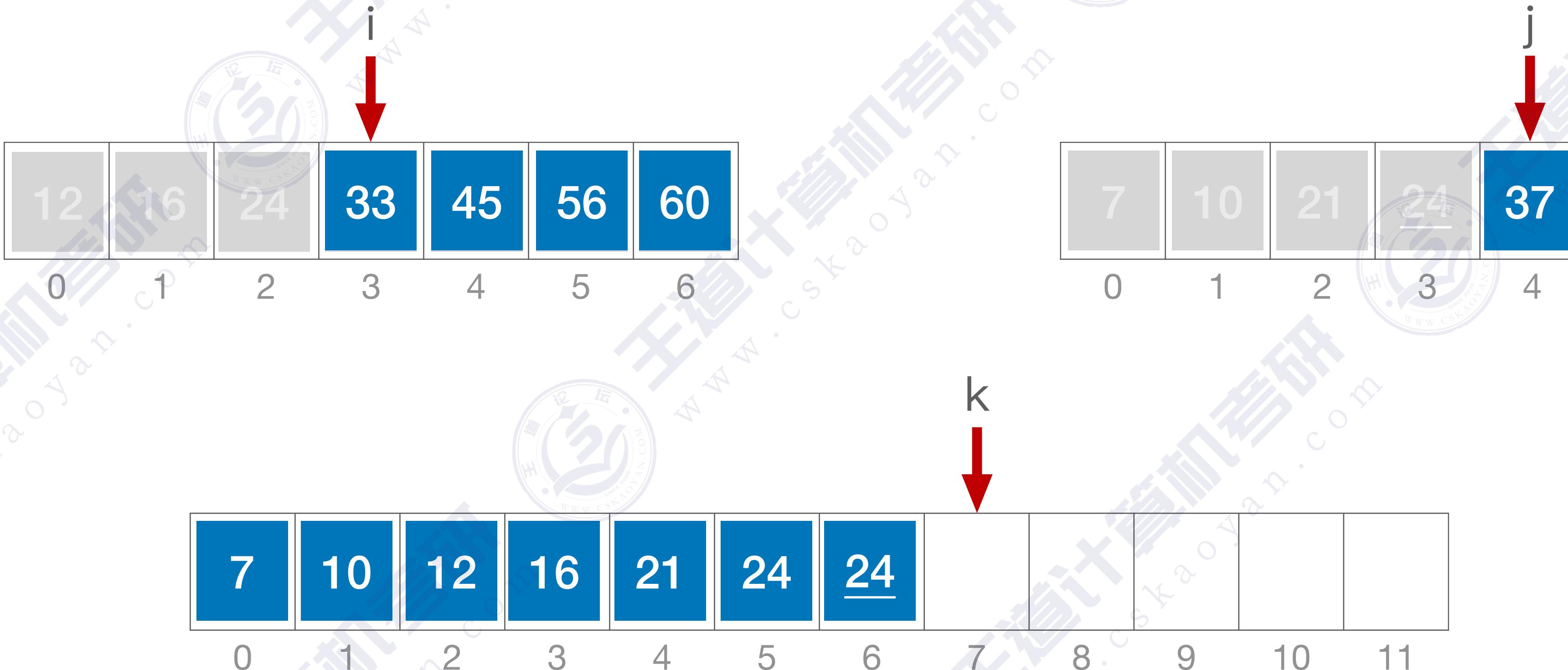


对比 i 、 j 所指元素，选择更小的一个放入 k 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个

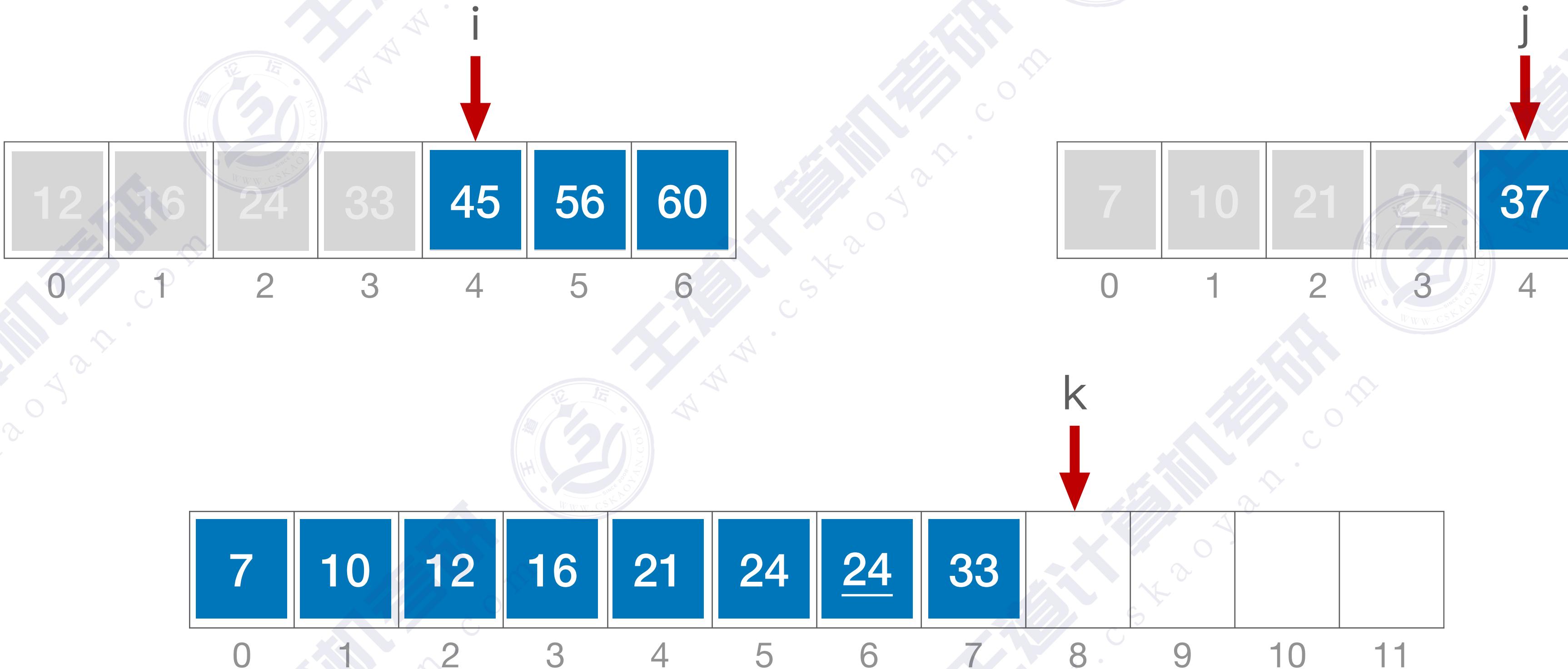


对比 i 、 j 所指元素，选择更小的一个放入 k 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个



对比 i 、 j 所指元素，选择更小的一个放入 k 所指位置

什么是 Merge (归并/合并) ?

归并：把两个或多个已经有序的序列合并成一个

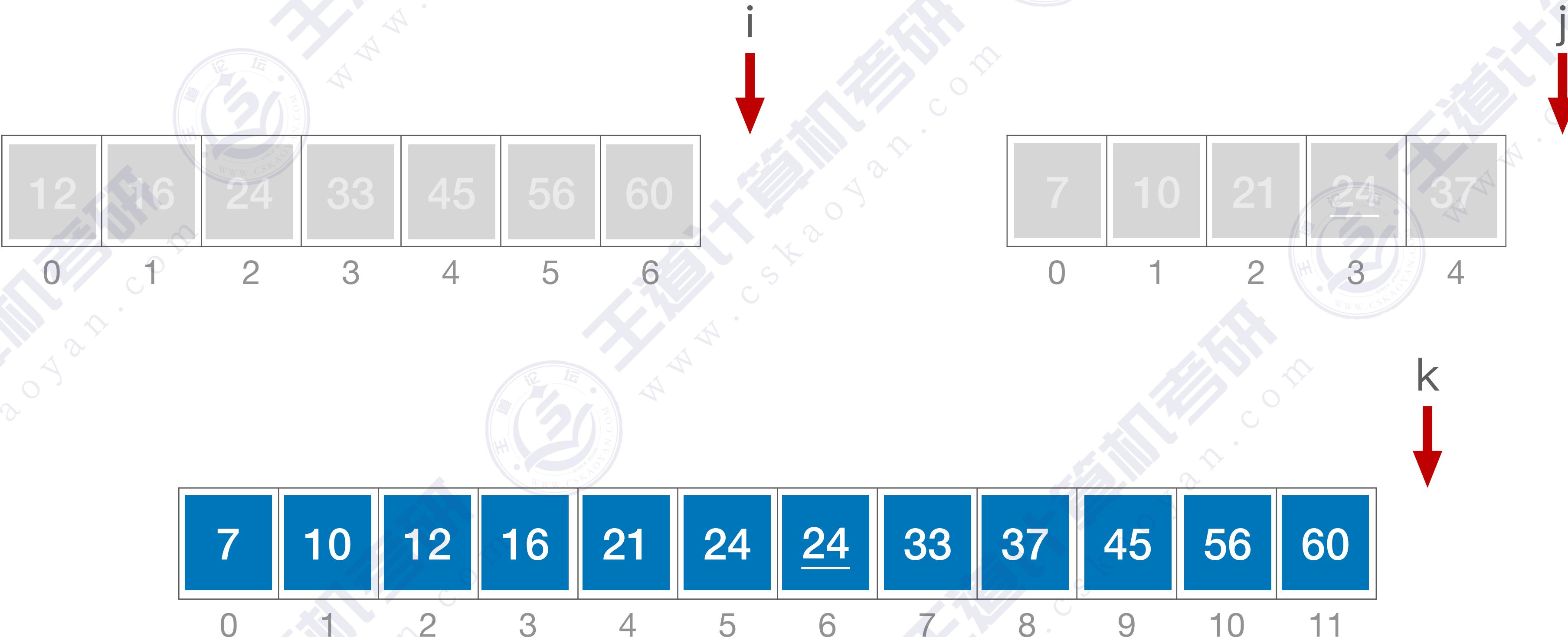


对比 i 、 j 所指元素, 选择更小的一个放入 k 所指位置

什么是 Merge (归并/合并) ?



归并：把两个或多个已经有序的序列合并成一个



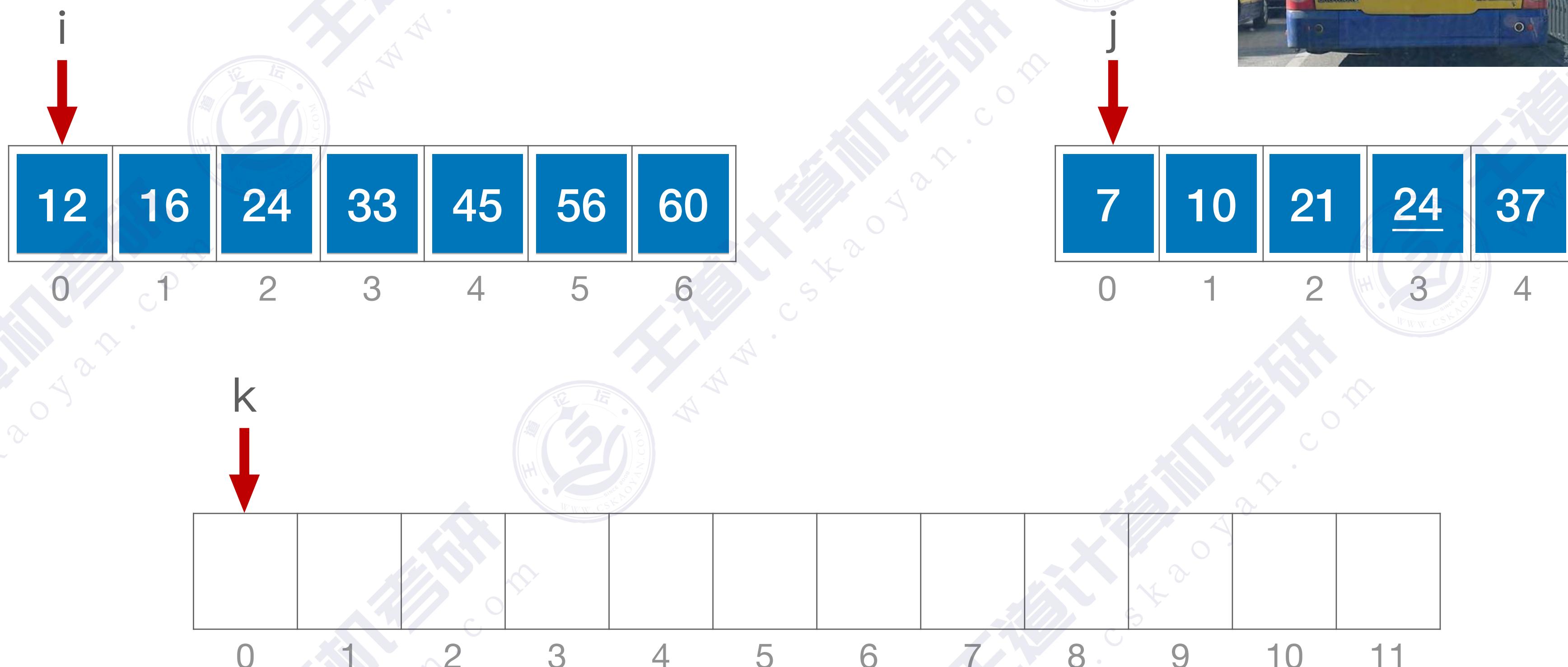
对比 i 、 j 所指元素，选择更小的一个放入 k 所指位置

“2路”——二合一

“2路”归并



归并：把**两个**或多个已经有序的序列合并成一个



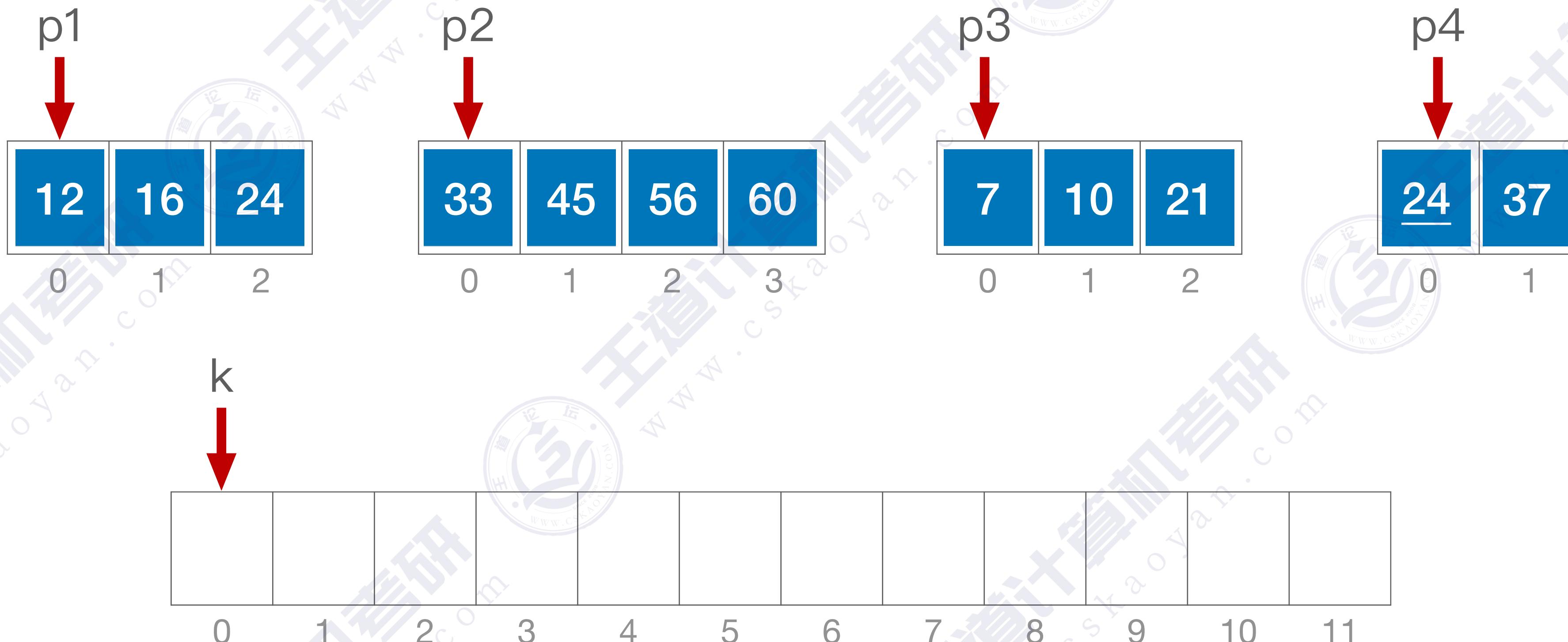
“2路”归并——每选出一个元素
注需对比关键字1次

对比 i、j 所指元素，选择更小的一个放入 k 所指位置



“4路”——四合一 → “4路”归并

归并：把两个或多个已经有序的序列合并成一个



“4路”归并——每选出一个元素
注需对比关键字3次

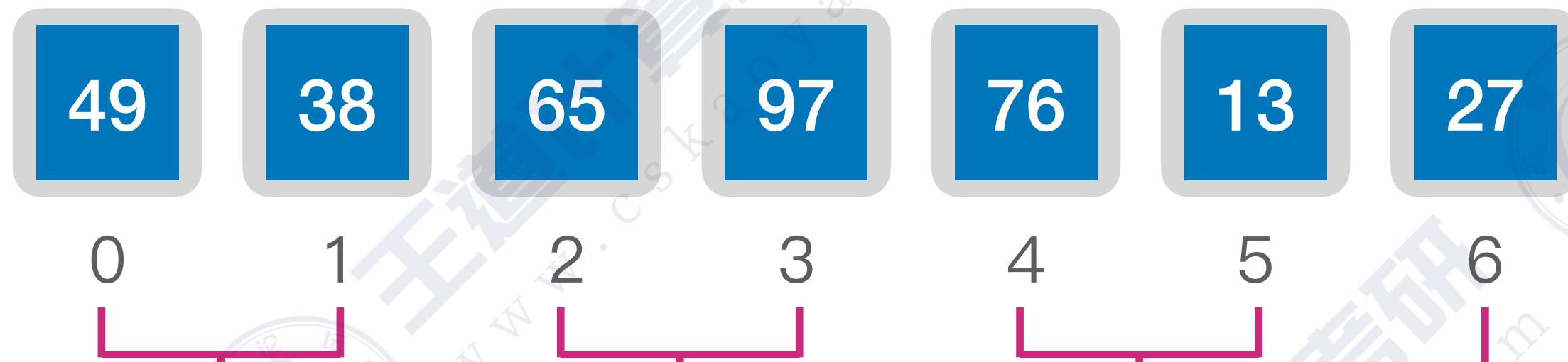
对比 p_1 、 p_2 、 p_3 、 p_4 所指元素，选择更小的一个放入 k 所指位置

结论：m路归并，每选出一个元素需要对比关键字 $m-1$ 次

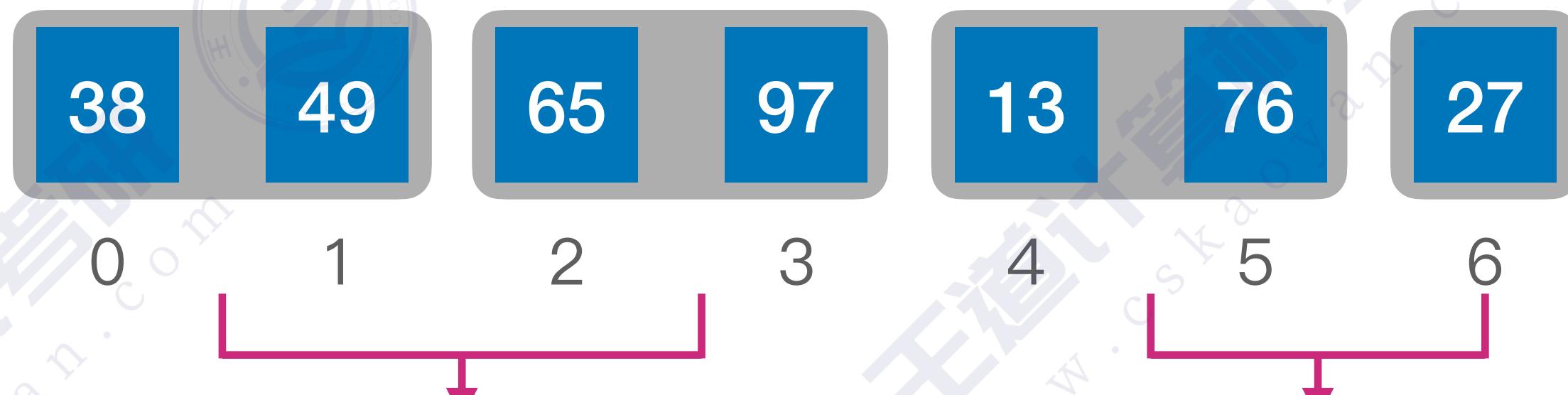
在内部排序中一般采用2路归并

归并排序 (手算模拟)

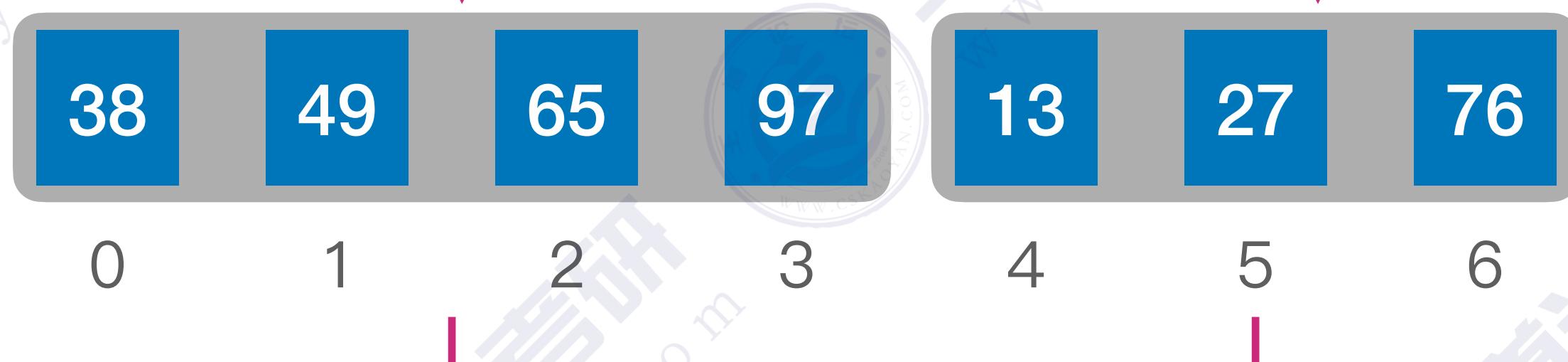
初始序列:



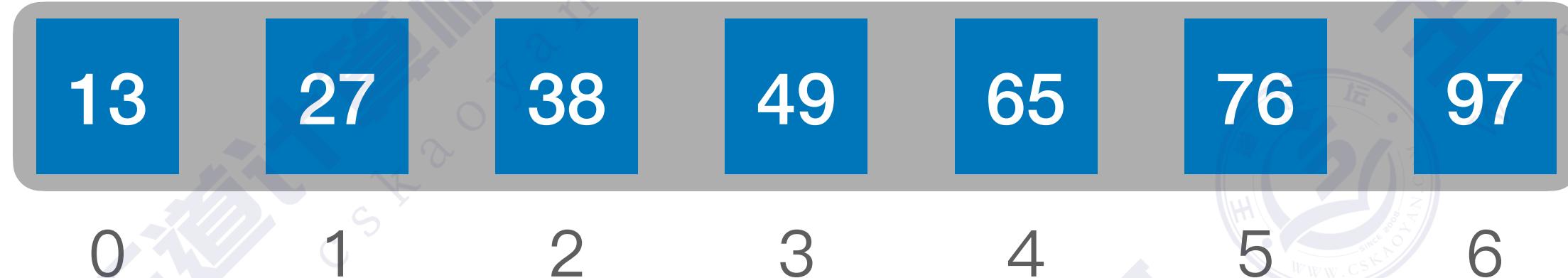
一趟归并后:



二趟归并后:

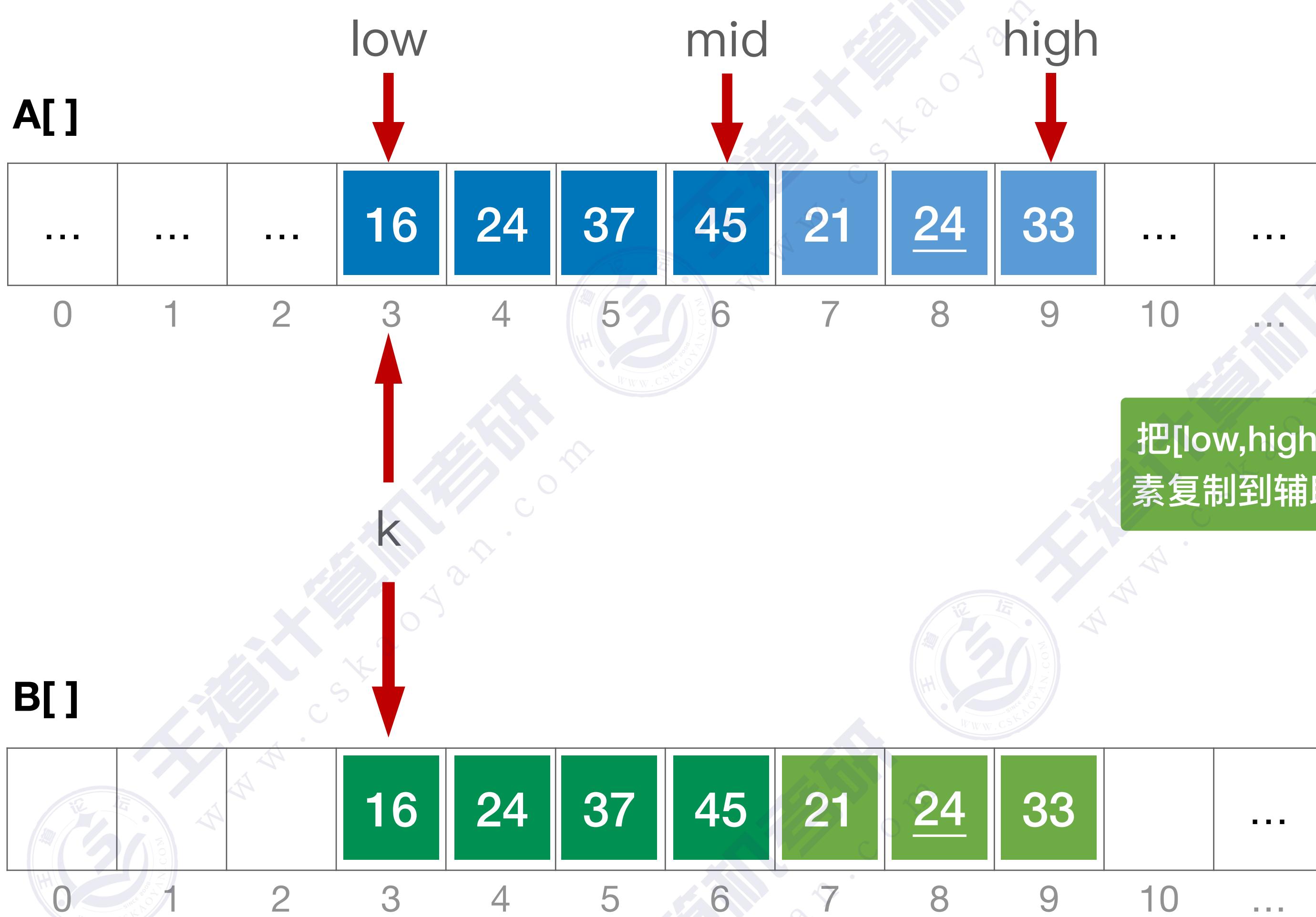


三趟归并后:



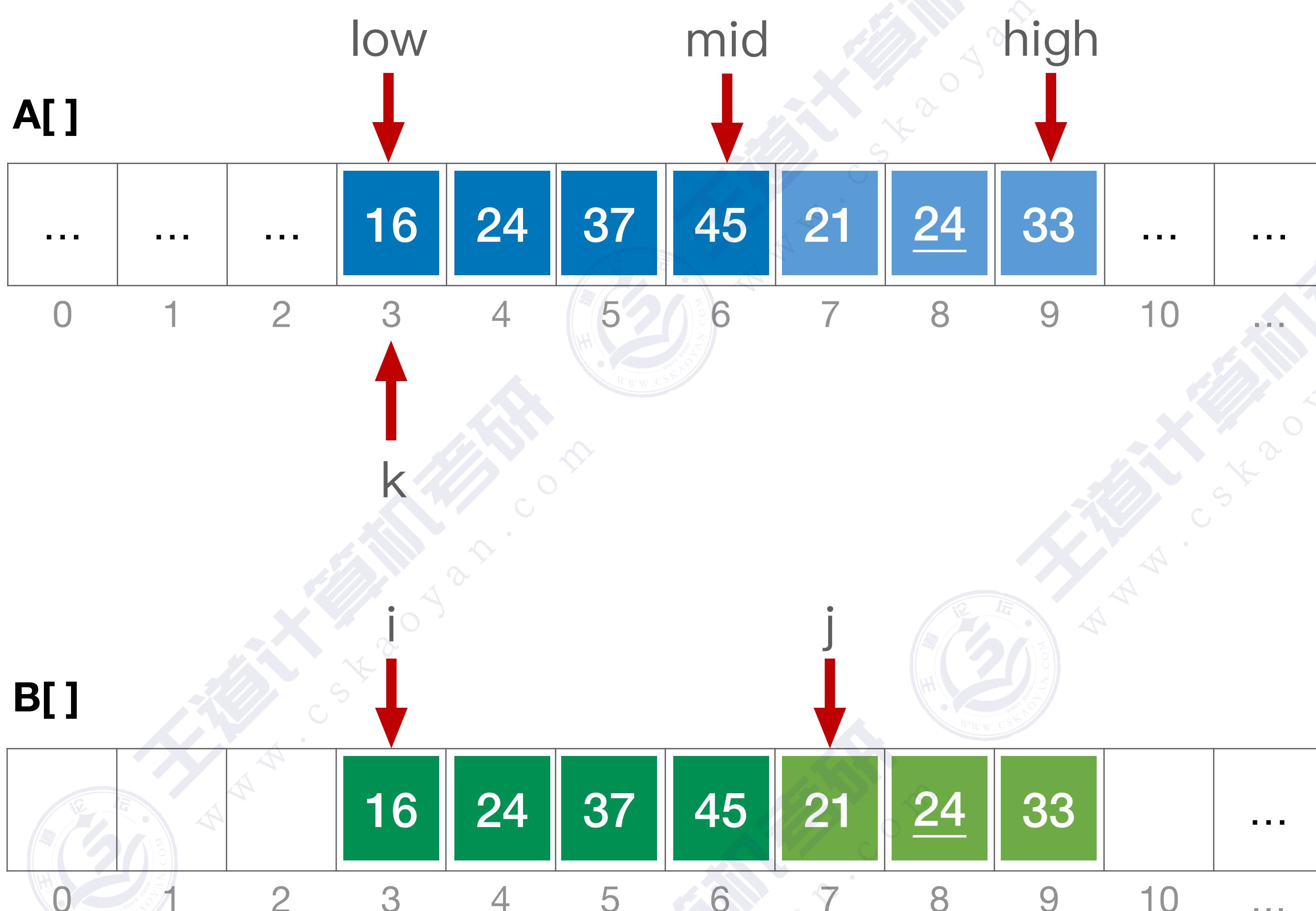
核心操作: 把数组内的两个有序序列归并为一个

代码实现



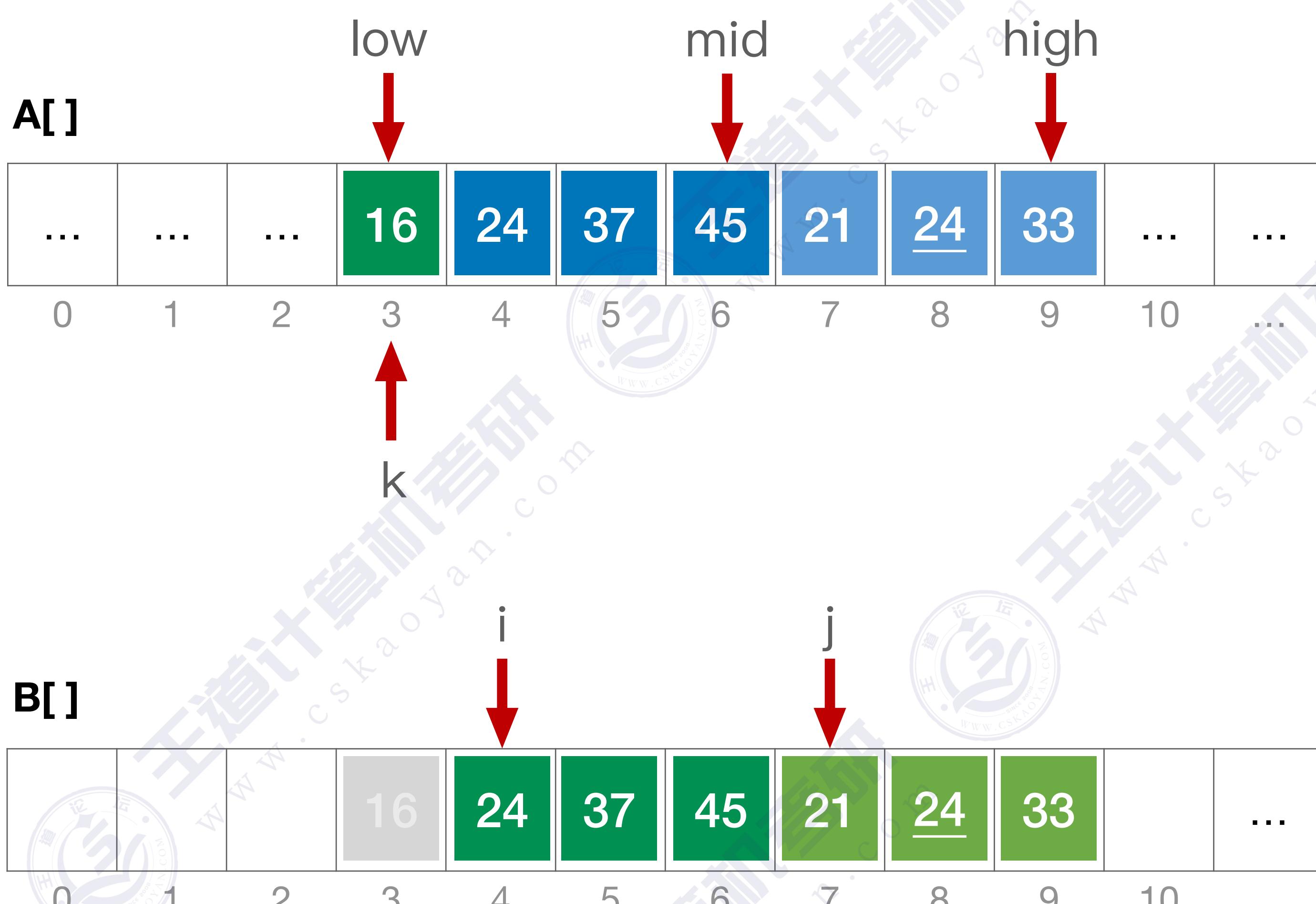
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B  
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并  
void Merge(int A[], int low, int mid, int high){  
    int i, j, k;  
    for(k=low; k<=high; k++)  
        B[k]=A[k]; //将A中所有元素复制到B中  
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){  
        if(B[i]<=B[j])  
            A[k]=B[i++]; //将较小值复制到A中  
        else  
            A[k]=B[j++];  
    } //for  
    while(i<=mid) A[k++]=B[i++];  
    while(j<=high) A[k++]=B[j++];  
}
```

代码实现



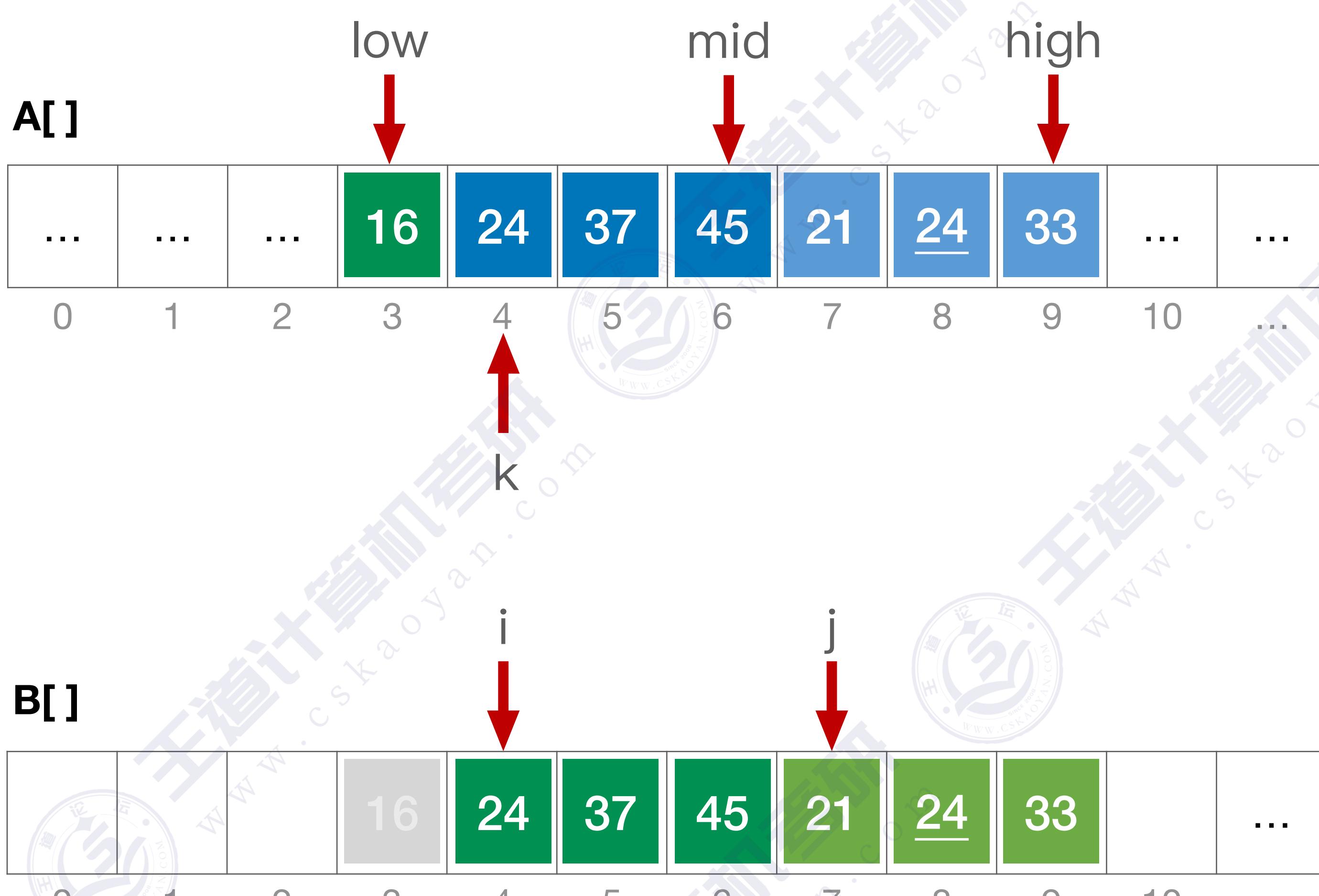
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



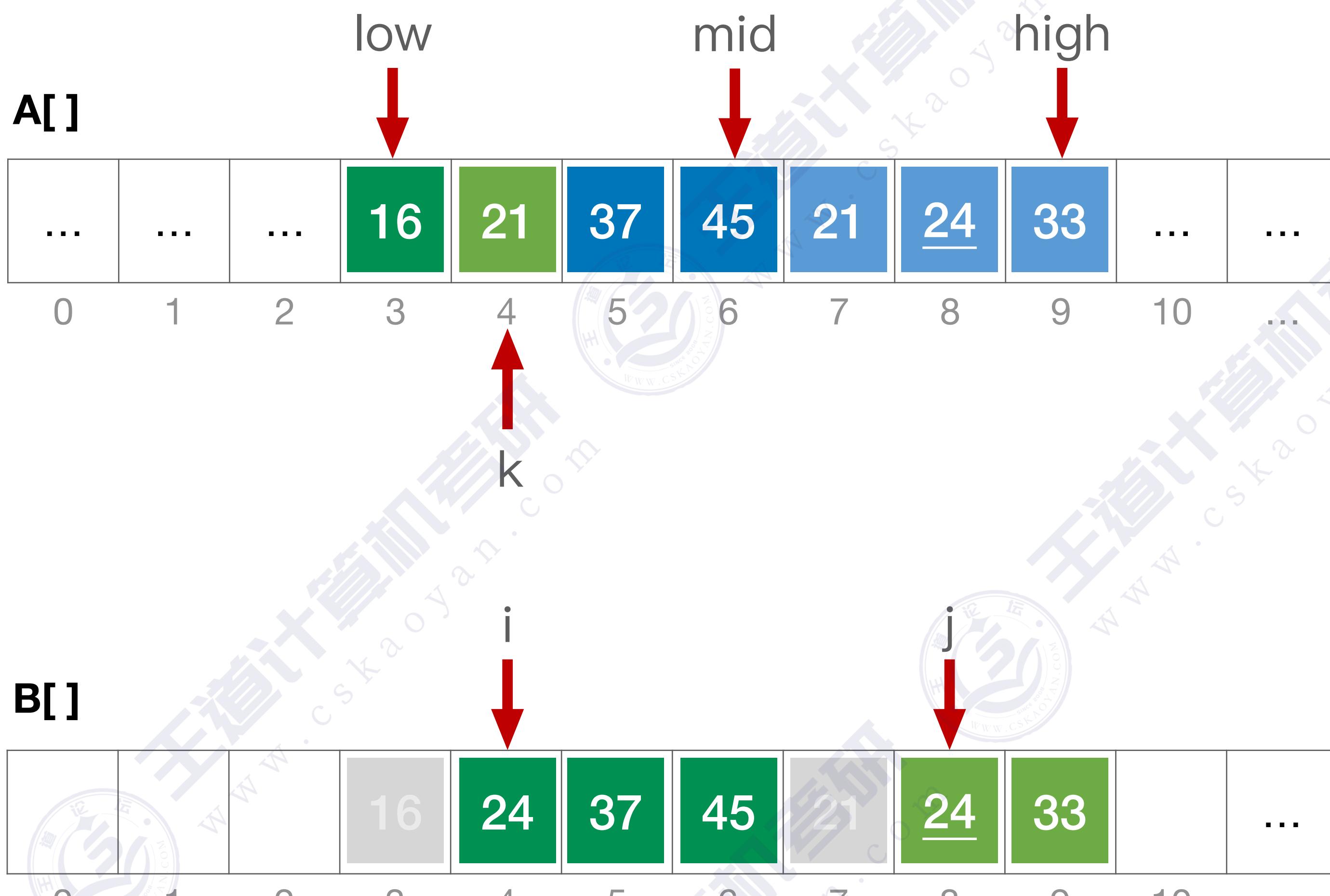
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



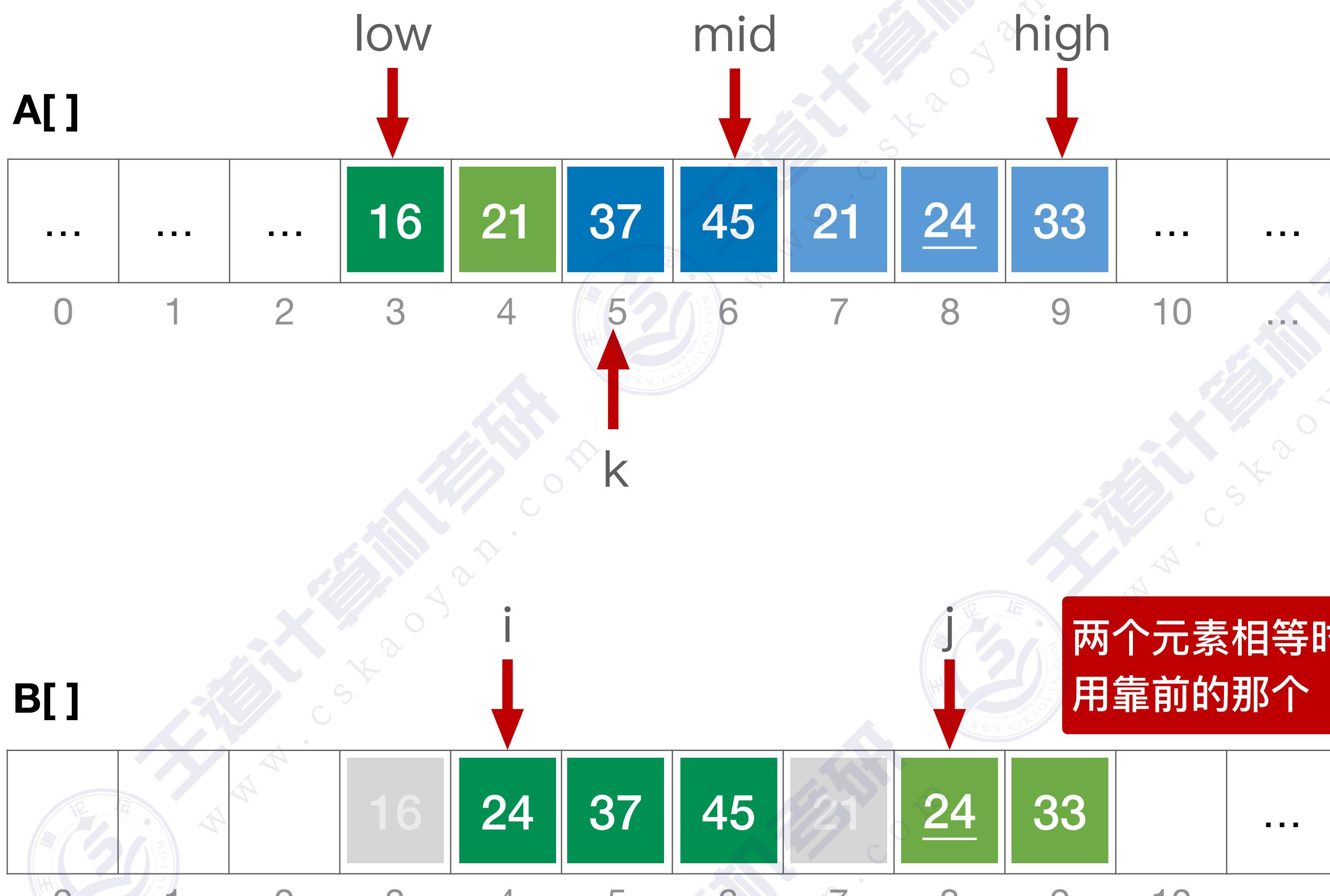
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



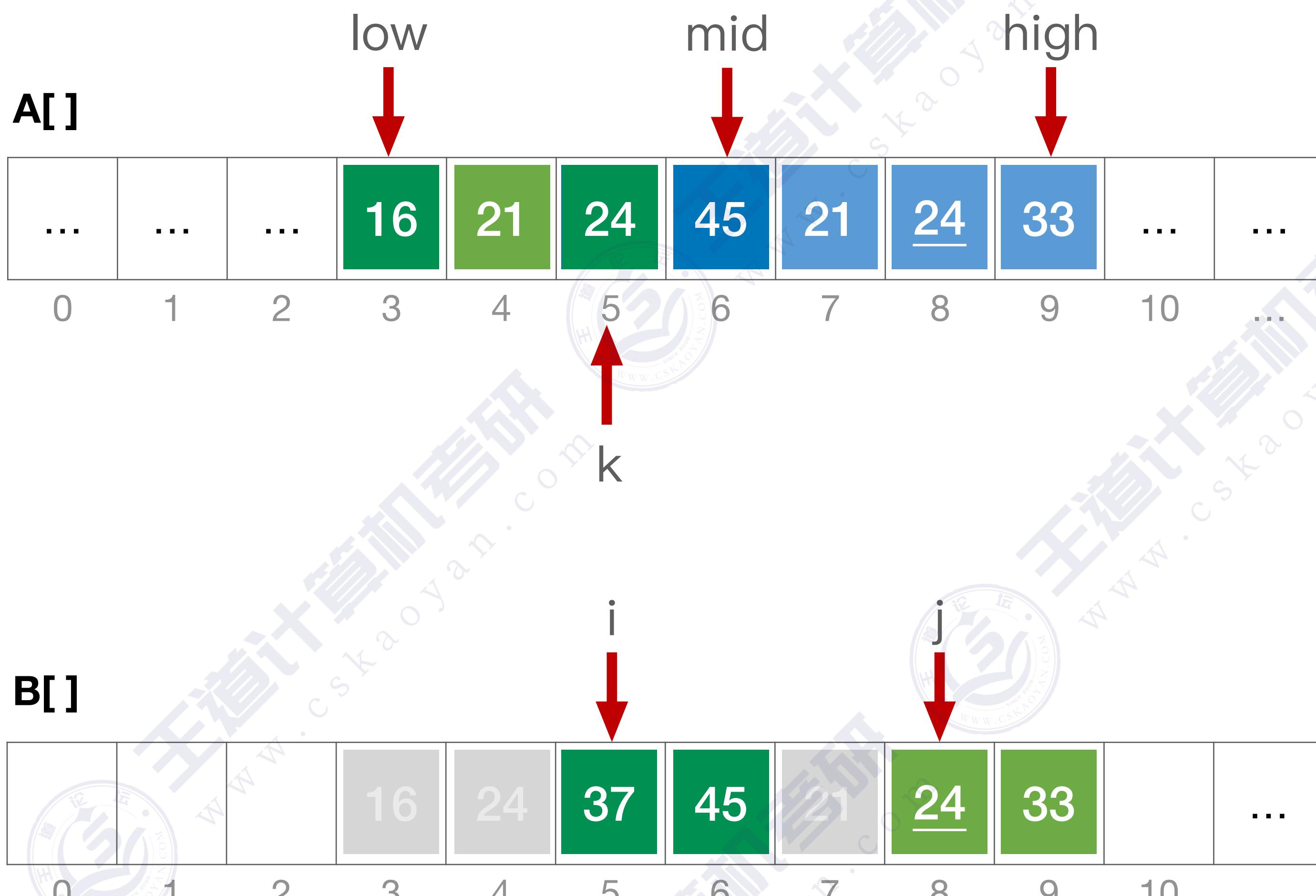
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



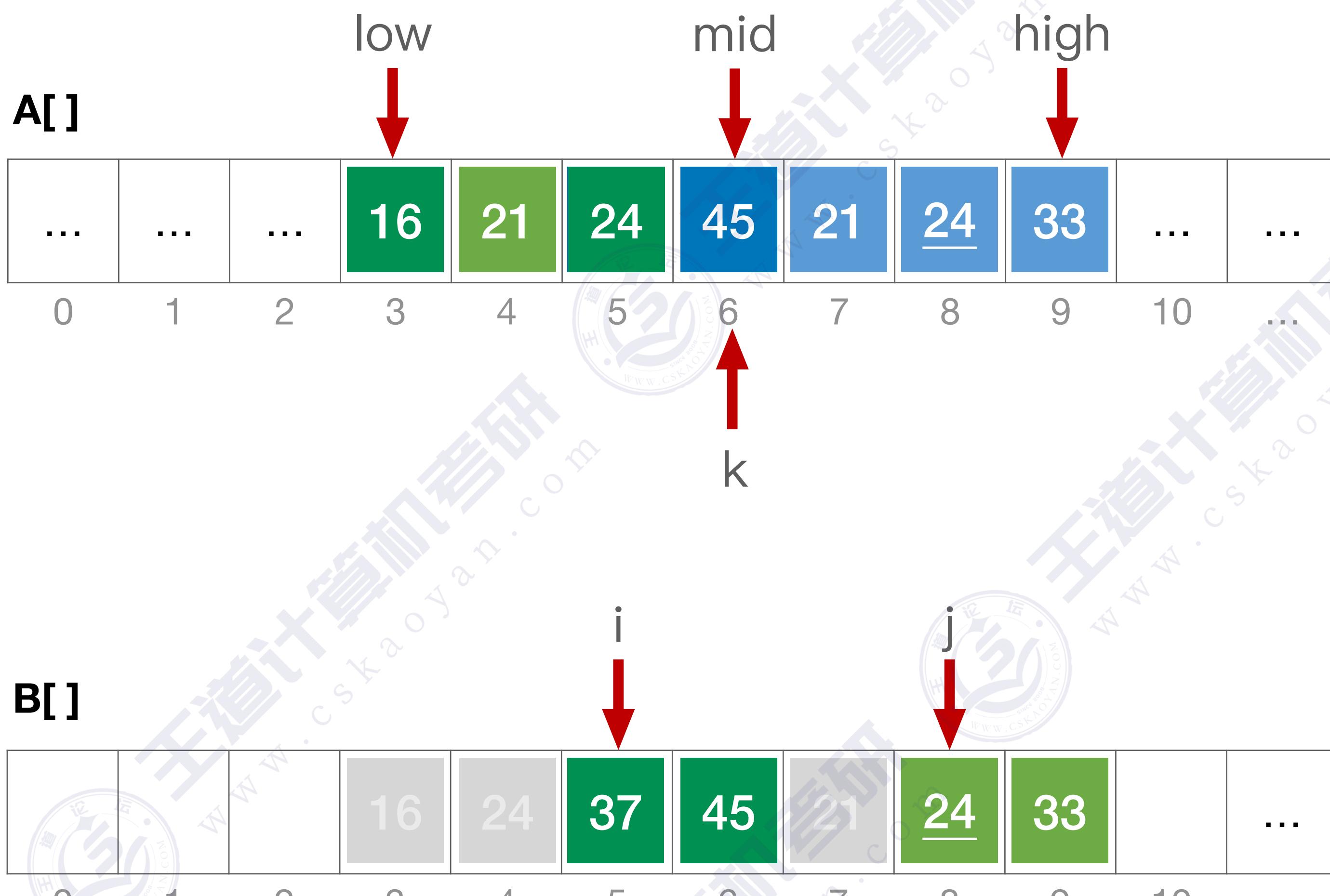
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



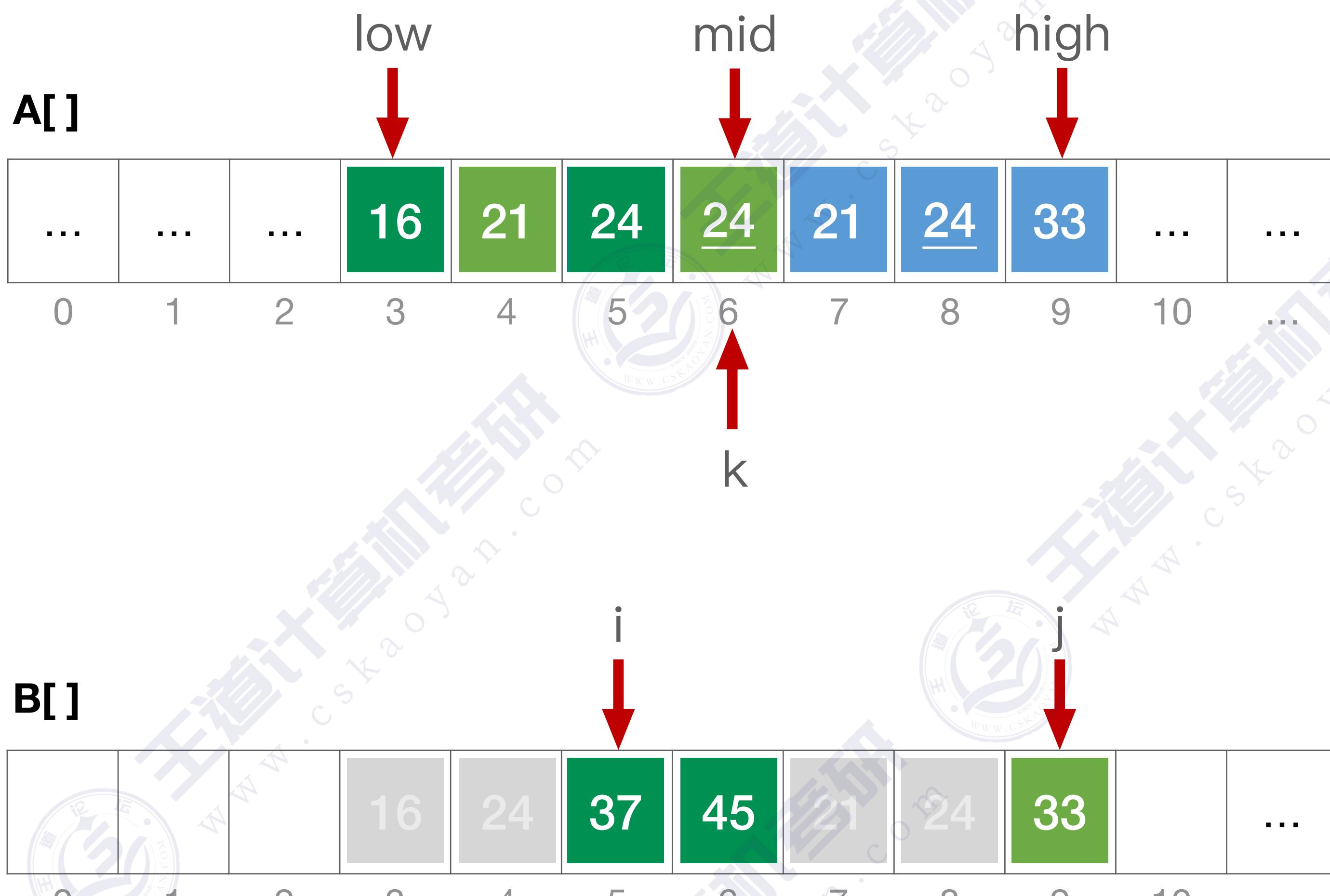
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



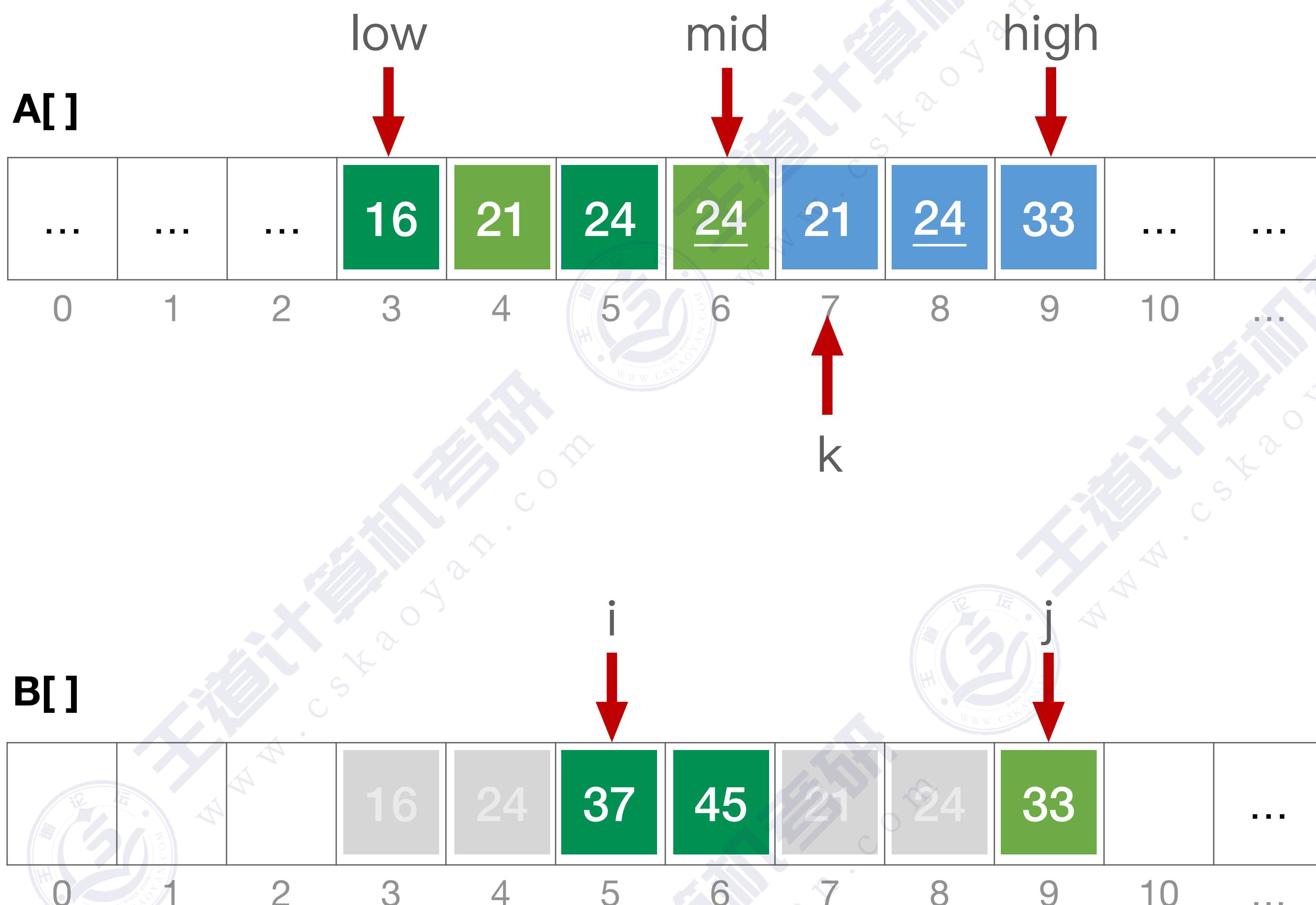
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



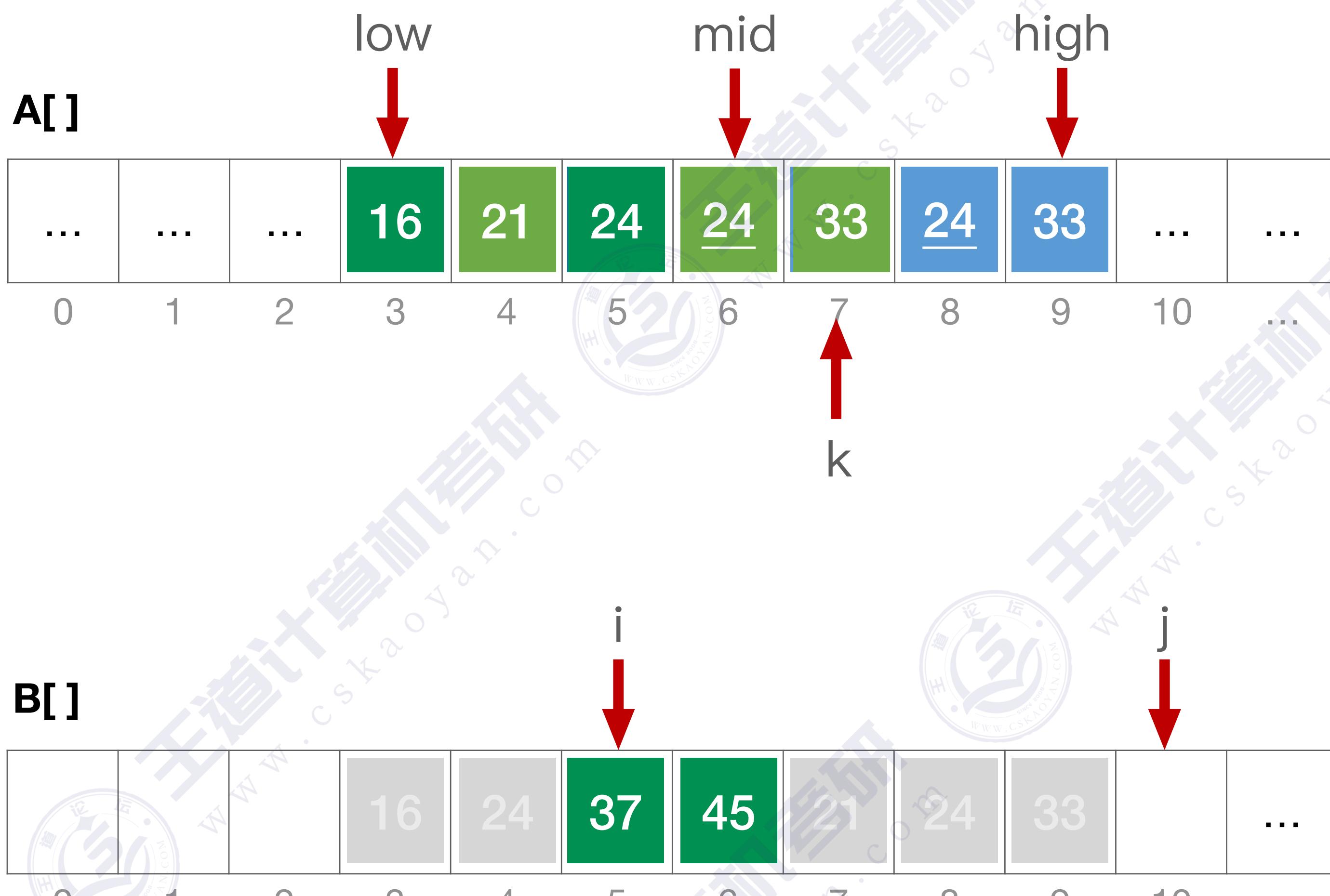
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



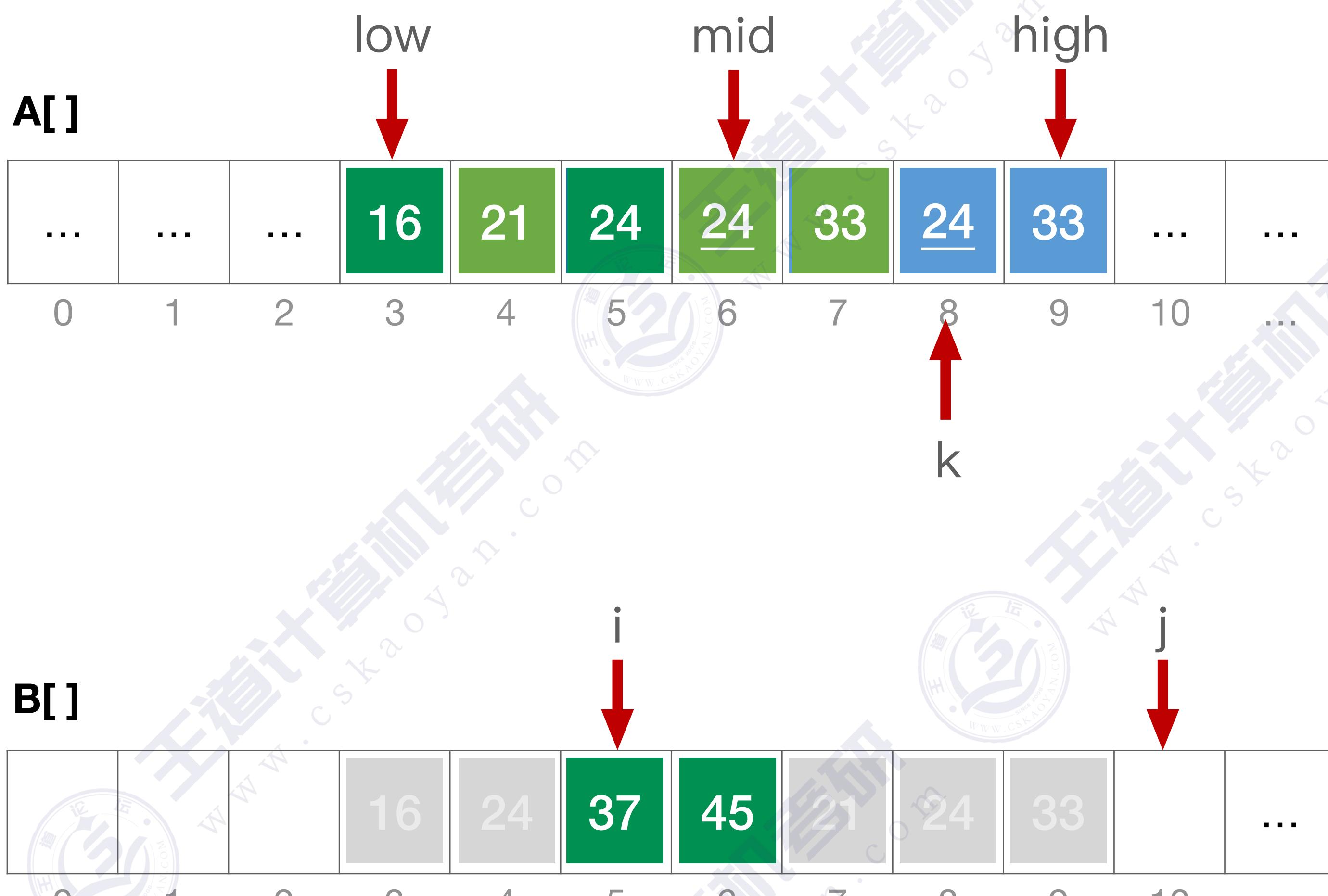
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

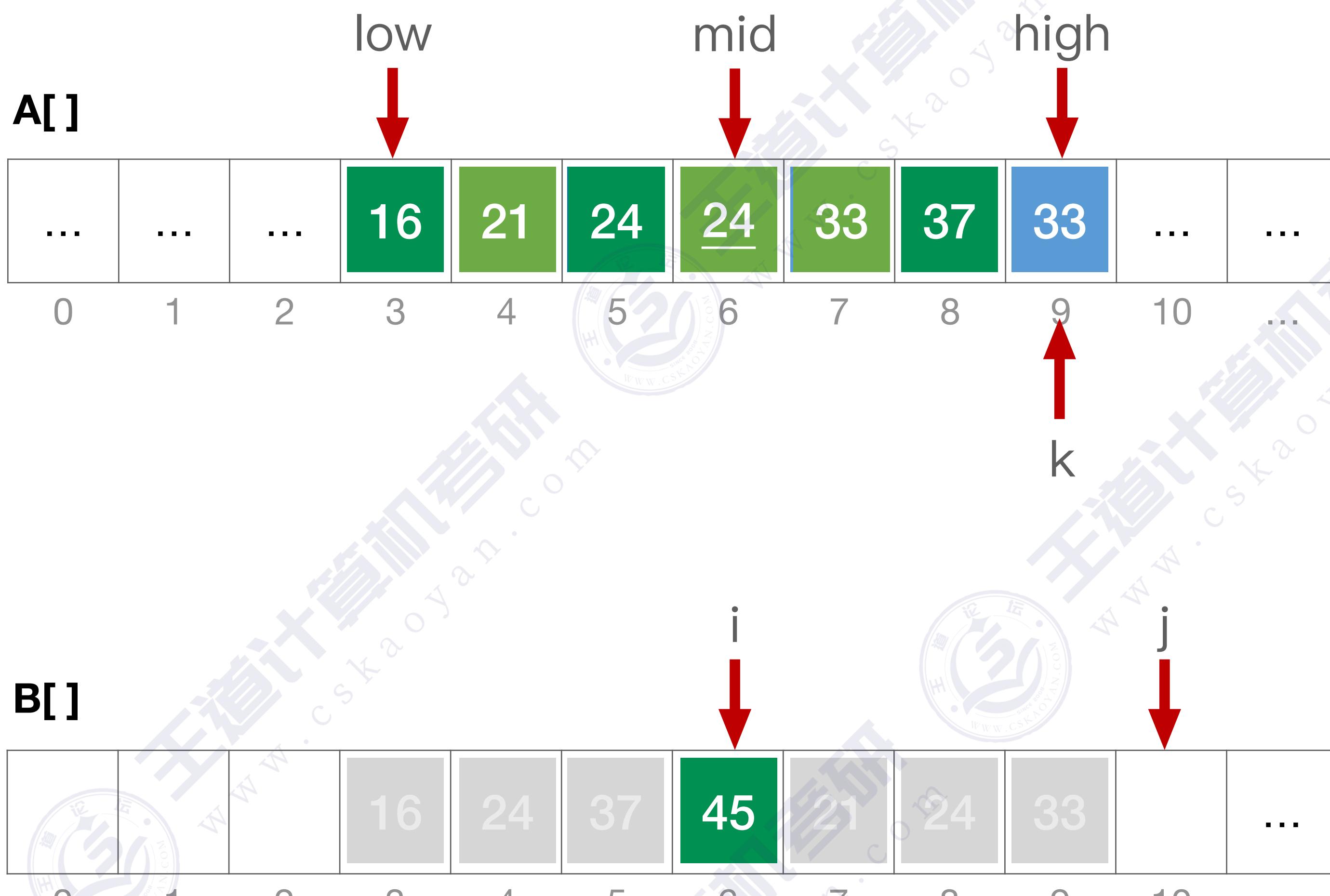
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

没有归并完的部分复制到尾部

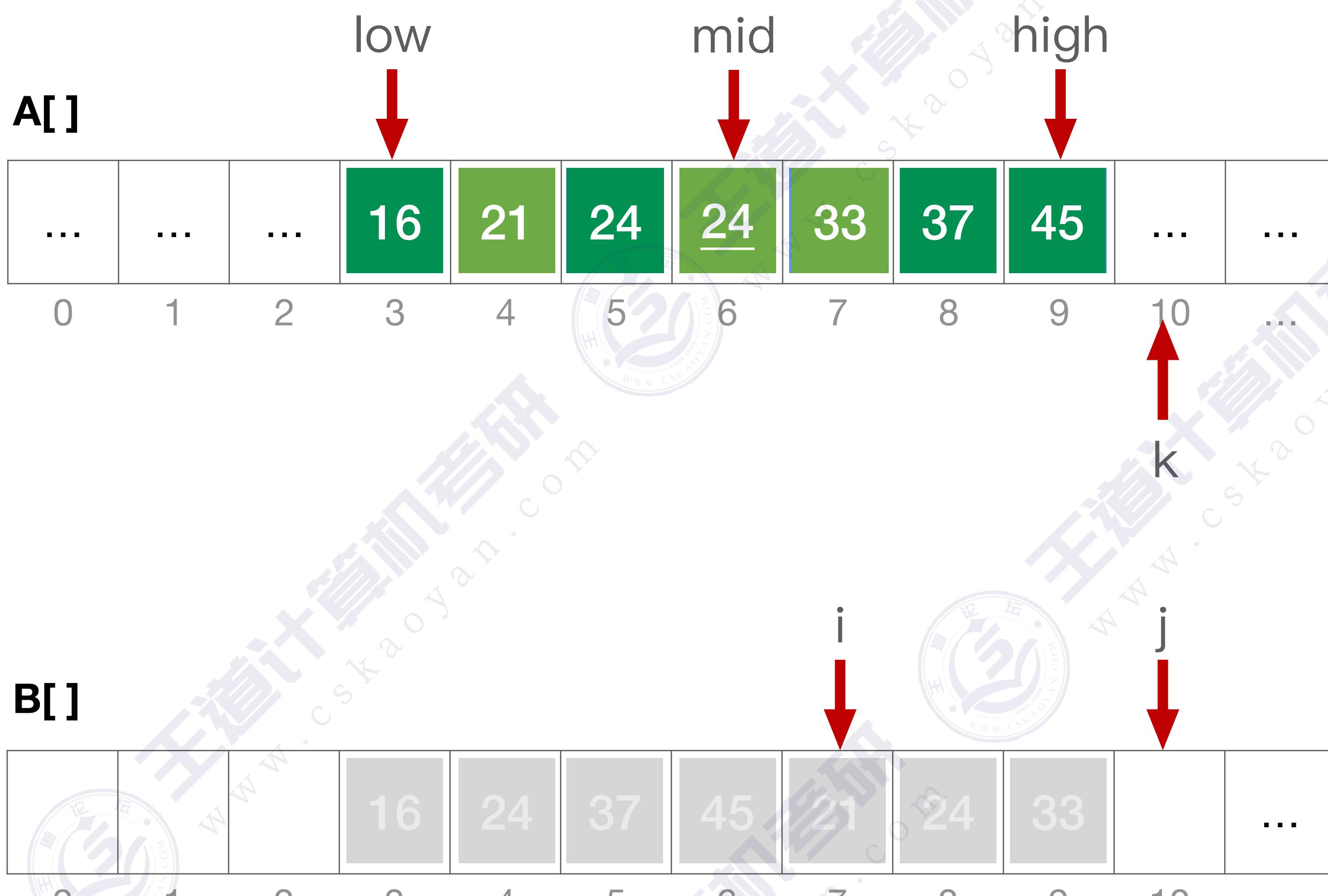
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

没有归并完的部分复制到尾部

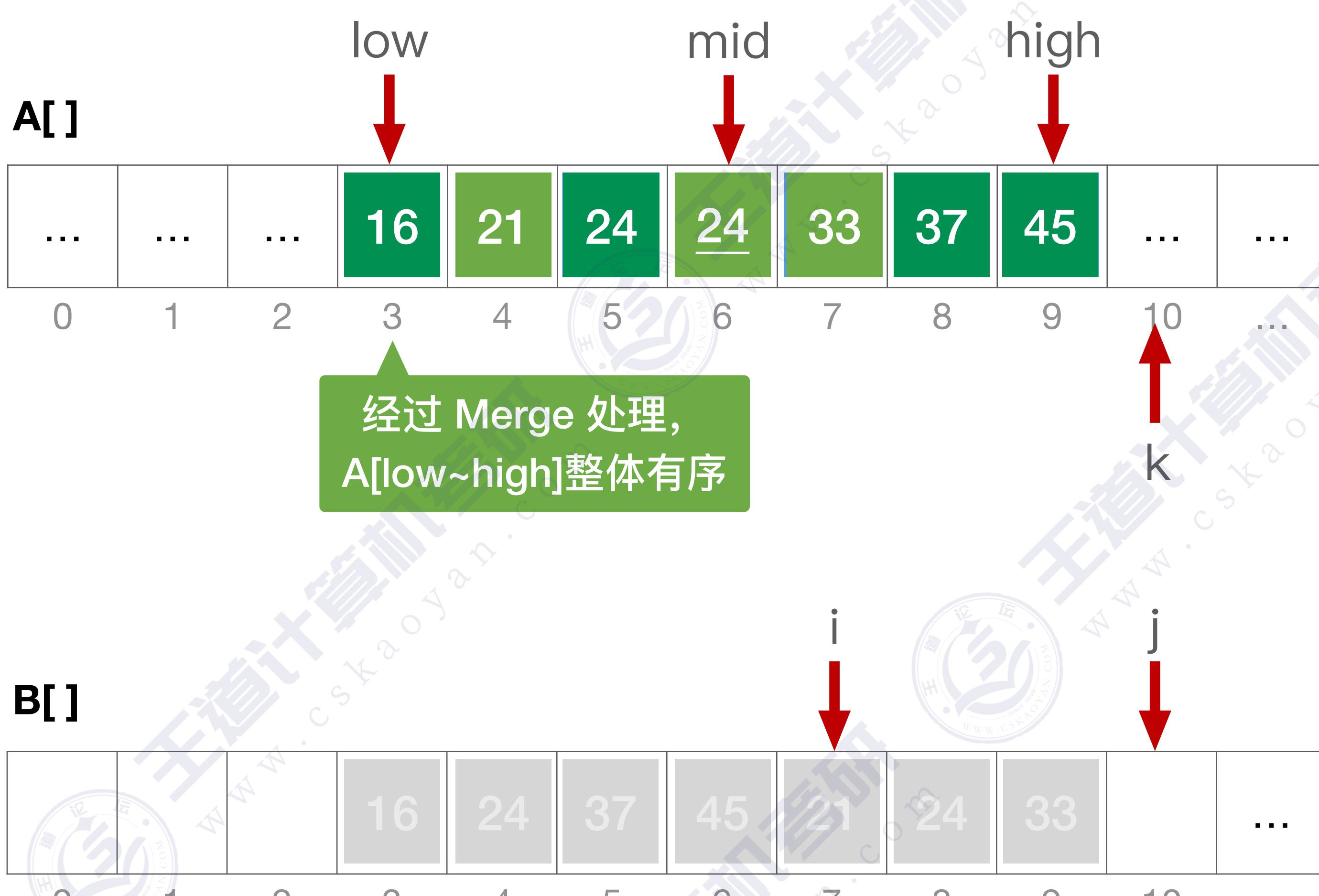
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

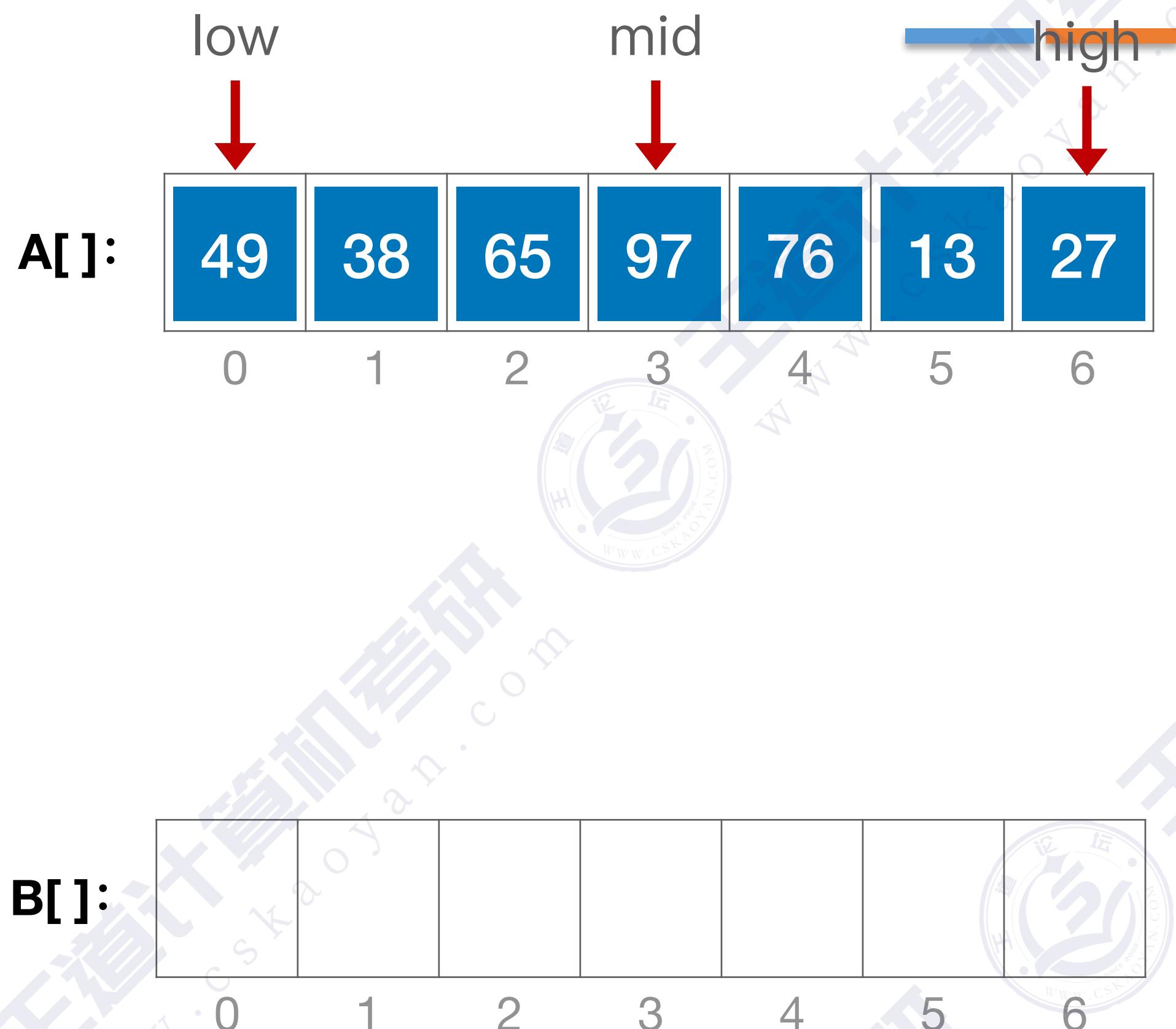
没有归并完的部分复制到尾部

代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[],int low,int mid,int high){
    int i,j,k;
    for(k=low;k<=high;k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}
```

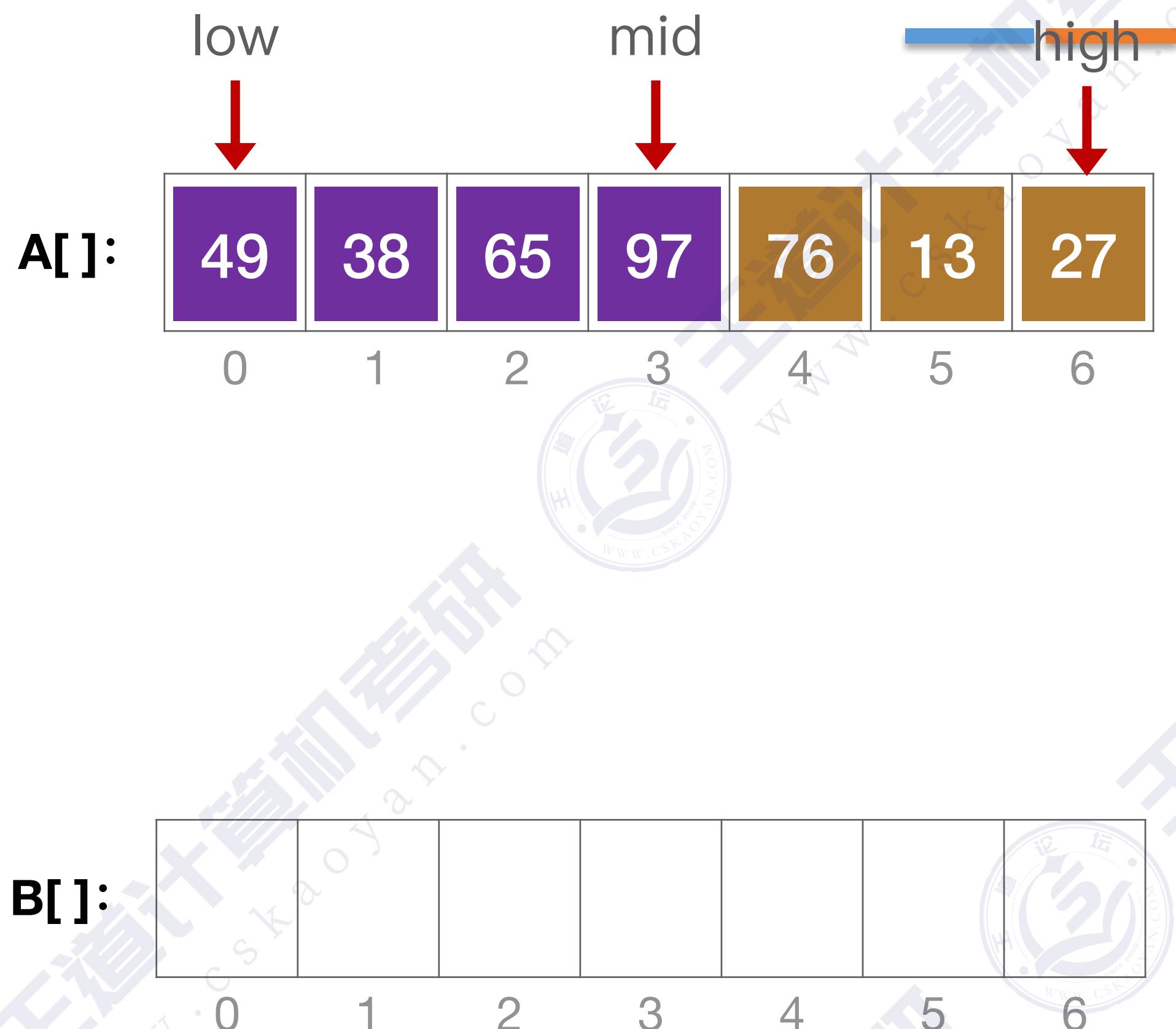
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    }//for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

void MergeSort(int A[], int low, int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A, low, mid); //对左半部分归并排序
        MergeSort(A, mid+1, high); //对右半部分归并排序
        Merge(A, low, mid, high); //归并
    }//if
}
```

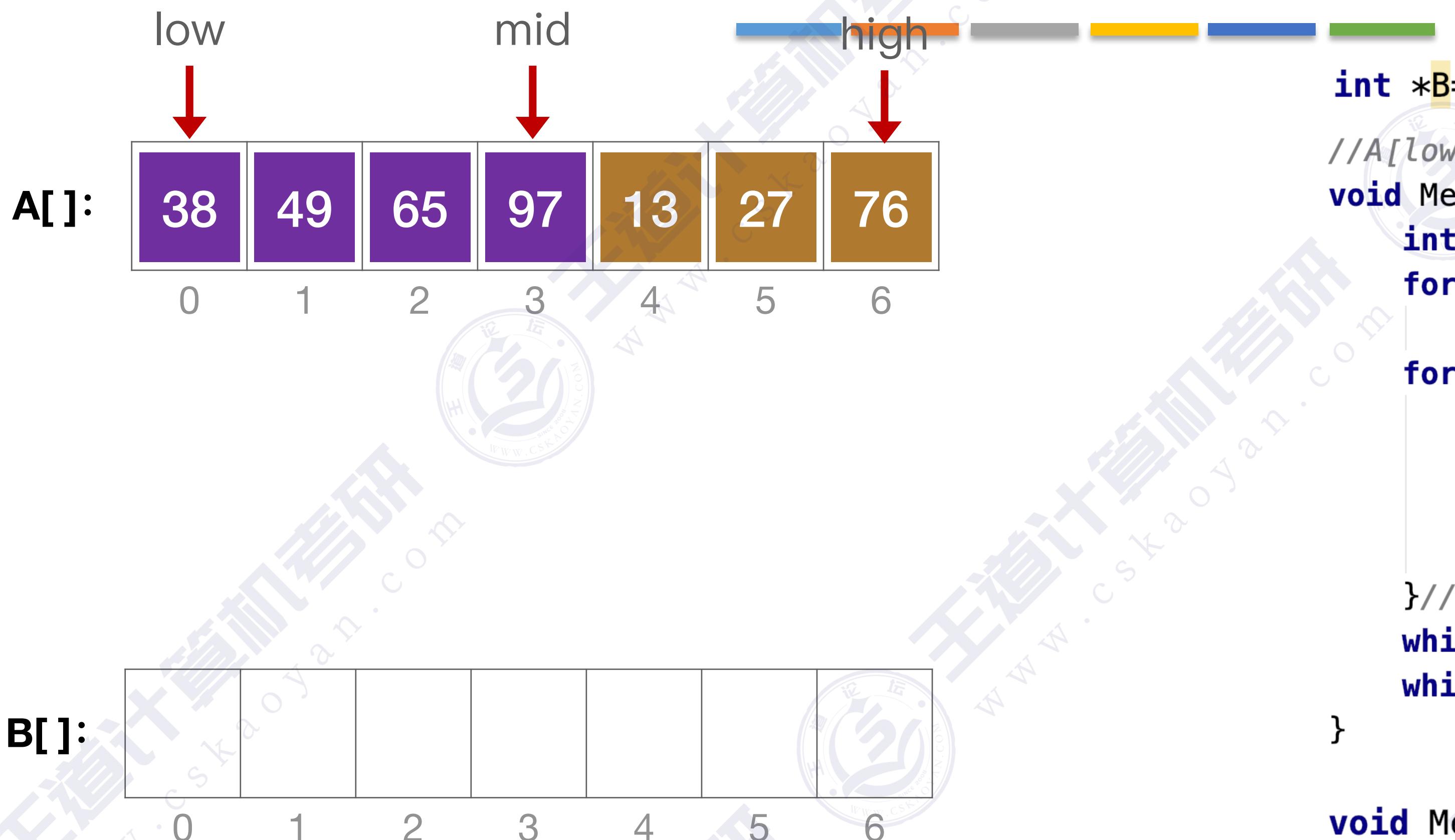
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low..mid]和A[mid+1..high]各自有序, 将两个部分归并
void Merge(int A[],int low,int mid,int high){
    int i,j,k;
    for(k=low;k<=high;k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    }//for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

void MergeSort(int A[],int low,int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A,low,mid); //对左半部分归并排序
        MergeSort(A,mid+1,high); //对右半部分归并排序
        Merge(A,low,mid,high); //归并
    }//if
}
```

代码实现

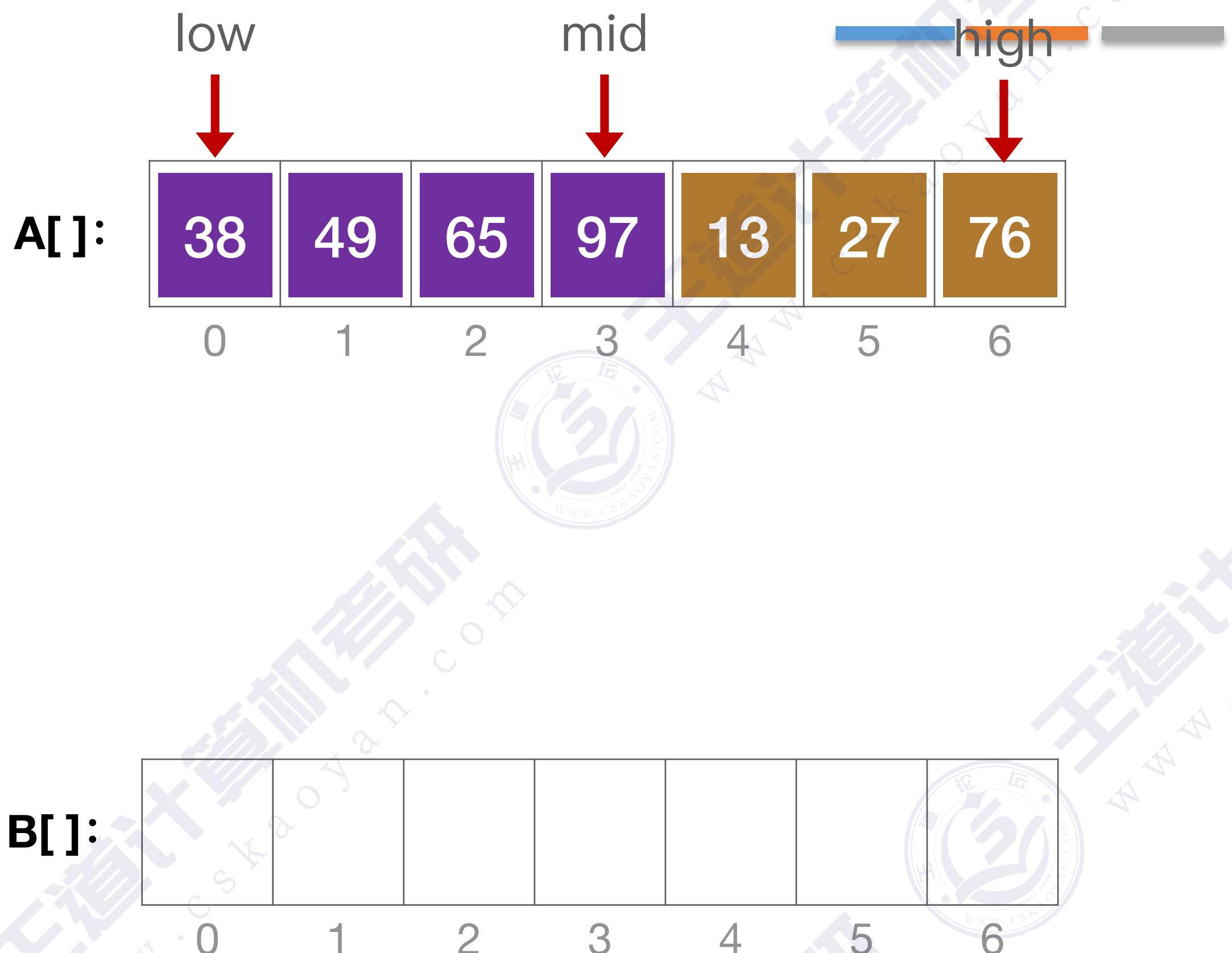


将左右两个子序列分别进行归并排序

```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[],int low,int mid,int high){
    int i,j,k;
    for(k=low;k<=high;k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    } //for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

void MergeSort(int A[],int low,int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A,low,mid); //对左半部分归并排序
        MergeSort(A,mid+1,high); //对右半部分归并排序
        Merge(A,low,mid,high); //归并
    } //if
}
```

代码实现

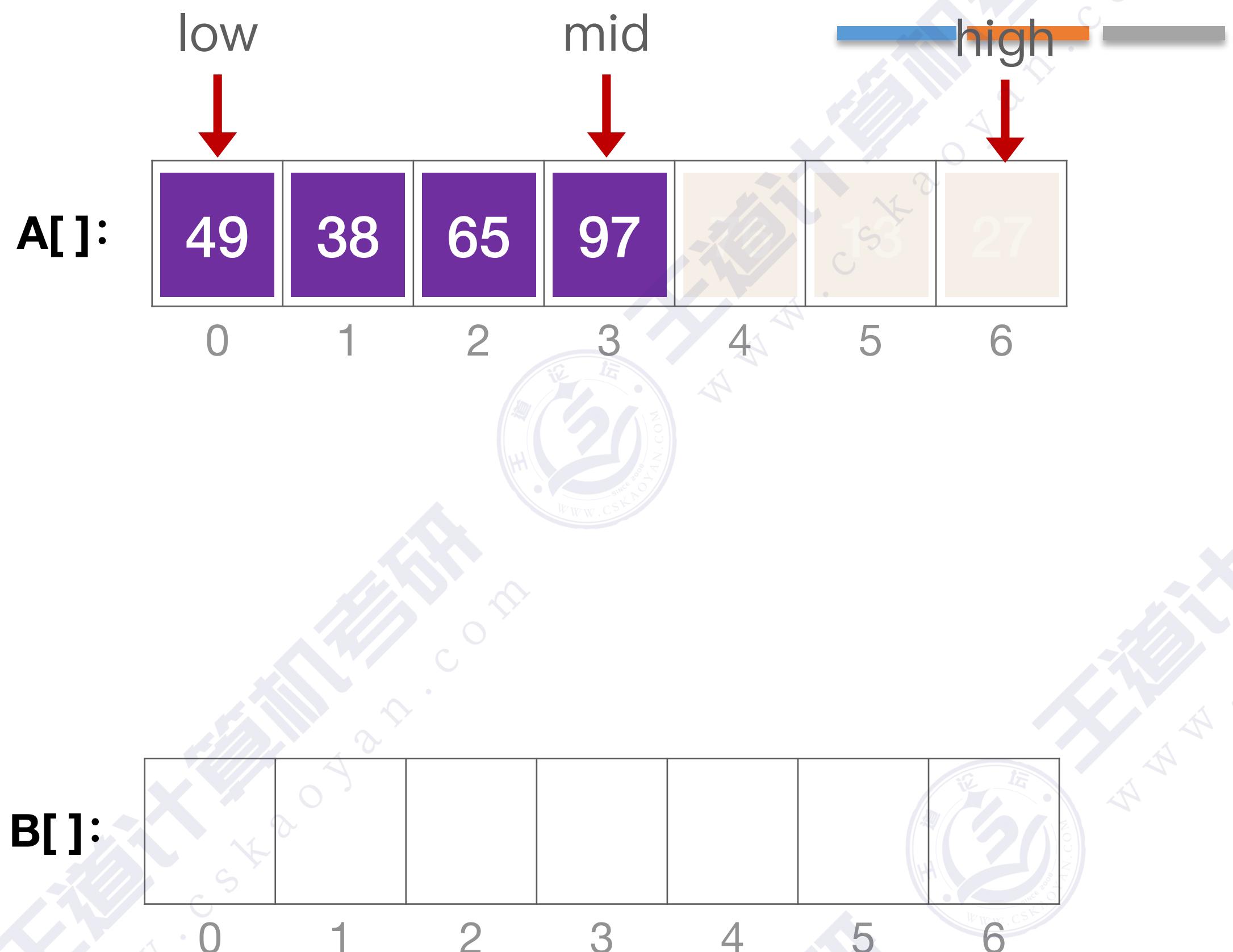


左右两个子序列分别有序之后再将二者归并

```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    }//for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

void MergeSort(int A[], int low, int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A, low, mid); //对左半部分归并排序
        MergeSort(A, mid+1, high); //对右半部分归并排序
        Merge(A, low, mid, high); //归并
    }//if
}
```

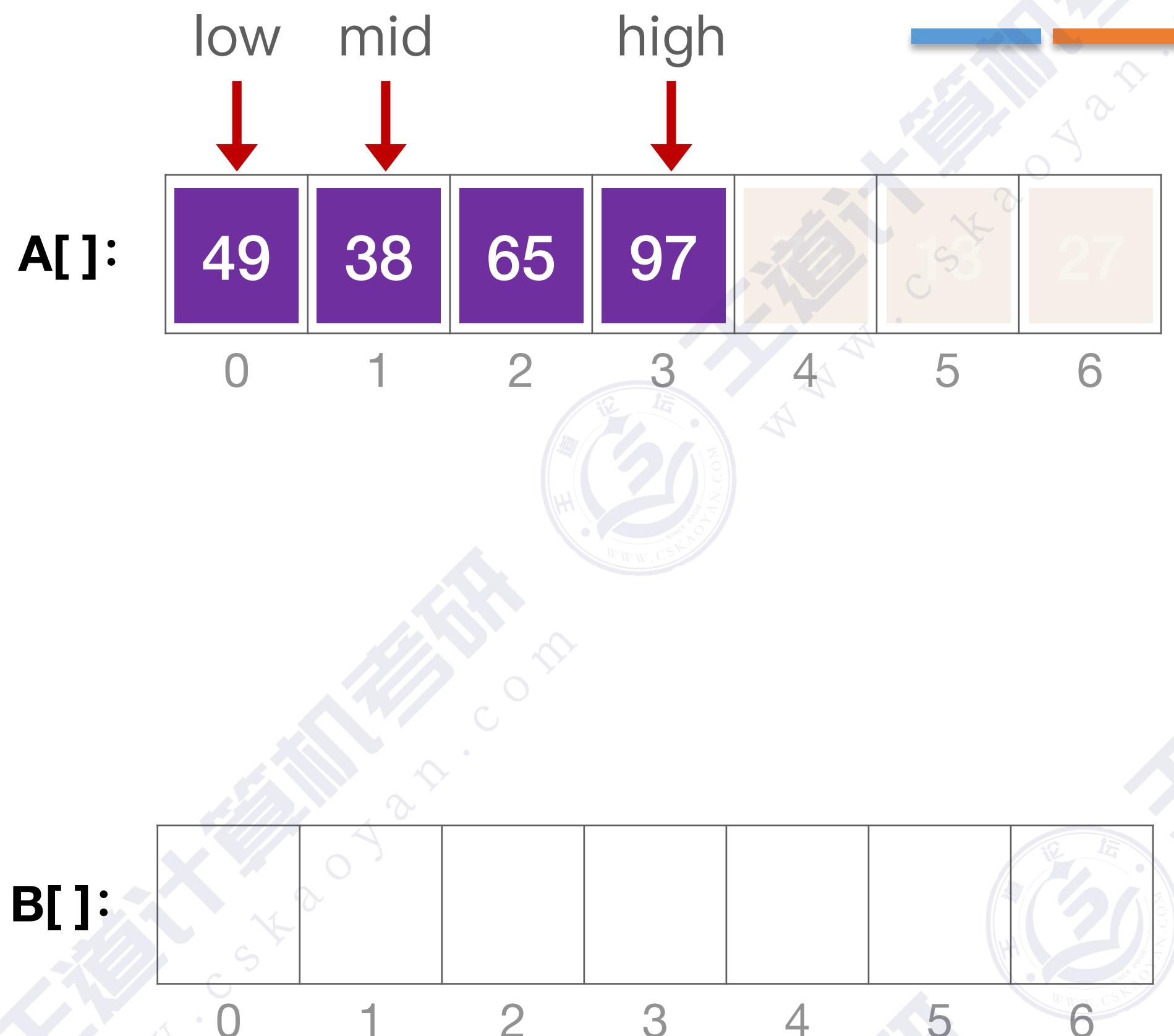
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    }//for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

void MergeSort(int A[], int low, int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A, low, mid); //对左半部分归并排序
        MergeSort(A, mid+1, high); //对右半部分归并排序
        Merge(A, low, mid, high); //归并
    }//if
}
```

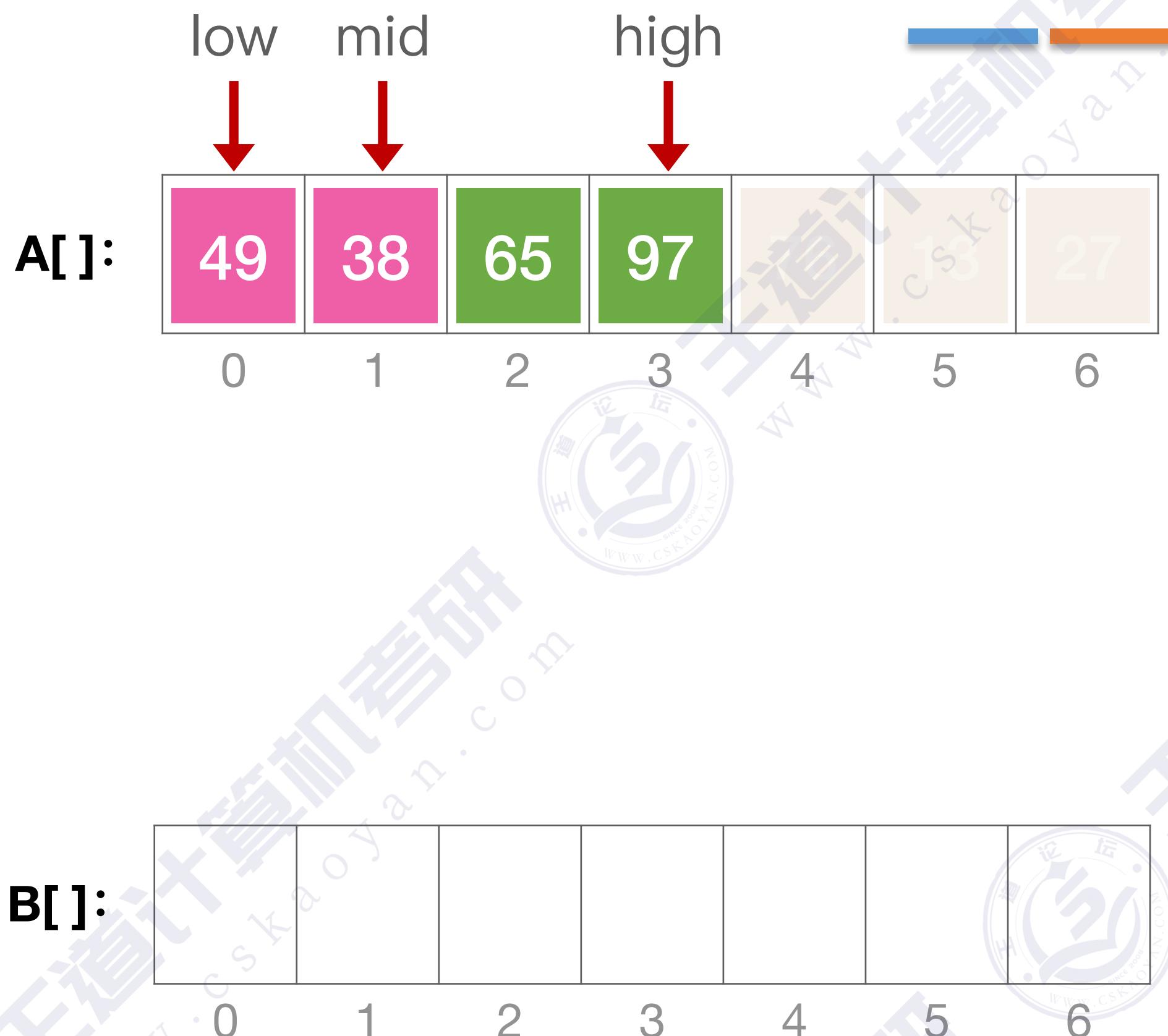
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[],int low,int mid,int high){
    int i,j,k;
    for(k=low;k<=high;k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    }//for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

void MergeSort(int A[],int low,int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A,low,mid); //对左半部分归并排序
        MergeSort(A,mid+1,high); //对右半部分归并排序
        Merge(A,low,mid,high); //归并
    }//if
}
```

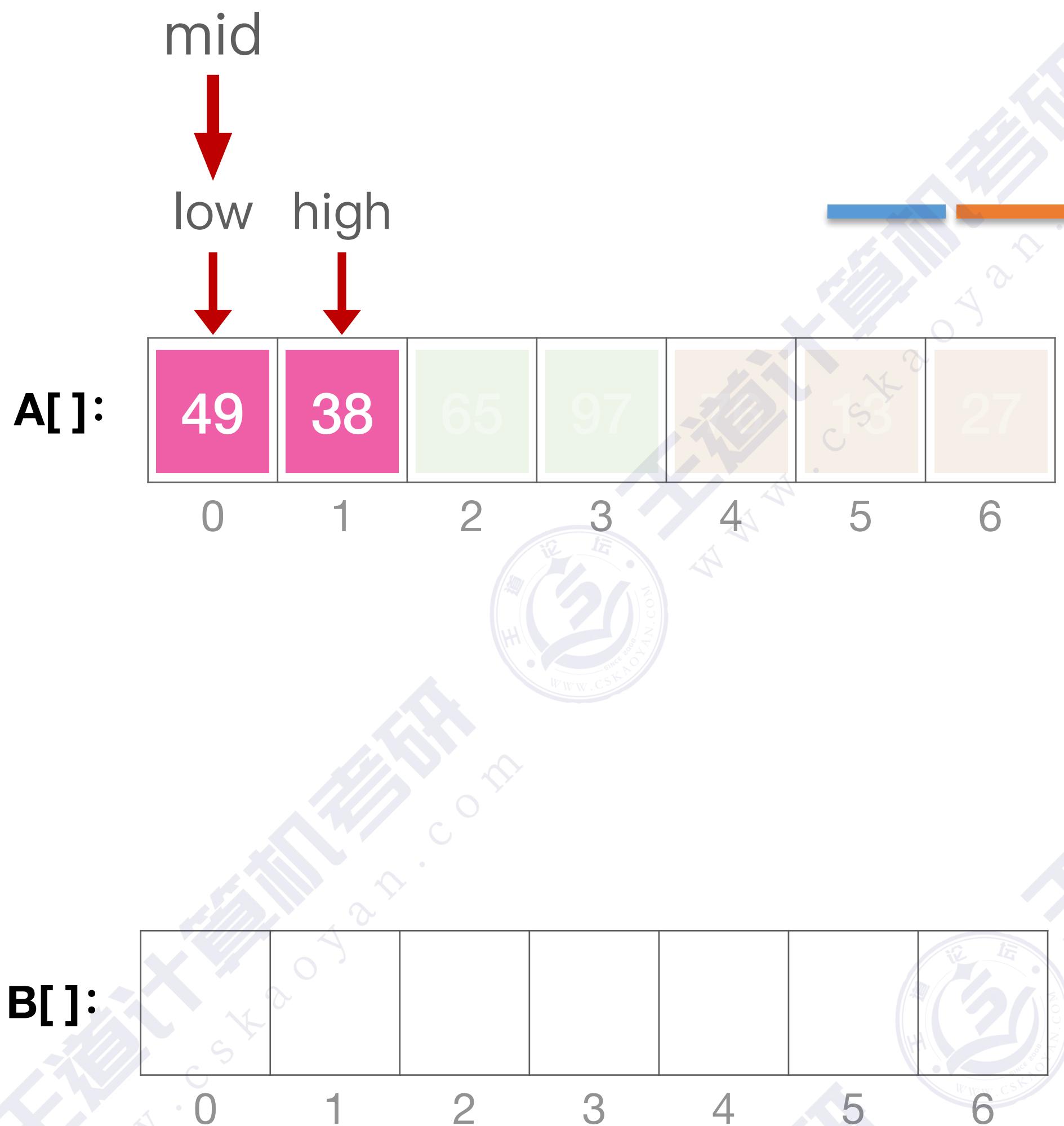
代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并
void Merge(int A[], int low, int mid, int high){
    int i, j, k;
    for(k=low; k<=high; k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    }//for
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

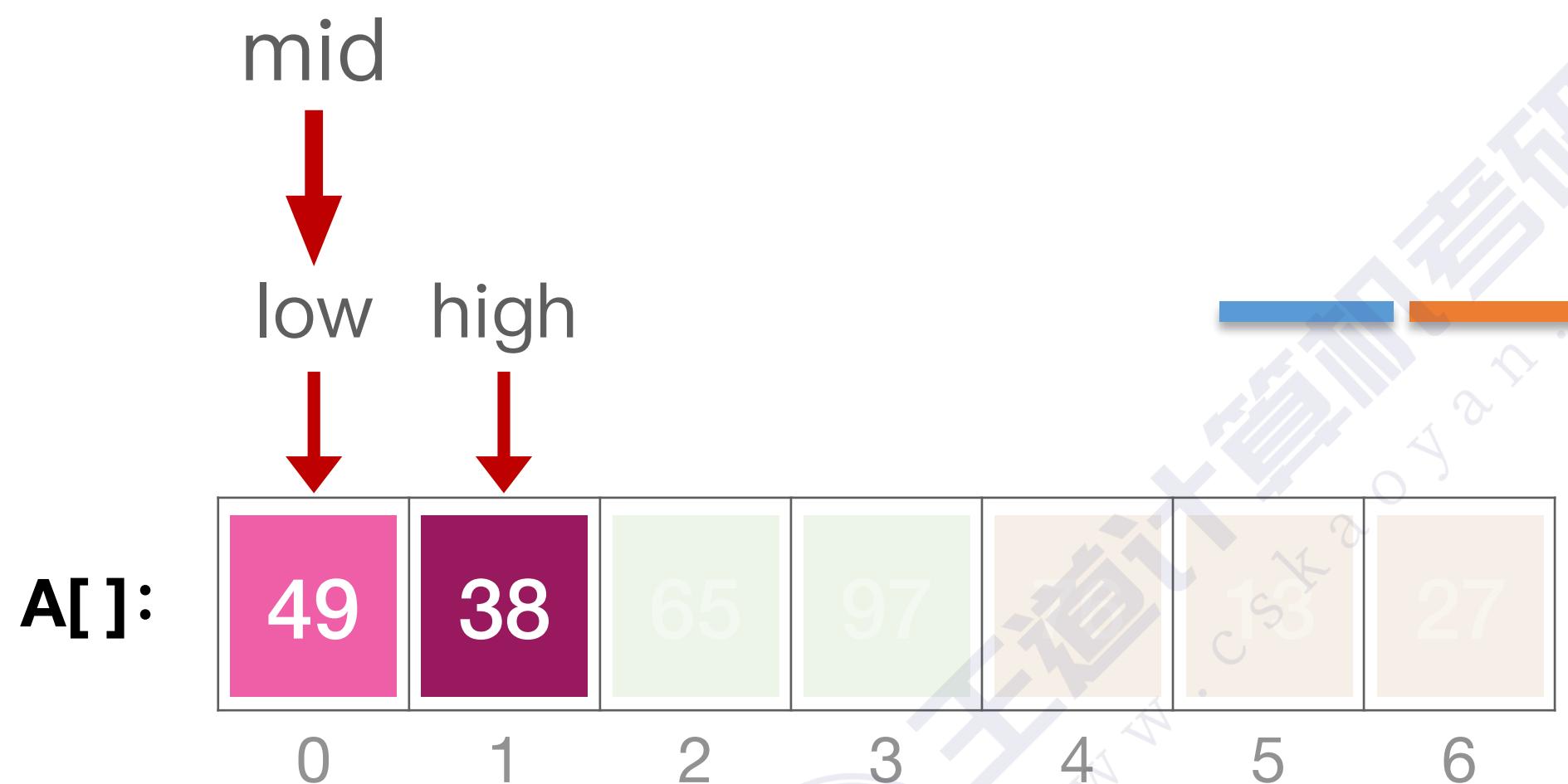
void MergeSort(int A[], int low, int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A, low, mid); //对左半部分归并排序
        MergeSort(A, mid+1, high); //对右半部分归并排序
        Merge(A, low, mid, high); //归并
    }//if
}
```

代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B  
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并  
void Merge(int A[], int low, int mid, int high){  
    int i, j, k;  
    for(k=low; k<=high; k++)  
        B[k]=A[k]; //将A中所有元素复制到B中  
    for(i=low, j=mid+1, k=i; i<=mid&&j<=high; k++){  
        if(B[i]<=B[j])  
            A[k]=B[i++]; //将较小值复制到A中  
        else  
            A[k]=B[j++];  
    } //for  
    while(i<=mid) A[k++]=B[i++];  
    while(j<=high) A[k++]=B[j++];  
}  
  
void MergeSort(int A[], int low, int high){  
    if(low<high){  
        int mid=(low+high)/2; //从中间划分  
        MergeSort(A, low, mid); //对左半部分归并排序  
        MergeSort(A, mid+1, high); //对右半部分归并排序  
        Merge(A, low, mid, high); //归并  
    } //if  
}
```

代码实现



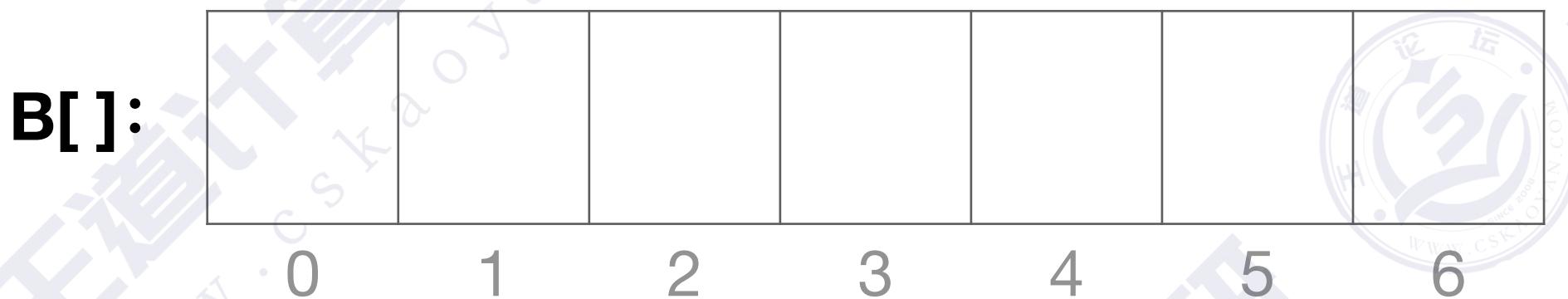
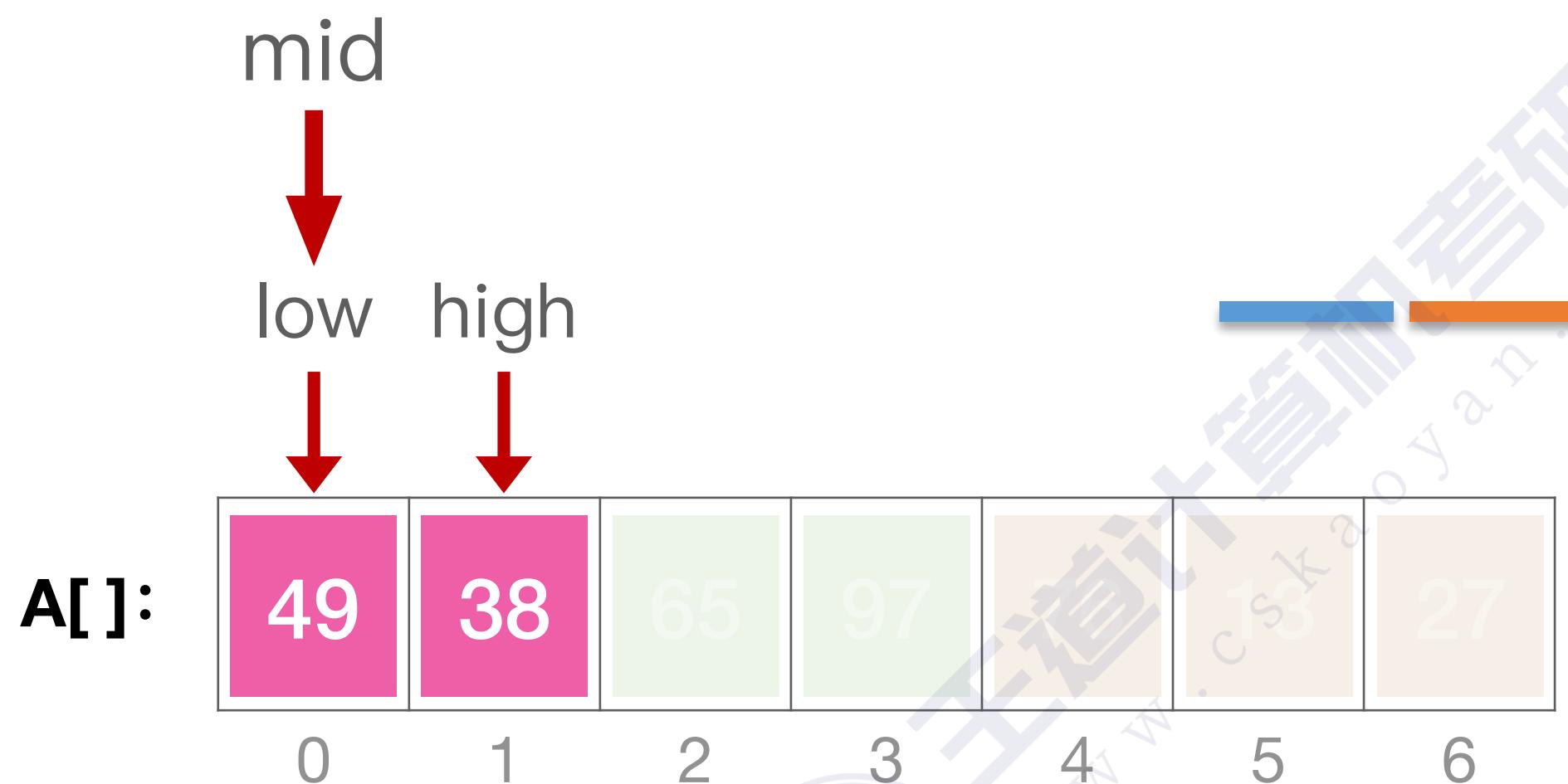
将左右两个子序列分别进行归并排序 (每个子序列只含有1个元素)



禁止套娃

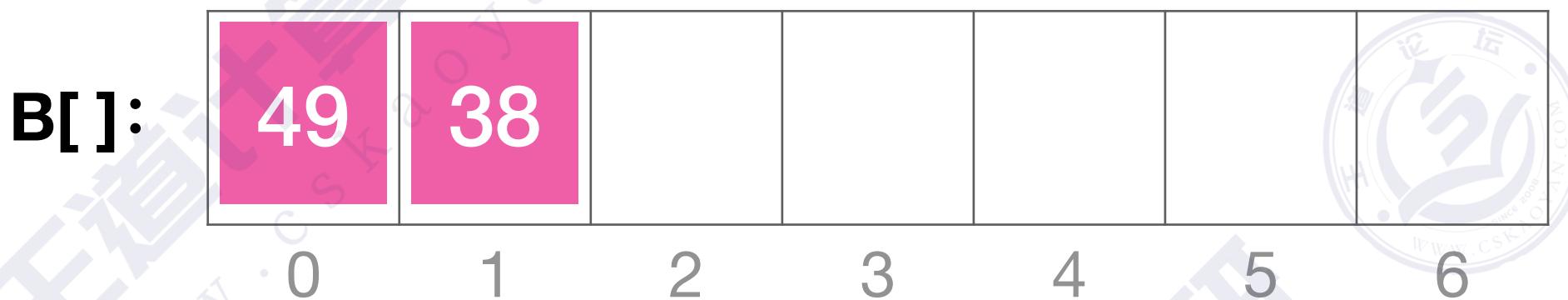
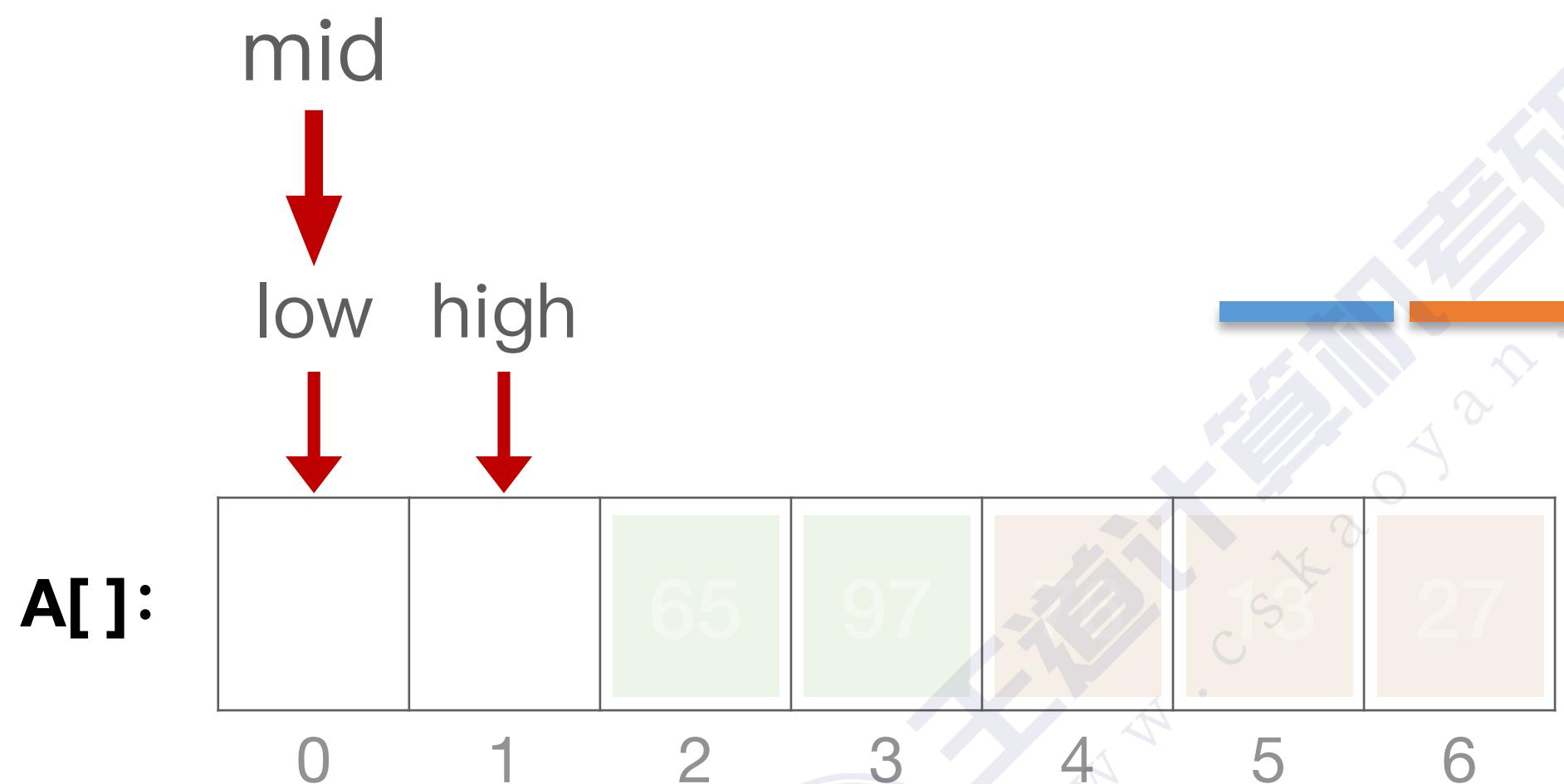
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B  
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并  
void Merge(int A[],int low,int mid,int high){  
    int i,j,k;  
    for(k=low;k<=high;k++)  
        B[k]=A[k]; //将A中所有元素复制到B中  
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){  
        if(B[i]<=B[j])  
            A[k]=B[i++]; //将较小值复制到A中  
        else  
            A[k]=B[j++];  
    } //for  
    while(i<=mid) A[k++]=B[i++];  
    while(j<=high) A[k++]=B[j++];  
}  
  
void MergeSort(int A[],int low,int high){  
    if(low<high){  
        int mid=(low+high)/2; //从中间划分  
        MergeSort(A,low,mid); //对左半部分归并排序  
        MergeSort(A,mid+1,high); //对右半部分归并排序  
        Merge(A,low,mid,high); //归并  
    } //if  
}
```

代码实现



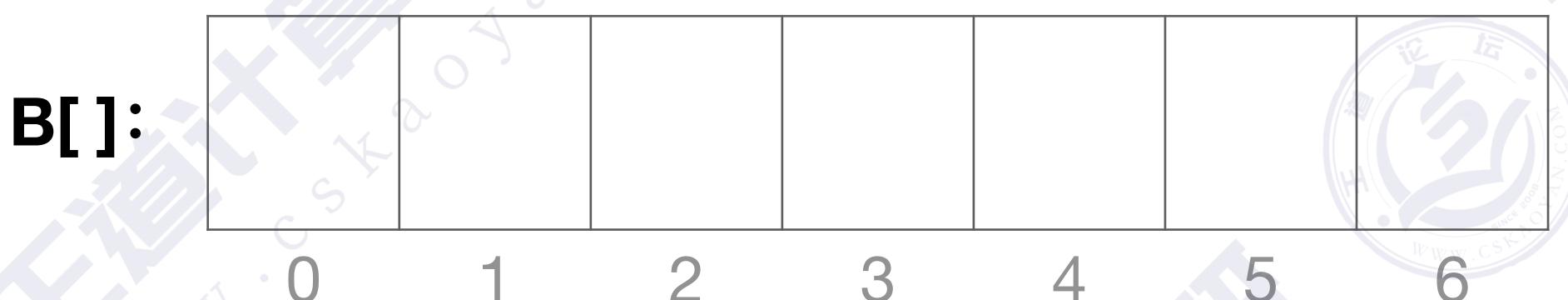
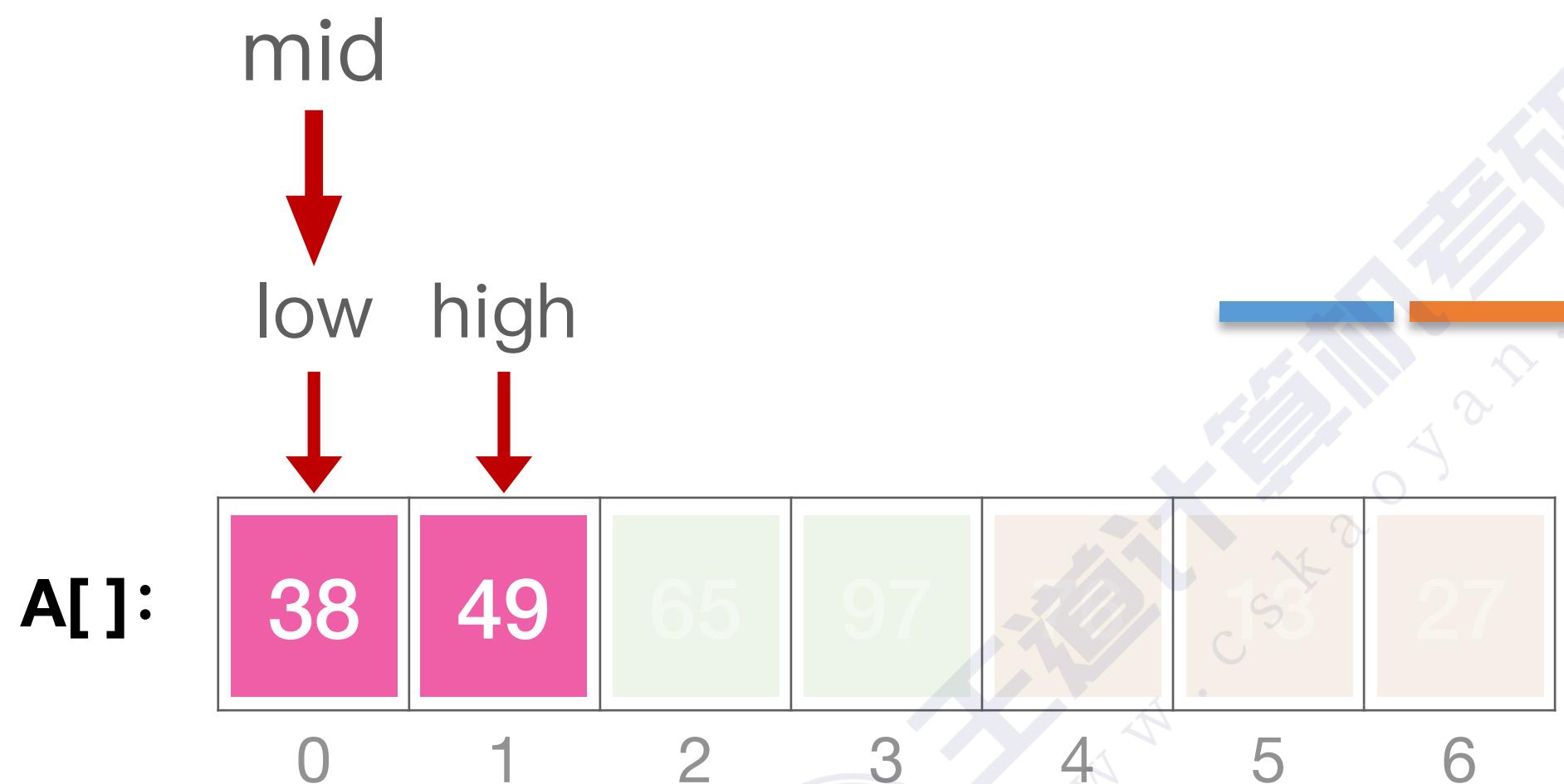
```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B  
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并  
void Merge(int A[],int low,int mid,int high){  
    int i,j,k;  
    for(k=low;k<=high;k++)  
        B[k]=A[k]; //将A中所有元素复制到B中  
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){  
        if(B[i]<=B[j])  
            A[k]=B[i++]; //将较小值复制到A中  
        else  
            A[k]=B[j++];  
    } //for  
    while(i<=mid) A[k++]=B[i++];  
    while(j<=high) A[k++]=B[j++];  
}  
  
void MergeSort(int A[],int low,int high){  
    if(low<high){  
        int mid=(low+high)/2; //从中间划分  
        MergeSort(A,low,mid); //对左半部分归并排序  
        MergeSort(A,mid+1,high); //对右半部分归并排序  
        Merge(A,low,mid,high); //归并  
    } //if  
}
```

代码实现



```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B  
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并  
void Merge(int A[],int low,int mid,int high){  
    int i,j,k;  
    for(k=low;k<=high;k++)  
        B[k]=A[k]; //将A中所有元素复制到B中  
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){  
        if(B[i]<=B[j])  
            A[k]=B[i++]; //将较小值复制到A中  
        else  
            A[k]=B[j++];  
    } //for  
    while(i<=mid) A[k++]=B[i++];  
    while(j<=high) A[k++]=B[j++];  
}  
  
void MergeSort(int A[],int low,int high){  
    if(low<high){  
        int mid=(low+high)/2; //从中间划分  
        MergeSort(A,low,mid); //对左半部分归并排序  
        MergeSort(A,mid+1,high); //对右半部分归并排序  
        Merge(A,low,mid,high); //归并  
    } //if  
}
```

代码实现

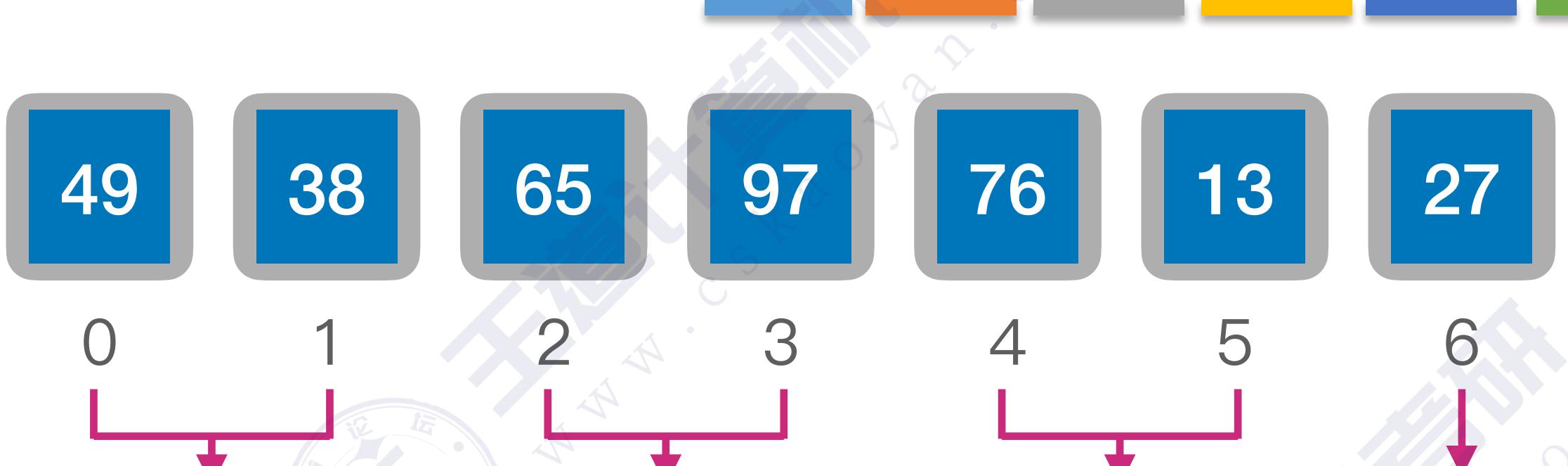


```
int *B=(int *)malloc(n*sizeof(int)); //辅助数组B  
//A[low...mid]和A[mid+1...high]各自有序, 将两个部分归并  
void Merge(int A[],int low,int mid,int high){  
    int i,j,k;  
    for(k=low;k<=high;k++)  
        B[k]=A[k]; //将A中所有元素复制到B中  
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){  
        if(B[i]<=B[j])  
            A[k]=B[i++]; //将较小值复制到A中  
        else  
            A[k]=B[j++];  
    } //for  
    while(i<=mid) A[k++]=B[i++];  
    while(j<=high) A[k++]=B[j++];  
}  
  
void MergeSort(int A[],int low,int high){  
    if(low<high){  
        int mid=(low+high)/2; //从中间划分  
        MergeSort(A,low,mid); //对左半部分归并排序  
        MergeSort(A,mid+1,high); //对右半部分归并排序  
        Merge(A,low,mid,high); //归并  
    } //if  
}
```

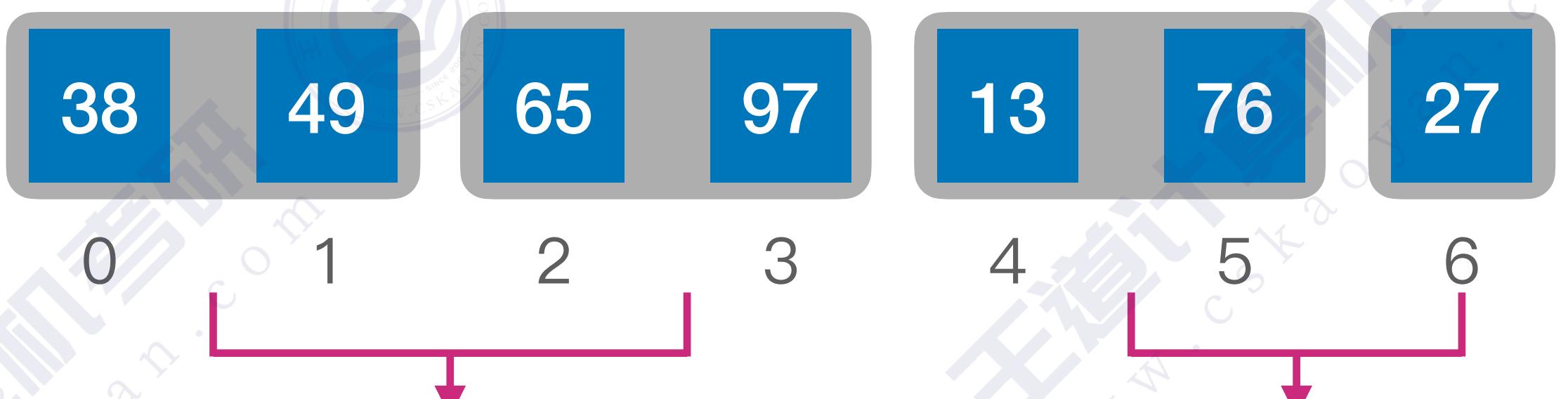
算法效率分析

2路归并的“归并树”——形态上就是一棵倒立的二叉树

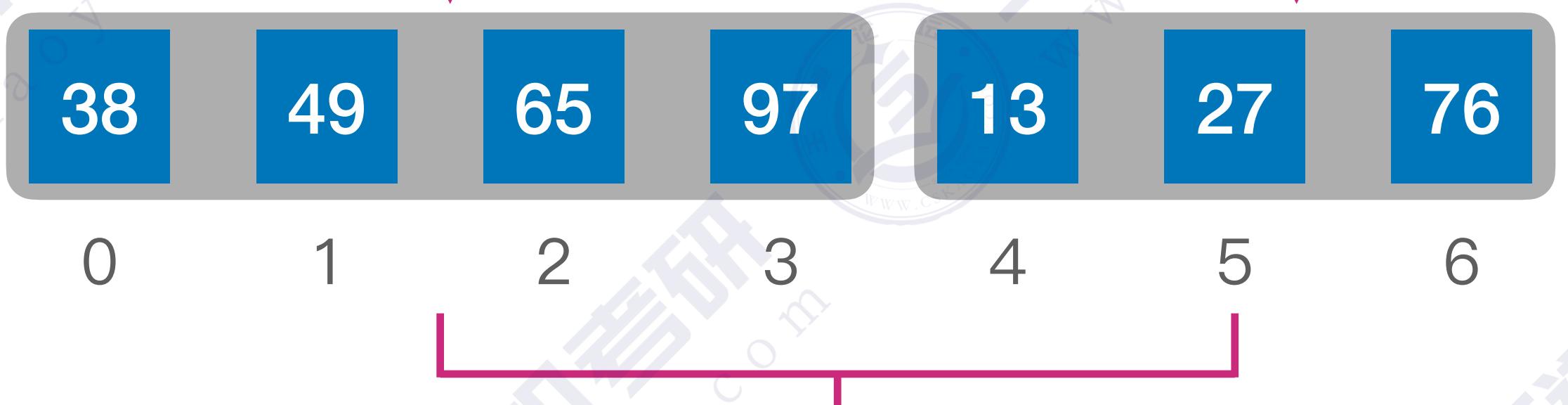
初始序列：



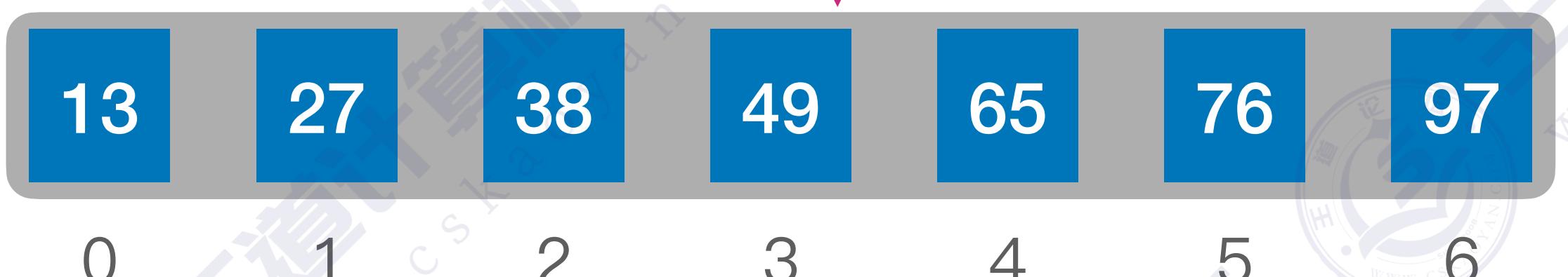
一趟归并后：



二趟归并后：



三趟归并后：



看左边

二叉树的第 h 层最多有 2^{h-1} 个结点
若树高为 h , 则应满足 $n \leq 2^{h-1}$
即 $h - 1 = \lceil \log_2 n \rceil$

结论: n 个元素进行2路归并排序, 归并趟数 $= \lceil \log_2 n \rceil$

每趟归并时间复杂度为 $O(n)$, 则算法时间复杂度为 $O(n \log_2 n)$

空间复杂度 $= O(n)$, 来自于辅助数组B

知识回顾与重要考点

