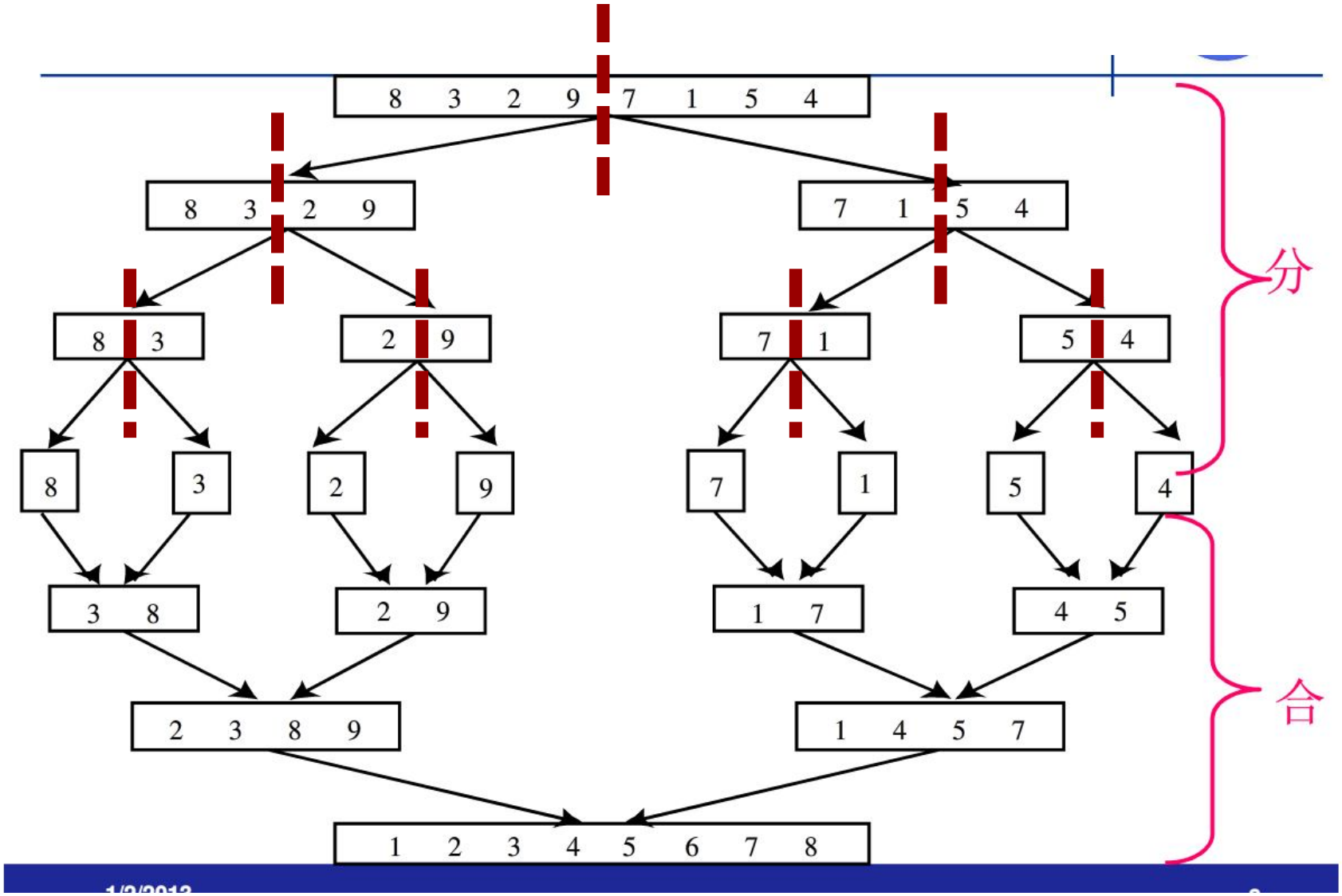
4.5-4.6归并排序

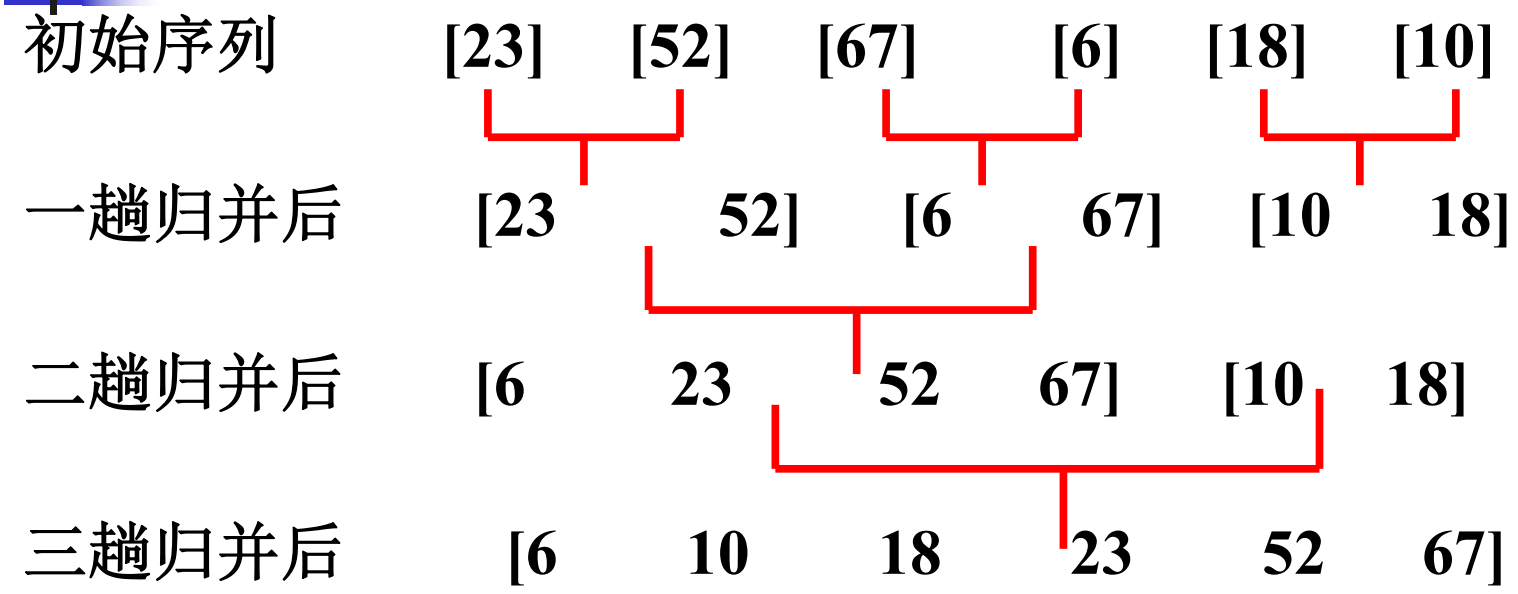
- 基本思想----分治：

将待排序序列**划分**成若干有序子序列；将两个或两个以上的有序子序列**合并**为一个有序序列。


8 3 2 9 7 1 5 4
8 3 2 9
7 1 5 4
8 3
2 9
7 1
5 4
8
3
2
9
7
1
5
4
3 8
2 9
1 7
4 5
2 3 8 9
1 4 5 7
1 2 3 4 5 6 7 8
分
合

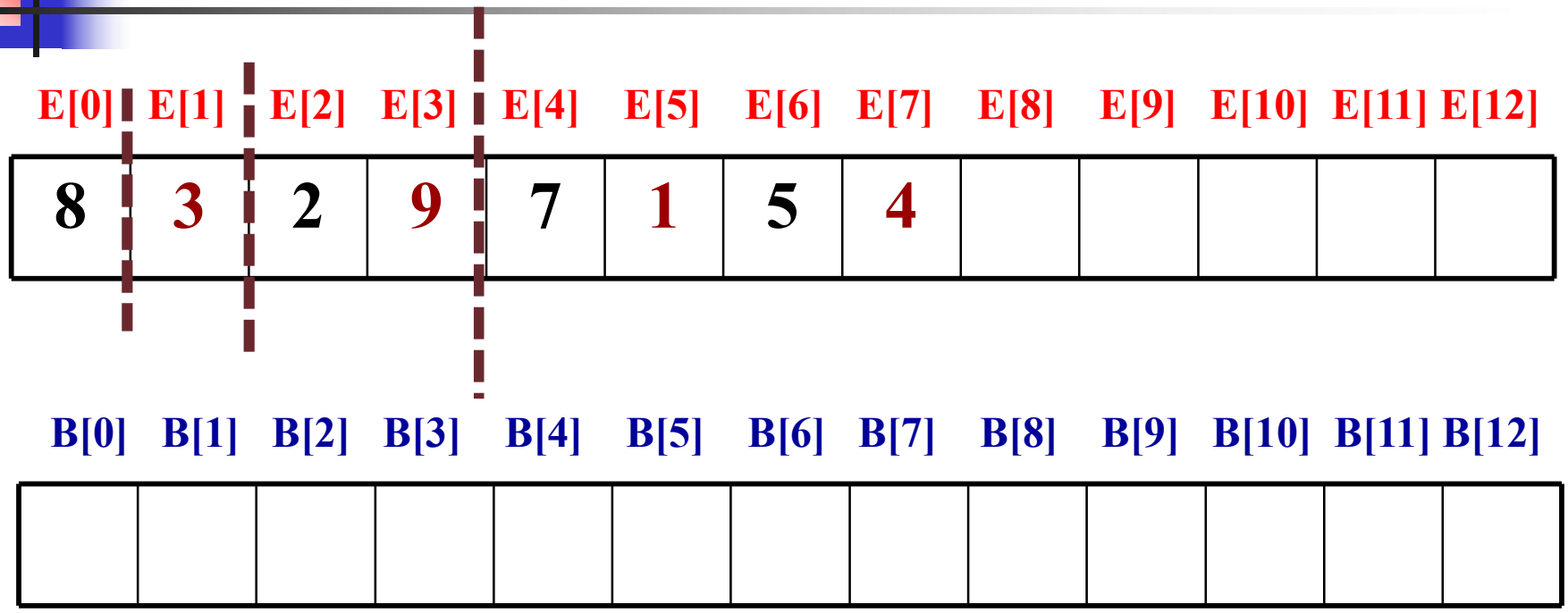
归并排序

或：将n个待排序的数据直接划分成n个规模为1的子序列



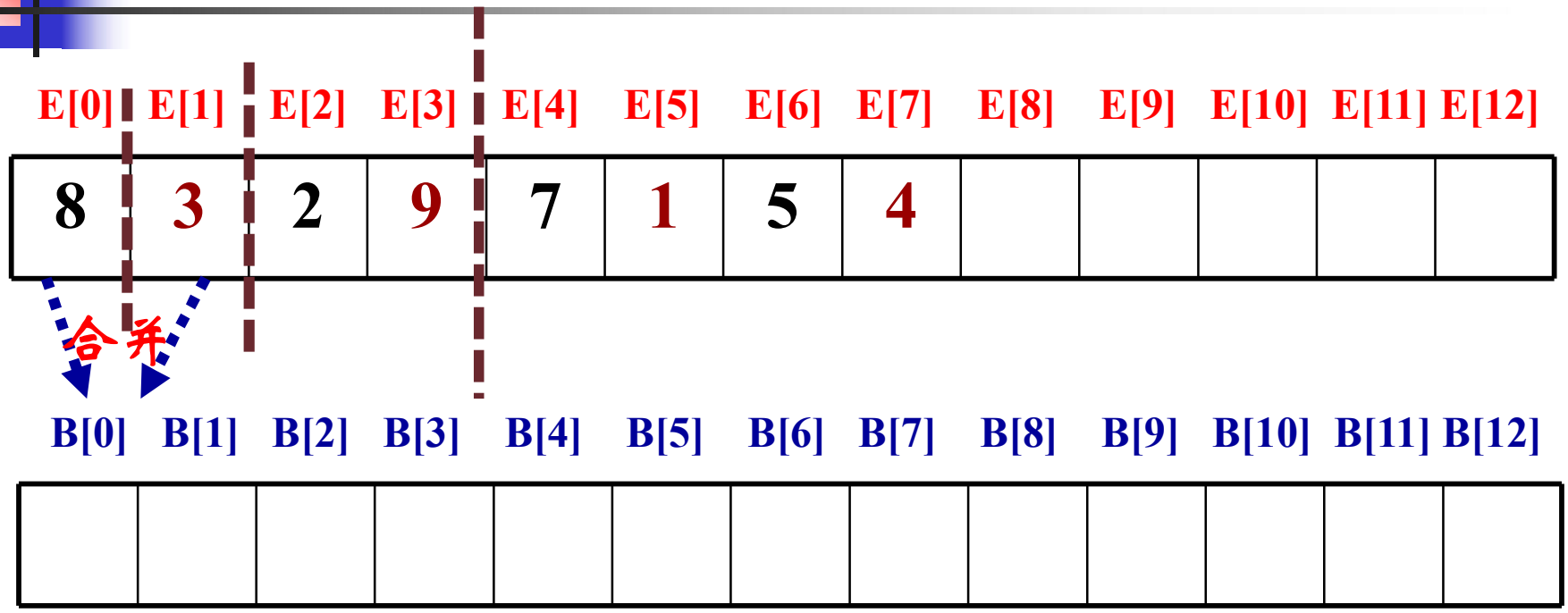
归并排序

二路归并排序



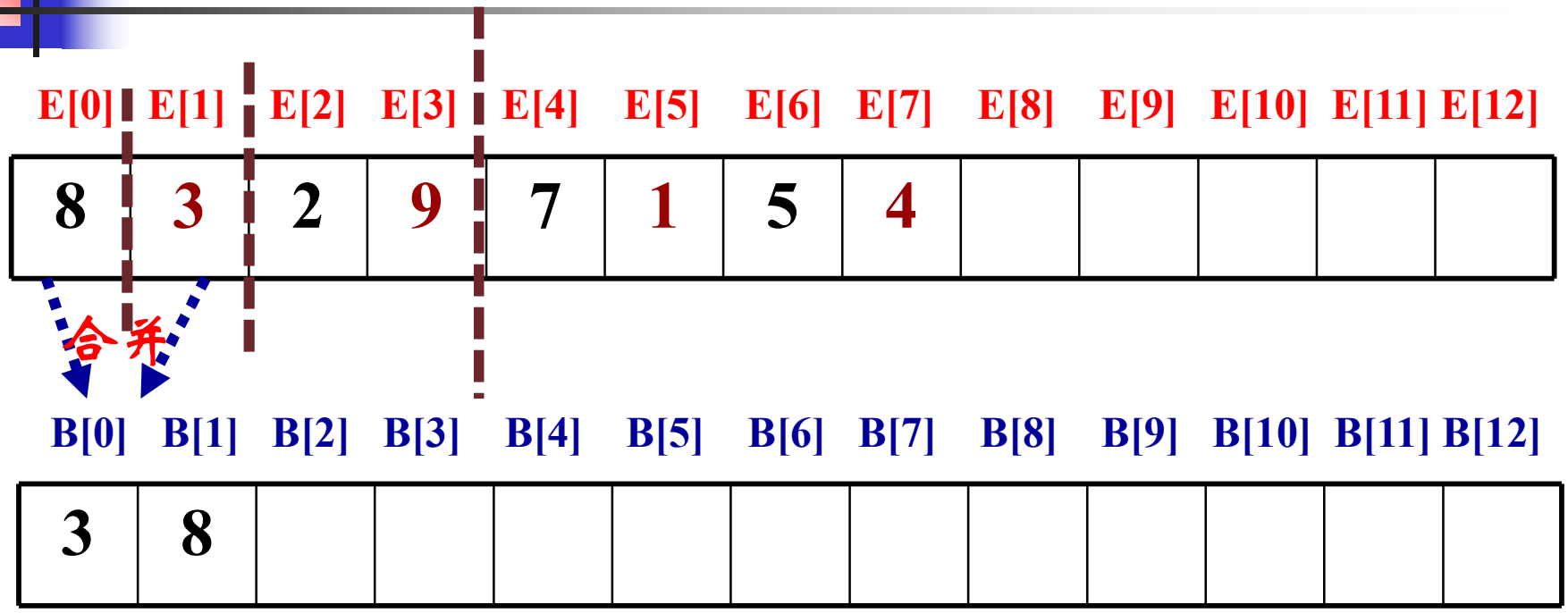
借助额外的辅助数组实现，空间复杂度O(n)

二路归并排序



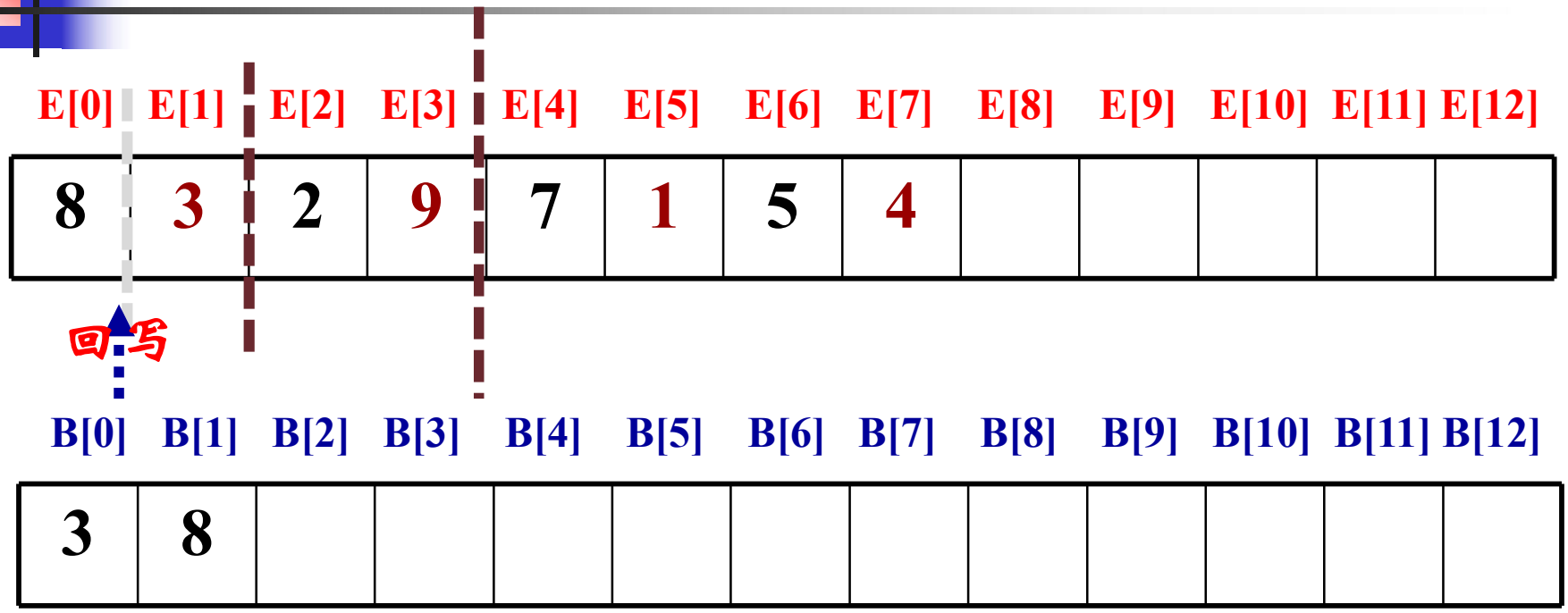
借助额外的辅助数组实现，空间复杂度O(n)

二路归并排序



借助额外的辅助数组实现，空间复杂度O(n)

二路归并排序



借助额外的辅助数组实现，空间复杂度O(n)

```
void Merge (Element E[ ] ,  int s, int m, int t)
{// 将有序的记录序列 E[s..m] 和 E[m+1..t]合并
   Element B[ ];
   for (k=s,p=s, j=m+1;  p<=m && j<=t;  ++k)
   {
       if (E[p].key<=E[j].key)  B [k] = E[p++];
        else B[k] = E[j++];
   }
   if (p<=m) B[k..t] = E[p..m];
   if (j<=t) B[k..t] = E[j..t];
  for (p=s;p<=t;p++) E[p] = B[p];
} // Merge
```

将有序的记录序列E[s..m]和E[m+1..t]合并到B[s..t]

将有序序列 B[s..t] 回写到E[s..t]

主要操作：子问题的解合并得到原问题的解
将2个有序序列 合并为一个有序序列

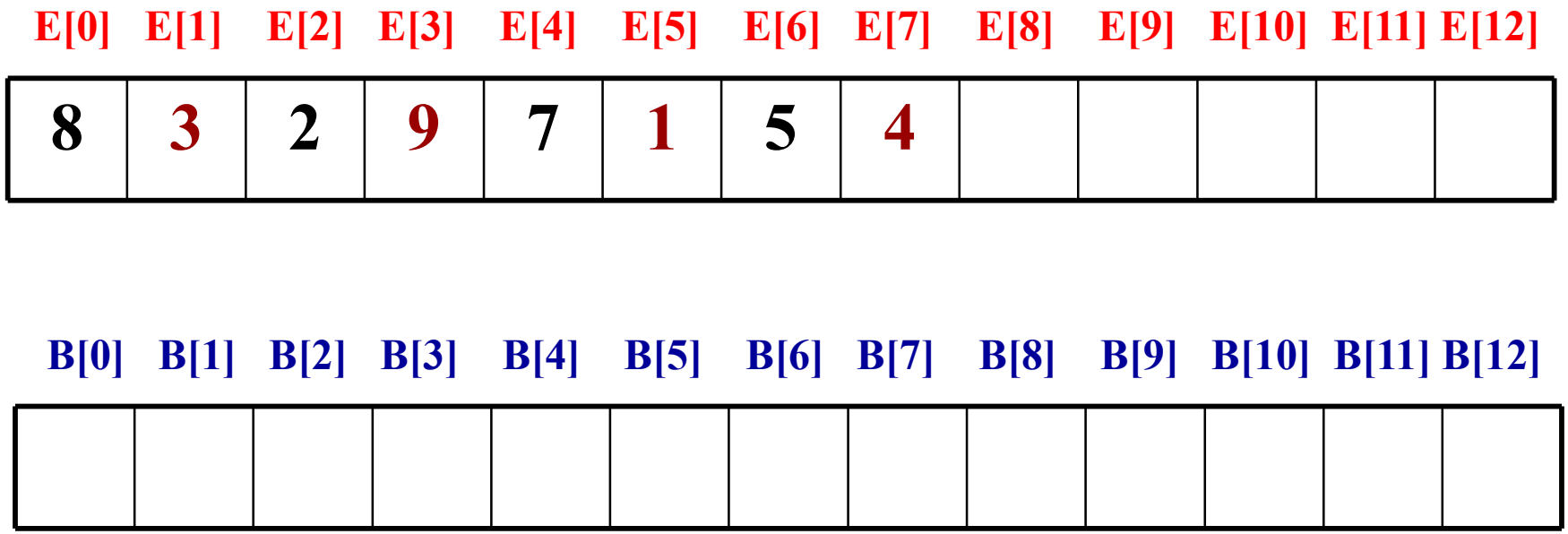
```
void Msort (Element E[], int s, int t )
{
  if (s<t)
  { m = (s+t)/2;
    Msort (E, s, m);
    Msort (E, m+1, t);
    Merge (E, s, m, t);
  }
} // Msort
```

```
Solve(I)
{   n=size(I);
   if (n<=SmallSize)
    Solution=directlySolve(I);
   else  Divide I into I1,I2,…,Ik;
         for each i ∈{1,2,…,k} Si=solve(Ii);
         Solution=  Combine(S1, S2,…, Sk);
   return Solution;}
```

归并排序

有序子序列个数=8

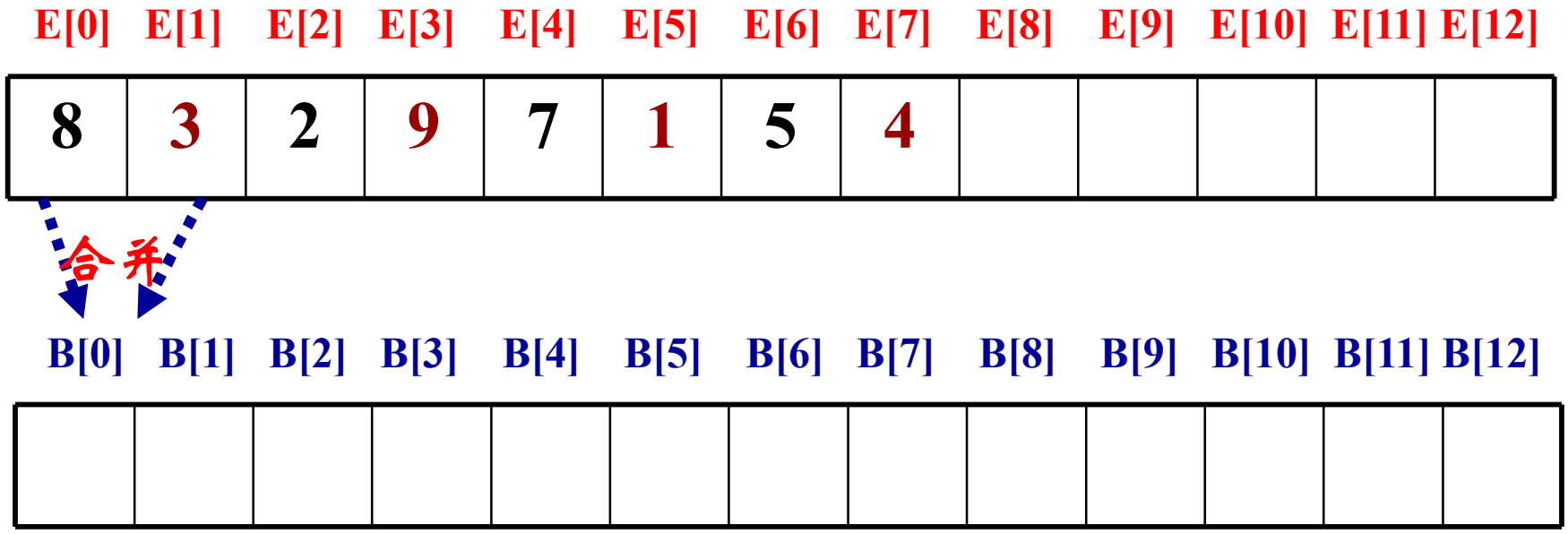
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

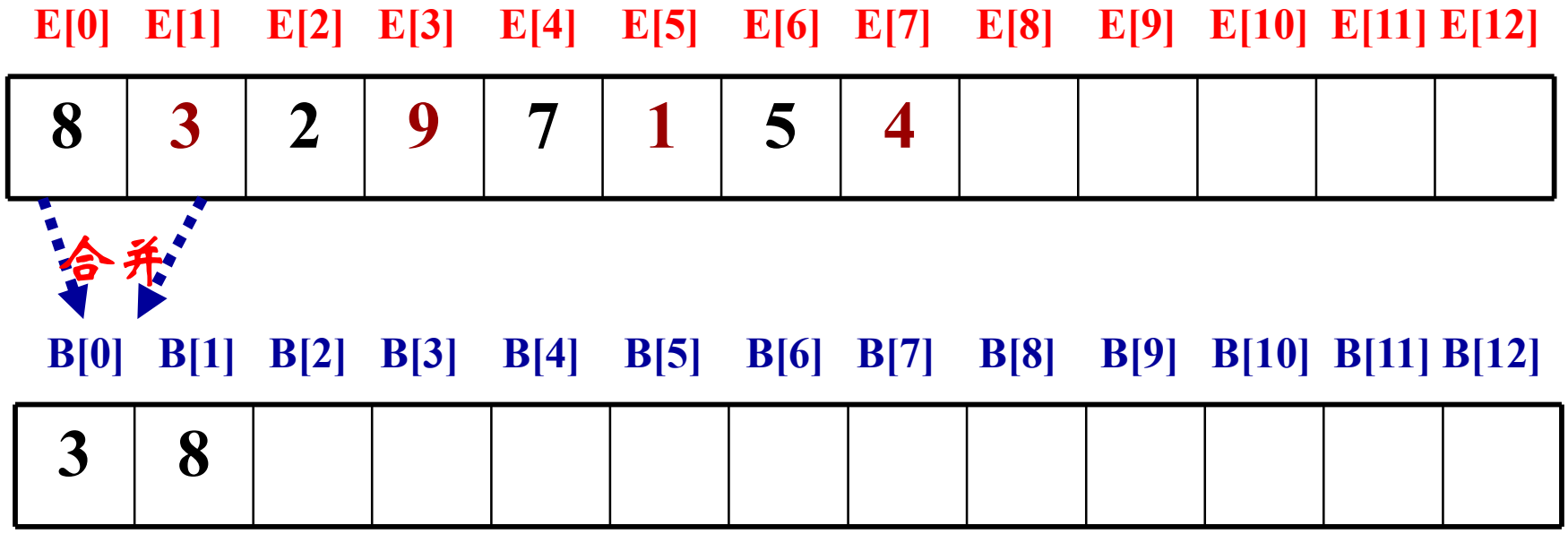
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

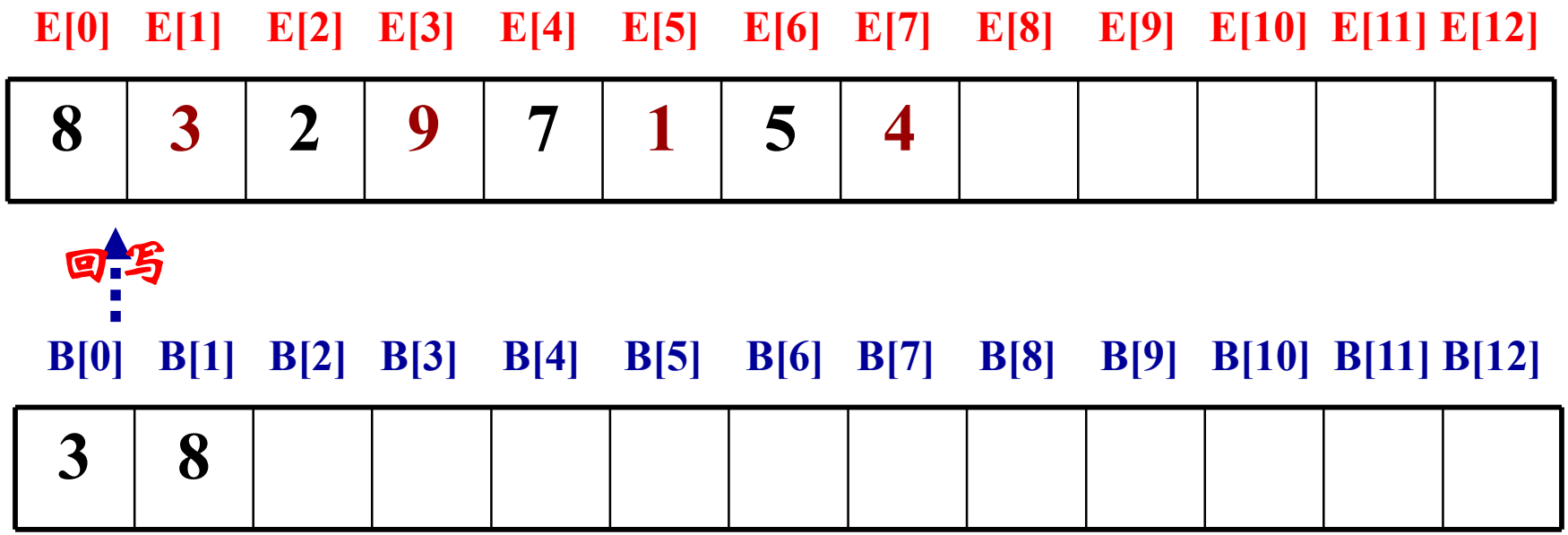
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

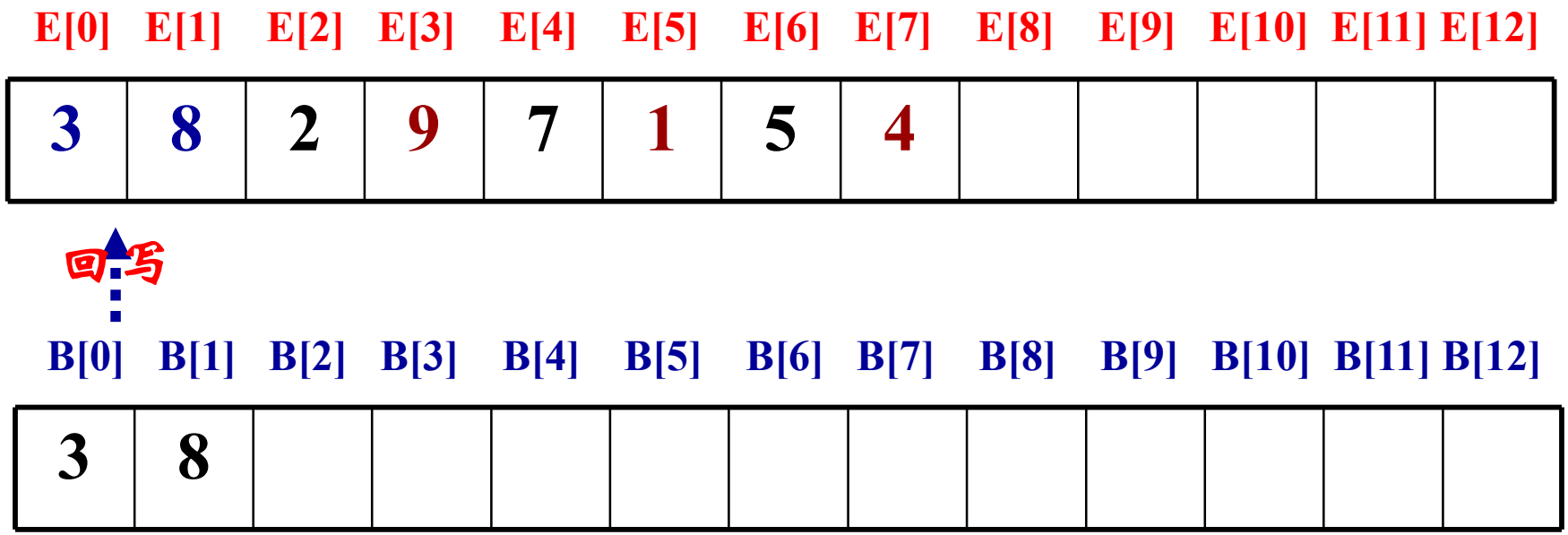
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

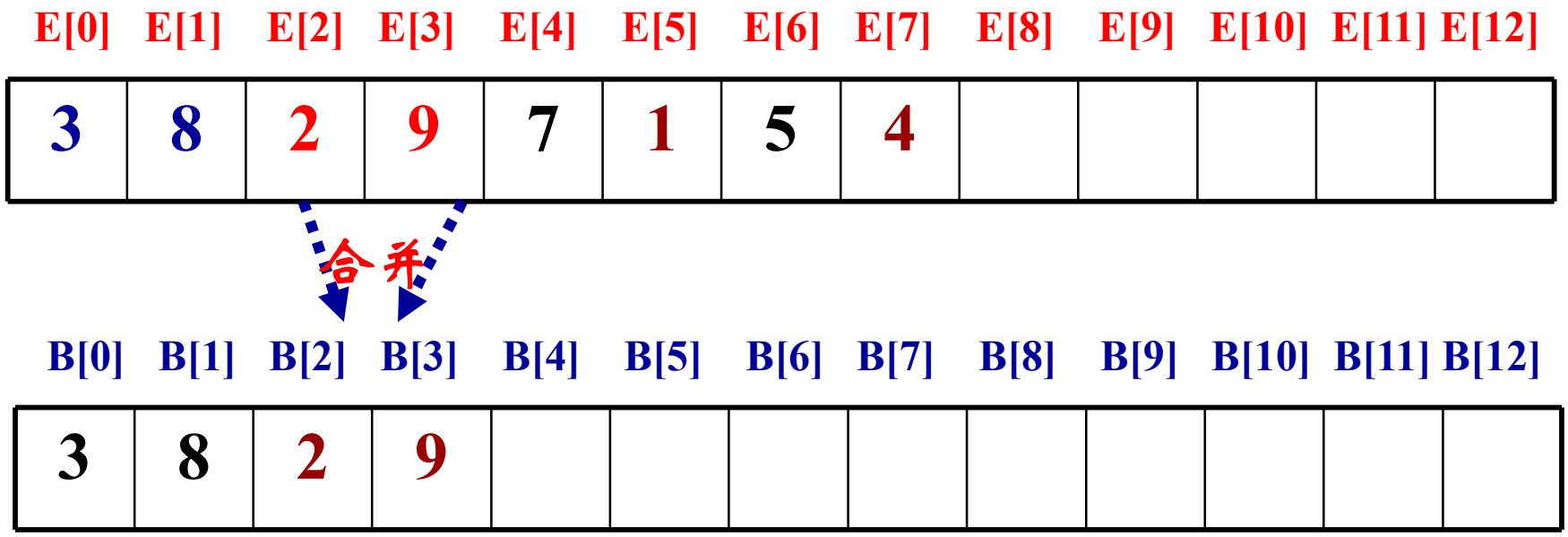
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

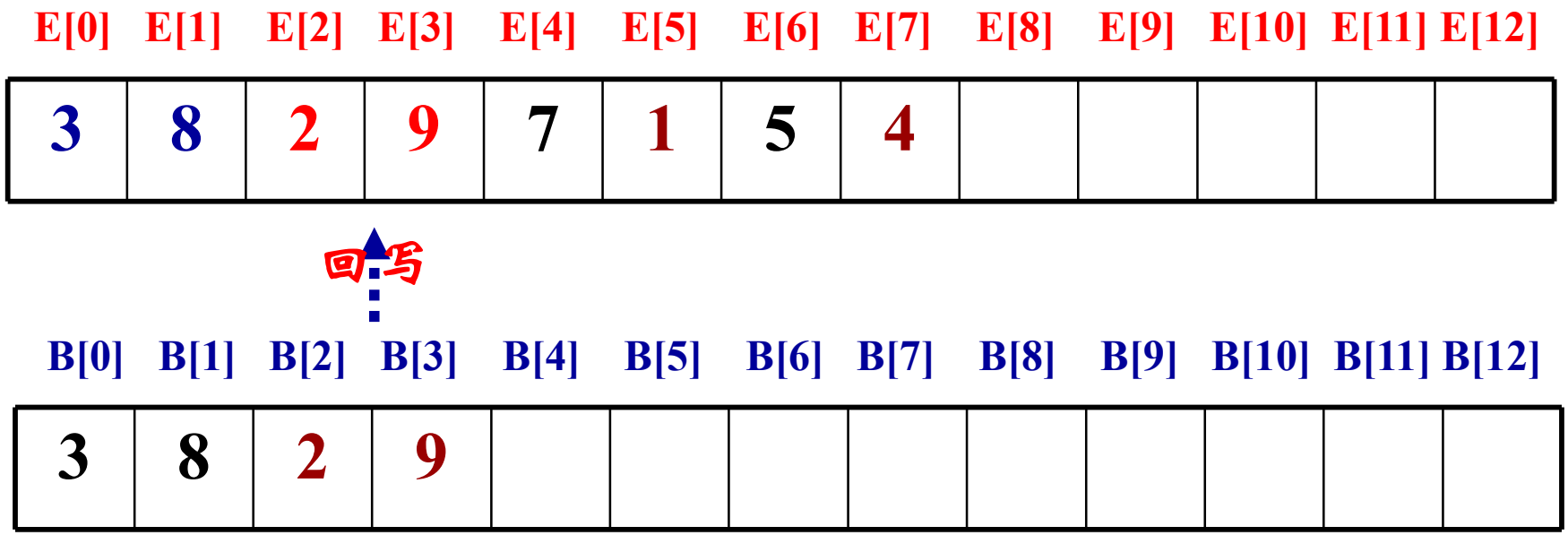
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

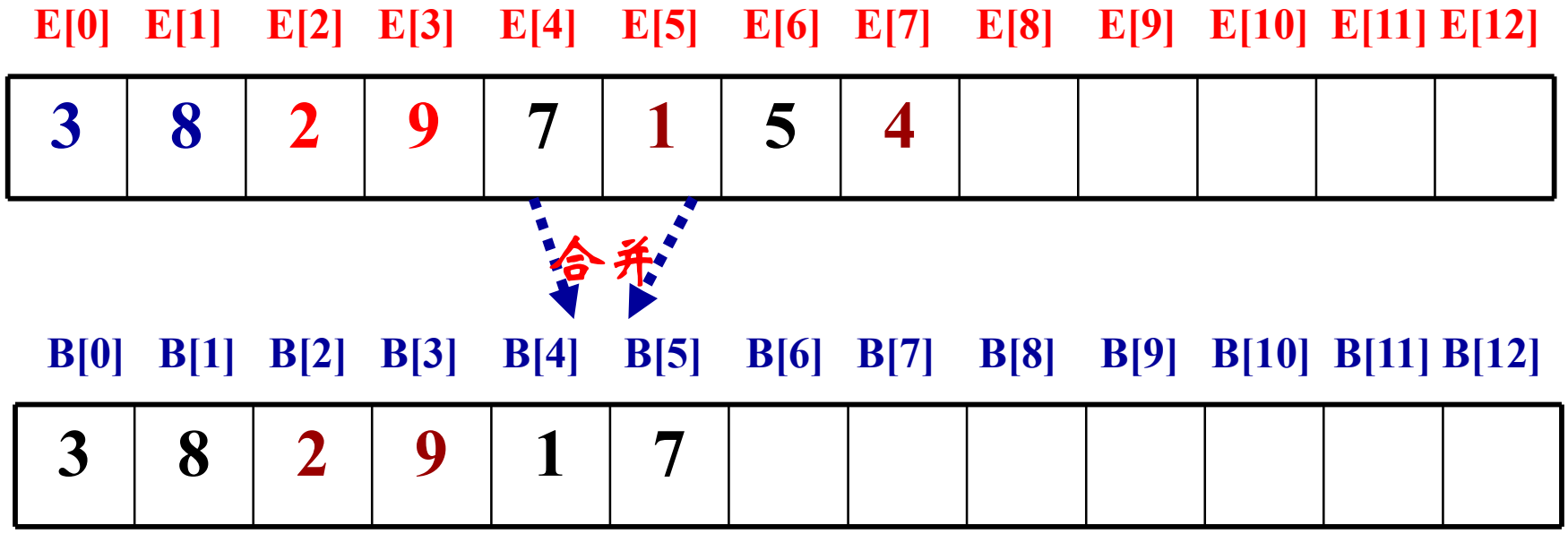
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

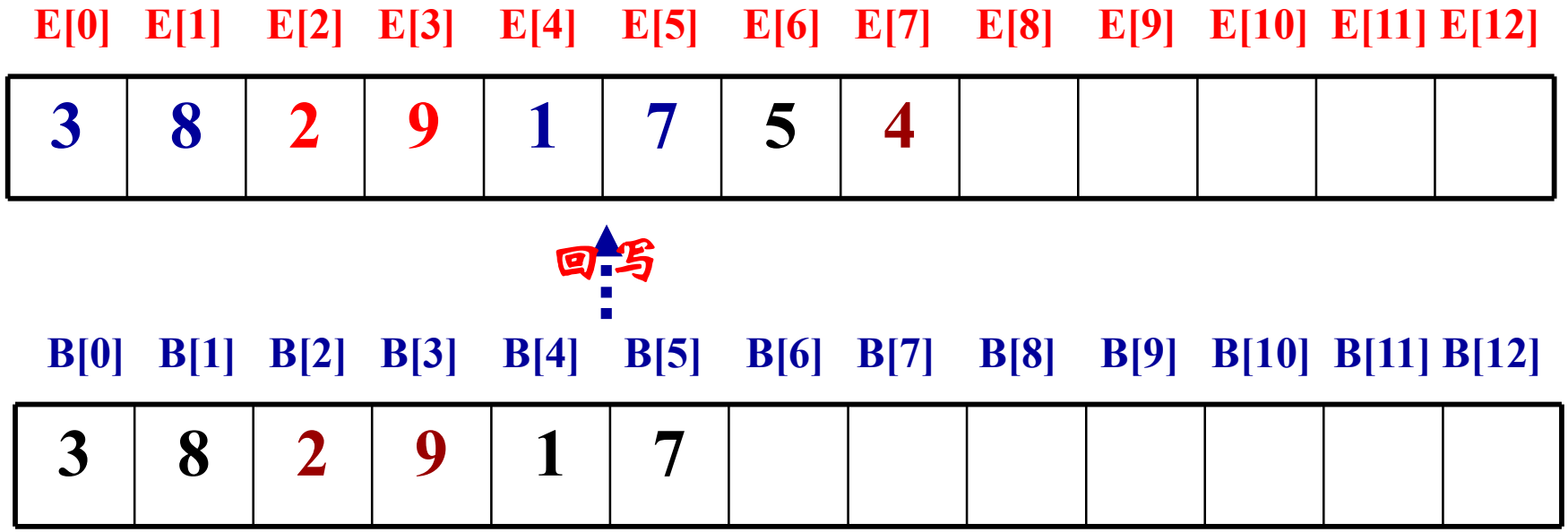
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

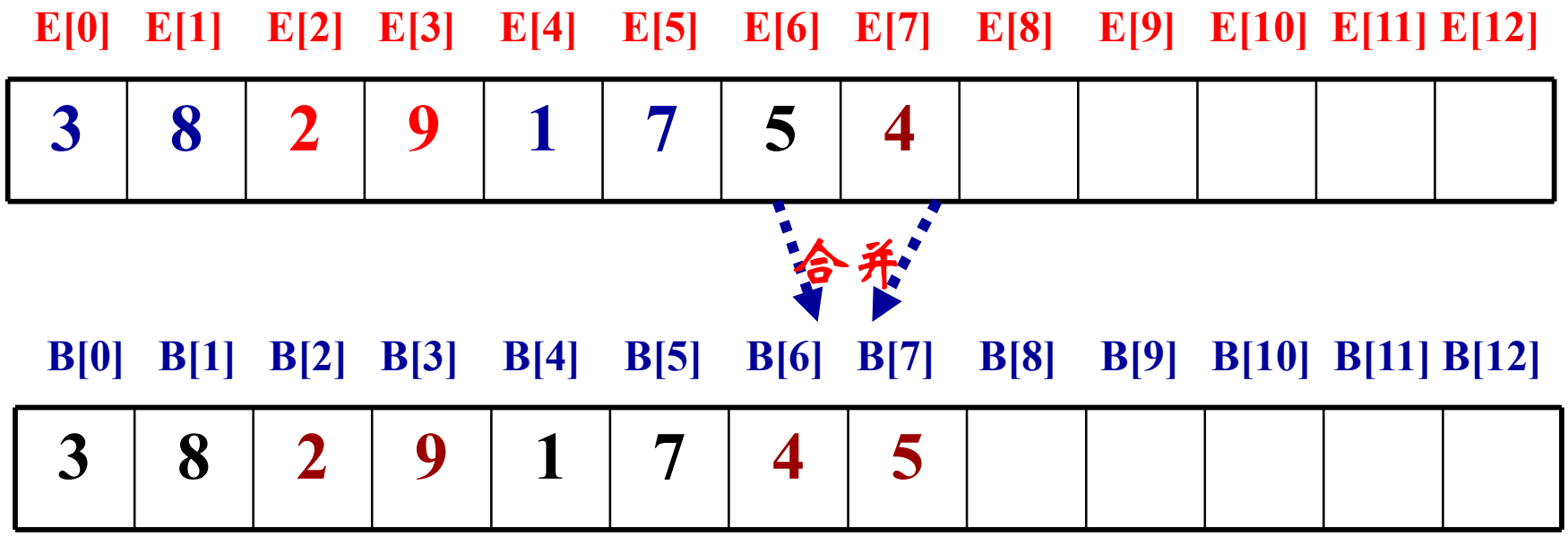
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=8

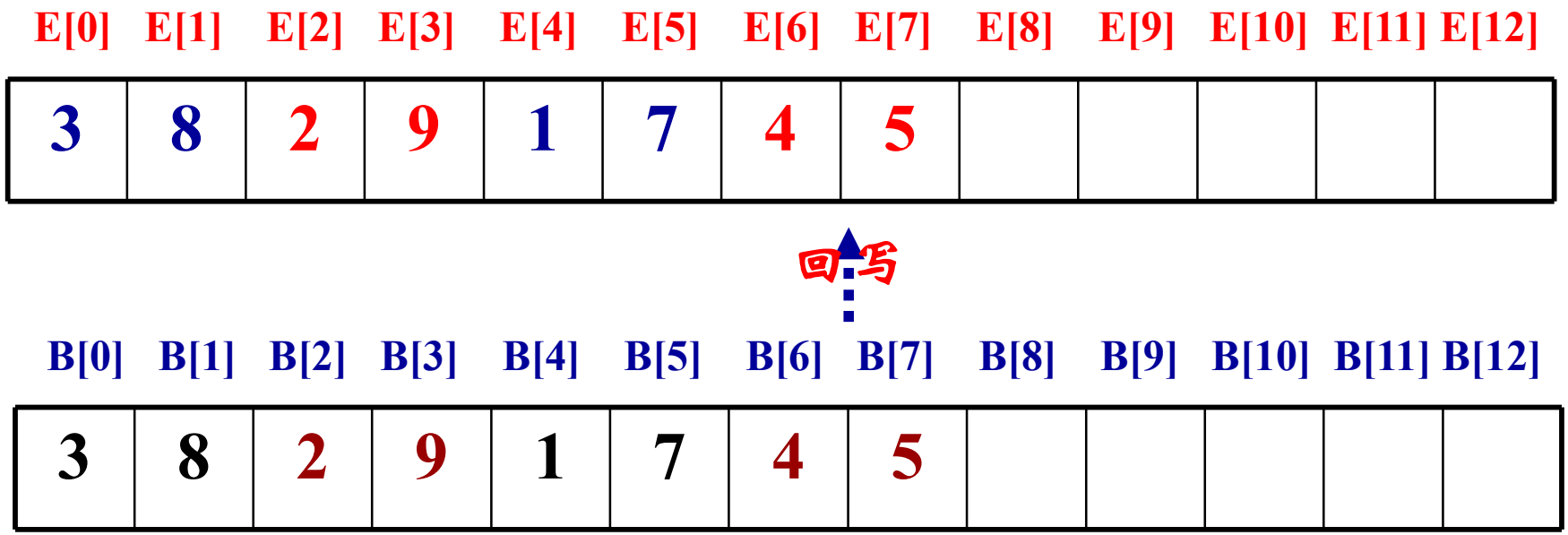
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=4

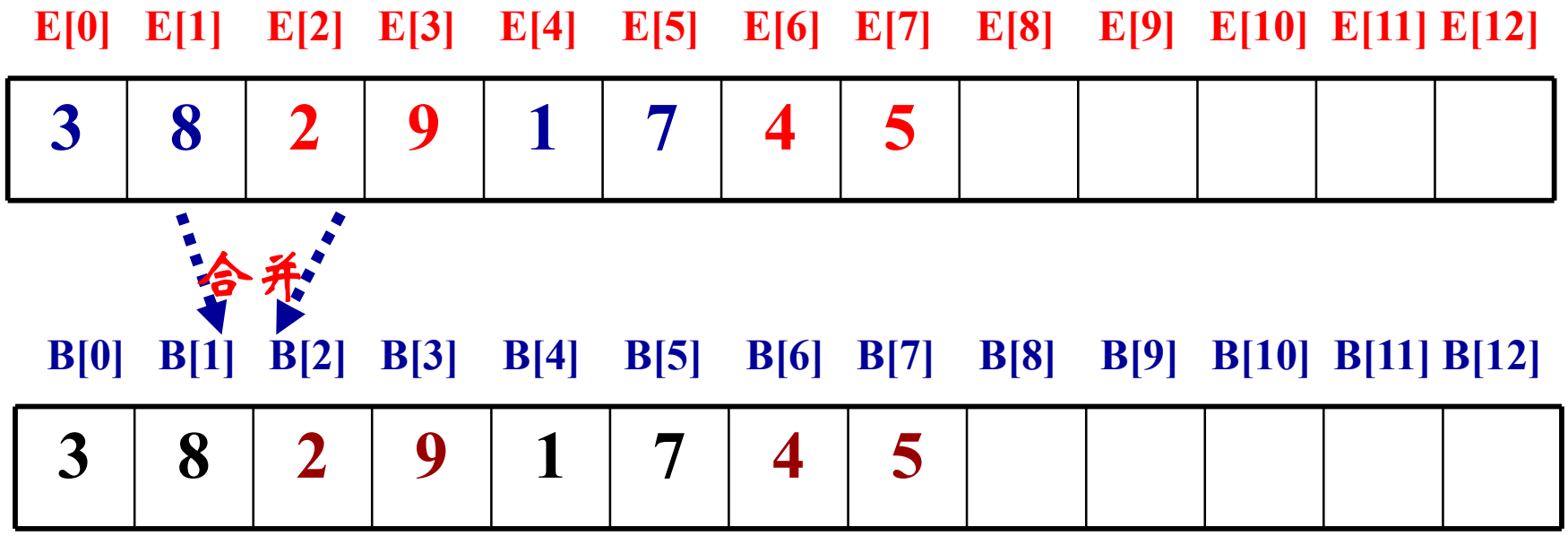
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=4

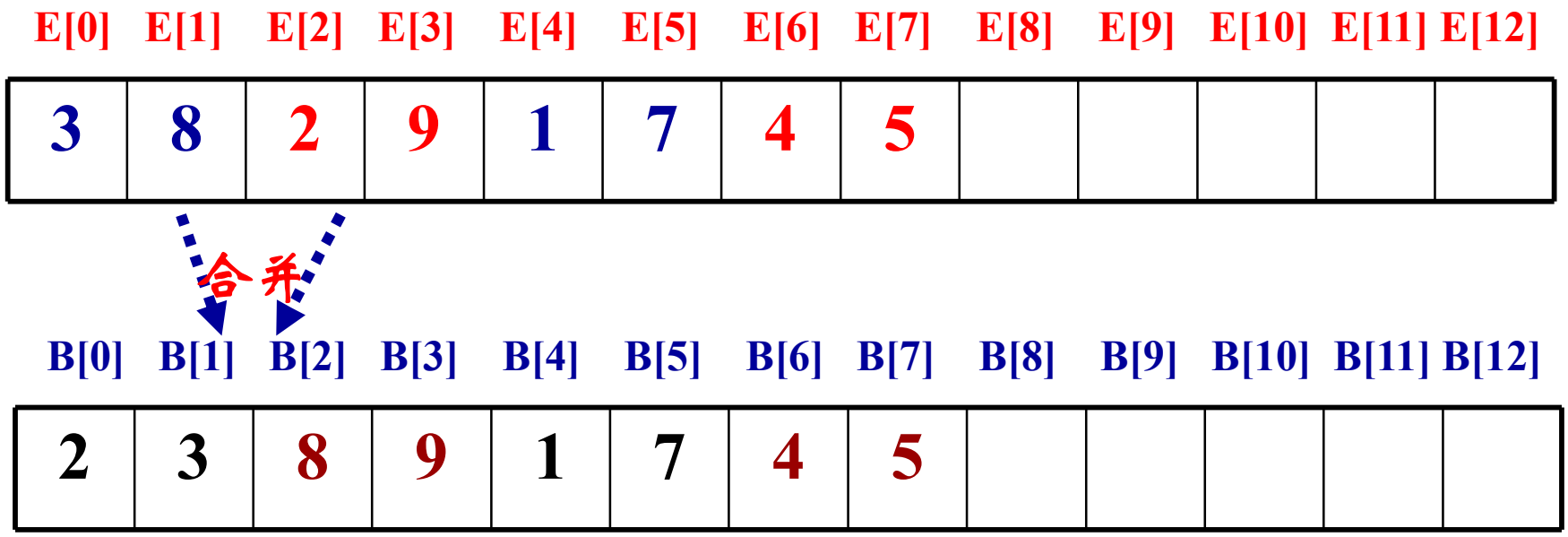
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=4

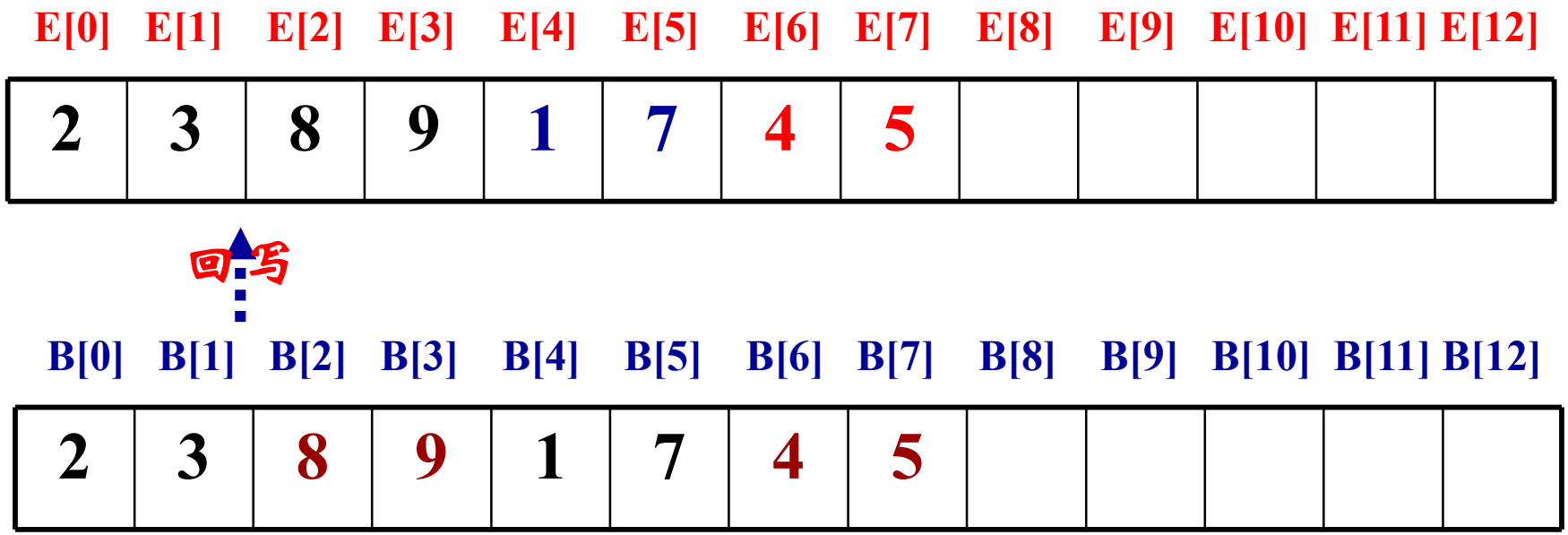
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

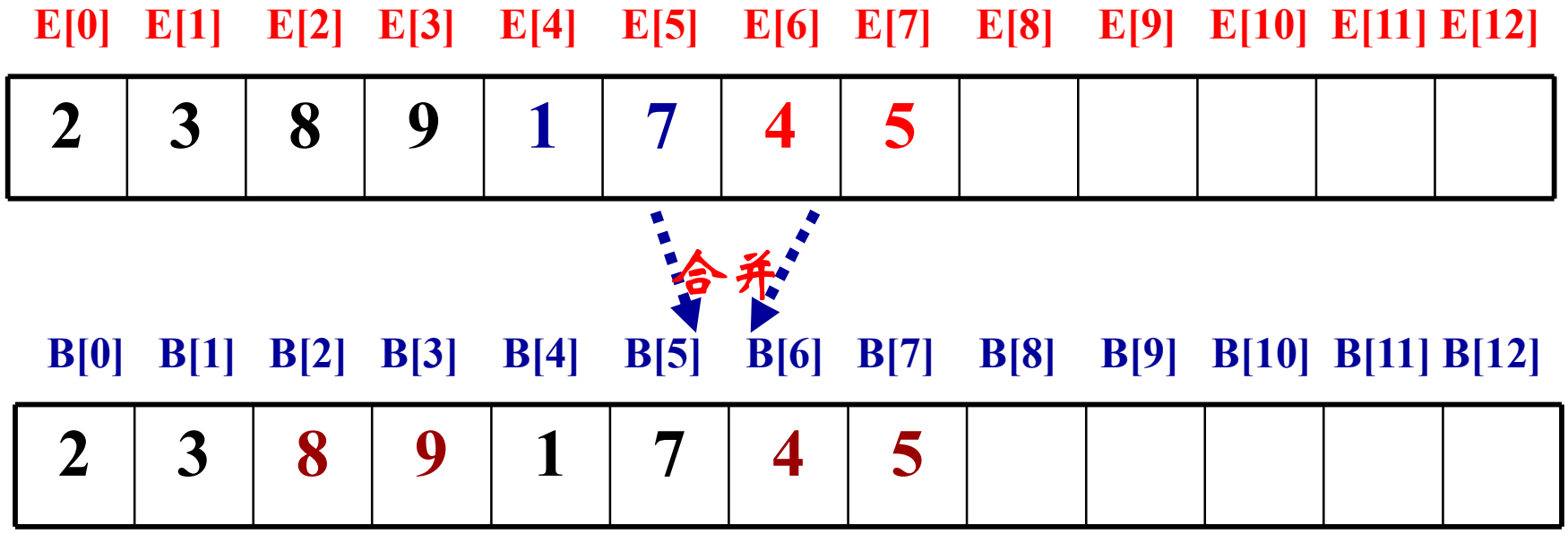
有序子序列个数=4

二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



有序子序列个数=4

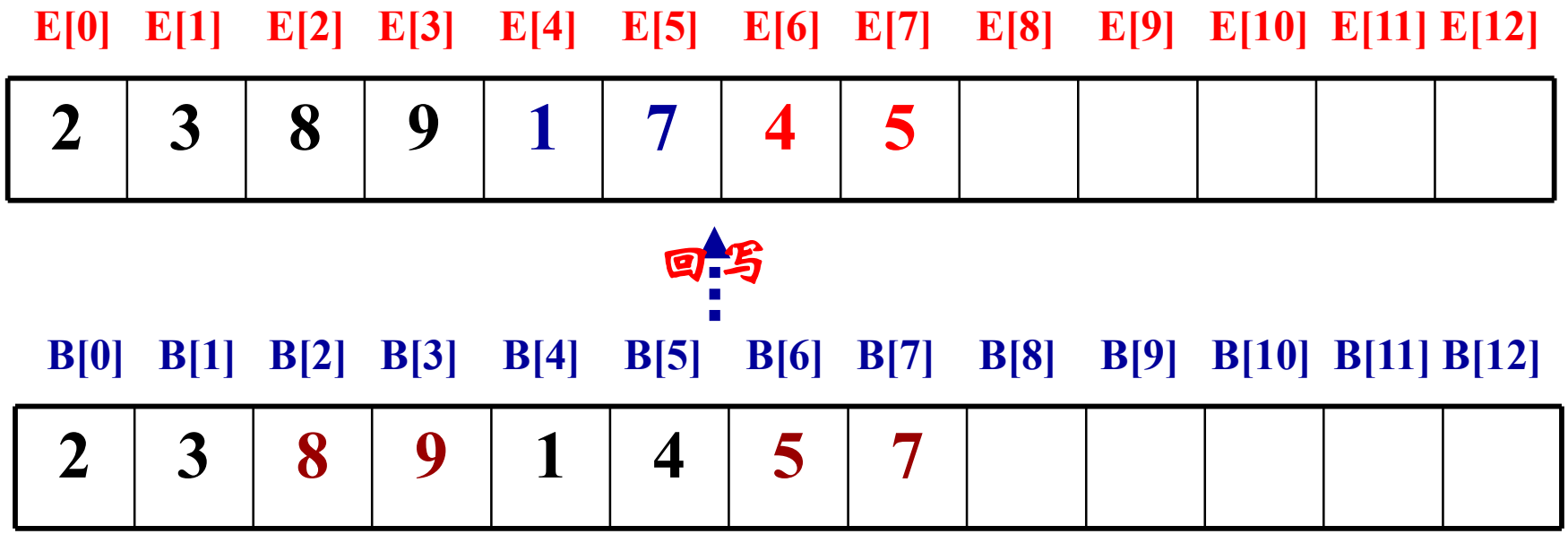
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=4

二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

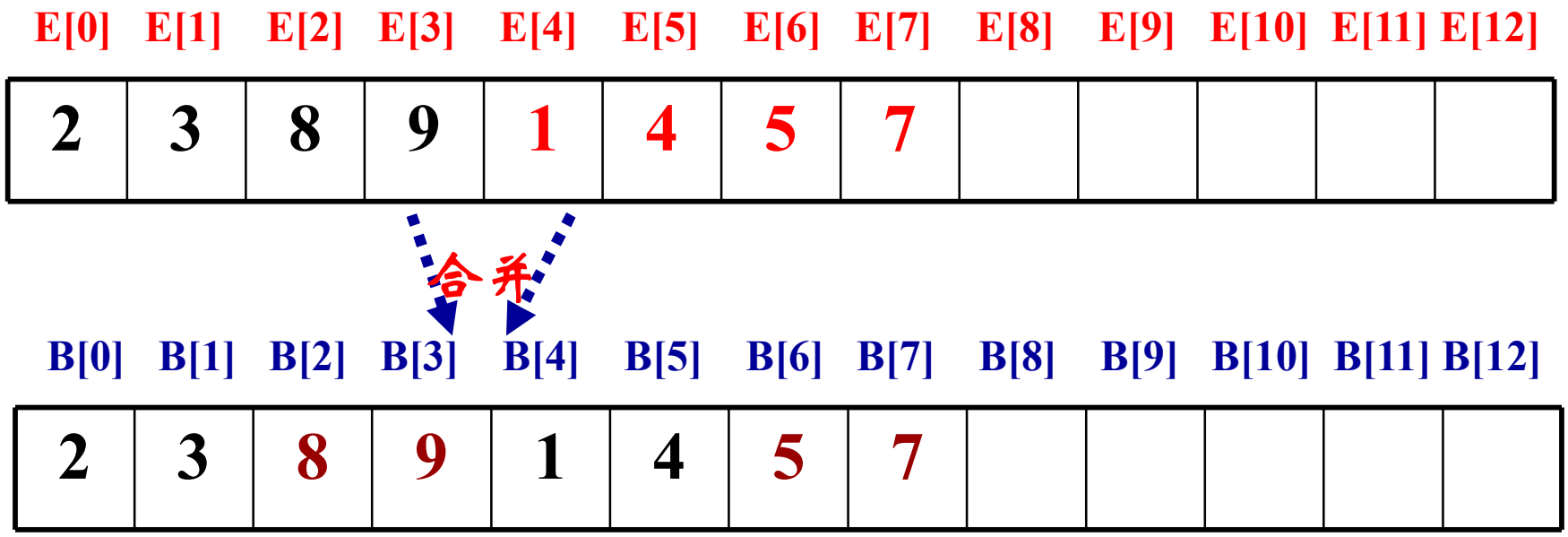
有序子序列个数=2

二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止

归并排序的非递归算法

有序子序列个数=2

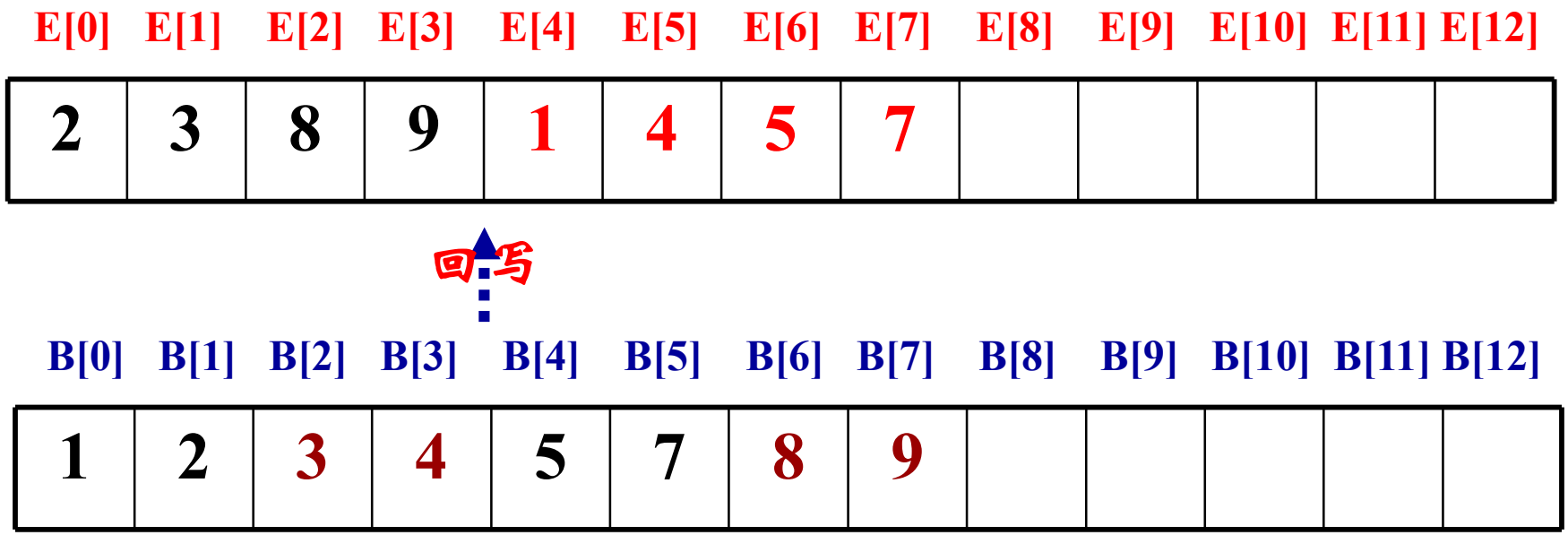
二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=2

二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止



归并排序的非递归算法

有序子序列个数=1

二路归并排序：n个待排序的数据元素看成n个有序子序列，依次合并两个相邻有序子序列，直到只剩一个有序子序列为止

E[0]	E[1]	E[2]	E[3]	E[4]	E[5]	E[6]	E[7]	E[8]	E[9]	E[10]	E[11]	E[12]
1	2	3	4	5	7	8	9					

B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]	B[10]	B[11]	B[12]
1	2	3	4	5	7	8	9					

写出该二路归并排序算法

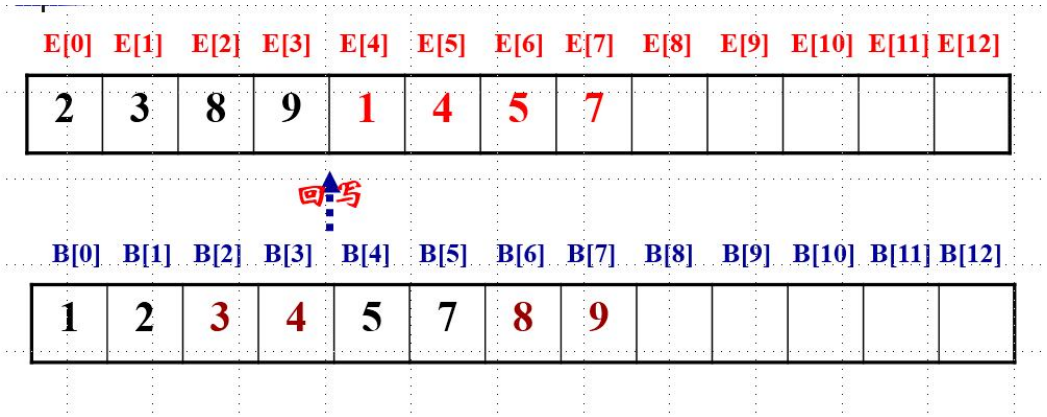
归并排序的非递归算法

Merge--将2个有序子序列合并成一个有序子序列

- 有序子序列的数据元素的个数≤Merge算法的比较次数
- Merge算法的比较次数≤2个子序列数据元素个数和-1

1. 序列1：1，2，3，4，5，6，序列2：7，8，9，10，11
2. 序列1：1，3，5，7，9，序列2：2，4，6，8，10

- 一趟归并，比较次数O(n)，移动次数2n
- 进行logn次合并-------O(nlogn)



w(n)--Merge合并2个有序序列的最坏情况

- 两个有序子序列长度分别为m和k, m+k=n，合并成一个有序序列C，则最多比较n-1次。

1. 一次比较至少将一个数据元素移入合并后的有序序列C。
2. 最坏的情况是最后一次比较两个有序子序列各自剩最后一个数据元素，而C中有n-2个数据元素，这n-2个数据元素最多经过n-2次比较产生。

- 所以合并2个有序序列的最坏情况：n-1次比较

Merge--合并2个有序序列

- Theorem 4.4：Any algorithm to merge two sorted arrays, each containing m=k=n/2 entries, by comparison of keys, does at least n-1 such comparisons in the worst case.
- 两个有序子序列长度分别为m=k=n/2，通过比较操作合并成一个有序序列C，则最坏情况下至少比较n-1次。

归并排序的性能分析

$$T(n)=\begin{cases}0, & n\le 1\\ D(n)+T(I_1)+T(I_2)+C(n), & n>1\end{cases}$$

$$T(n)=\begin{cases}0, & n\le 1\\ 0+T(n/2)+T(n/2)+C(n), & n>1\end{cases}$$

(1)一次比较至少将一个数据元素移入合并后的有序序列C。

(2)前C(n)-1次比较，每次比较将一个数据元素移入合并后的有序序列C。第C(n)次比较，一个子序列剩一个数据元素，另一个子序列剩n1>0个数据元素

(3)第C(n)次比较前共有n/2-1+n/2-n1个数据元素移入合并后的有序序列C，进行了n-1-n1次比较，则C(n)-1= n-1-n1，从而C(n)= n--n1≤n-1

最坏的情况是最后一次比较两个有序子序列各自剩最后一个数据元素，而C中有n-2个数据元素， C(n)=n-1。

最好的情况是最后一次比较一个有序子序列剩最后一个数据元素，一个有序子序列剩n/2个数据元素，C(n)=n/2。

归并排序W(n)分析

$w(n)=w(\lfloor n/2 \rfloor)+w(\lceil n/2 \rceil)+(n-1)\quad w(1)=0;\quad n\approx 2^k$

$=2w(n/2)+(n-1)$

$=2(2w(n/2^2)+(n/2-1))+(n-1)$

$=2^2w(n/2^2)+n-2+(n-1)$

$=2^2w(n/2^2)+2n-2-1$

$=2^2(2w(n/2^3)+n/2^2-1)+2n-2-1$

$=2^3w(n/2^3)+3n-2^2-2-1$

$=\cdots\cdots$

$=2^kw(n/2^k)+kn-2^{k-1}-\cdots\cdots-2^2--2-1$

$=nw(1)+nlogn-((1-2^k)/(1-2))$

$=nlogn-(n-1)\in O(nlogn)$

归并排序最好情况分析

- 待排序数据元素已经有序,合并logn趟,每趟n/2次比较, O (nlogn)
- $T(n)=T(\lfloor n/2 \rfloor)+T(\lceil n/2 \rceil)+n/2 \quad T(1)=0; \quad n\approx 2^k$

$=2T(n/2)+n/2$

$=2(2T(n/2^2)+n/2^2)+n/2$

$=2^2T(n/2^2)+n/2+n/2$

$=2^2(2T(n/2^3)+n/2^3)+2(n/2)$

$=2^3T(n/2^3)+n/2+2(n/2)$

$=2^3T(n/2^3)+3(n/2)$

$=\cdots\cdots$

$=2^kT(n/2^k)+k(n/2)$

$=nT(1)+(n/2)\log n$

$\in O(n\log n)$

改进措施

- 不回写
- 递归
- 最长无逆序子序列
- 小排序问题:划分为小序列（长度为20），做直接插入排序，在采用归并排序

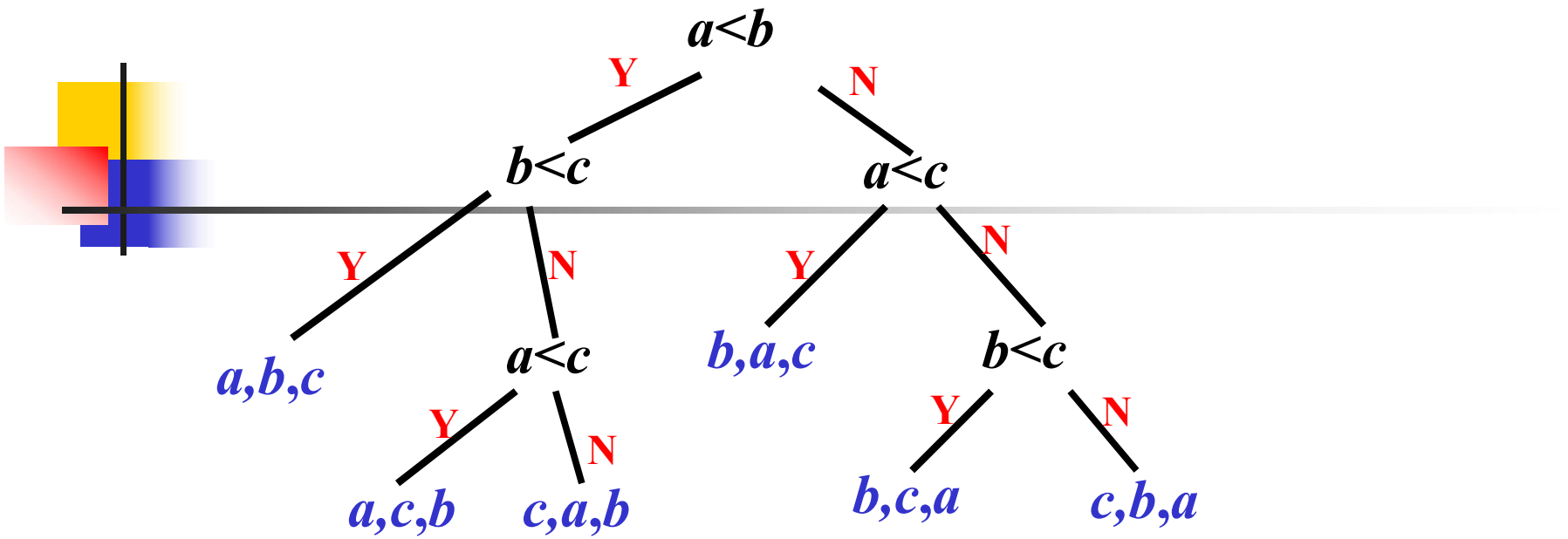
```
void Merge (Element E[ ] , Element B[ ] , int s, int m, int t)
{// 将有序的记录序列 E[s..m] 和 E[m+1..t]合并到B[s..t]
for (k=s,p=s, j=m+1;  p<=m && j<=t;  ++k)
    {
        if (E[p].key<=E[j].key)  B [k] = E[p++];
         else B[k] = E[j++];
    }
  if (p<=m) B[k..t] = E[p..m];
  if (j<=t) B[k..t] = E[j..t];

} // Merge
```

主要操作：子问题的解合并得到原问题的解将2个有序序列合并为一个有序序列----不回写！！！

4.7 Lower bounds for sorting by comparison of keys

- 输入*a*,*b*,*c*三个数排序，可能的结果为一下3!=6种：
  - *a*, *b*, *c*
  - *a*, *c*, *b*
  - *b*, *a*, *c*
  - *b*, *c*, *a*
  - *c*, *a*, *b*
  - *c*, *b*, *a*
- 输入*n*个数排序，可能的结果有*n*!个

a,b,c三个数排序



1. 不论采用何种排序方法，只要是基于比较操作完成的排序都可得到一棵高度相同、具有相同6个叶子结点的判定树。

2. 判定树为二叉树，非叶结点为比较操作，叶结点为可能的排序结果。

3. 判定树的高度反映最坏情况下比较次数。

4.对n个数据元素排序，其判定树的叶子有n!个。

令：根结点的深度为0，任一结点的深度=父结点深度+1；二叉树的高度=结点深度最大值

对n个数据元素排序，最坏至少需要比较多少次？

$w(n) \geq n\log n$?

基于“比较”操作的排序算法w(*n*)的下界

引理：高度为*h*的二叉树最多有2^h个叶子结点

定理：*n*个不同数据元素排序的判定树的高度至少为 log(*n*!)

证明：*n*个不同数据元素排序，可能的结果有的*n*!个，对应的判定树有*n*!个叶子结点。设有*n*!个叶子结点的二叉树高度为*h*的二叉树，则有： $n! \le 2^h$

$w(n)=h \ge \log(n!)$

$n!=n(n-1)(n-2)\cdots\cdots 3*2*1 \ge n(n-1)(n-2)\cdots\cdots\lceil n/2 \rceil$

$n! \ge (n/2)^{n/2}$

$w(n)=h \ge \log(n!) \ge \log (n/2)^{n/2} = n/2\log(n/2) \in O(n\log n)$

推论：比较排序$w(n) \ge O(n\log n)$

基于“比较”操作的排序算法A(*n*)的下界

A(*n*)=判定树的平均叶子高度

$$A(n)=\frac{1}{n!}\sum_{i=1}^{n!}h_i \qquad A(n)=\frac{1}{n!}\sum_{i=1}^{n!}h_i \geq \frac{1}{n!}\text{(最高层所有叶子结点的高度和)}$$

高度为log(*n*!)-1的二叉树最多有 $2^{(\log n!-1)}=\frac{n!}{2}$ 个叶子结点

$$A(n)\geq \frac{1}{n!}\sum_{i=1}^{\frac{n!}{2}}\log(n!)=\frac{1}{n!}\frac{n!}{2}\log(n!)=\frac{1}{2}\log(n!)$$

***n*!=*n*(*n*-1)(*n*-2)……3*2*1 ≥ *n*(*n*-1)(*n*-2)……⌈*n*/2⌉**

$n!\geq (n/2)^{n/2}$

A(*n*)≥(1/2)log(*n*!) ≥(1/2)log $(n/2)^{n/2}$= (*n*/4)log(*n*/2) ∈ O (*n*log*n*)

推论：比较排序A(*n*) ≥ O(*n*log*n*)

归并排序--最优的基于比较操作的排序

比较排序： $w(n) \geq O(n\log n)$

比较排序在最坏情况下至少做 $O(n\log n)$ 次的比较

$n=5: \lceil \log 5! \rceil = 7$

归并排序？ 8次

归并排序W(n)分析

$w(n)=w(\lfloor n/2 \rfloor)+w(\lceil n/2 \rceil)+(n-1) \quad w(1)=0; \quad n\approx 2^k$

$=2w(n/2)+(n-1)$

$=2(2w(n/2^2)+(n/2-1))+(n-1)$

$=2^2w(n/2^2)+n-2+(n-1)$

$=2^2w(n/2^2)+2n-2-1$

$=2^2(2w(n/2^3)+n/2^2-1)+2n-2-1$

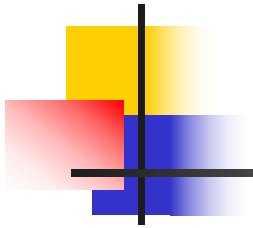
$=2^3w(n/2^3)+3n-2^2-2-1$

$=\ldots\ldots$

$=2^kw(n/2^k)+kn-2^{k-1}-\ldots\ldots-2^2-2-1$

$=nw(1)+n\log n-((1-2^k)/(1-2))$

$=n\log n-(n-1) \in O(n\log n)$



习题4.32

- 给出在最坏情况下，只需5次比较的4个元素的排序算法
  - 2趟归并排序
- 给出在最坏情况下，只需7次比较的5个元素的排序算法