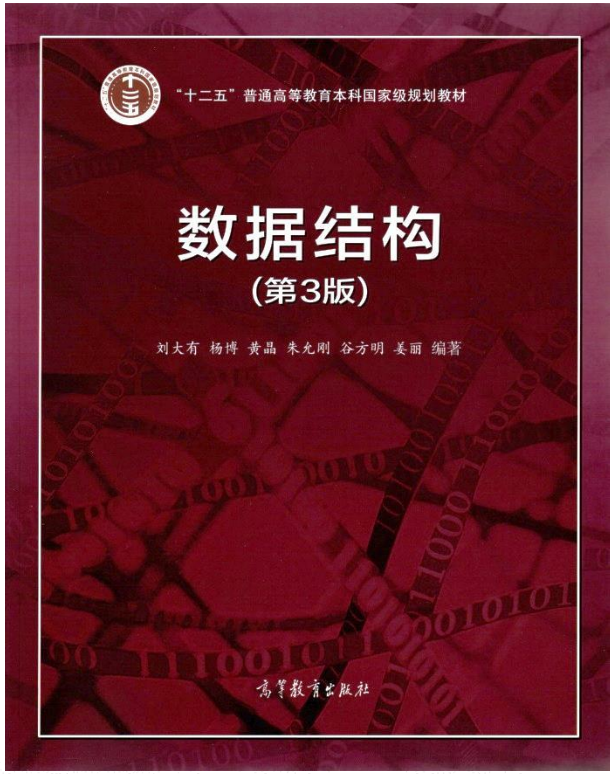
计算机学院王湘浩班

2024级



归并排序及其他

- 归并排序
- 排序算法时间下界
- 外排序方法简介

数据之法
结构之美
算法之道

Last updated on 2025.5

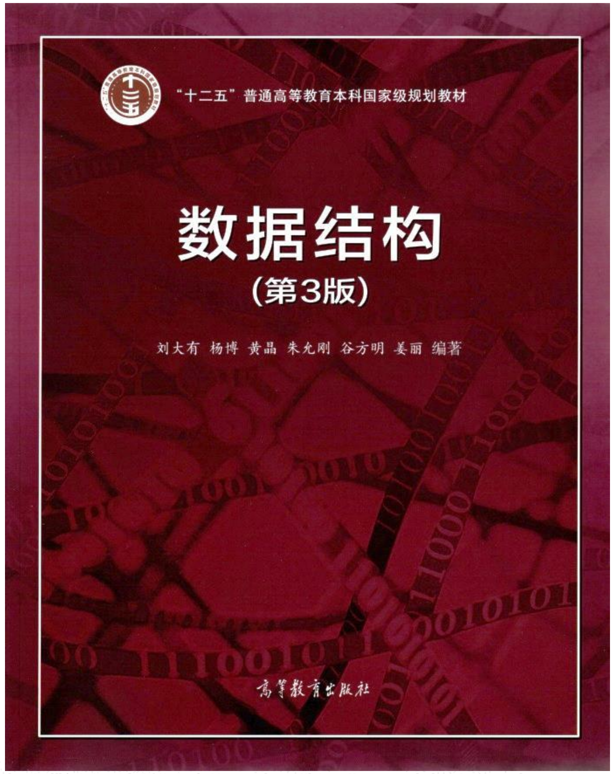
zhuyungang@jlu.edu.cn

归并排序算法提出者

John von Neumann
冯·诺依曼
美国科学院院士
普林斯顿大学教授
现代计算机之父

If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.

如果有人不相信数学是简单的，那是因为他们没有意识到生活有多复杂。



归并排序及其他

- 归并排序
- 排序算法时间下界
- 外排序方法简介

数据之法
结构之美
算法之道

Last updated on 2025.5

zhuyungang@jlu.edu.cn

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(R_low,…,R_mid)和(R_mid+1,…,R_high)合并成一个有序数组
```

low *mid* *mid*+1 *high*

R	8	21	25	26	50	62	72	93	16	37	39

↑ *i* ↑ *j*

X	8

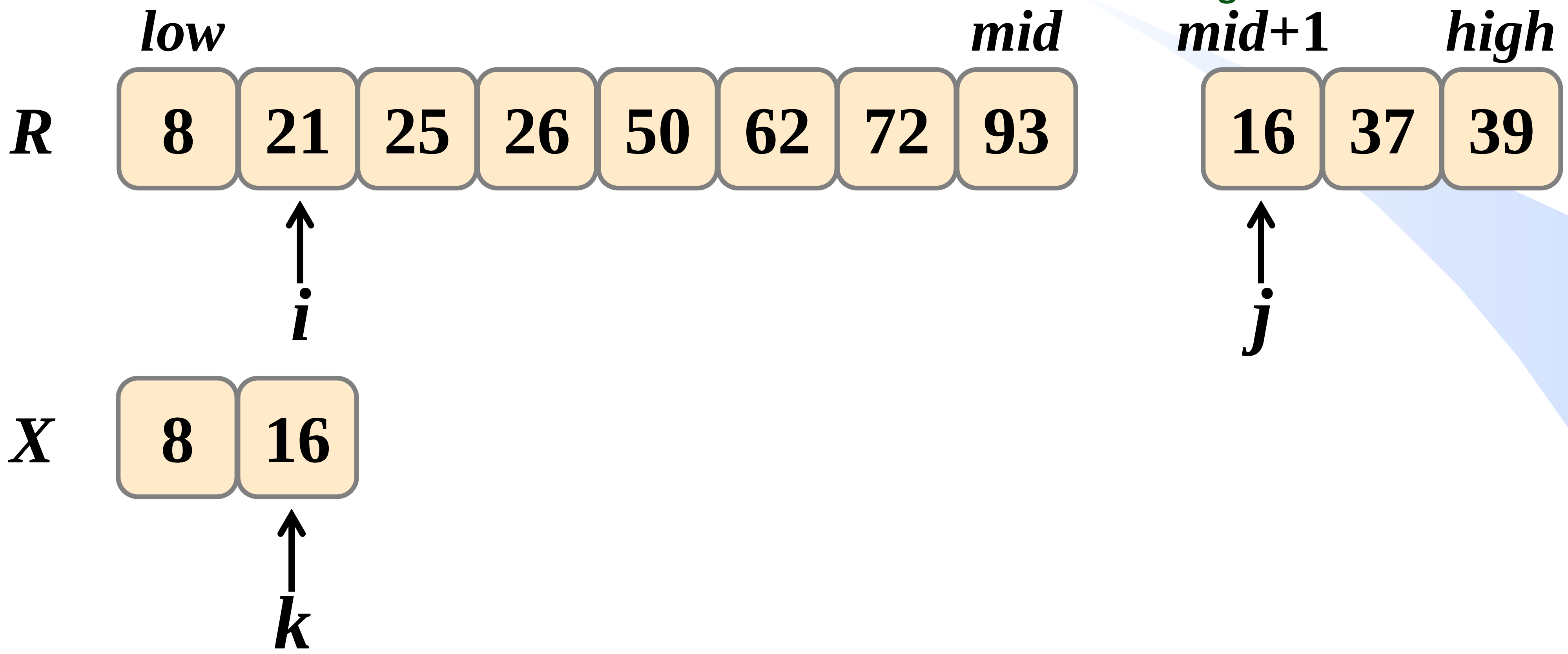
↑ *k*

- 通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- 当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(R_low,…,R_mid)和(R_mid+1,…,R_high)合并成一个有序数组
```

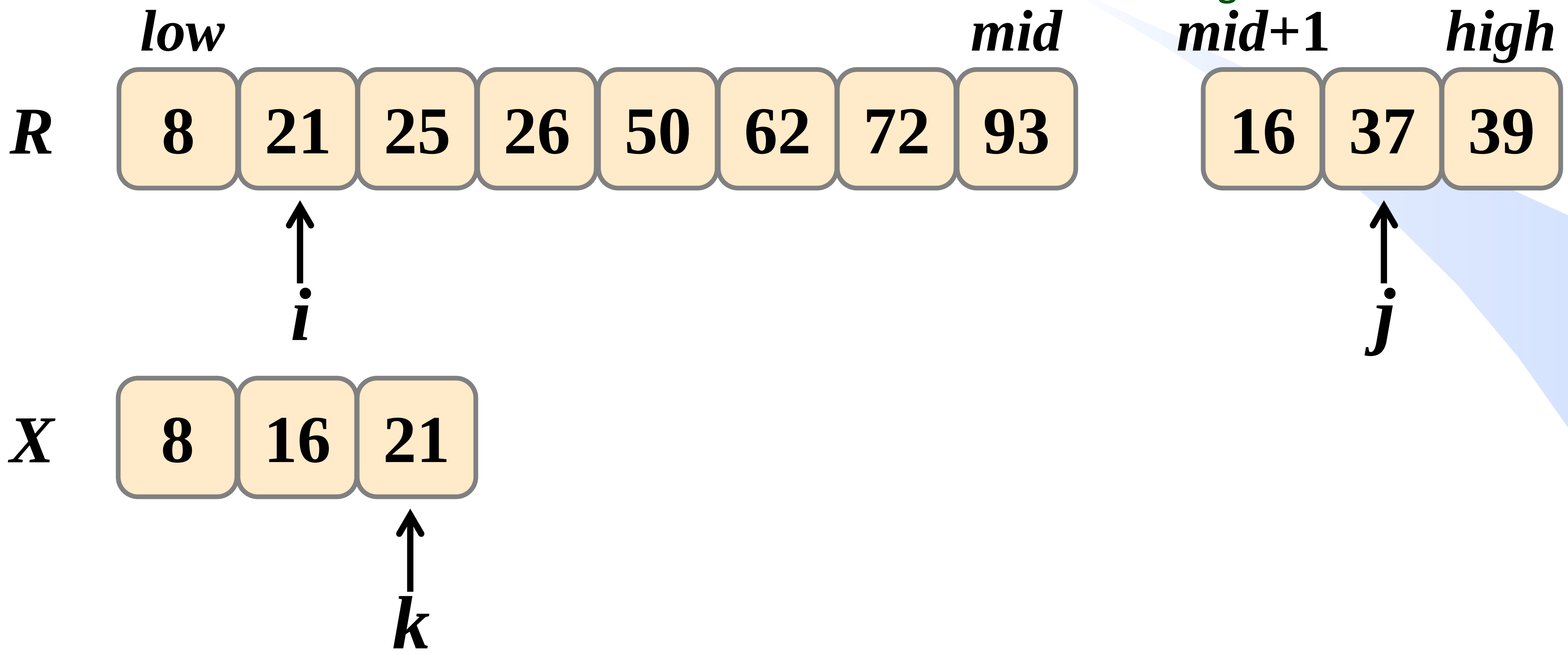


- ➢通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- ➢当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(Rlow,…,Rmid)和(Rmid+1,…,Rhigh)合并成一个有序数组
```

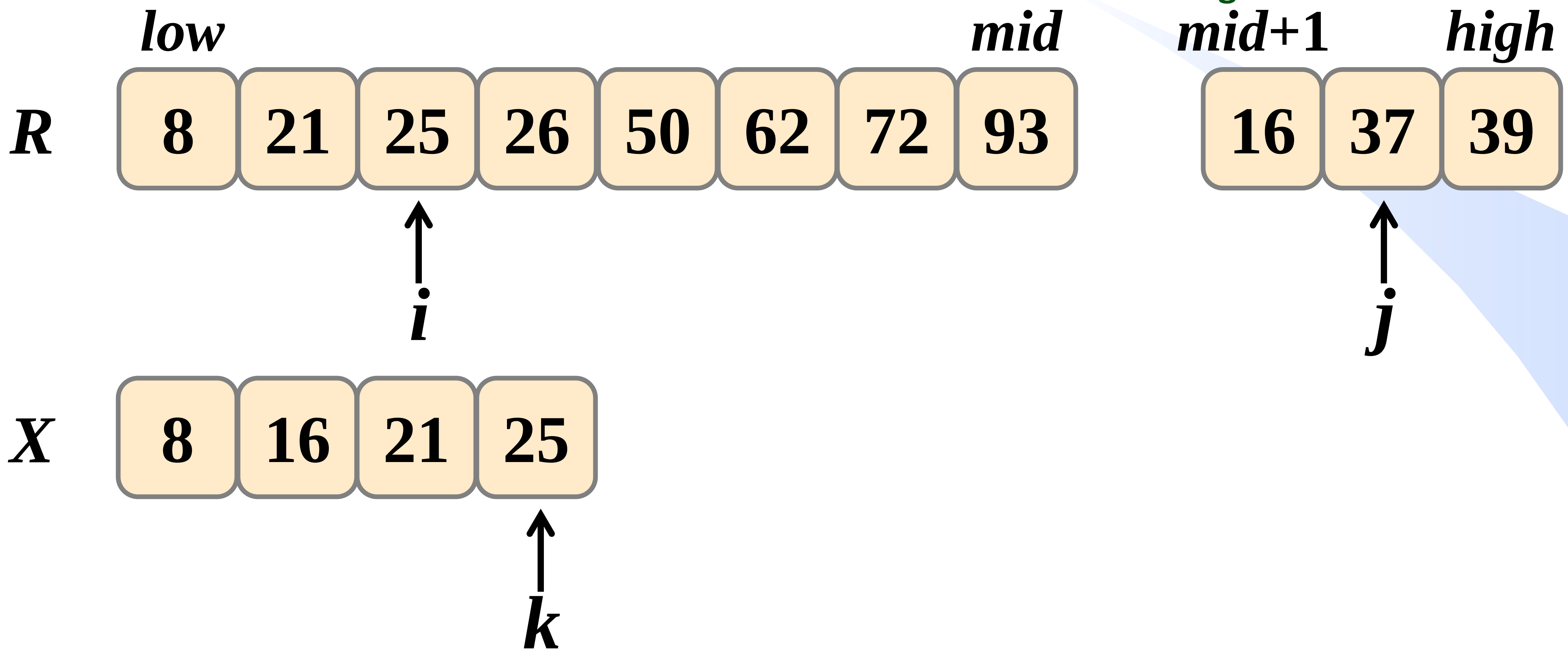


- ➢通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- ➢当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(R_low,…,R_mid)和(R_mid+1,…,R_high)合并成一个有序数组
```

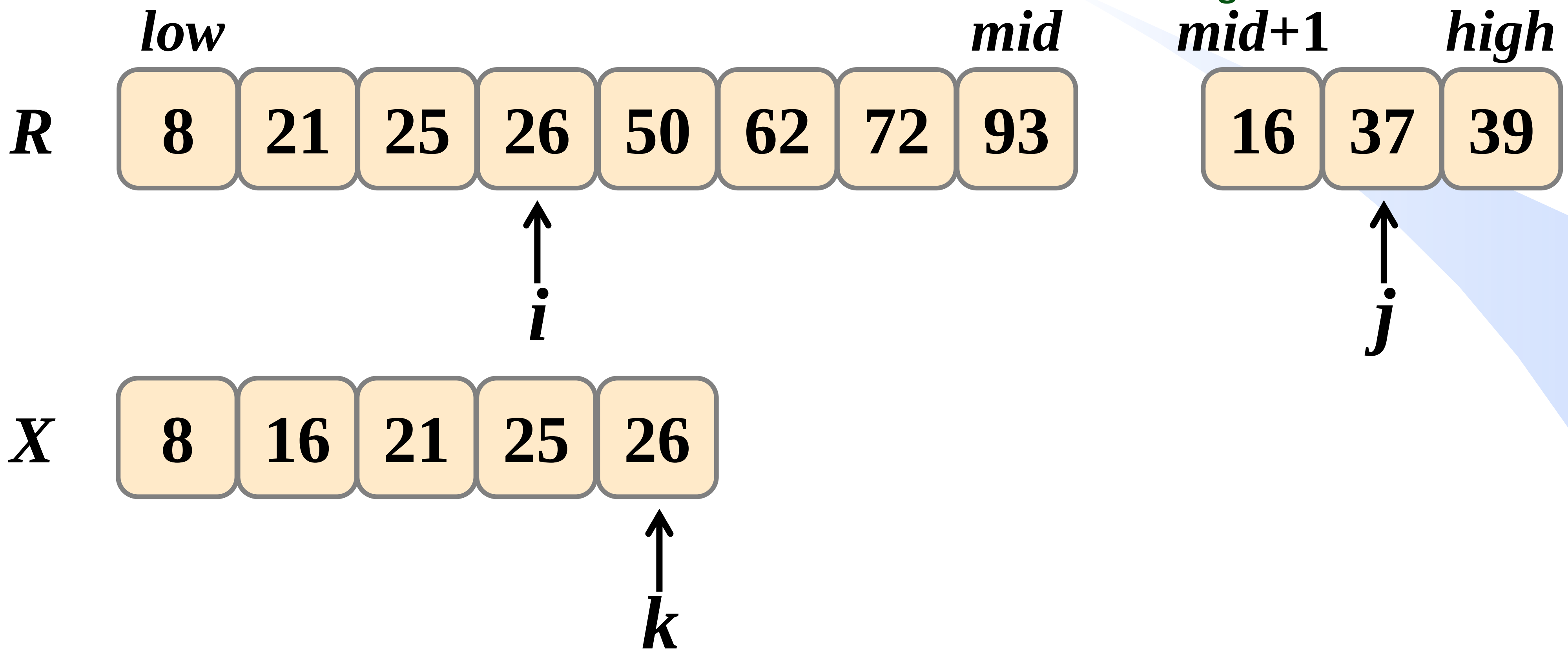


- 通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- 当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(Rlow,…,Rmid)和(Rmid+1,…,Rhigh)合并成一个有序数组
```

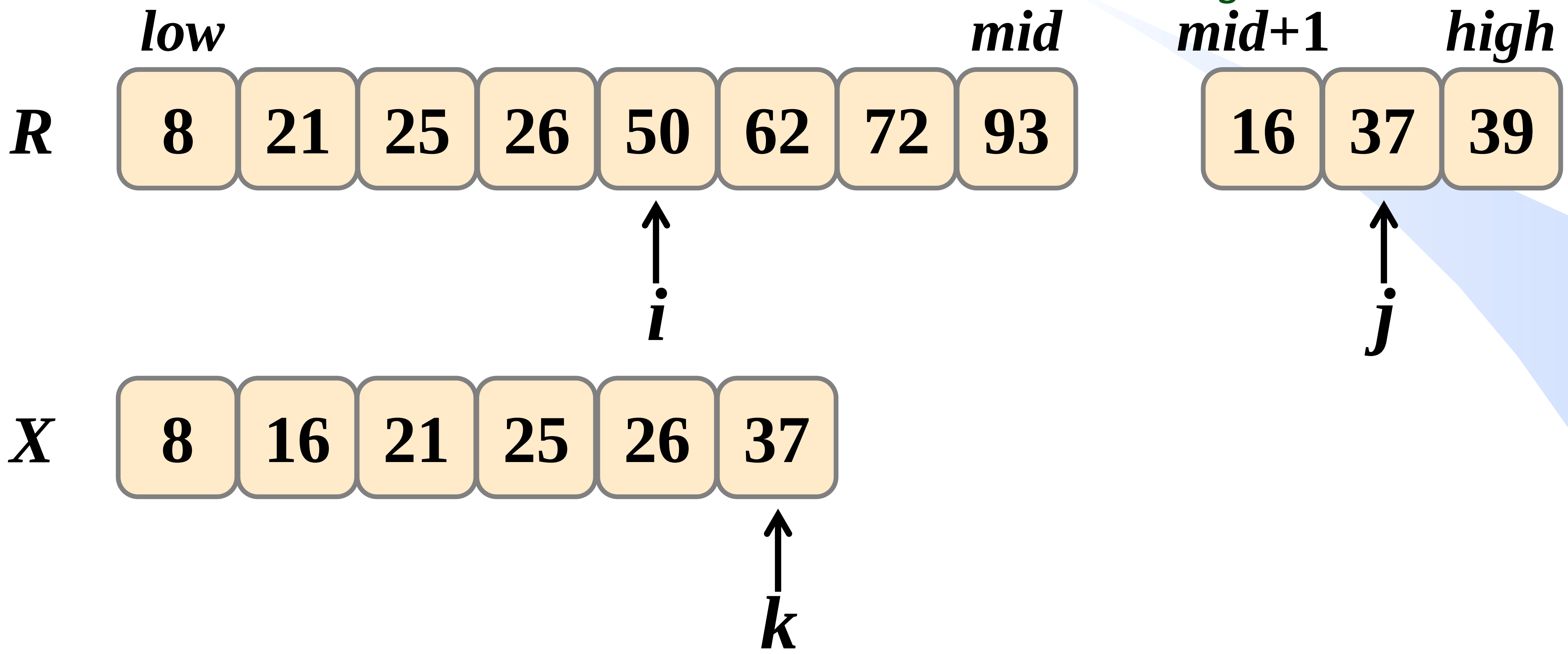


- 通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- 当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(Rlow,…,Rmid)和(Rmid+1,…,Rhigh)合并成一个有序数组
```



- ➢通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- ➢当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(R_low,…,R_mid)和(R_mid+1,…,R_high)合并成一个有序数组
```

low … *mid*　　*mid*+1 … *high*

R	8	21	25	26	50	62	72	93		16	37	39
					↑ *i*							↑ *j*

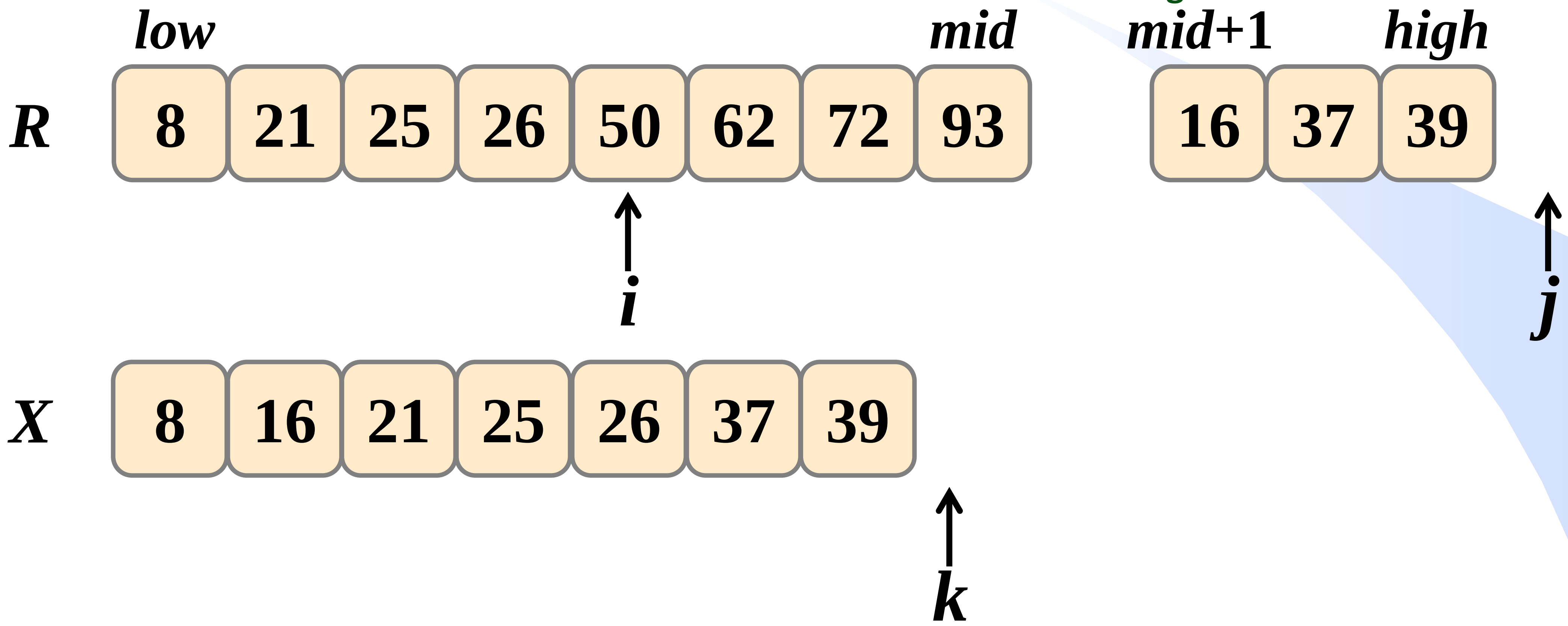
X	8	16	21	25	26	37	39
							↑ *k*

- ➢通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- ➢当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(Rlow,…,Rmid)和(Rmid+1,…,Rhigh)合并成一个有序数组
```



- 通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- 当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(R_low,…,R_mid)和(R_mid+1,…,R_high)合并成一个有序数组
```

	low							*mid*	*mid*+1		*high*
R	8	21	25	26	50	62	72	93	16	37	39
								↑ *i*			↑ *j*

X	8	16	21	25	26	37	39	50	62	72	93
											↑ *k*

- ➢通过 *i* 和 *j* 扫描两个子数组，将R[*i*]与R[*j*]的关键词较小者放入*X*[*k*]中
- ➢当一个子数组扫描完毕后，将另一个子数组的剩余部分直接放入*X*中

两个有序子数组合并成一个大的有序子数组

A

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(Rlow,…,Rmid)和(Rmid+1,…,Rhigh)合并成一个有序数组
    int i=low, j=mid+1, k=0;
    int *X=new int[high-low+1];
    while(i<=mid  && j<=high)
       if(R[i]<=R[j]) X[k++]=R[i++];
       else X[k++]=R[j++];
    while(i<=mid) X[k++]=R[i++]; //复制余留记录
    while(j<=high) X[k++]=R[j++];
    for(i=0; i<=high-low; i++)  //将X拷贝回R
       R[low+i]=X[i];
    delete []X;
}
```

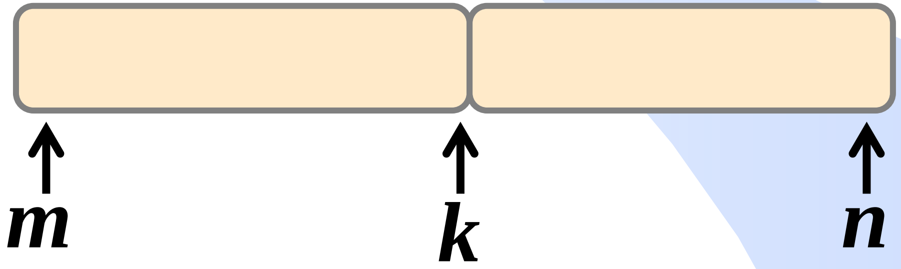
时间复杂度O(n)
空间复杂度O(n)

X_0	X_1	X_2	…	X_i	…	$X_{high\text{-}low}$

R_{low}	R_{low+1}	R_{low+2}	…	R_{low+i}	…	R_{high}

归并排序（递归形式）

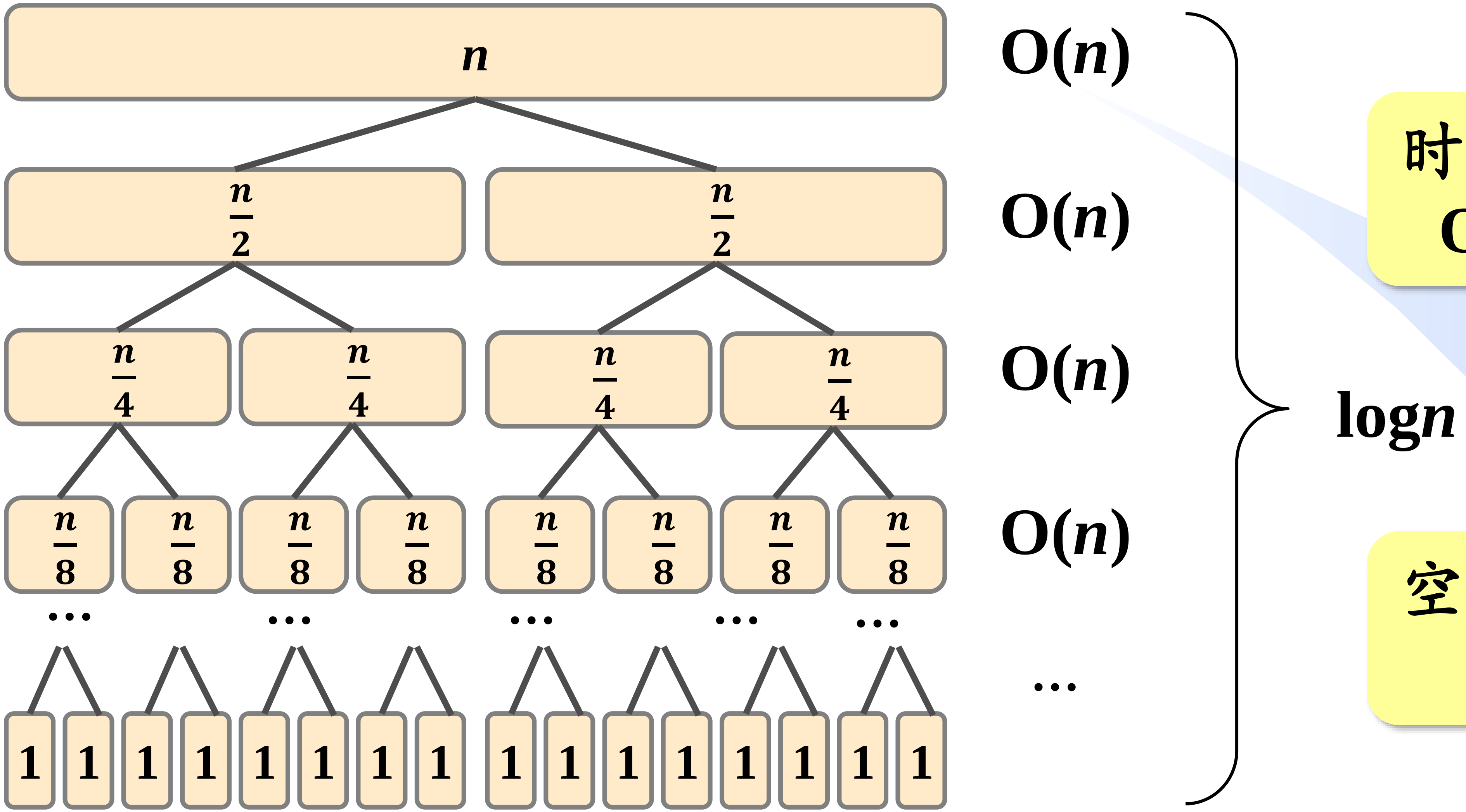
```
void MergeSort(int R[], int m, int n){
    if(m < n){
        int k = (m+n)/2; //将待排序序列等分为两部分
        MergeSort(R, m, k);
        MergeSort(R, k+1, n);
        Merge(R, m, k, n);
    }
}
```



分治法

归并排序——时间复杂度

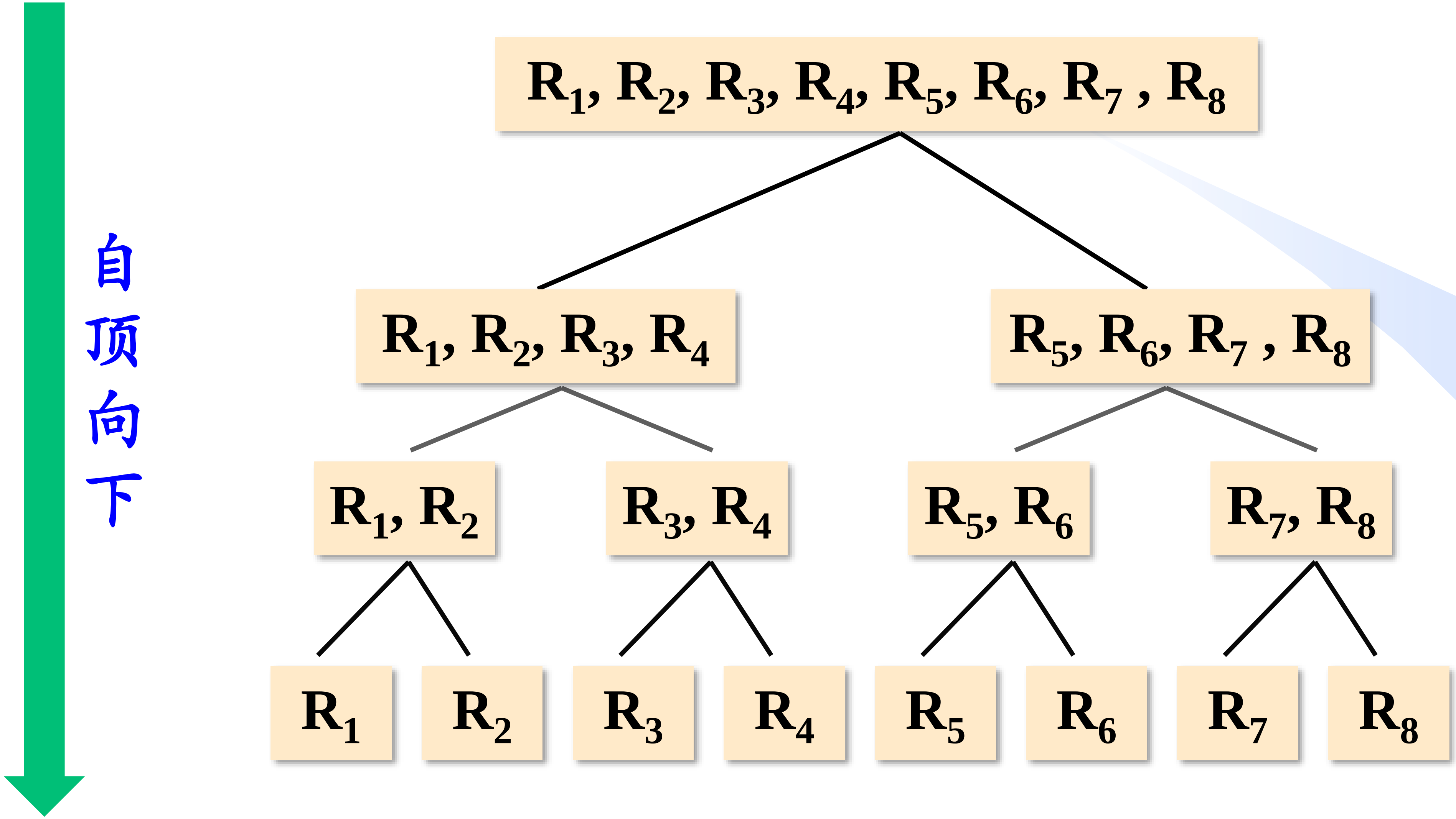
A



时间复杂度
O($n$log$n$)

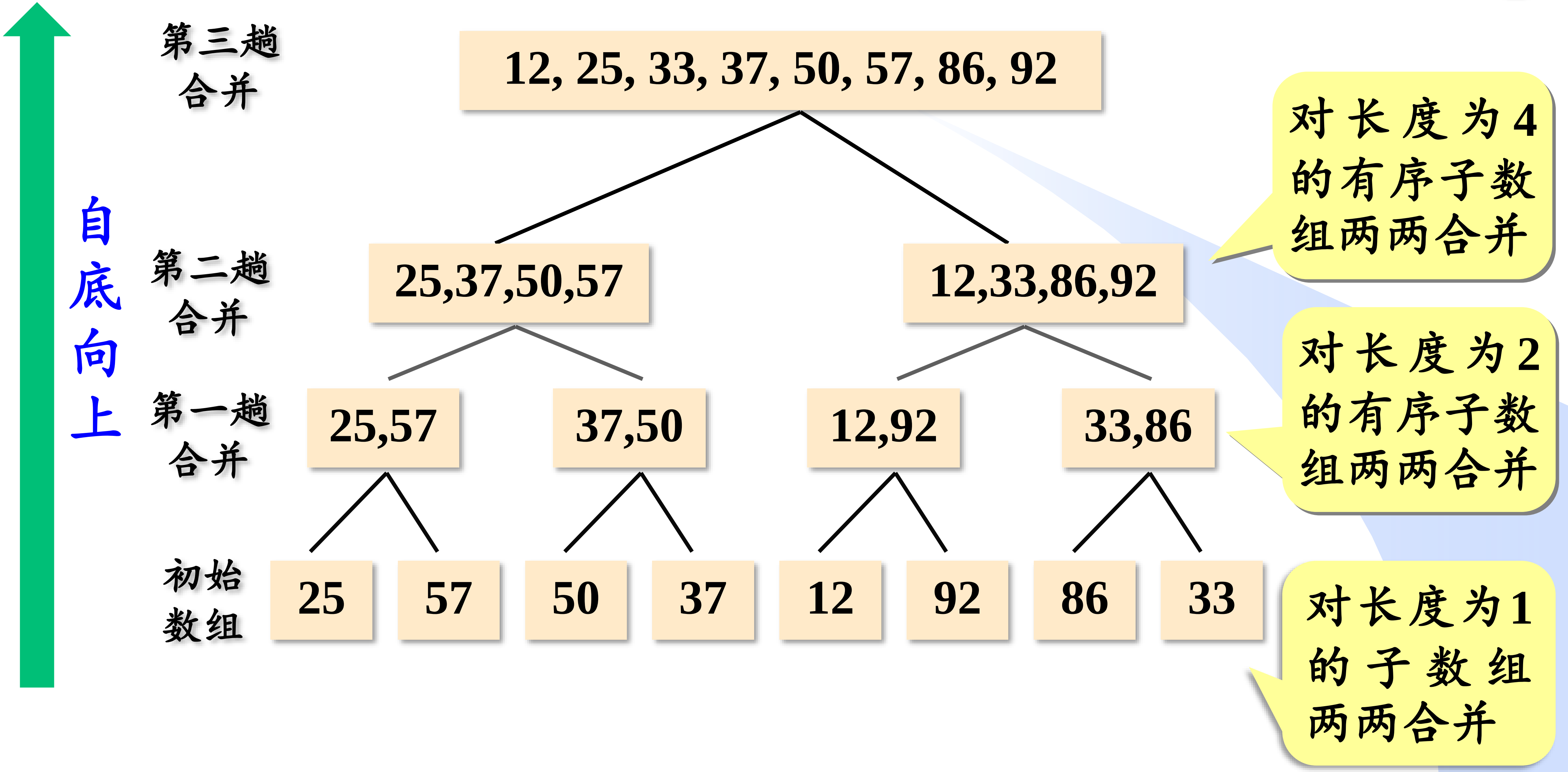
空间复杂度
O(n)

归并排序——递归过程



自顶向下
$R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8$
R_1, R_2, R_3, R_4
R_5, R_6, R_7, R_8
R_1, R_2
R_3, R_4
R_5, R_6
R_7, R_8
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8

归并排序——非递归过程

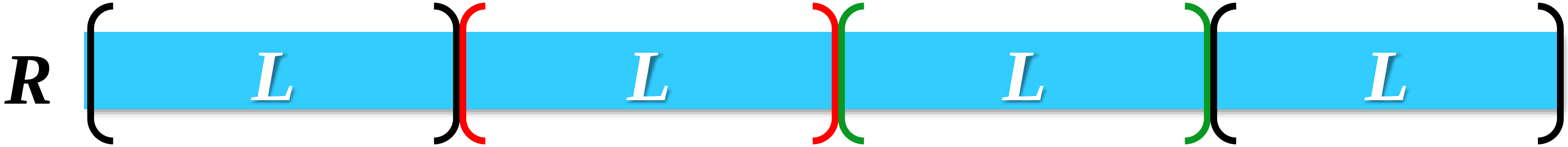


A
第三趟合并
12, 25, 33, 37, 50, 57, 86, 92
第二趟合并
25,37,50,57
12,33,86,92
第一趟合并
25,57
37,50
12,92
33,86
初始数组
25
57
50
37
12
92
86
33
自底向上
对长度为4的有序子数组两两合并
对长度为2的有序子数组两两合并
对长度为1的子数组两两合并

归并排序——非递归形式

```
void MergeSort(int R[], int n){
    for(int L=1; L<n; L*=2)
        MergePass(R, n, L);
}
```

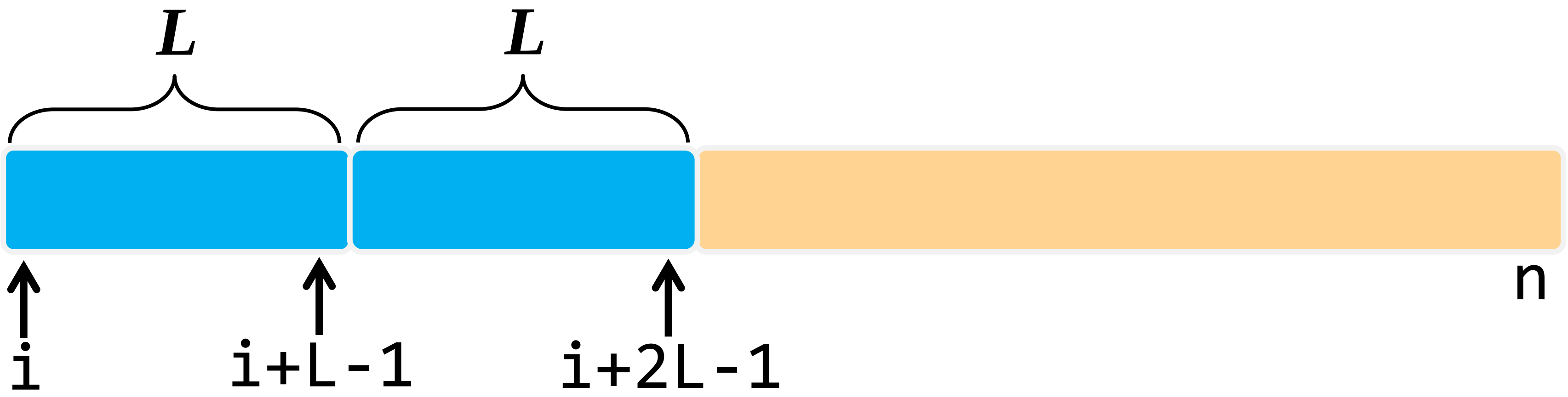
执行一趟合并过程，将数组*R*中相邻的、长度为*L*的各有序子数组依次两两合并



一趟合并

```
void MergePass(int R[], int n, int L){
   int i;  //标识合并的起点
   for(i=1; i+2*L-1<=n; i+=2*L)
      Merge(R, i, i+L-1, i+2*L-1);
```

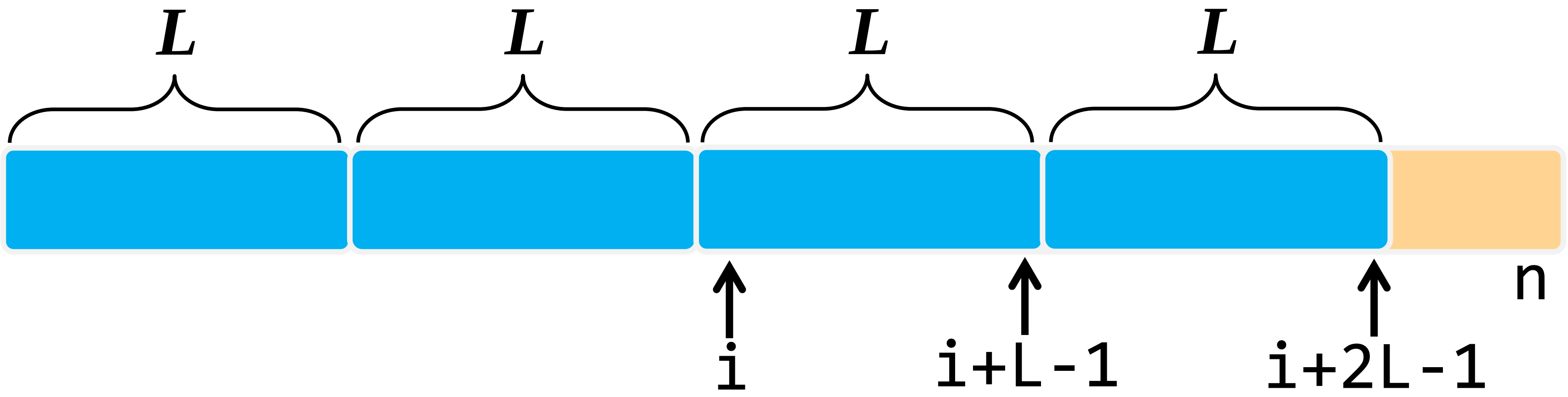
执行一趟合并过程，将数组*R*中长度为*L*的相邻有序子数组两两合并



一趟合并

```
void MergePass(int R[], int n, int L){
   int i;
   for(i=1; i+2*L-1<=n; i+=2*L)
       Merge(R, i, i+L-1, i+2*L-1);
```

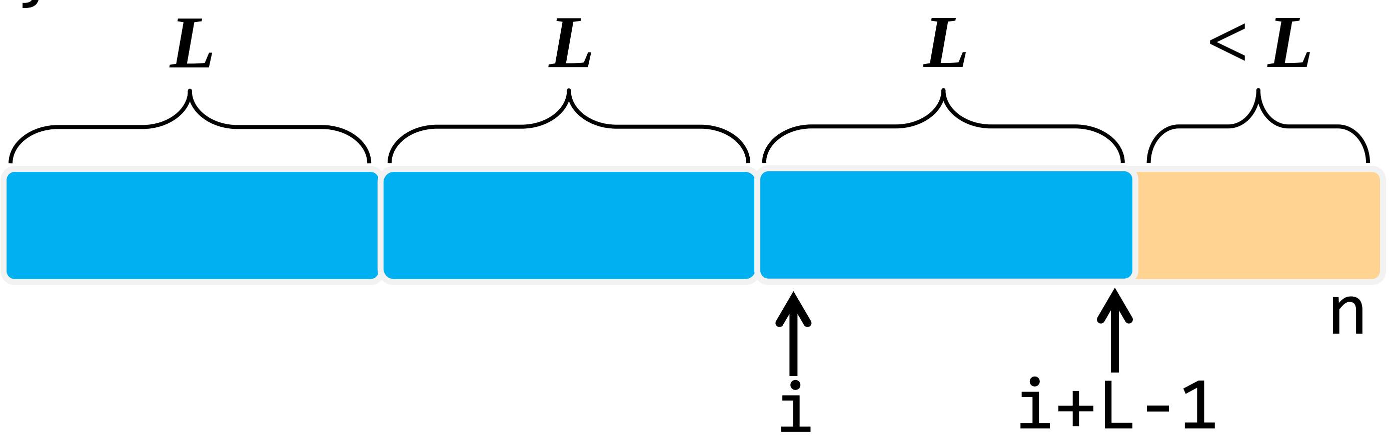
执行一趟合并过程，将数组*R*中长度为*L*的相邻有序子数组两两合并



一趟合并

```
void MergePass(int R[], int n, int L){
   int i;
   for(i=1; i+2*L-1<=n; i+=2*L)
      Merge(R, i, i+L-1, i+2*L-1);
   //处理余留的长度小于2*L的子数组
   if(i+L-1<n)
      Merge(R, i, i+L-1, n);    //L<剩余部分长度<2L
}
```

若剩余子数组长度不够2*L*，则该for循环结束



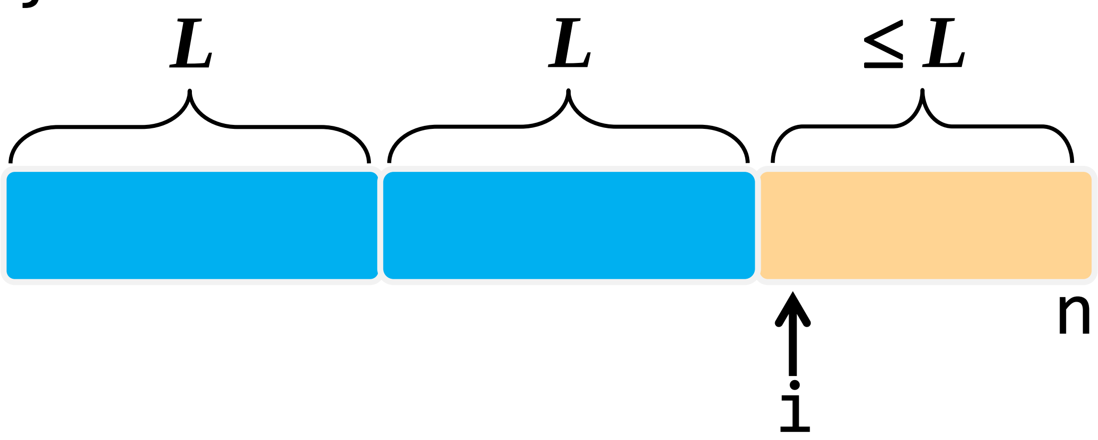
情况1

L <剩余部分< 2*L*

方案：将剩余的长度为*L*和长度<*L*的两个子数组合并

一趟合并

```
void MergePass(int R[], int n, int L){
   int i;
   for(i=1; i+2*L-1<=n; i+=2*L)
      Merge(R, i, i+L-1, i+2*L-1);
   //处理余留的长度小于2*L的子数组
   if(i+L-1<n)
      Merge(R, i, i+L-1, n);   //L<剩余部分长度<2L
}
```



时间复杂度O(n)

情况2
剩余部分≤L

方案：剩余部分已有序，故不做处理

归并排序——示例

A

21 25 49 25* 93 62 72 08 37 16 56 L=1

21 25 | 25* 49 | 62 93 | 08 72 | 16 37 | 56 L=2

21 25 25* 49 | 08 62 72 93 | 16 37 56 L=4

08 21 25 25* 49 62 72 93 | 16 37 56 L=8

08 16 21 25 25* 37 49 56 62 72 93 L=16

归并排序——非递归形式

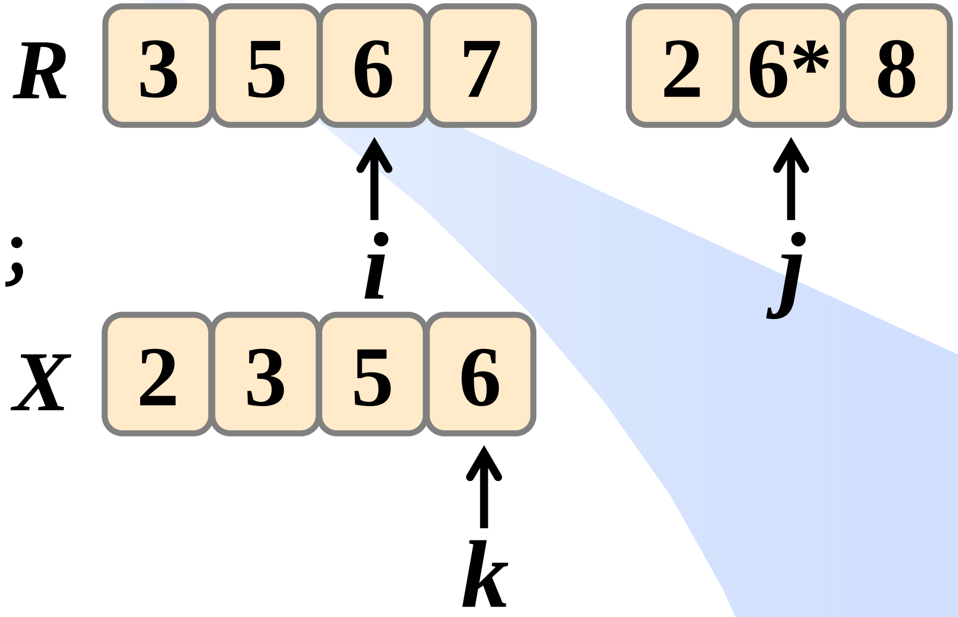
```
void MergeSort(int R[], int n){
    for(int L=1; L<n; L*=2)
        MergePass(R, n, L);
}
```

时间复杂度

O(*n*log*n*)

归并排序的稳定性

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(R_low,…,R_mid)和(R_mid+1,…,R_high)合并成一个有序数组
    int i=low, j=mid+1, k=0;
    int *X=new int[high-low+1];
    while(i<=mid  && j<=high)
        if(R[i]<=R[j]) X[k++]=R[i++];
        else X[k++]=R[j++];
    while(i<=mid) X[k++]=R[i++];
    while(j<=high) X[k++]=R[j++];
    for(i=0; i<high-low+1; i++)
        R[low+i]=X[i];
    delete []X;
}
```



- ✓若关键词相等先把i指向的元素放入X
- ✓使两个关键词相等的元素合并后左面的元素还在左面

归并排序算法总结

排序算法	时间复杂度			空间复杂度	稳定性
	最好	平均	最坏		
归并排序	O(*n*log*n*)	O(*n*log*n*)	O(*n*log*n*)	O(*n*)	稳定

最快的稳定性排序算法

归并排序优化策略

A

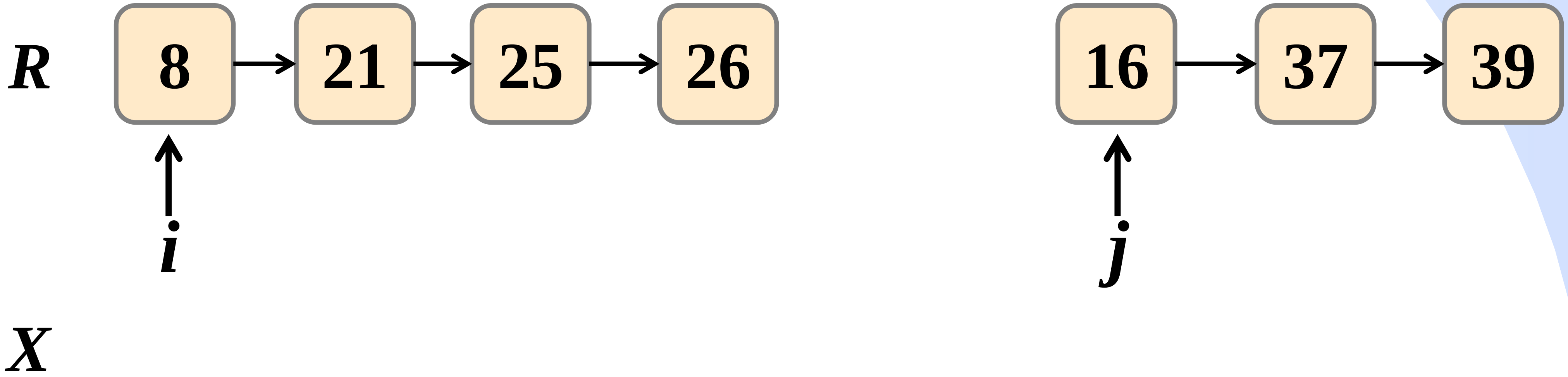
- **问题**：当数据量非常小时，若仍然采用分治策略，效率不高。
- **优化策略**：对于非常小的数据集，以及前几次合并动作，调用直接插入排序算法。

- **问题**：Merge操作基于元素移动，当元素比较大时，赋值操作会比较费时。
- **优化策略**：将数组存储改为链表存储，这样记录移动就变为指针移动了。

B

课下思考

- 对单链表进行排序，哪种排序算法最适合？【腾讯、华为、阿里、字节跳动、百度、快手、美团、谷歌、微软、苹果面试题】
- 给定一个包含哨位结点的单链表，请设计一个时空效率尽可能高效的算法，对该链表进行递增排序，*n*为链表长度。【吉林大学21级期末考试题，15分】



归并排序的经典应用——求逆序对数目

B

求数组中逆序对的个数【华为、阿里、字节跳动、百度、B站、美团、谷歌、京东、滴滴面试题 LeetCode LCR170】

8	21	5	26	10	12	72	6

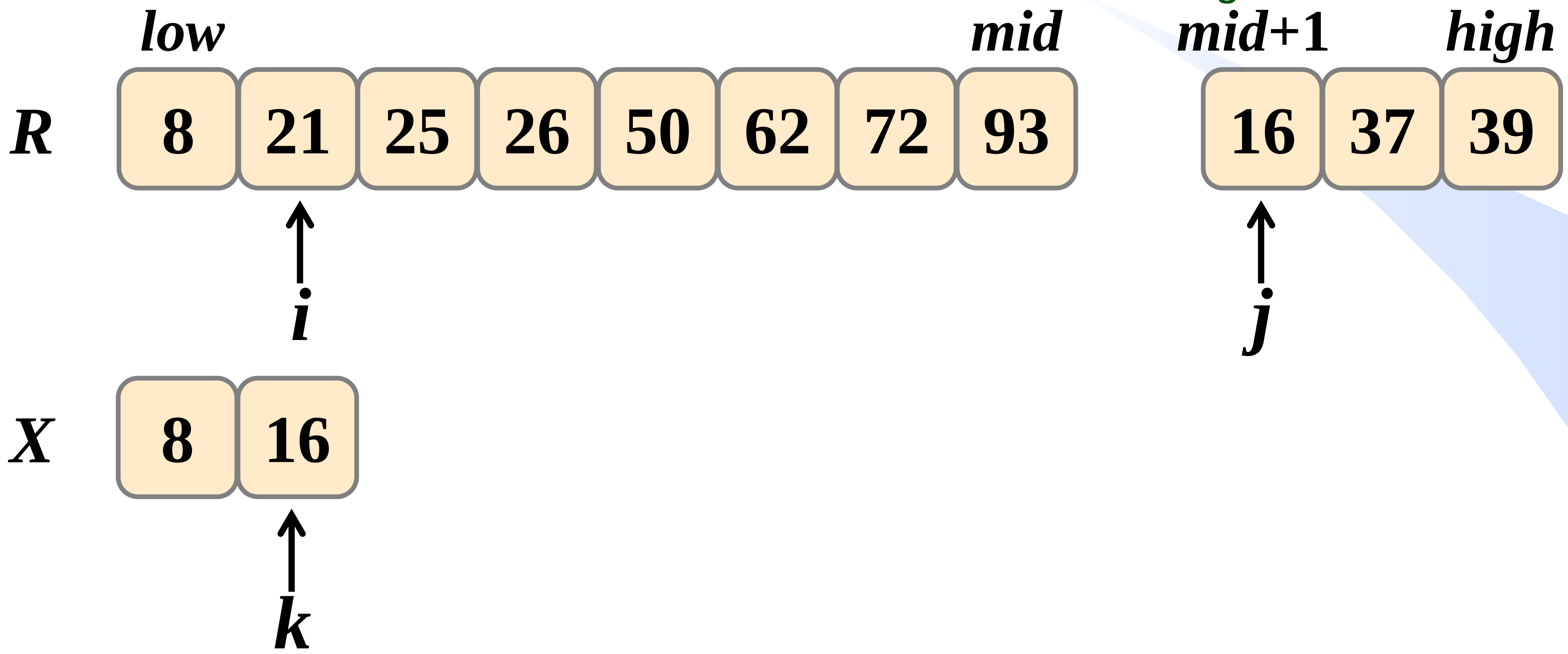
```
int cnt=0;
for(int i=1;i<=n;i++)
    for(int j=i+1;j<=n;j++)
        if(R[i]>R[j]) cnt++;
```

时间复杂度
$O(n^2)$

计算逆序对个数

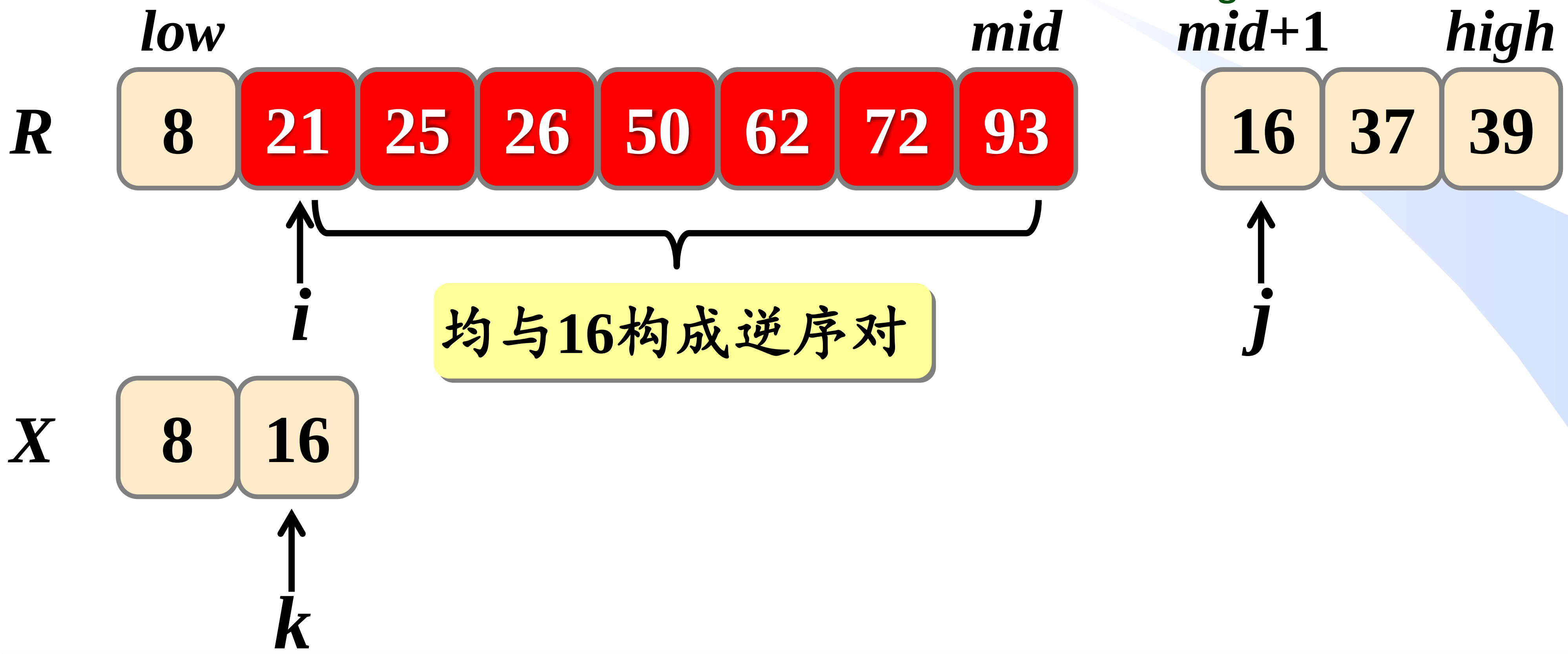
B

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(Rlow,…,Rmid)和(Rmid+1,…,Rhigh)合并成一个有序数组
```



计算逆序对个数

```
void Merge(int R[],int low, int mid, int high){
//将两个相邻的有序数组(R_low,…,R_mid)和(R_mid+1,…,R_high)合并成一个有序数组
```



R[*i*] >R[*j*] ⇔ R[*i*]…R[*mid*] >R[*j*] ⇔ 逆序对增加*mid*-*i*+1个

计算逆序对个数

B

```
int cnt=0;
void Merge(int R[],int low, int mid, int high){
    int i=low, j=mid+1, k=0;
    int *X=new int[high-low+1];
    while(i<=mid  && j<=high)
        if(R[i]<=R[j]) X[k++]=R[i++];
        else { X[k++]=R[j++]; cnt += mid-i+1;}
    while(i<=mid) X[k++]=R[i++];
    while(j<=high) X[k++]=R[j++];
    for(i=0; i<high-low+1; i++)
        R[low+i]=X[i];
    delete []X;
}
```

归并排序时，把调用**Merge**时算出来的*cnt*值累加起来

时间复杂度**O(*n*log*n*)**

基于分治法（*Divide and Conquer*）的排序算法

A

- 分治法思想：将一个输入规模为n的问题分解为多个规模较小的子问题，这些子问题互相独立且与原问题相同，然后递归地求解这些子问题，最后用适当的方法将各子问题的解合并，以获得原问题的解。
- 分治排序包括三个步骤：
  - ✓“分”：将数组划分为若干子数组，一般为二分；
  - ✓“治”：对子数组递归排序；
  - ✓“合”，将子数组排序后的结果整合在一起。

基于分治法（*Divide and Conquer*）的排序算法

A

- 由于“治”是递归调用过程，分治排序的重点在“分”和“合”两步，具体的算法往往侧重其中的某一步：
  - ✓ 快速排序：侧重在“分”（调用**Partition**算法），在“分”的过程中调整了元素位置，而“合”过程无需任何操作。
  - ✓ 归并排序：侧重在“合”（调用**Merge**算法），分的过程就是简单的对半二分。

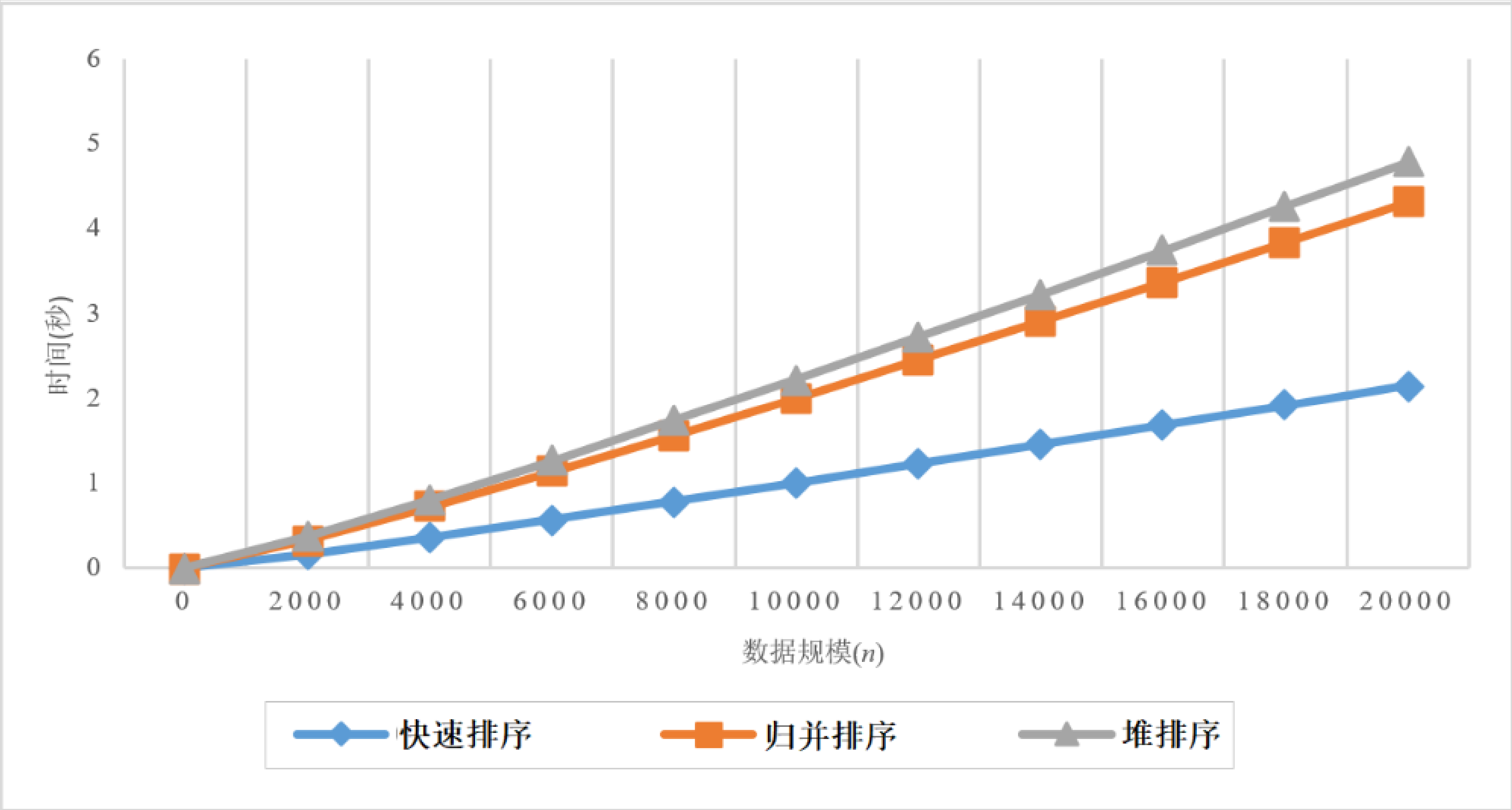
基于关键词比较的排序算法对比

A

排序算法	时间复杂度			空间复杂度	稳定性
	最好	平均	最坏		
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
直接选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	不稳定
快速排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$-$O(n)$	不稳定
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	稳定

O(*n*log*n*)排序算法实验对比

B



练习

A

现有n条词以及对应的拼音串，对其排序，排序规则：首先按拼音串的字母序排序，如果拼音串相同，则按当前词所在的顺序排序，下列排序算法中______符合条件。【搜狗校园招聘笔试题】

A. 直接插入排序　　B. 快速排序　　C. 堆排序　　D.冒泡排序

长春 *changchun*	→	北京 *beijing*	
北京 *beijing*		背景 *beijing*	
吉大 *jida*		长春 *changchun*	
数据 *shuju*		吉大 *jida*	
背景 *beijing*		数据 *shuju*	

课下思考

A

下列排序算法中，不稳定的是______。【2023年考研题全国卷】

I. 希尔排序　　II. 合并排序　　III. 快速排序

IV. 堆排序　　V. 冒泡排序

A. 仅 I 和 II　　B. 仅 II 和 V

C. 仅 I、III 和 IV　　D. 仅 III、IV 和 V

课下思考

A

下列排序方法中，每趟排序结束都至少能确定一个元素最终位置的方法是______。【考研题全国卷】

I.直接选择排序　　II.希尔排序　　III.快速排序

IV.堆排序　　V.合并排序

A. I、III、IV　　B. I、III、V

C. II、III、IV　　D. III、IV、V

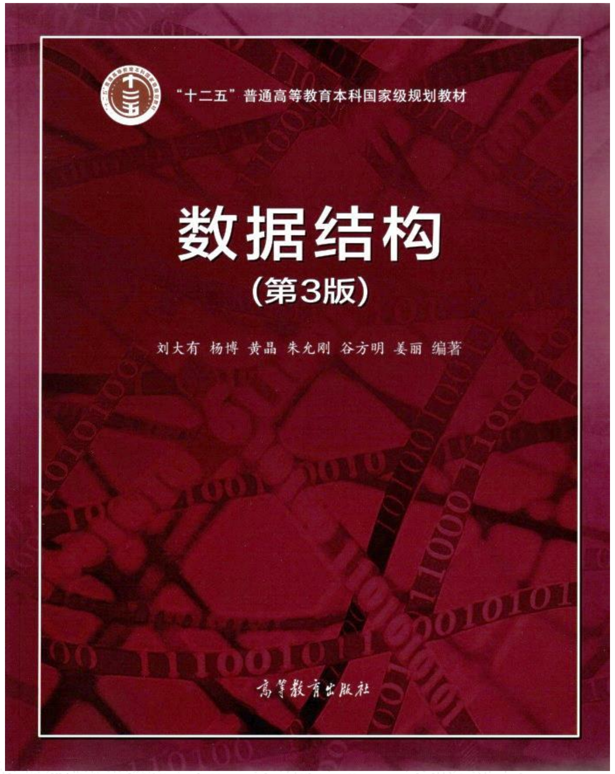
课下思考

A

下列哪个算法可能出现下列情况：在最后一趟开始之前，所有的元素都不在其最终的位置上。

A. 堆排序　　B. 冒泡排序　　C. 插入排序　　D. 快速排序

2 3 4 5 6 ← 1



归并排序及其他

- 归并排序
- **排序算法时间下界**
- 外排序方法简介

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

基于关键词比较的排序算法时间下界

B

下界 对于输入规模为n的问题，若不存在时间复杂度比T(n)小的算法，则称针对该问题的算法的时间复杂性下界为T(n)。

K_1, K_2, K_3 有6种可能排序结果。

$K_1<K_2<K_3$

$K_3<K_2<K_1$

$K_1<K_3<K_2$

$K_3<K_1<K_2$

$K_2<K_1<K_3$

$K_2<K_3<K_1$

基于关键词比较的排序算法时间下界

B

$K_1 : K_2$

$K_1<K_2<K_3$

$K_3<K_2<K_1$

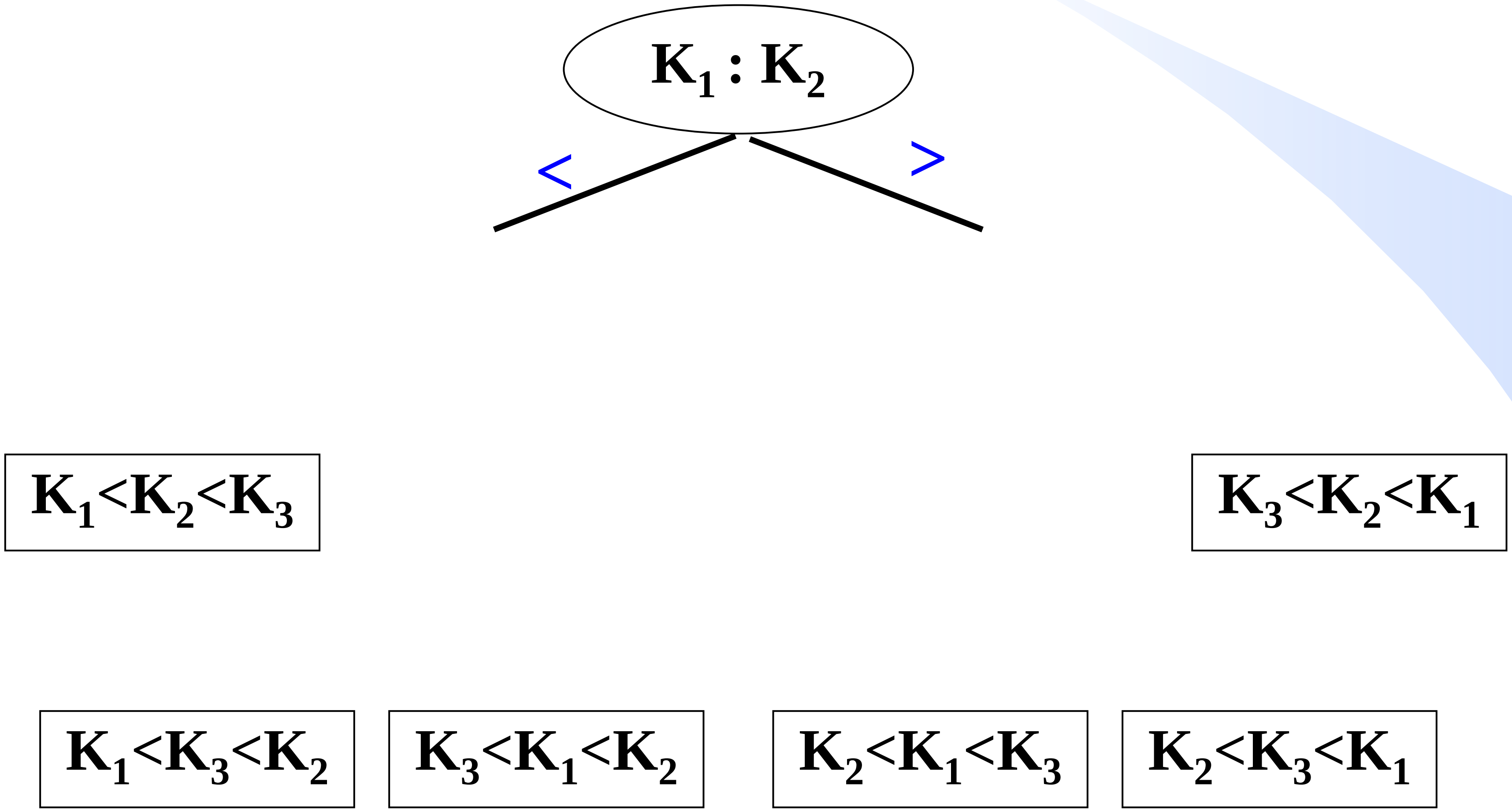
$K_1<K_3<K_2$

$K_3<K_1<K_2$

$K_2<K_1<K_3$

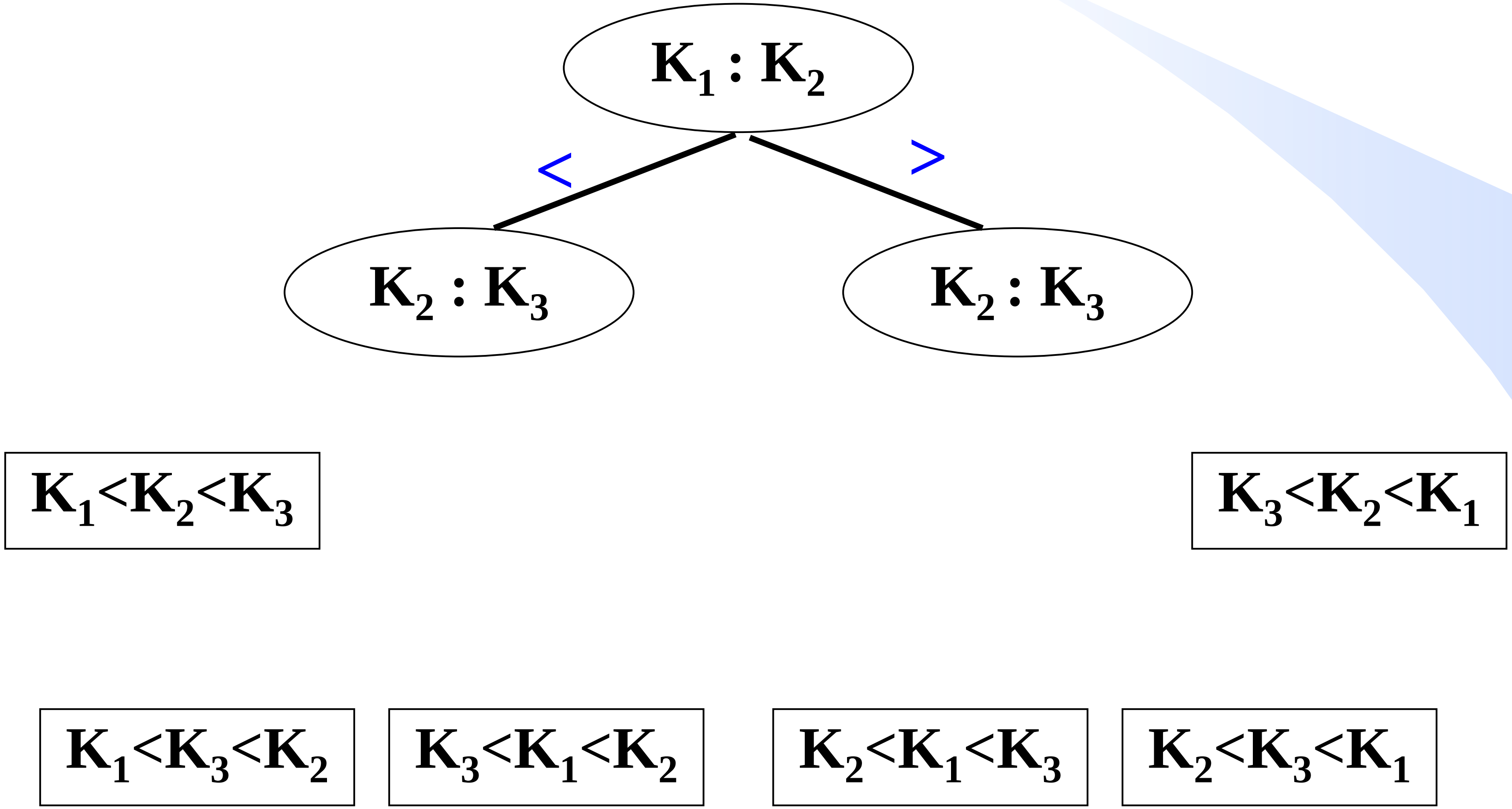
$K_2<K_3<K_1$

基于关键词比较的排序算法时间下界



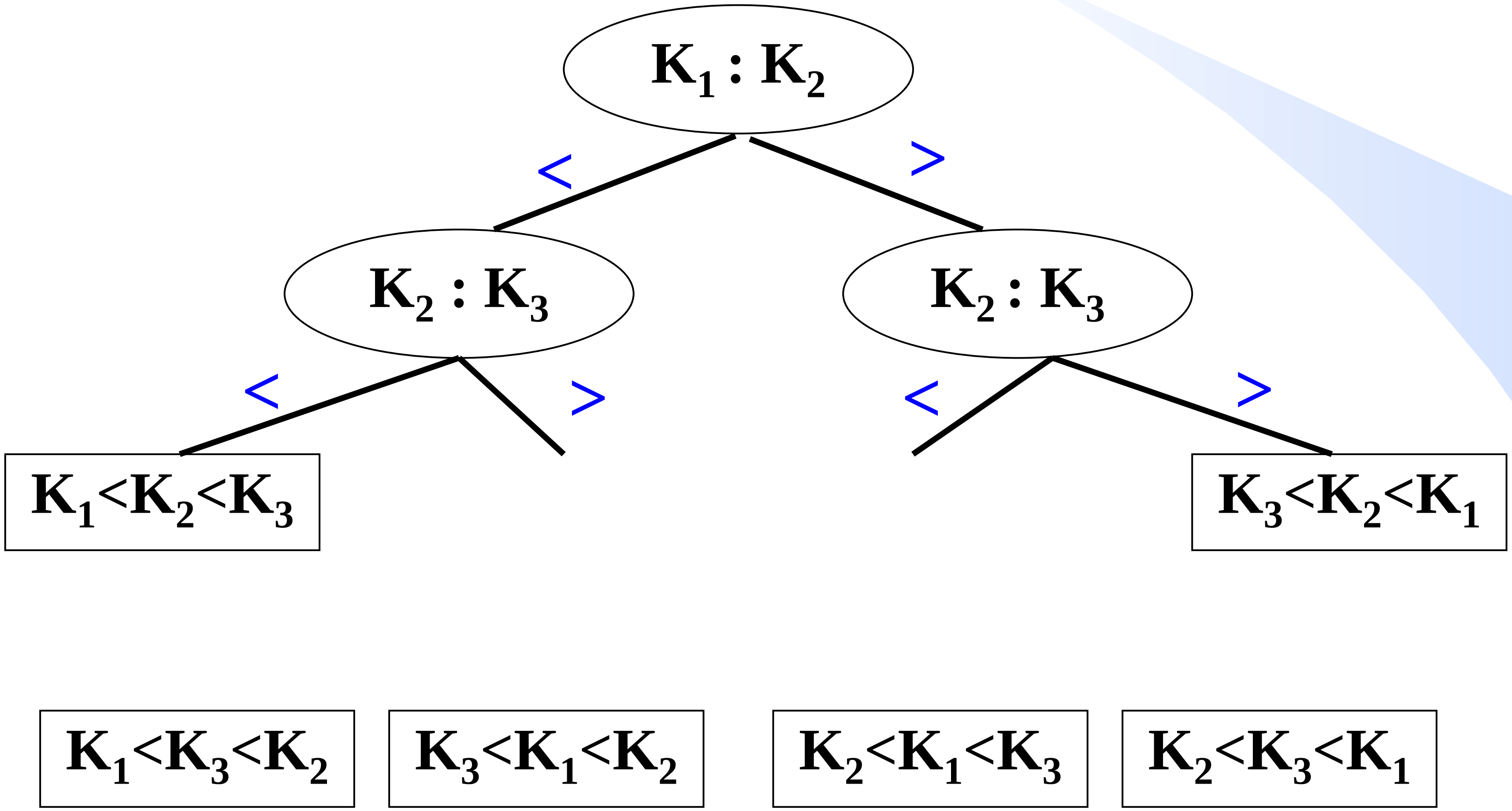
基于关键词比较的排序算法时间下界

B



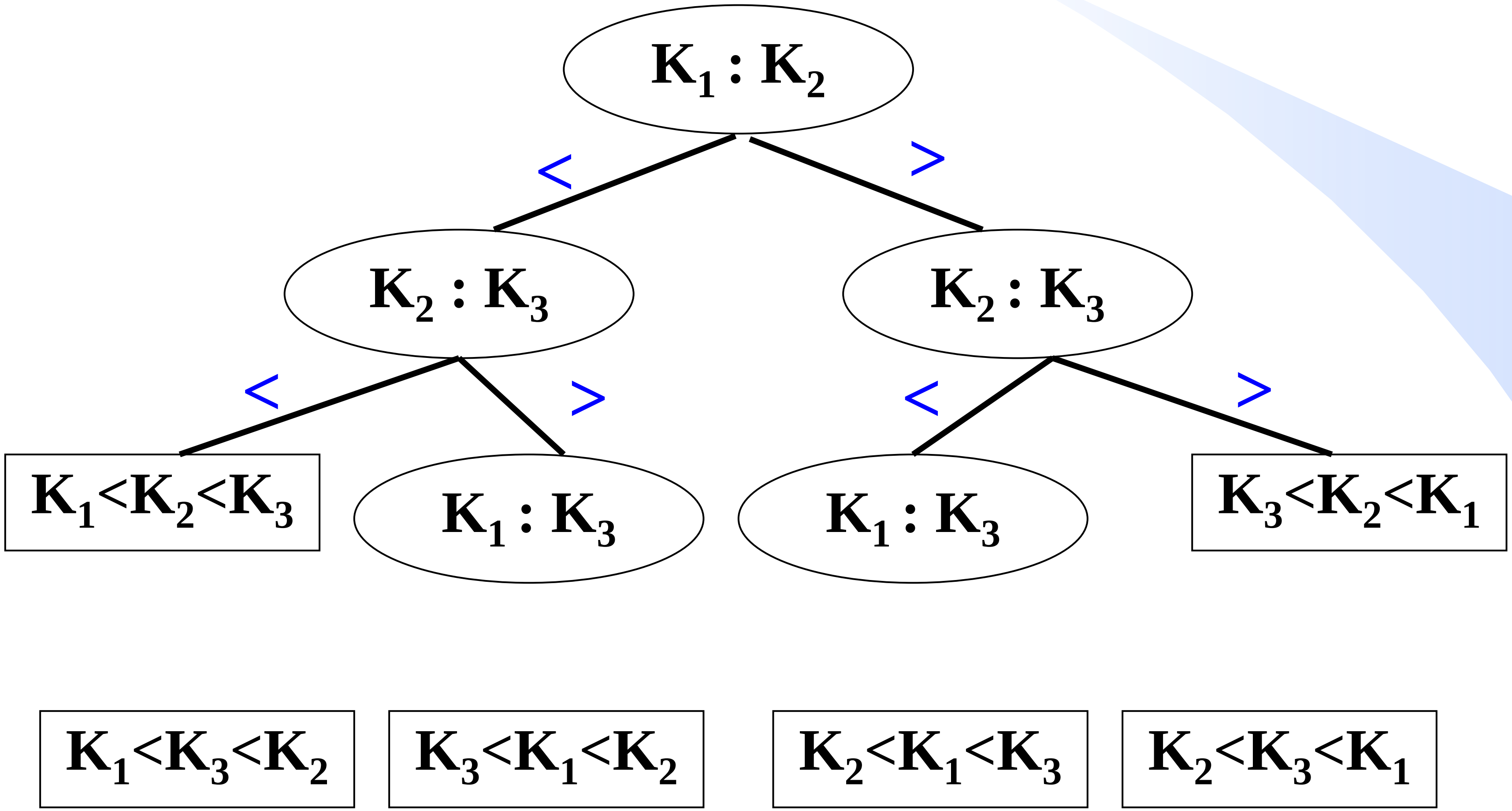
基于关键词比较的排序算法时间下界

B



基于关键词比较的排序算法时间下界

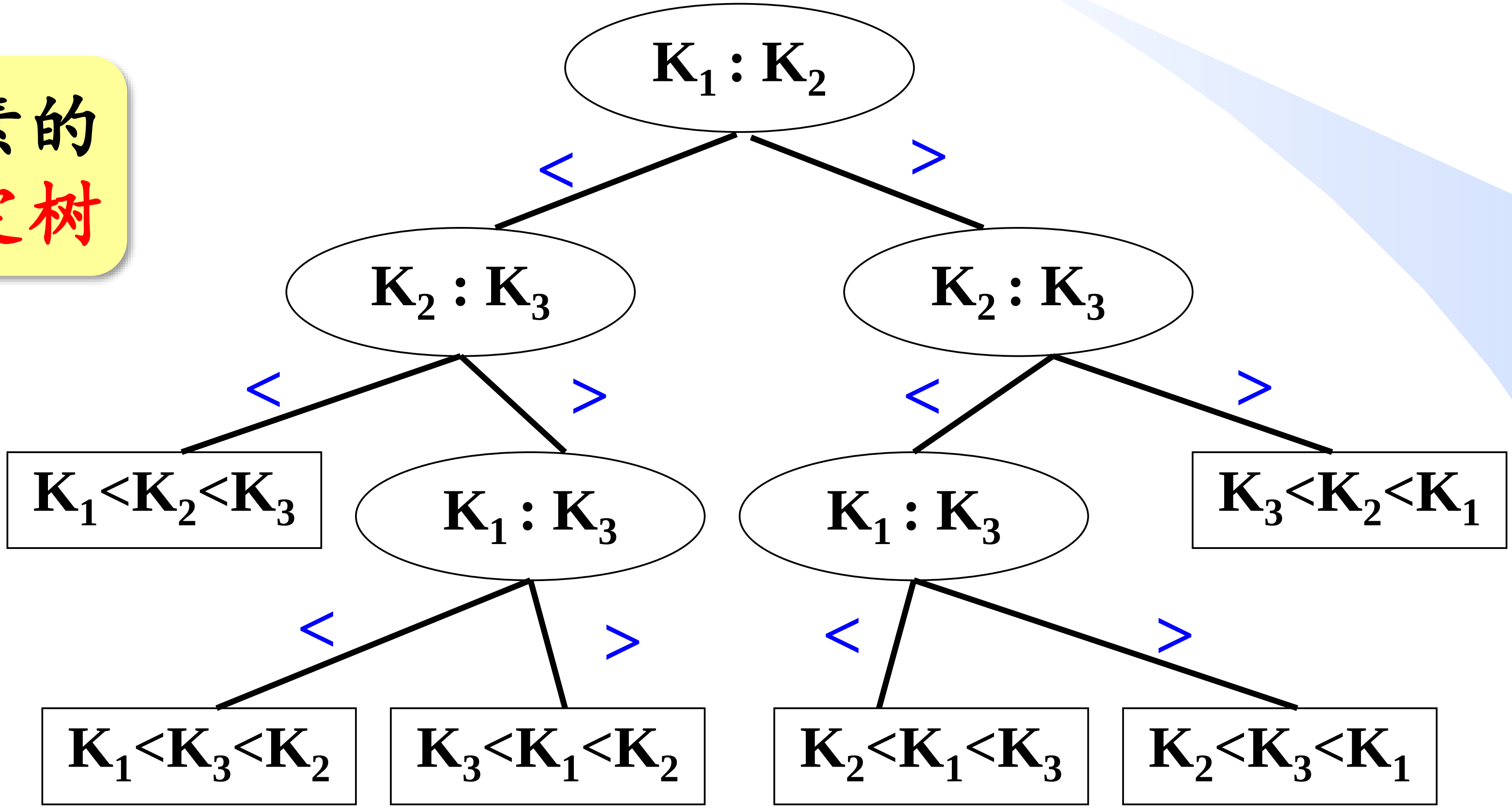
B



基于关键词比较的排序算法时间下界

B

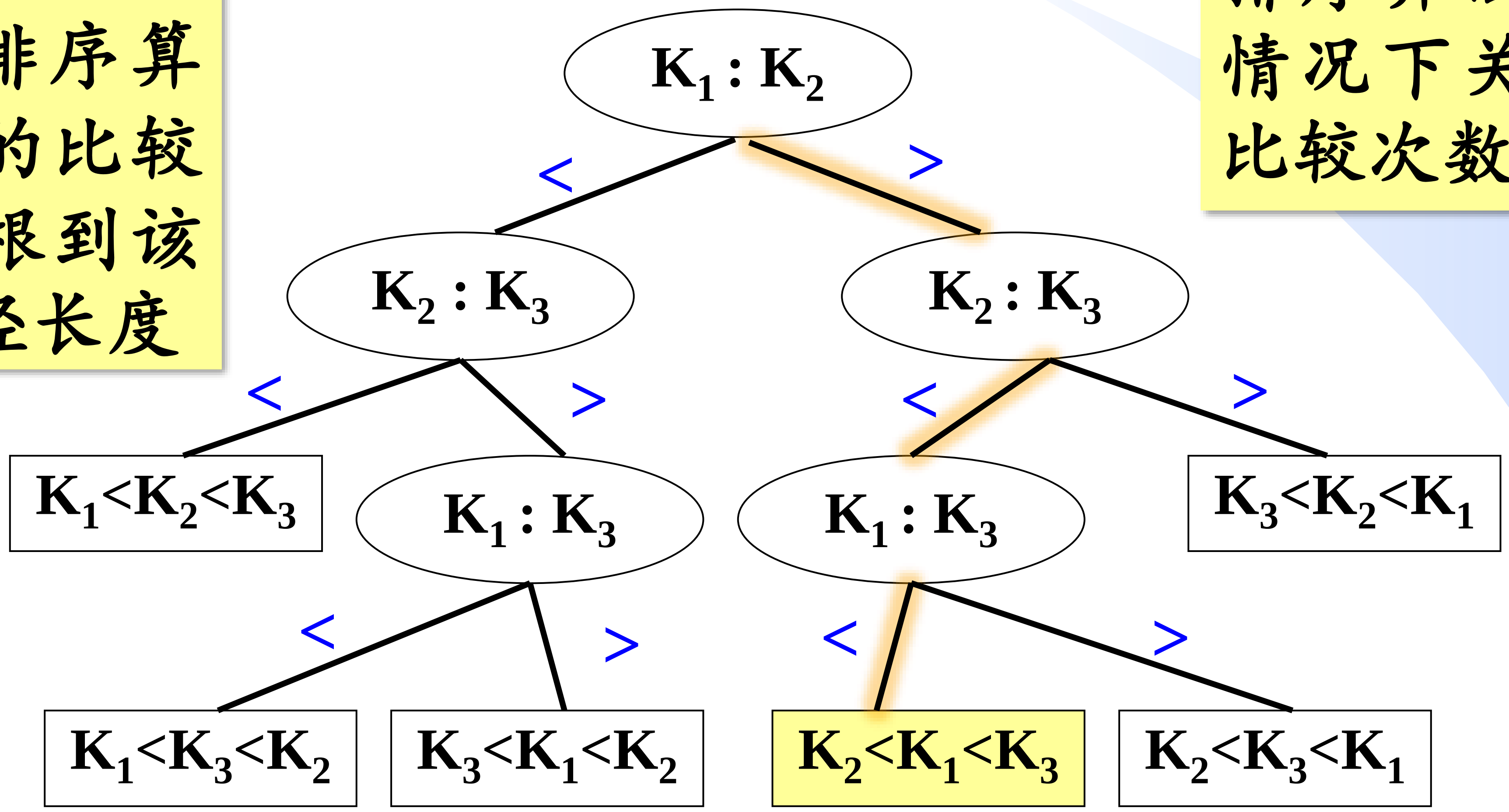
三个元素的排序判定树



基于关键词比较的排序算法时间下界

B

每种排列看成一种输入，对于每种输入，排序算法所执行的比较次数为从根到该排列的路径长度



判定树的高度是排序算法在最坏情况下关键词的比较次数。

基于关键词比较的排序算法时间下界

B

- n个元素的排列数为$n!$，故n个元素的判定树有$n!$个叶结点；
- 假定判定树高度为h，因高度为h的二叉树最多有2^h个叶结点，即叶结点个数小于等于2^h，又当前实际的叶结点个数为$n!$，故有： $n! \le 2^h$

即 $h \ge \log_2(n!)$，

$$\log(n!) = \log n + \log(n-1) + \log(n-2) + \cdots + \log 2 + \log 1$$

$$\ge \log n + \log(n-1) + \log(n-2) + \ldots + \log(n/2)$$

$$\ge \frac{n}{2}\log\frac{n}{2}$$

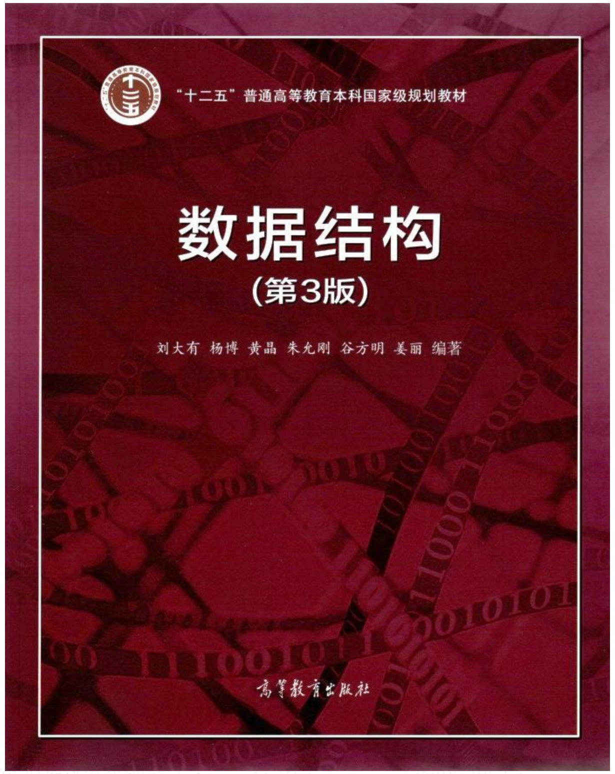
$$= \Omega(n\log n)$$

✓ 基于比较的排序算法的最坏时间复杂度下界是Ω(nlogn)。

✓ 亦可证明基于比较的排序算法的平均时间复杂度下界是Ω(nlogn)。

归并排序及其他

- 归并排序
- 排序算法时间下界
- 外排序方法简介



数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

外排序原理简介

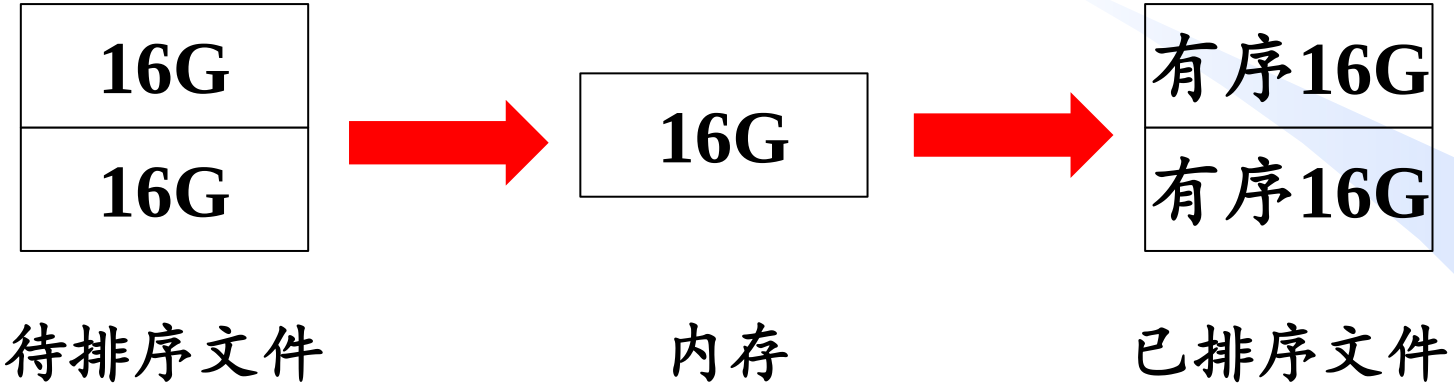
32G

待排序文件

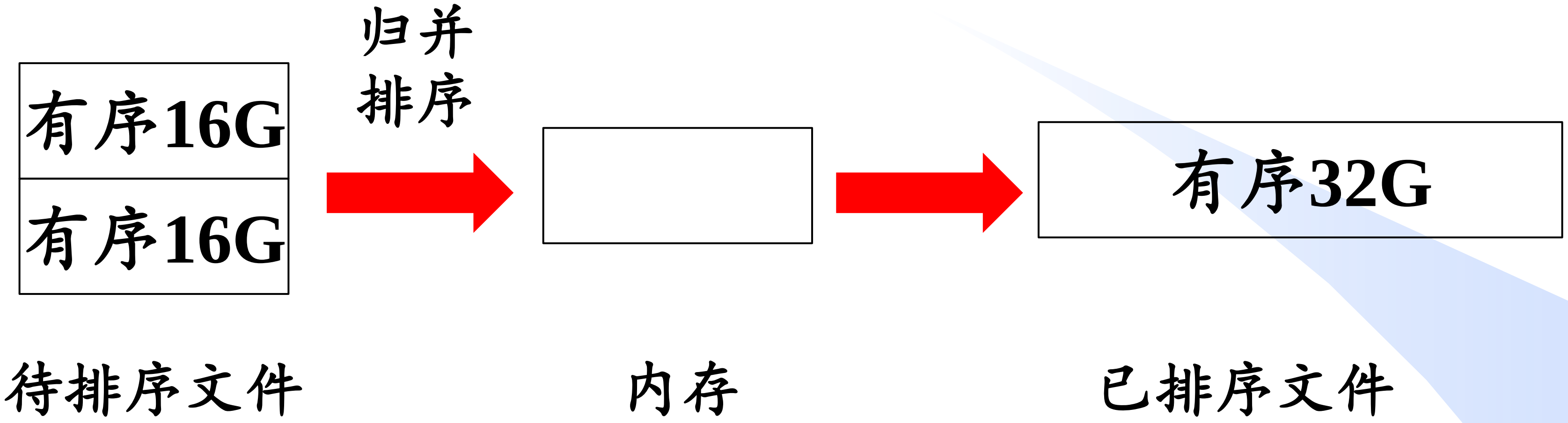
16G

内存

外排序原理简介



外排序原理简介



C

练习

对10TB的数据文件进行排序，应使用的方法是_____。【考研题全国卷】

A.希尔排序　　B.堆排序

C.快速排序　　D.归并排序