数据结构

（C语言版）（第2版）

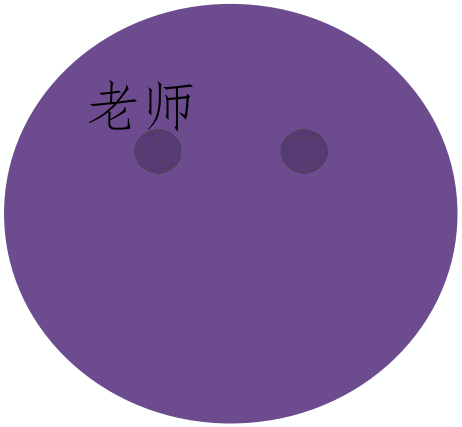
排序

交换排序

主讲教师：汪红松

教 学 内 容 Contents

一、串的定义
二、串的存储结构
三、串的模式匹配BF算法



交换排序

基本思想：

两两比较，如果发生逆序则交换，直到所有记录都排好序为止。

起泡排序$O(n^2)$

快速排序$O(\ n\log_2{}^n\ )$

一、起泡排序

基本思想：

每趟不断将记录两两比较，并按“前小后大” 规则交换

21，25，49， 25*，16， 08
21，25，25*，16， 08 ， 49
21，25， 16， 08 ，25*，49
21，16， 08 ，25， 25*，49
16，08 ，21， 25， 25*，49
08，16， 21， 25， 25*，49

一、起泡排序

优点：

每趟结束时，不仅能挤出一个最大值到最后面位置，还能同时部分理顺其他元素；

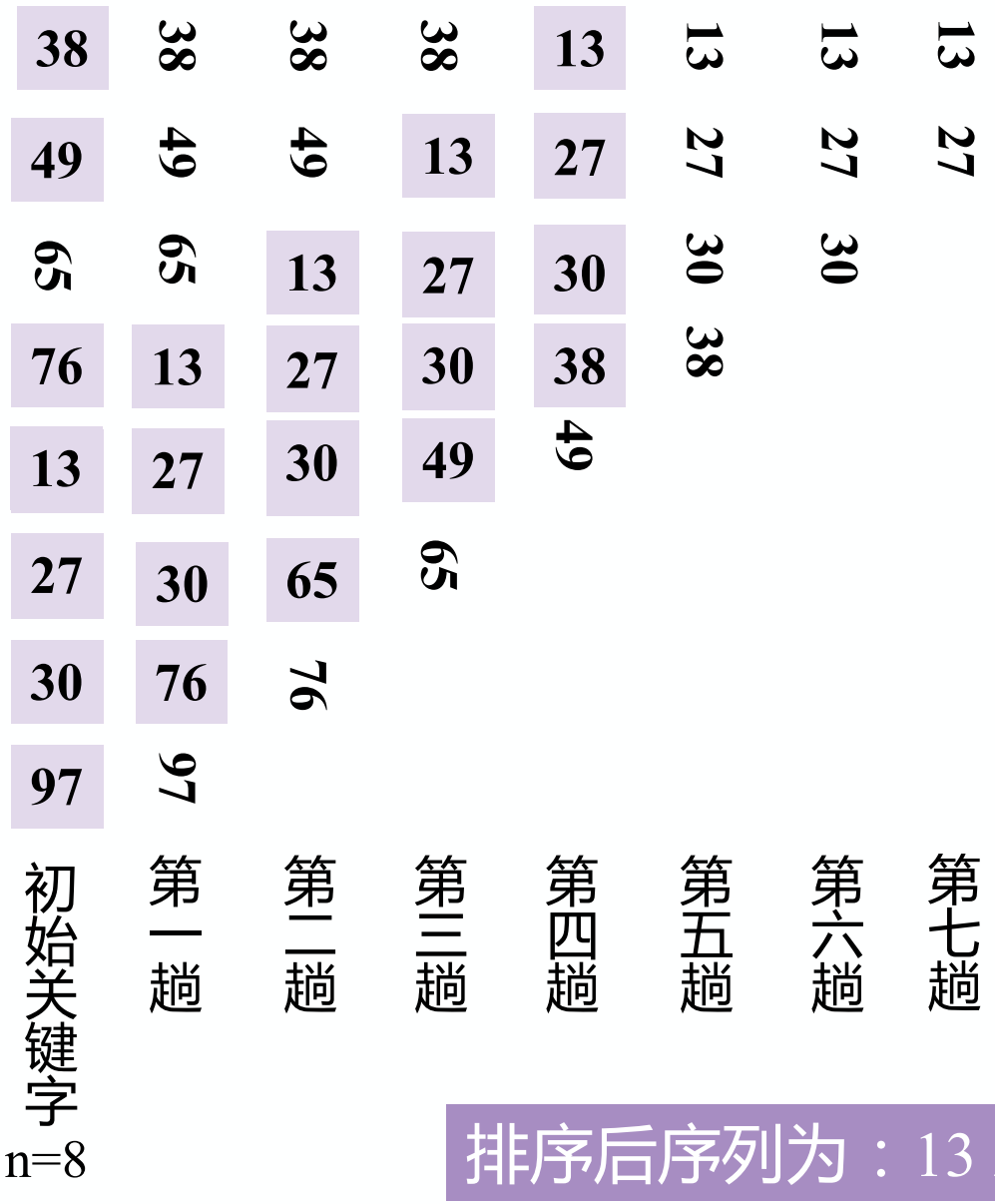
一旦下趟没有交换，还可提前结束排序。

一、起泡排序

```
void main()
{   int a[11];                    /*a[0]不用，之用a[1]~a[10]*/
    int i,j,t;
    printf("\nInput 10 numbers: \n");
    for(i=1;i<=10;i++)     scanf("%d",&a[i]);     printf("\n");
    for(j=1;j<=9;j++)
        for(i=1;i<=10-j;i++)
          if(a[i]>a[i+1])       {t=a[i];a[i]=a[i+1];a[i+1]=t;}//交换
    for(i=1;i<=10;i++)     printf("%d ",a[i]);
}
```

一、起泡排序

例



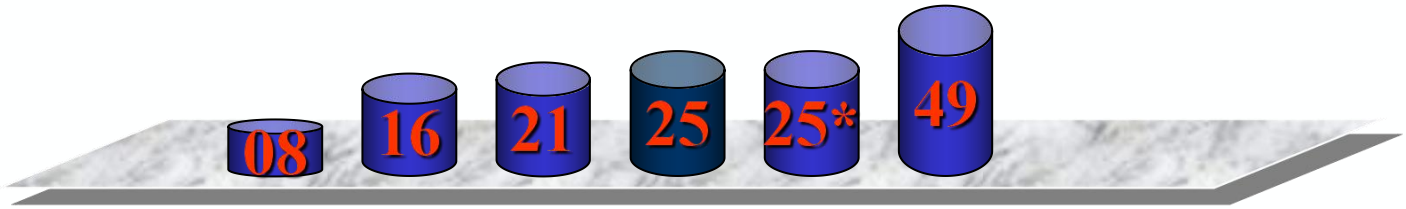
排序后序列为：13 27 30 38 49 65 76 97

一、起泡排序

```
void bubble_sort(SqList &L)
{ int m,i,j,flag=1;   RedType x;
  m=n-1;
  while((m>0)&&(flag==1))
  {  flag=0;
     for(j=1;j<=m;j++)
        if(L.r[j].key>L.r[j+1].key)
         {  flag=1;
            x=L.r[j];L.r[j]=L.r[j+1];L.r[j+1]=x; //交换
         }//endif
     m--;
  }//endwhile
}
```

一、起泡排序

1.算法分析



- 设对象个数为n
- 比较次数和移动次数与初始排列有关

最好情况下：

只需 1趟排序，比较次数为 n-1 ，不移动。

```
while((m>0)&&(flag==1))
   {  flag=0;
      for(j=1;j<=m;j++)
        if(L.r[j].key>L.r[j+1].key)
          {  flag=1;  x=L.r[j];L.r[j]=L.r[j+1];L.r[j+1]=x; }
            ……
```

一、起泡排序

1.算法分析

最坏情况下：

需 n-1趟排序，第i趟比较n-i次，移动3(n-i)次

$$\sum_{i=1}^{n-1}(n-i)=\frac{1}{2}(n^2-n)$$

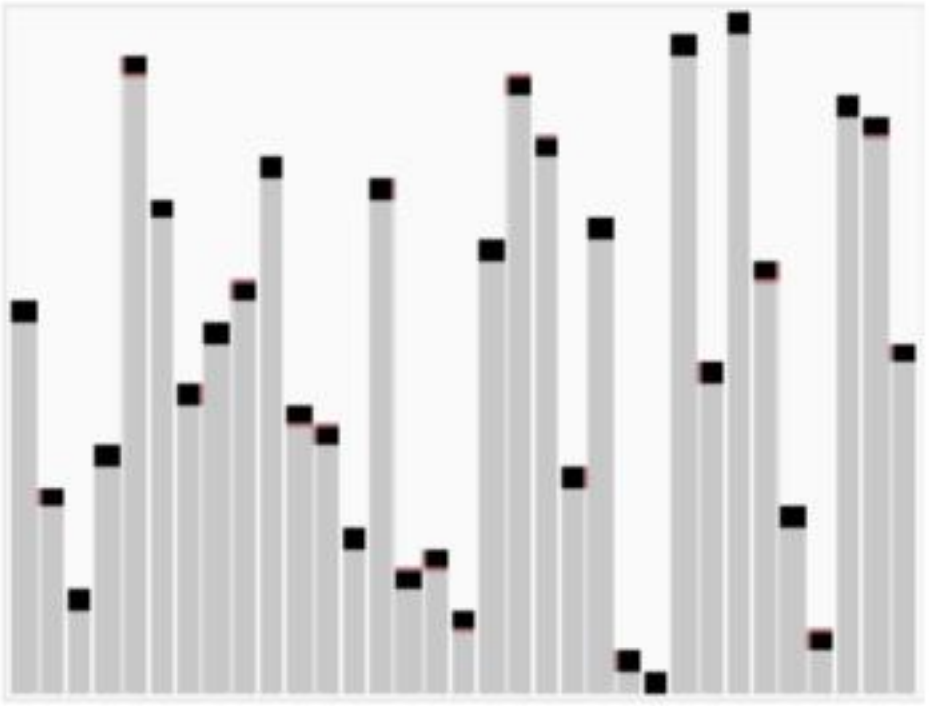
$$3\sum_{i=1}^{n}(n-i)=\frac{3}{2}(n^2-n)$$

- 时间复杂度为 o(n^2)
- 空间复杂度为 o(*1*)
- 是一种稳定的排序方法。

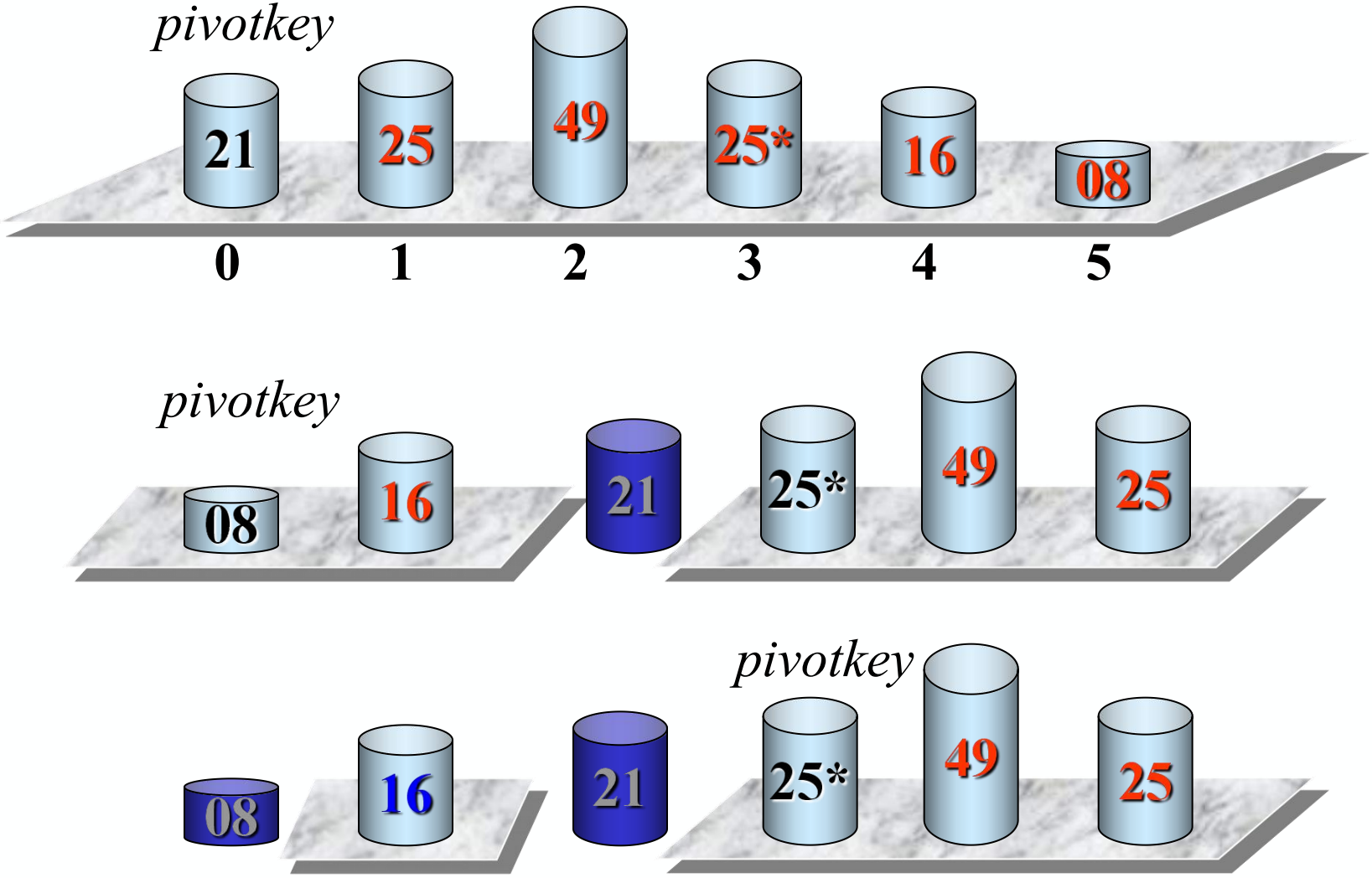
二、快速排序

基本思想：

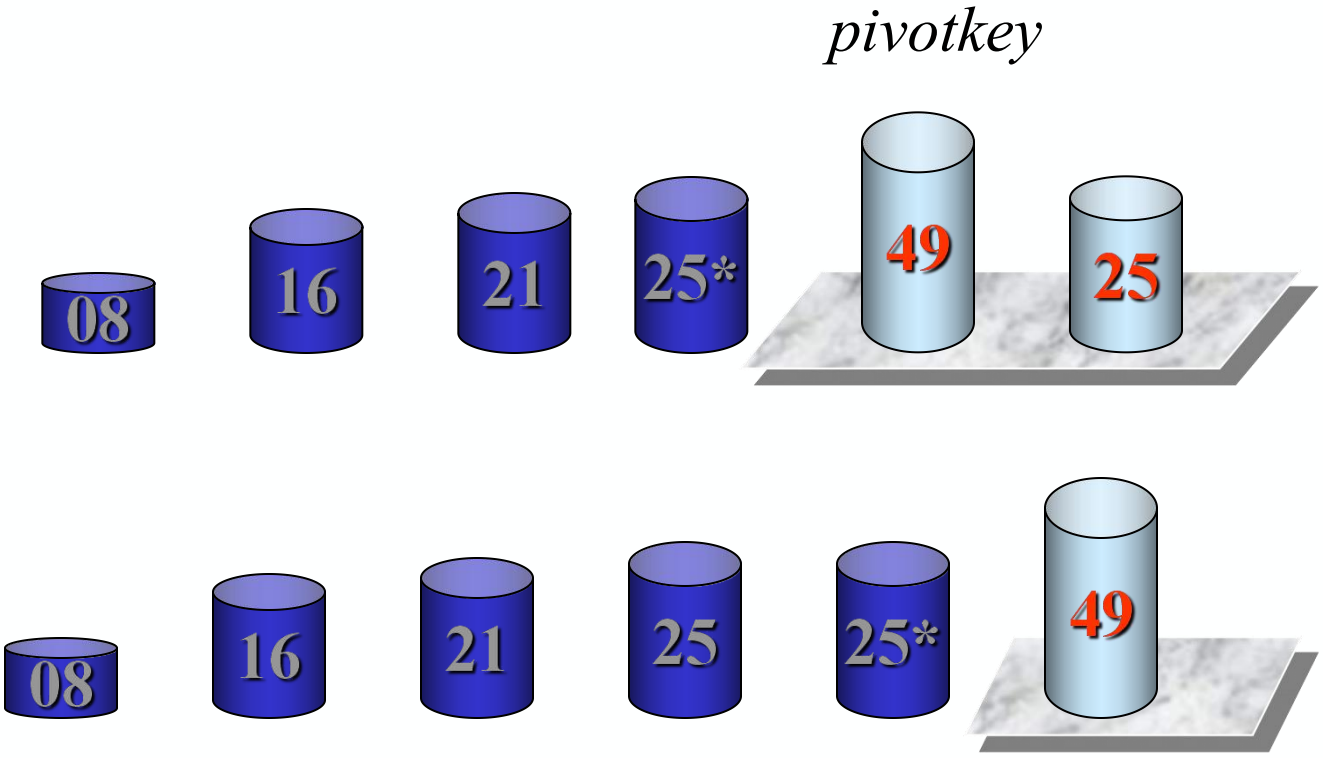
- 任取一个元素 (如第一个) 为中心；
- 所有比它小的元素一律前放，比它大的元素一律后放，形成左右两个子表；
- 对各子表重新选择中心元素并依此规则调整，直到每个子表的元素只剩一个。



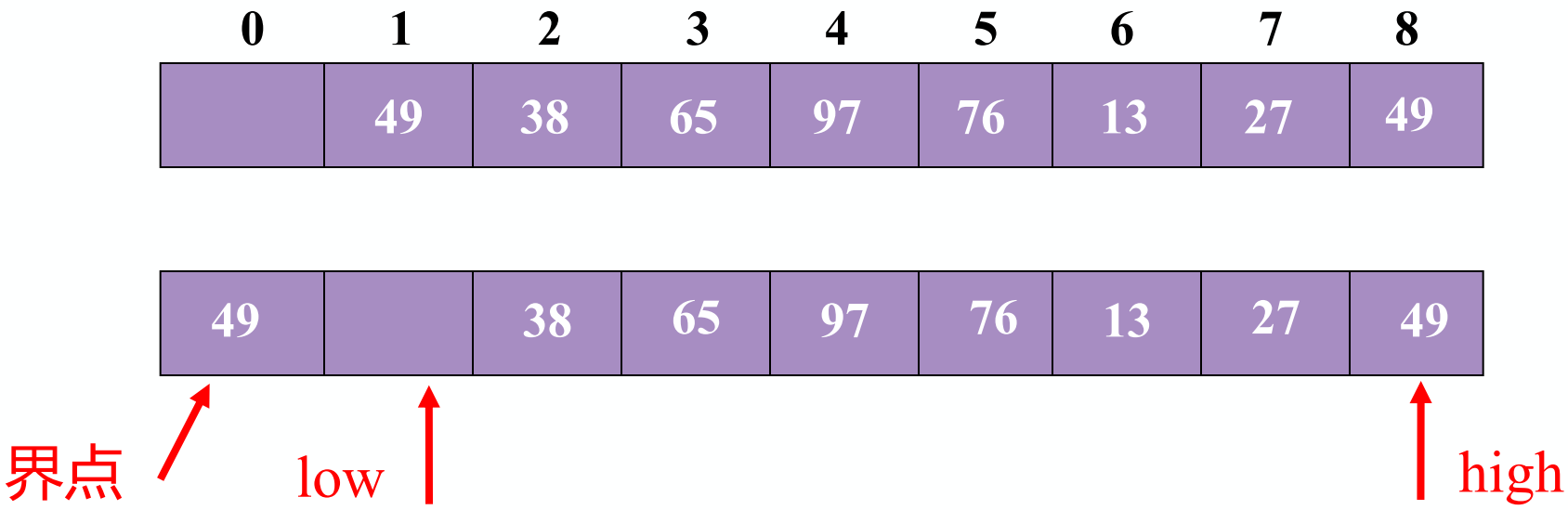
二、快速排序


pivotkey
21
25
49
25*
16
08
0
1
2
3
4
5
pivotkey
08
16
21
25*
49
25
pivotkey
08
16
21
25*
49
25

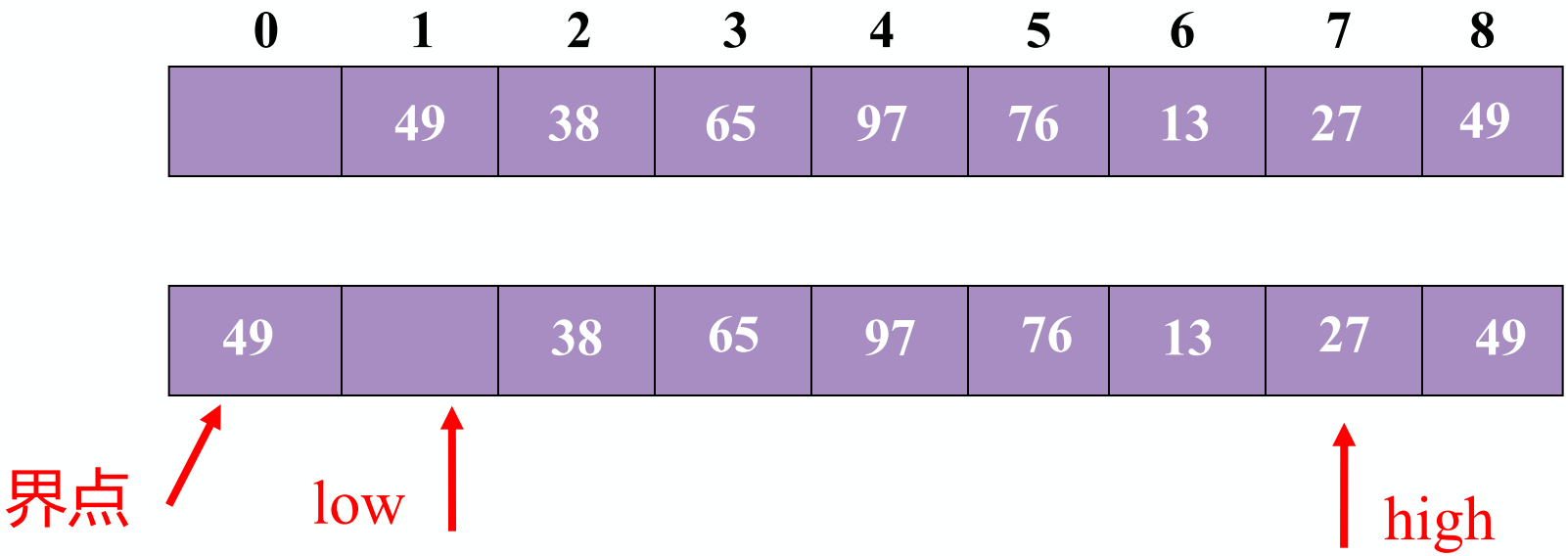
二、快速排序



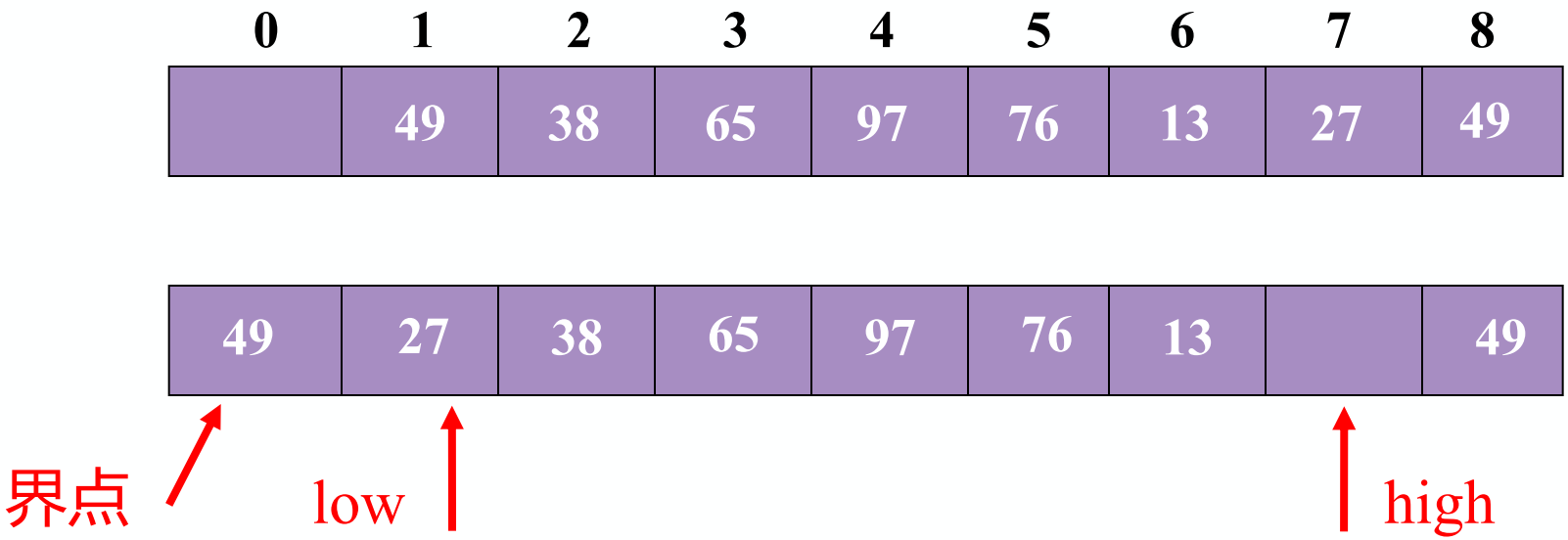
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
38
65
97
76
13
27
49
界点
low
high

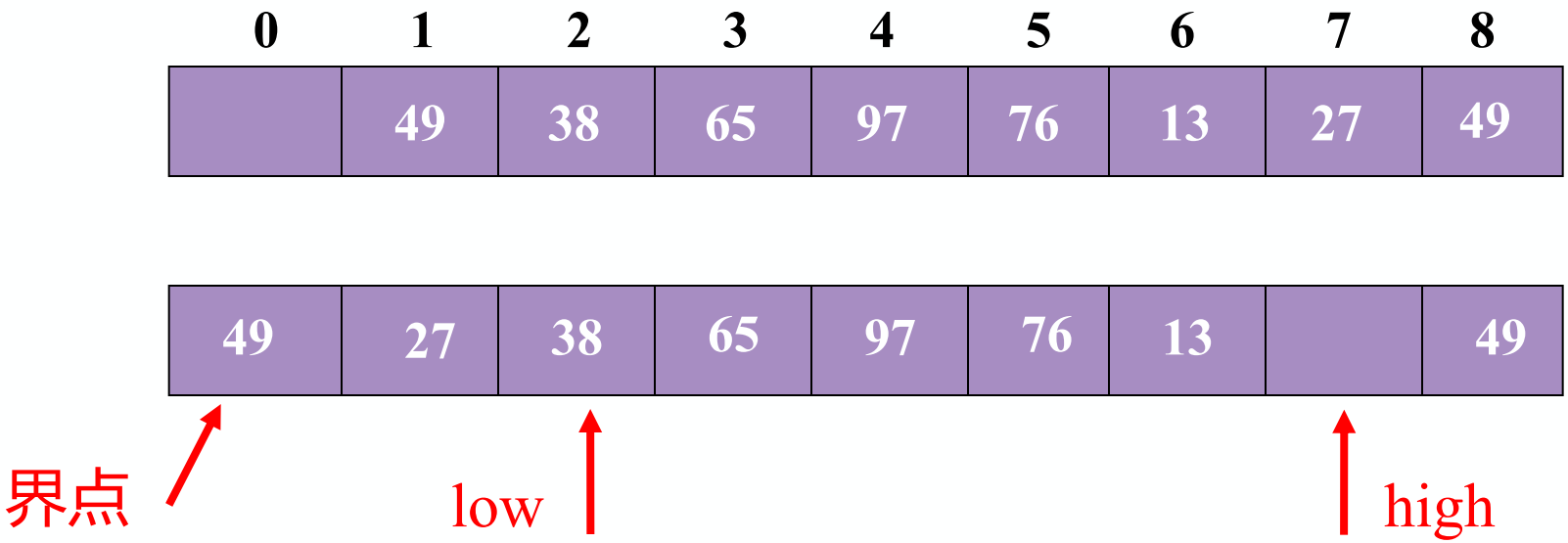
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
38
65
97
76
13
27
49
界点
low
high

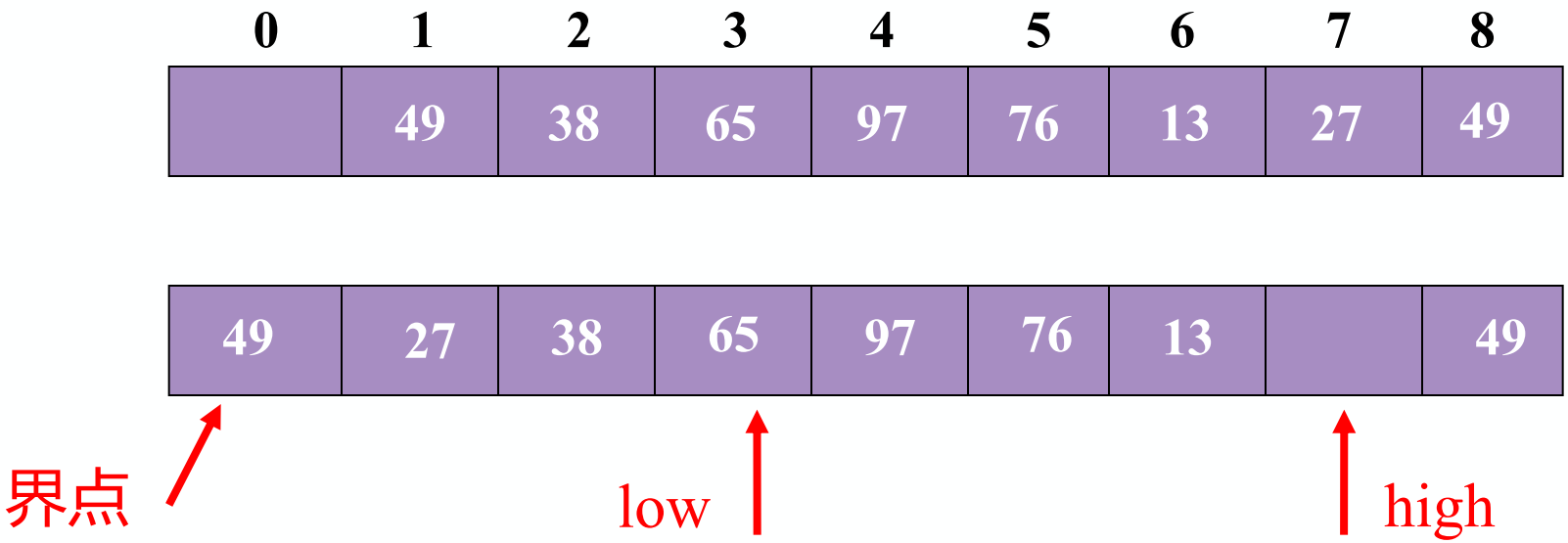
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
27
38
65
97
76
13
49
界点
low
high

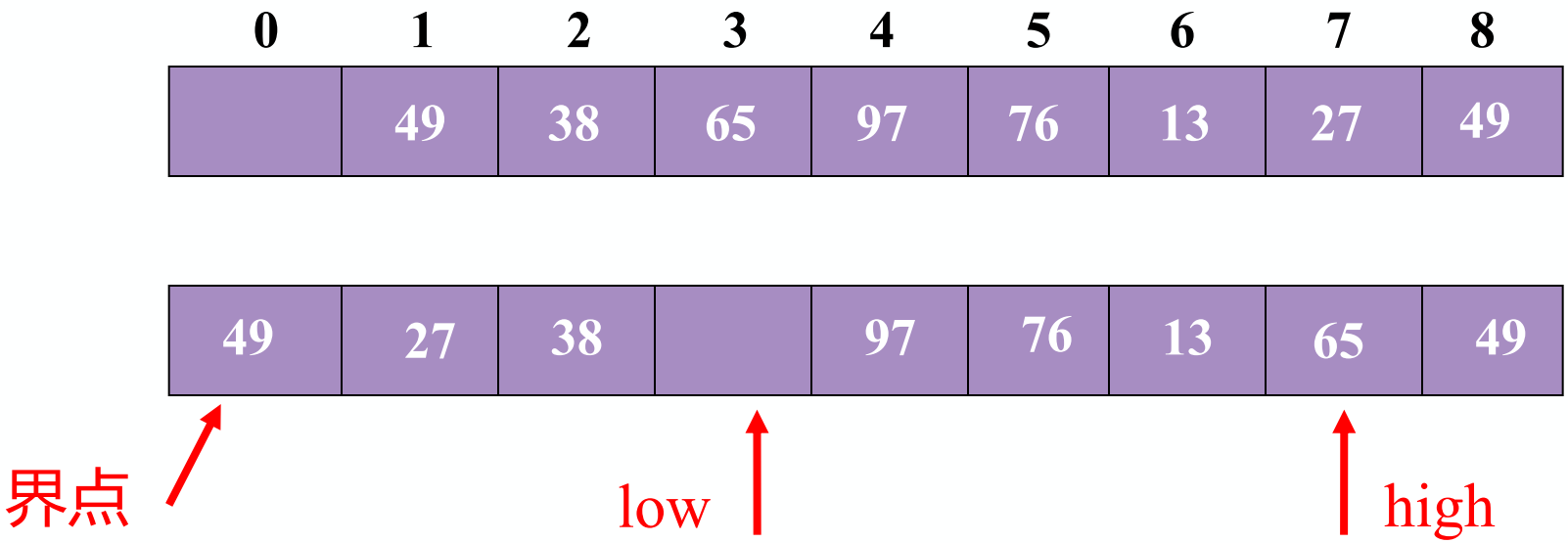
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
27
38
65
97
76
13
49
界点
low
high

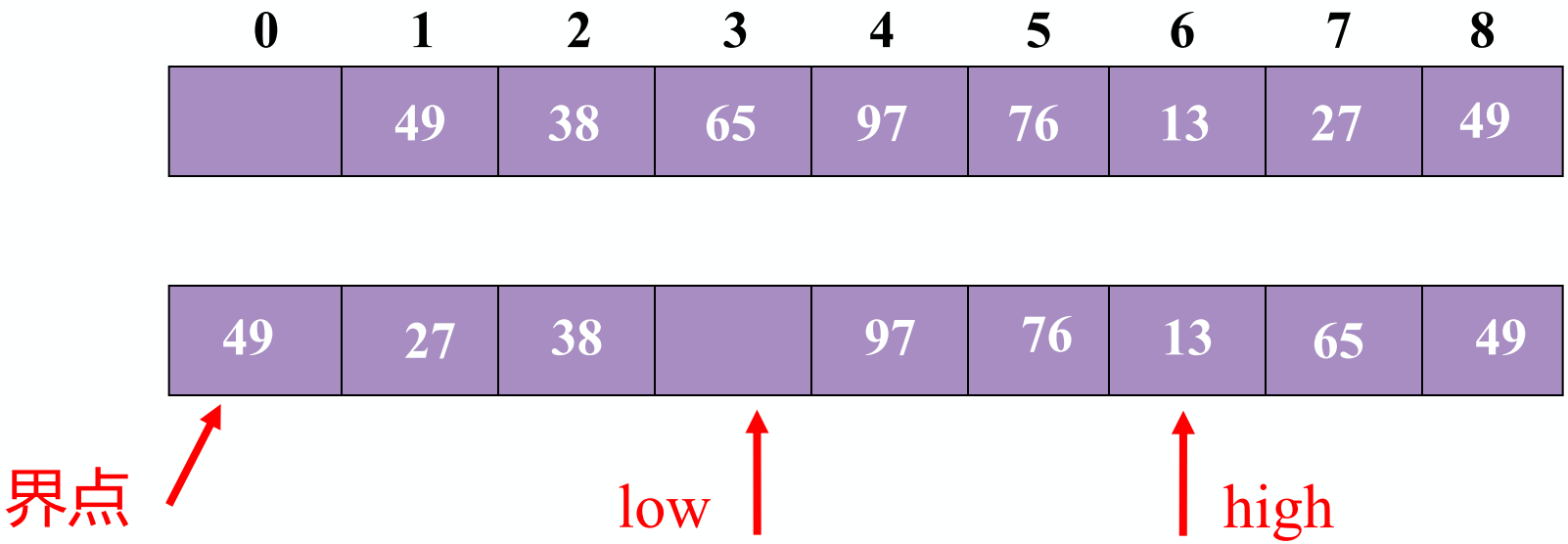
二、快速排序


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49
49 27 38 65 97 76 13 49
界点
low
high

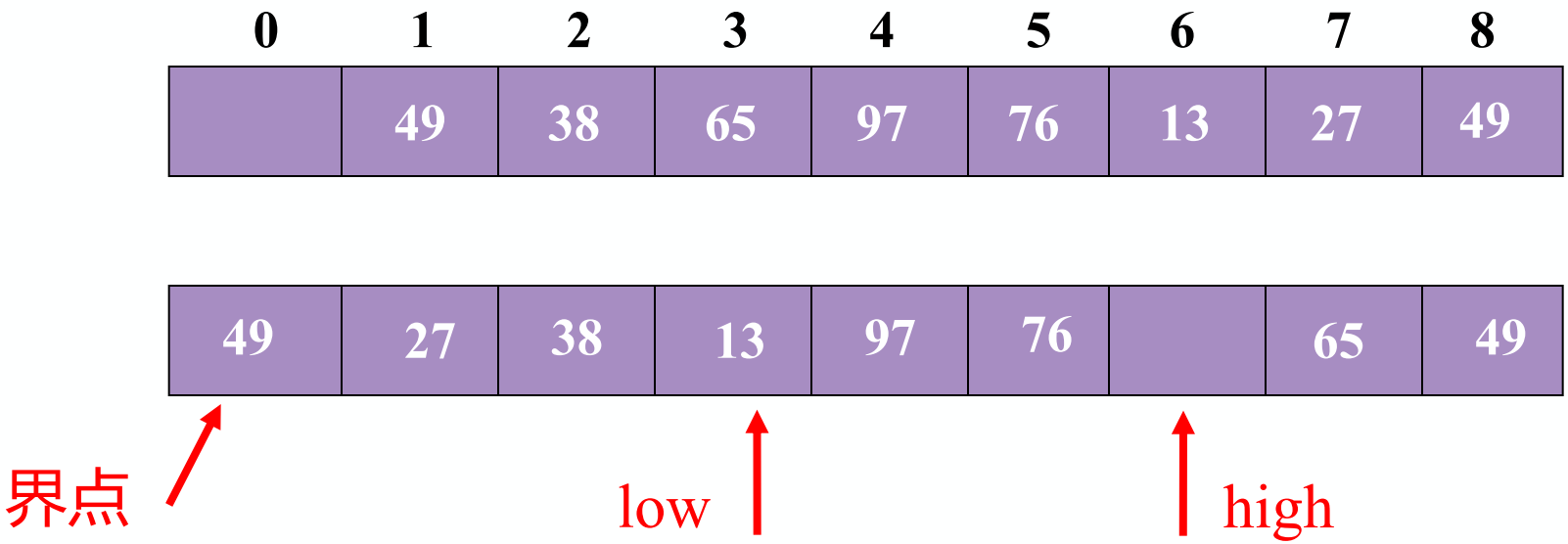
二、快速排序


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49
49 27 38 97 76 13 65 49
界点
low
high

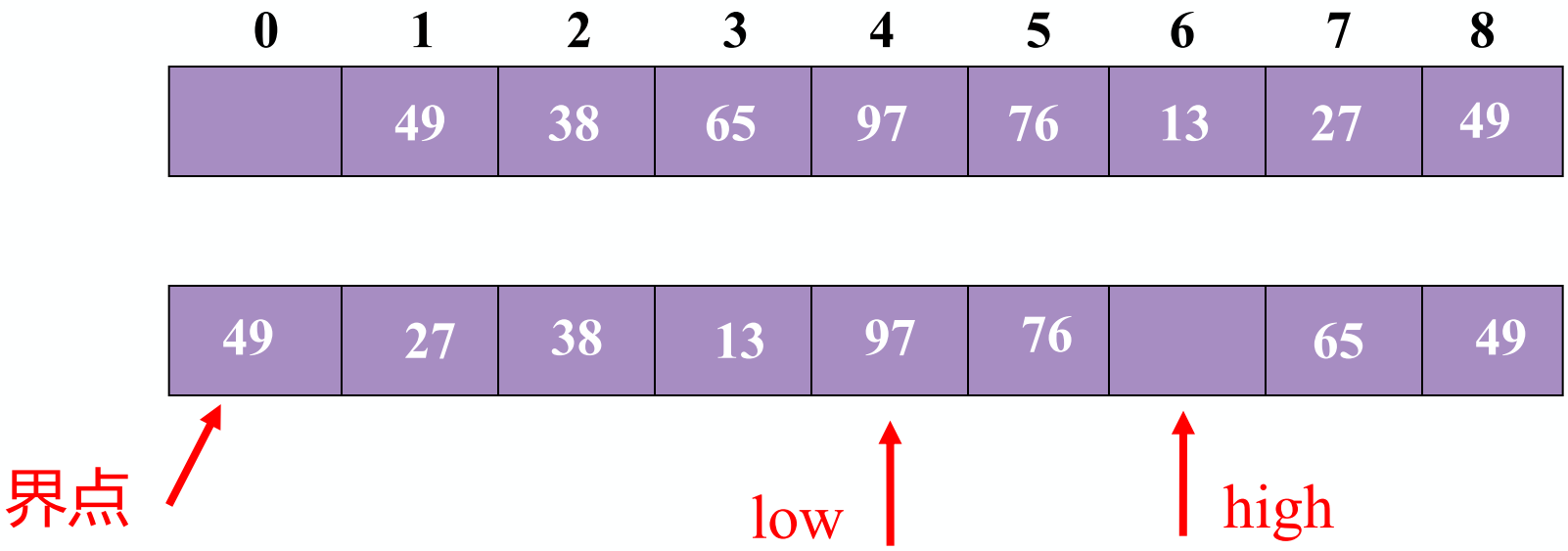
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
27
38
97
76
13
65
49
界点
low
high

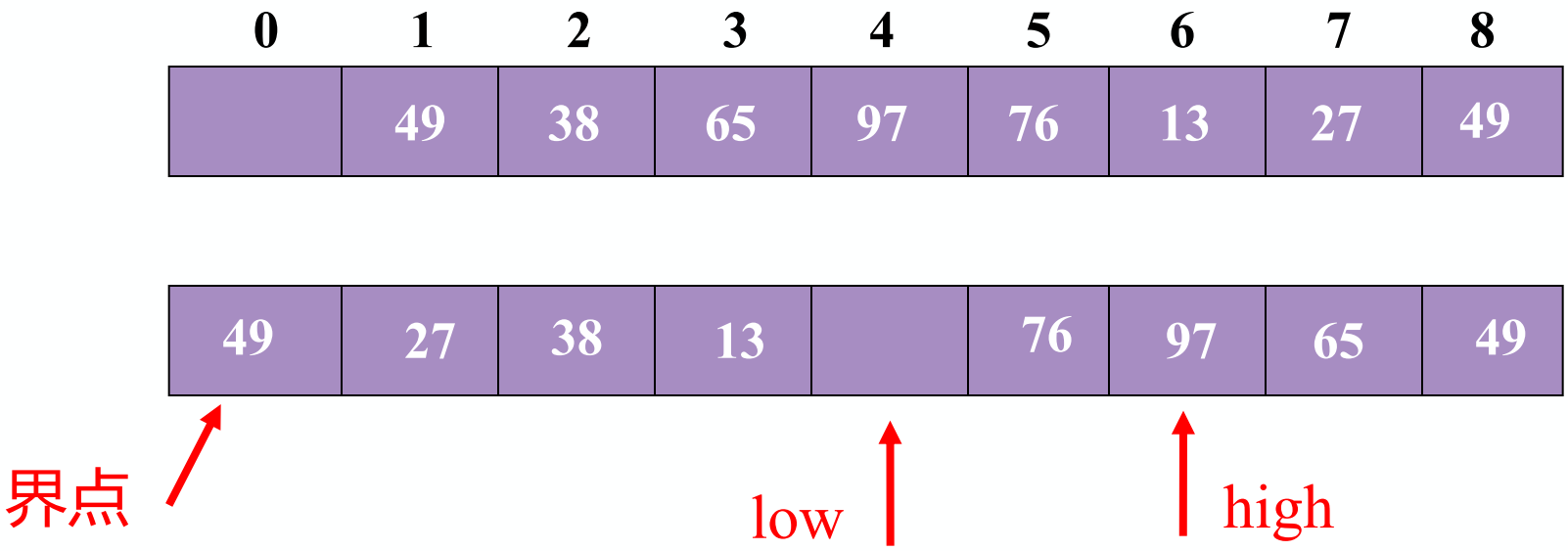
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
27
38
13
97
76
65
49
界点
low
high

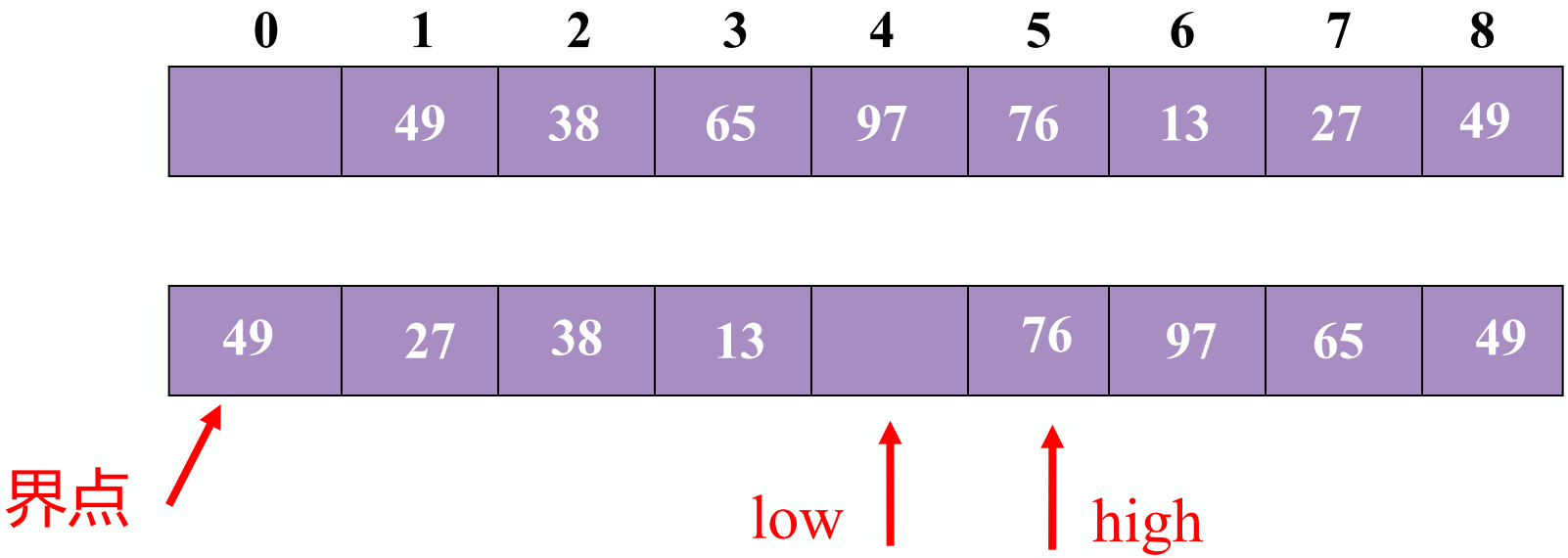
二、快速排序


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49
49 27 38 13 97 76 65 49
界点
low
high

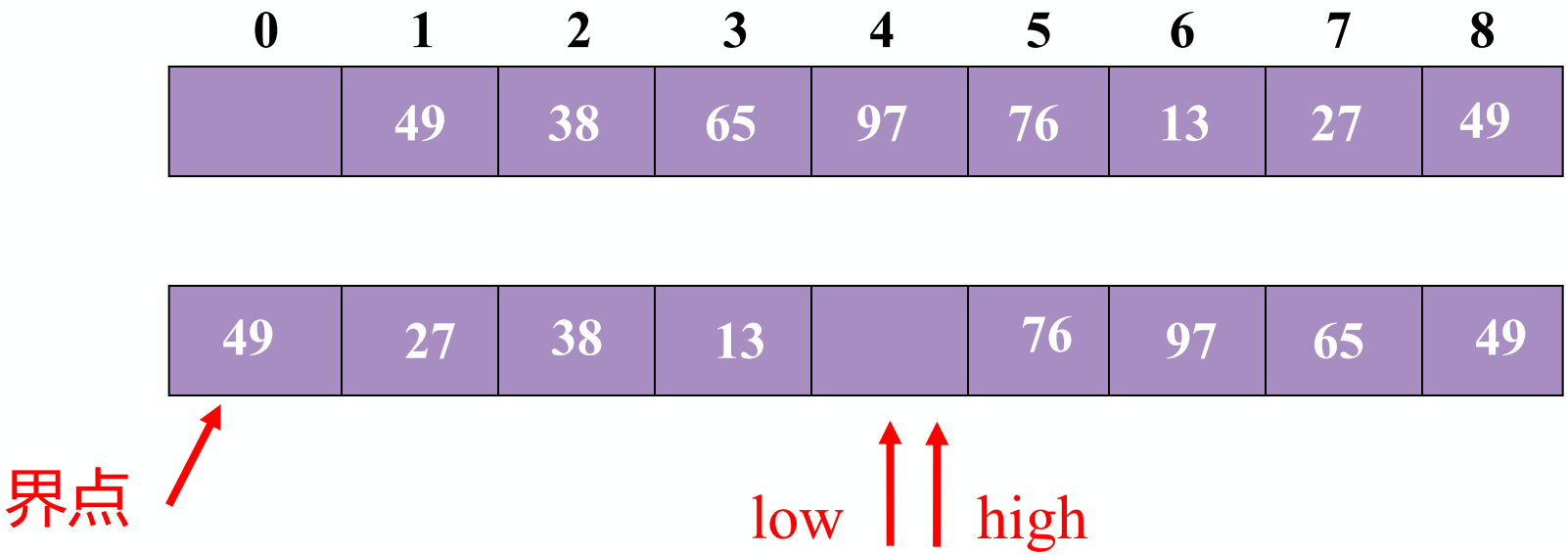
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
27
38
13
76
97
65
49
界点
low
high

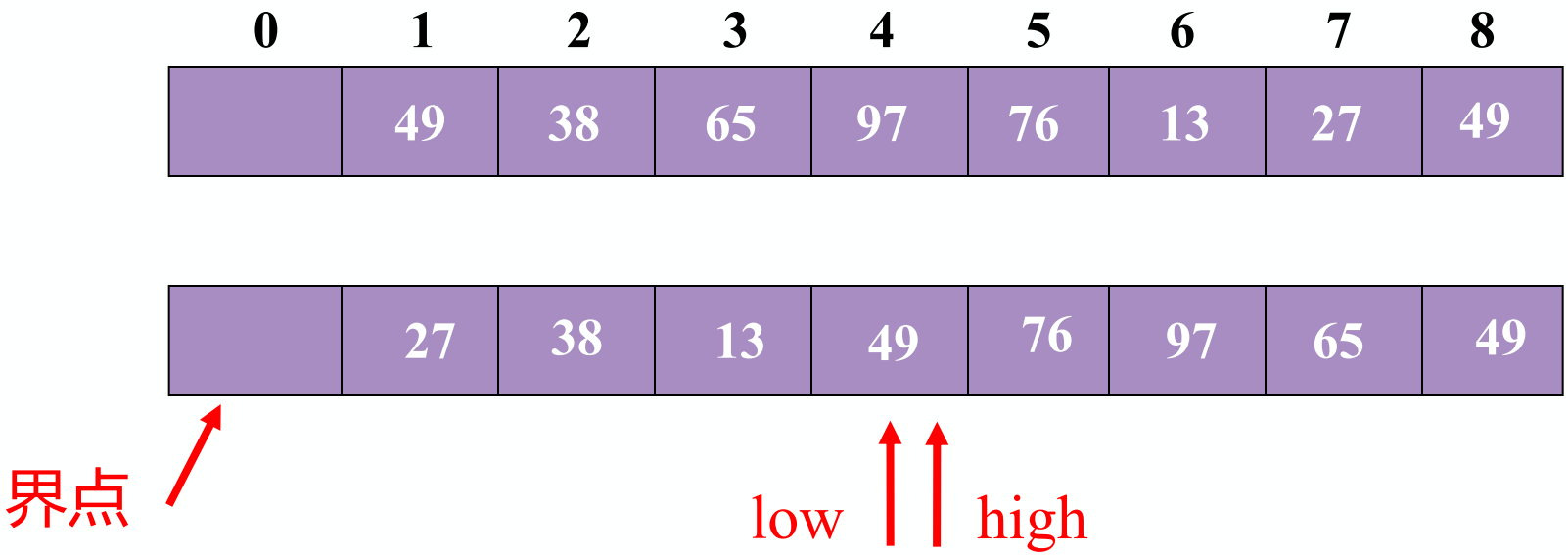
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
27
38
13
76
97
65
49
界点
low
high

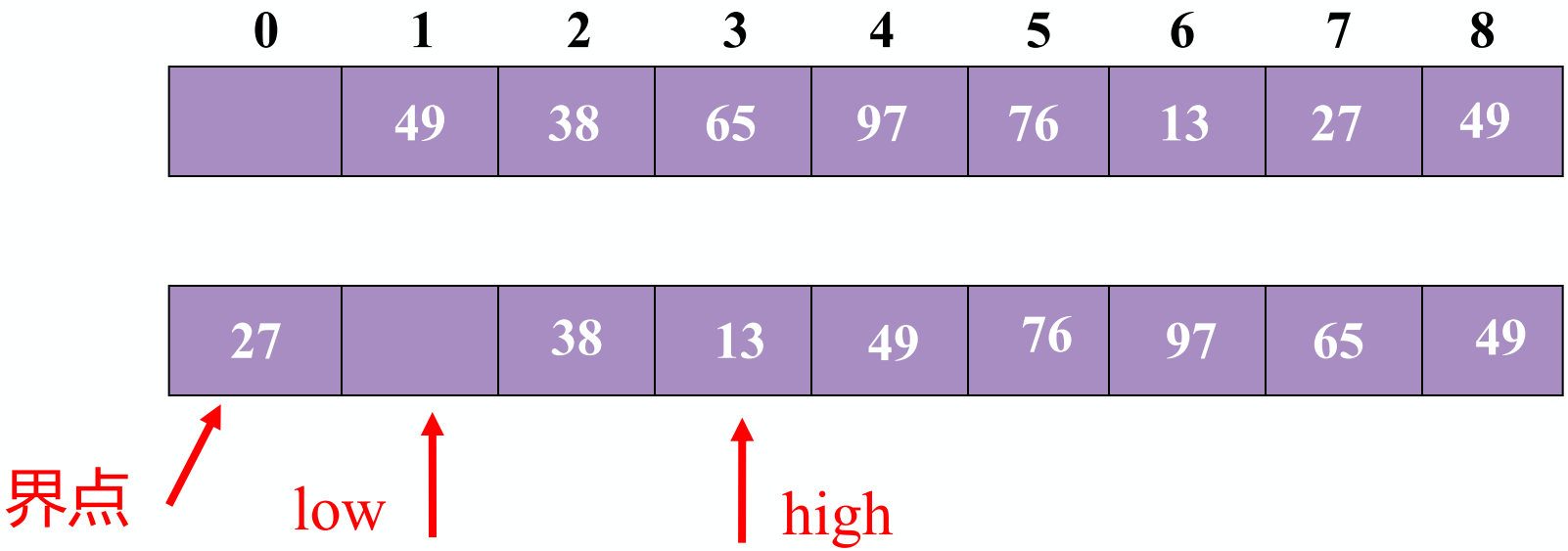
二、快速排序


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49
49 27 38 13 76 97 65 49
界点
low
high

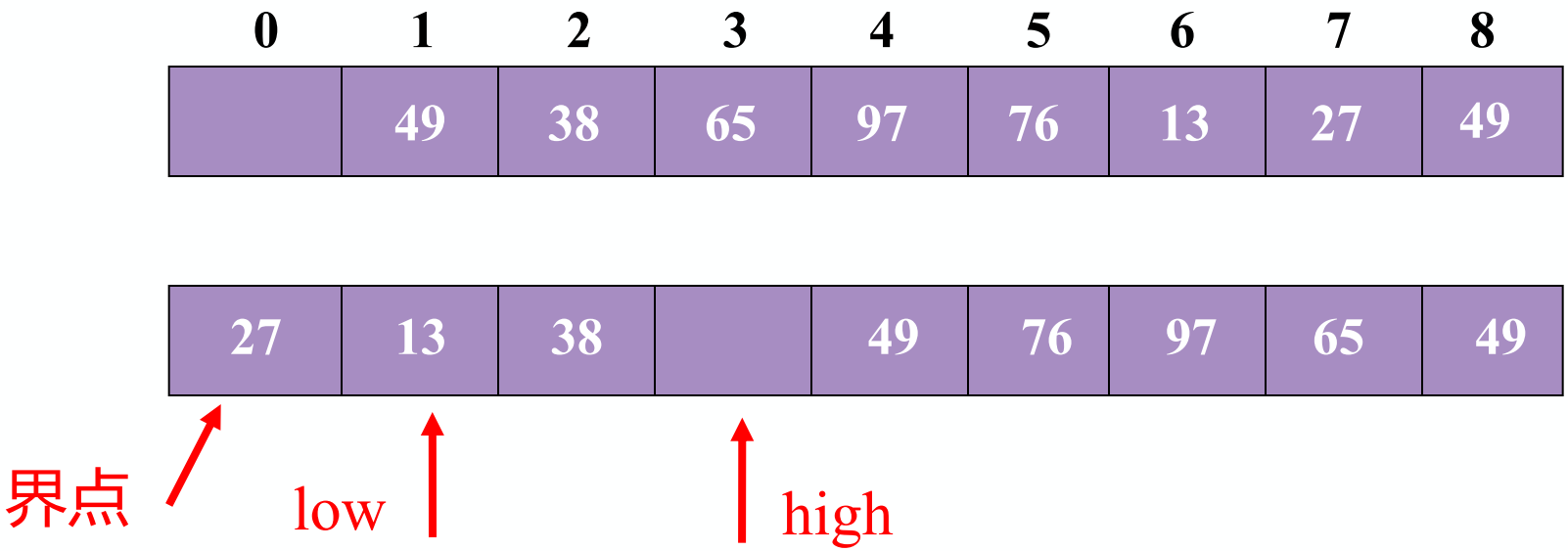
二、快速排序


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49
27 38 13 49 76 97 65 49
界点
low
high

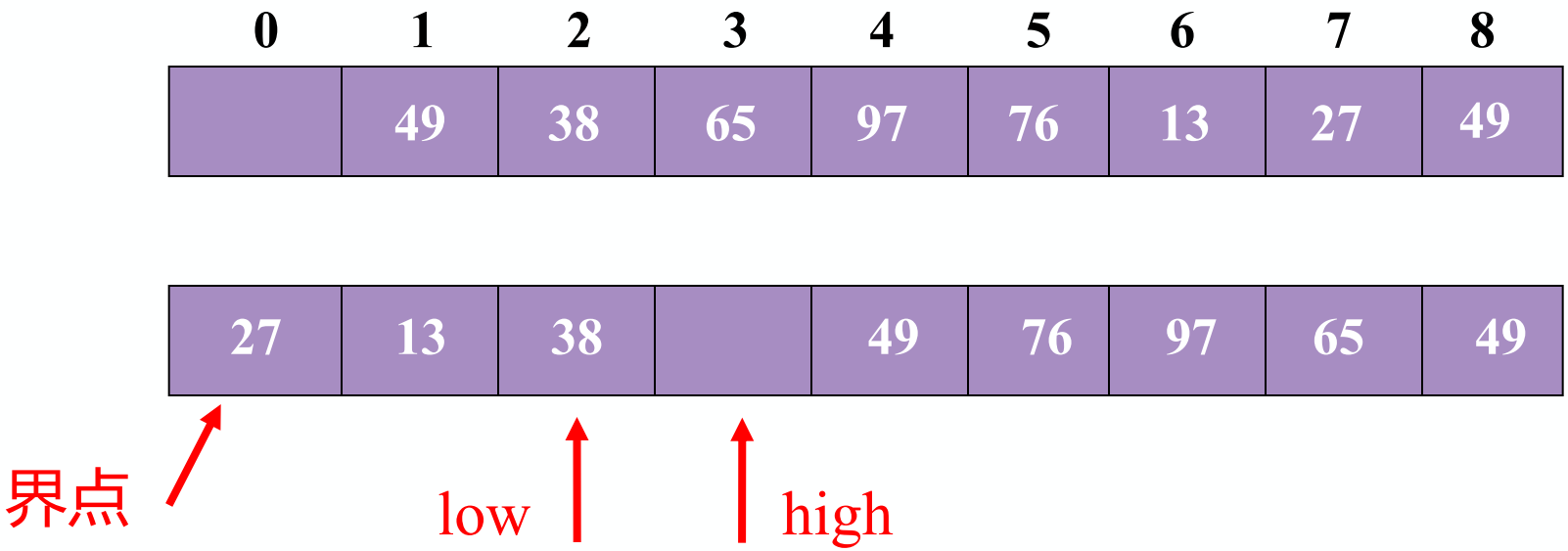
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
27
38
13
49
76
97
65
49
界点
low
high

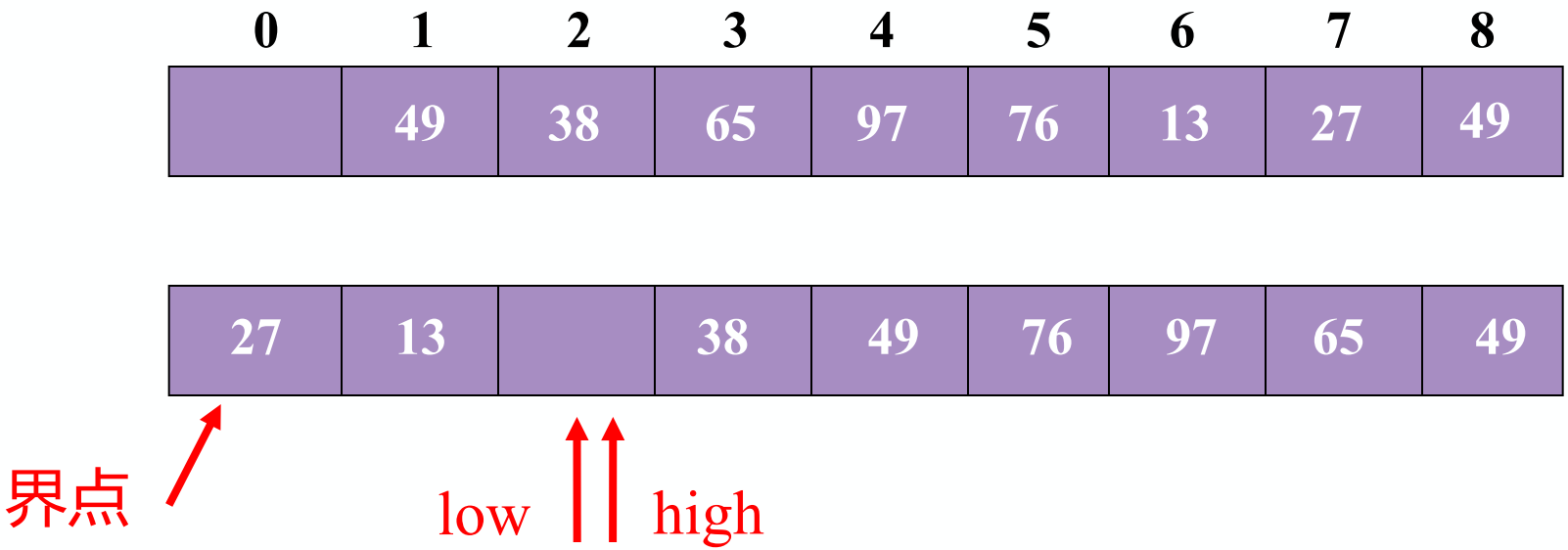
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
27
13
38
49
76
97
65
49
界点
low
high

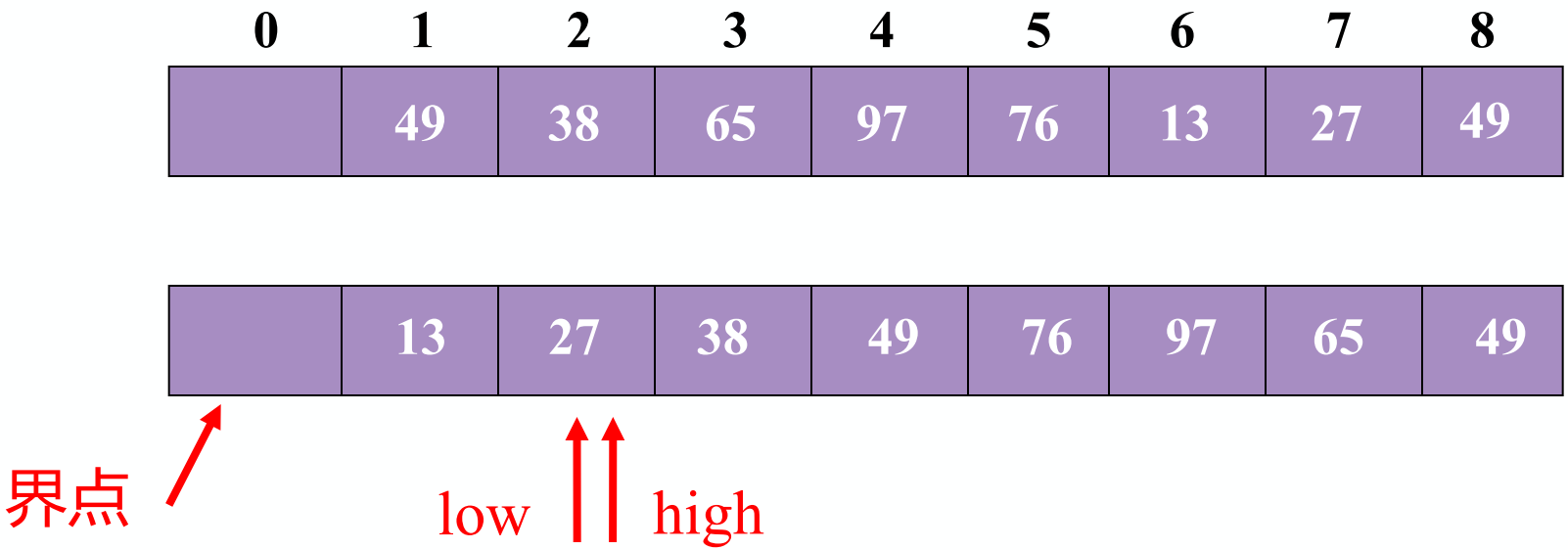
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
27
13
38
49
76
97
65
49
界点
low
high

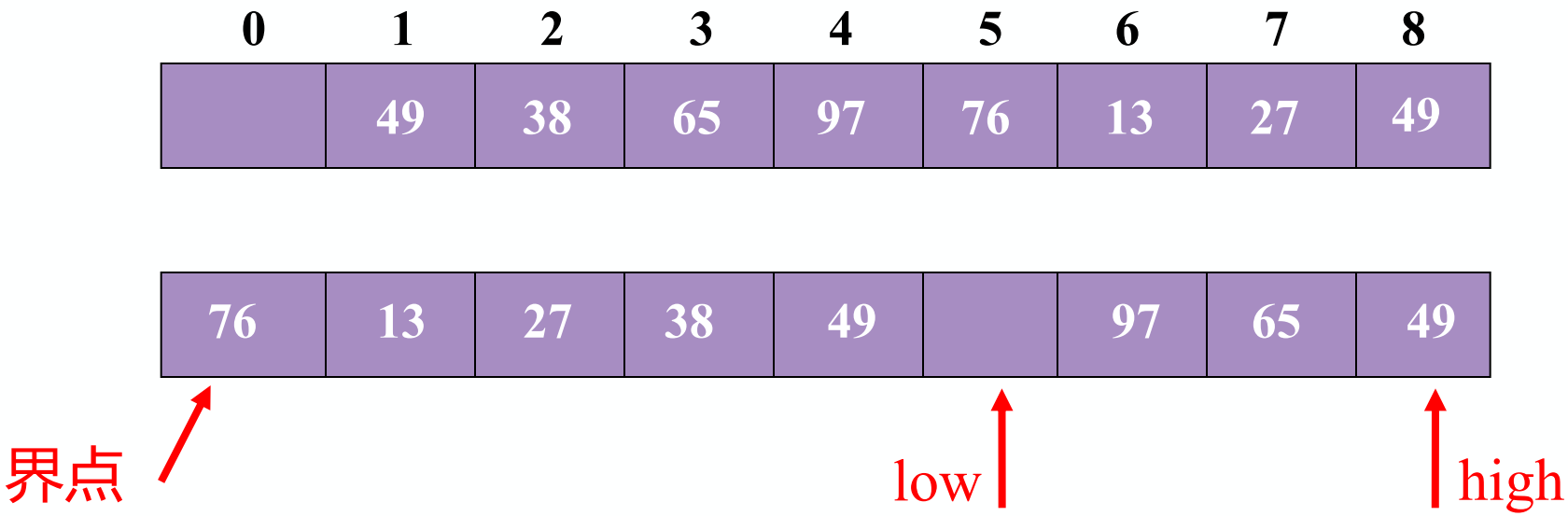
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
27
13
38
49
76
97
65
49
界点
low
high

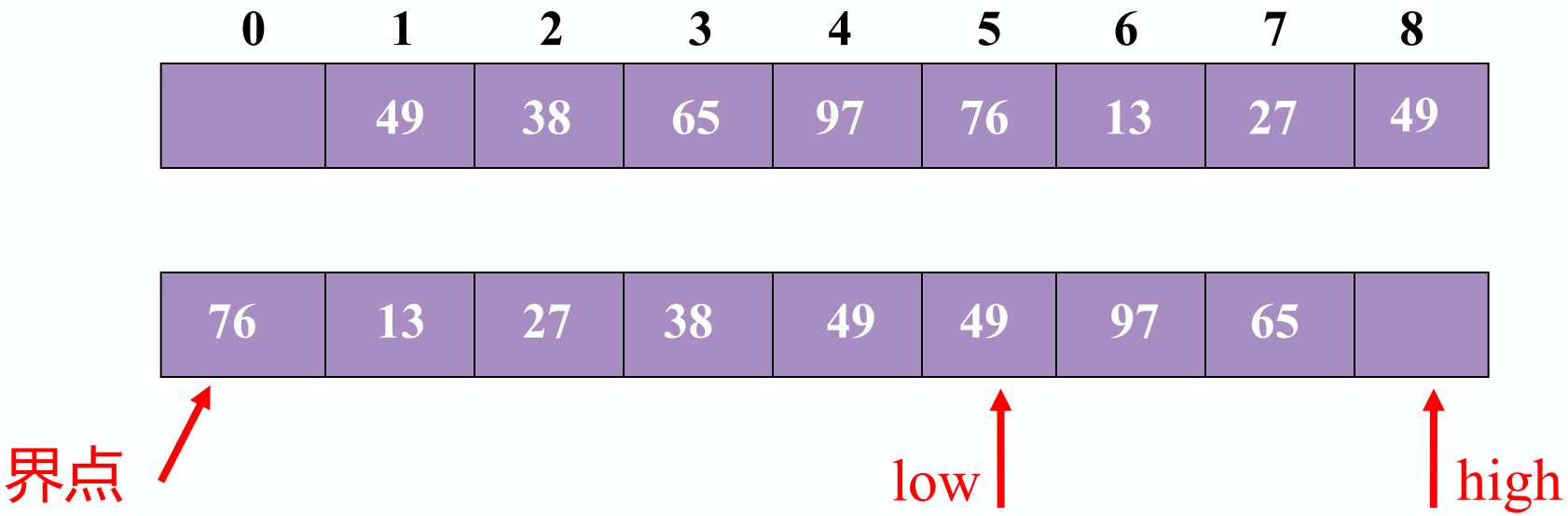
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
13
27
38
49
76
97
65
49
界点
low
high

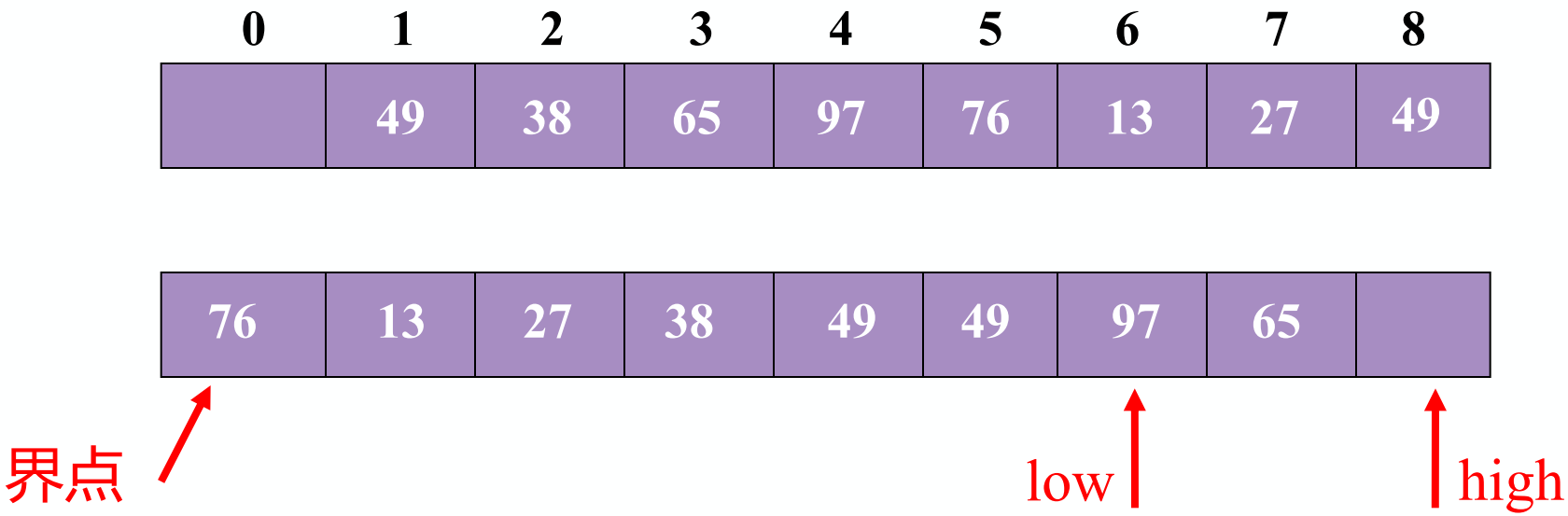
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
76
13
27
38
49
97
65
49
界点
low
high

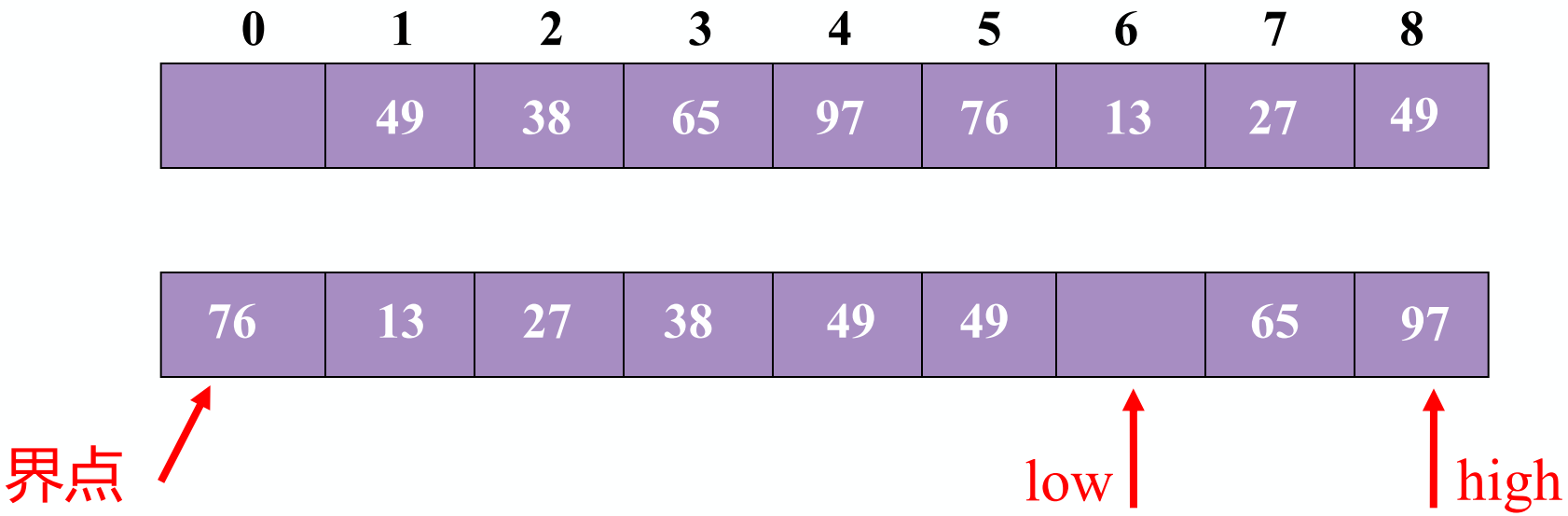
二、快速排序


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49
76 13 27 38 49 49 97 65
界点
low
high

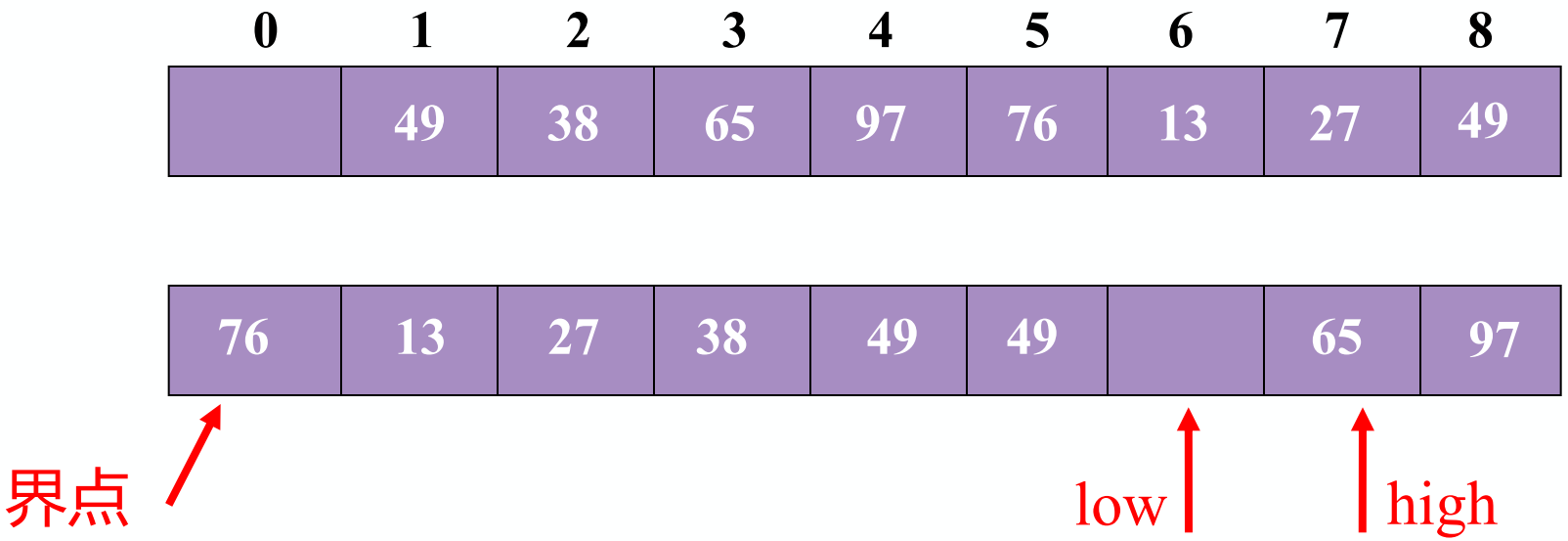
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
76
13
27
38
49
49
97
65
界点
low
high

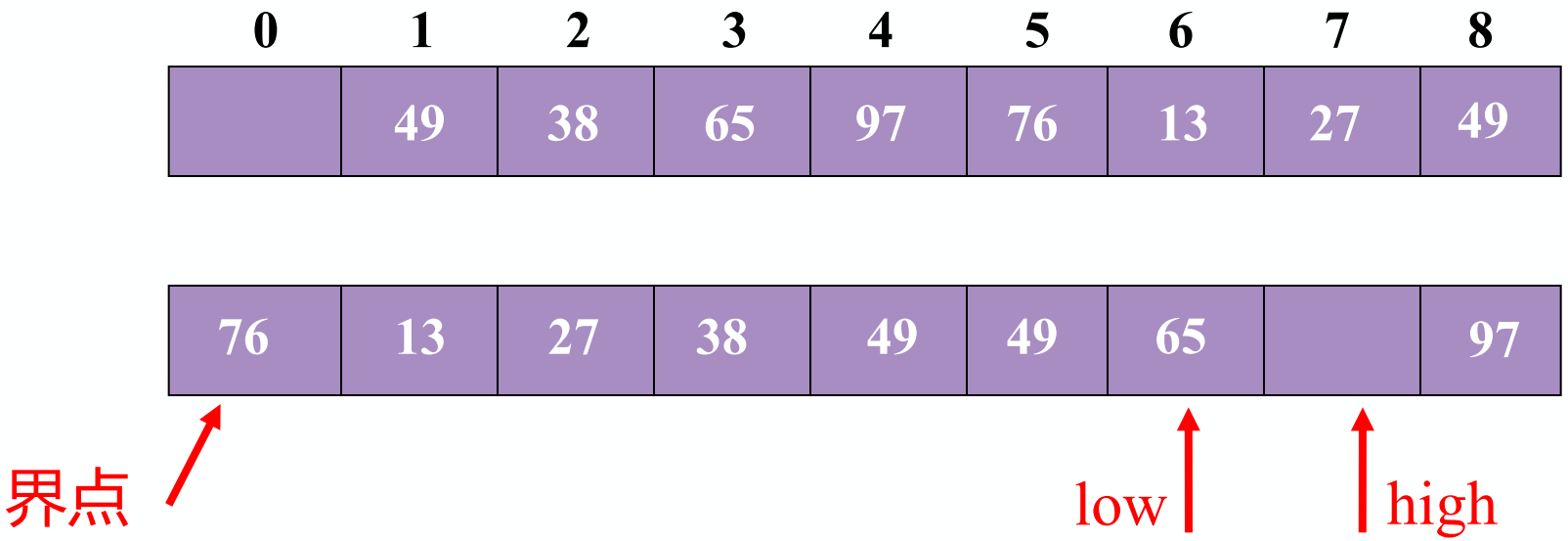
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
76
13
27
38
49
49
65
97
界点
low
high

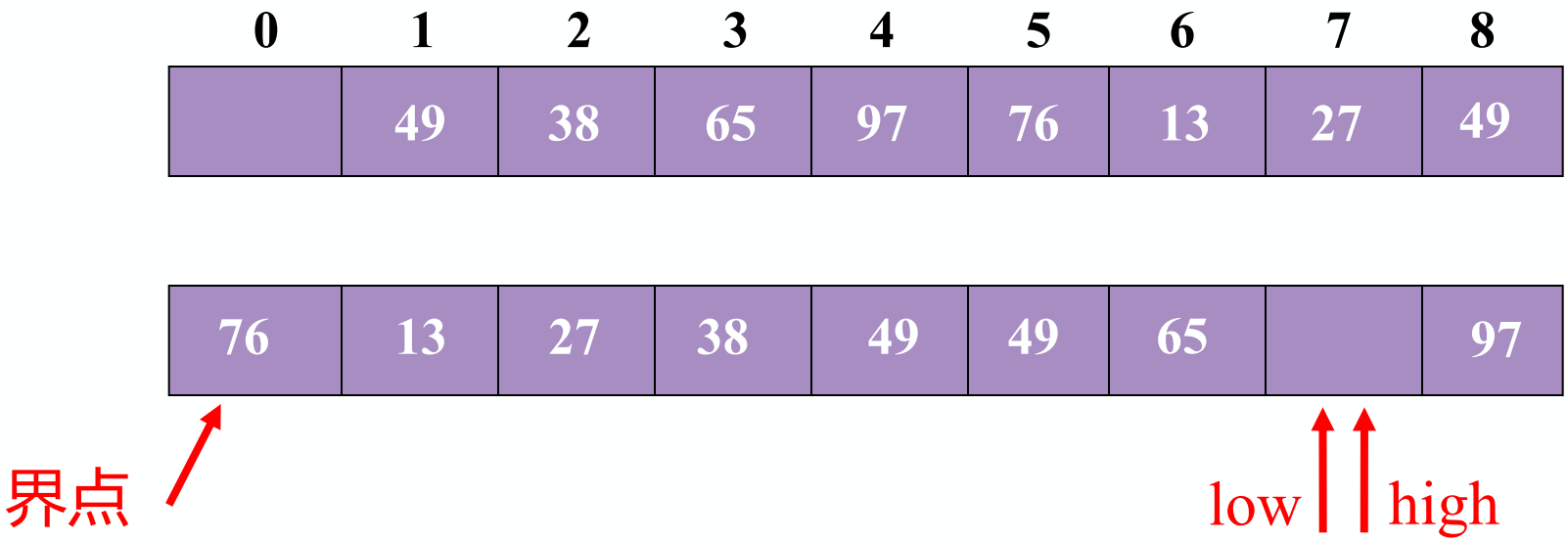
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
76
13
27
38
49
49
65
97
界点
low
high

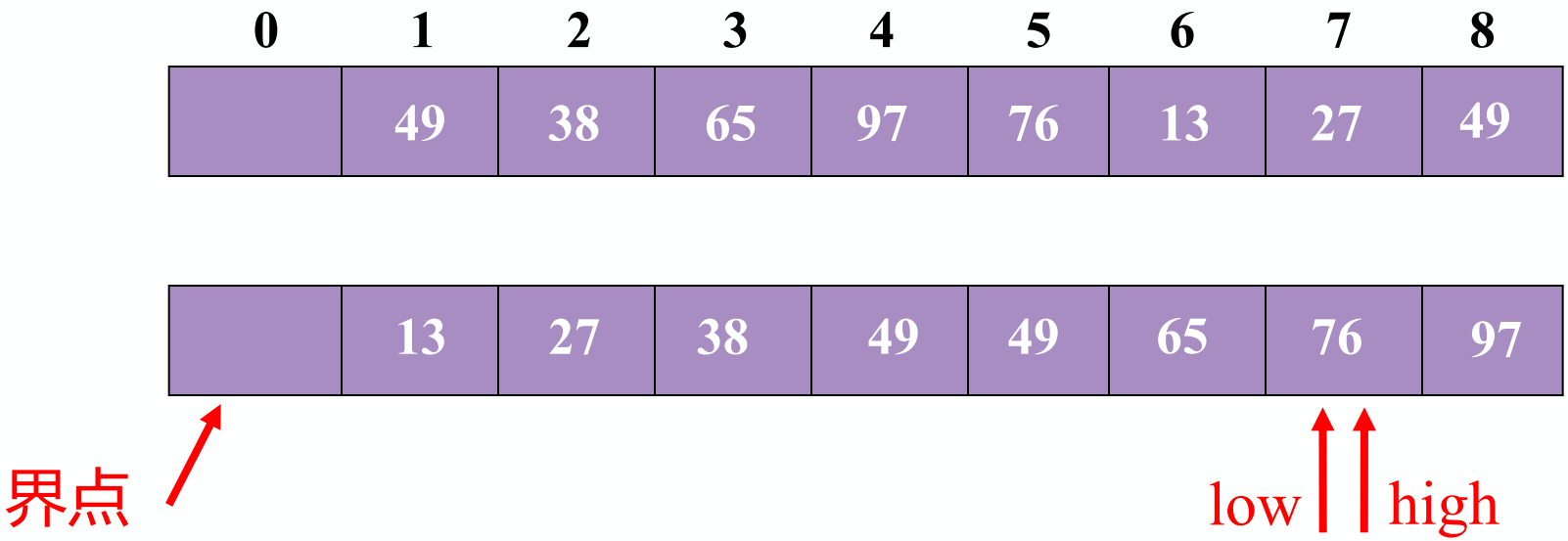
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
76
13
27
38
49
49
65
97
界点
low
high

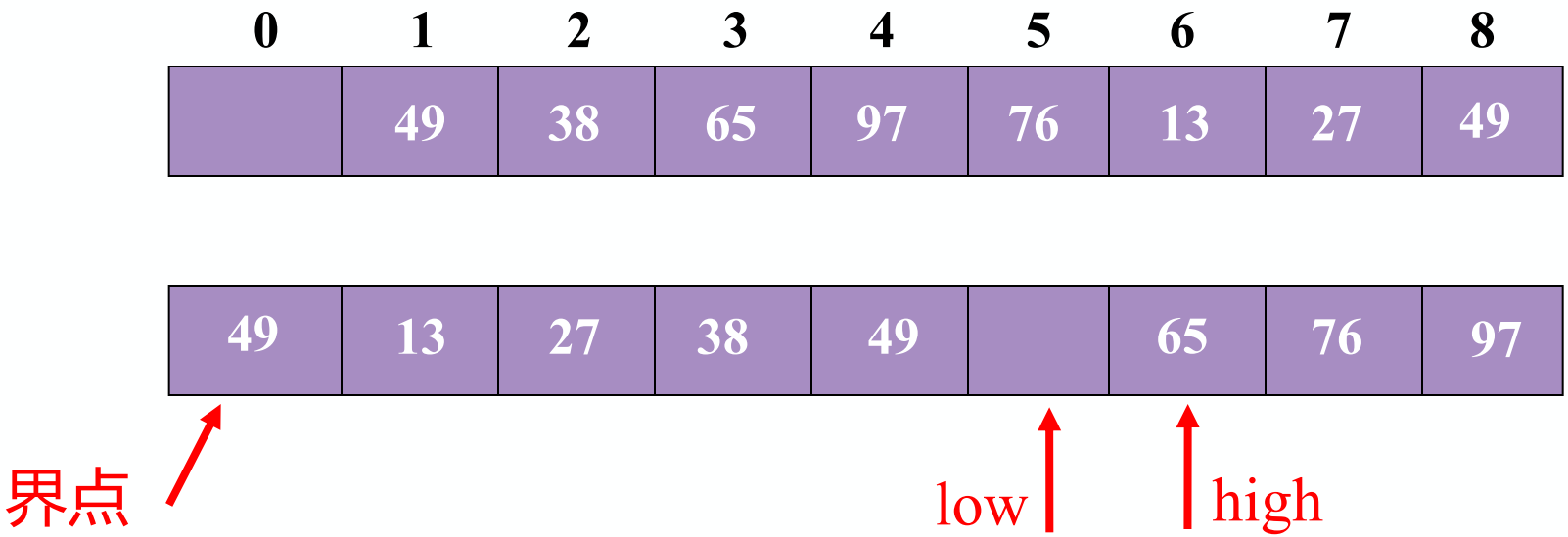
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
76
13
27
38
49
49
65
97
界点
low
high

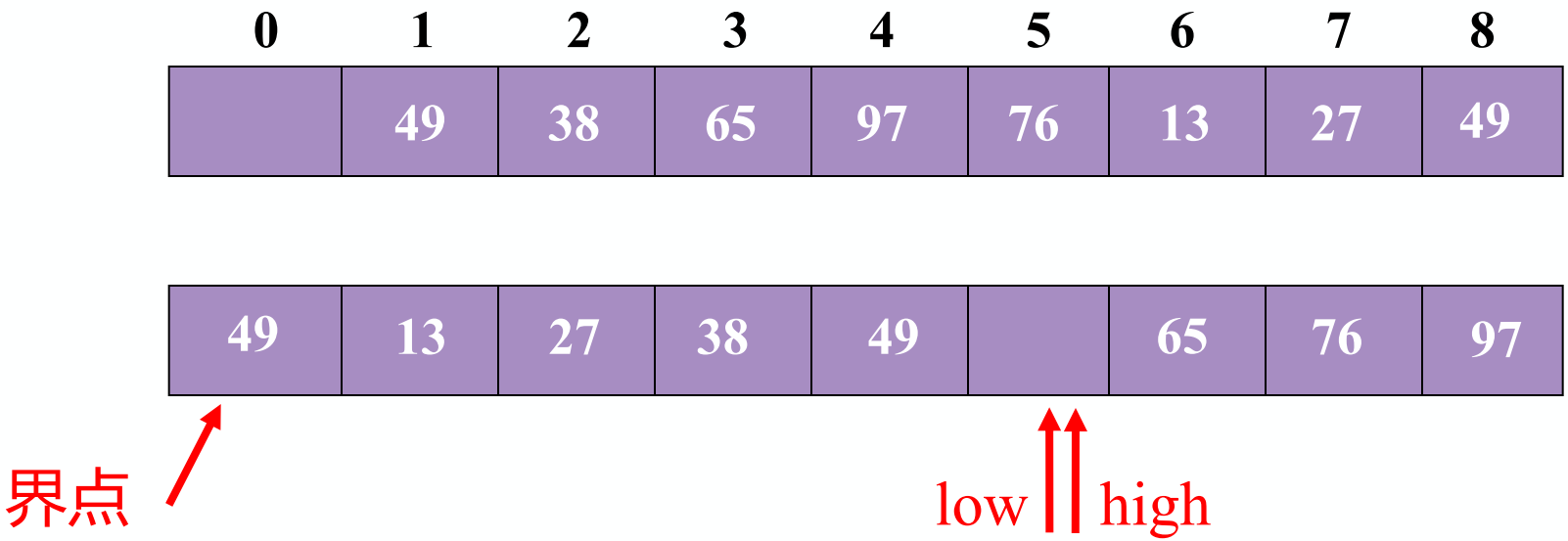
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
13
27
38
49
49
65
76
97
界点
low
high

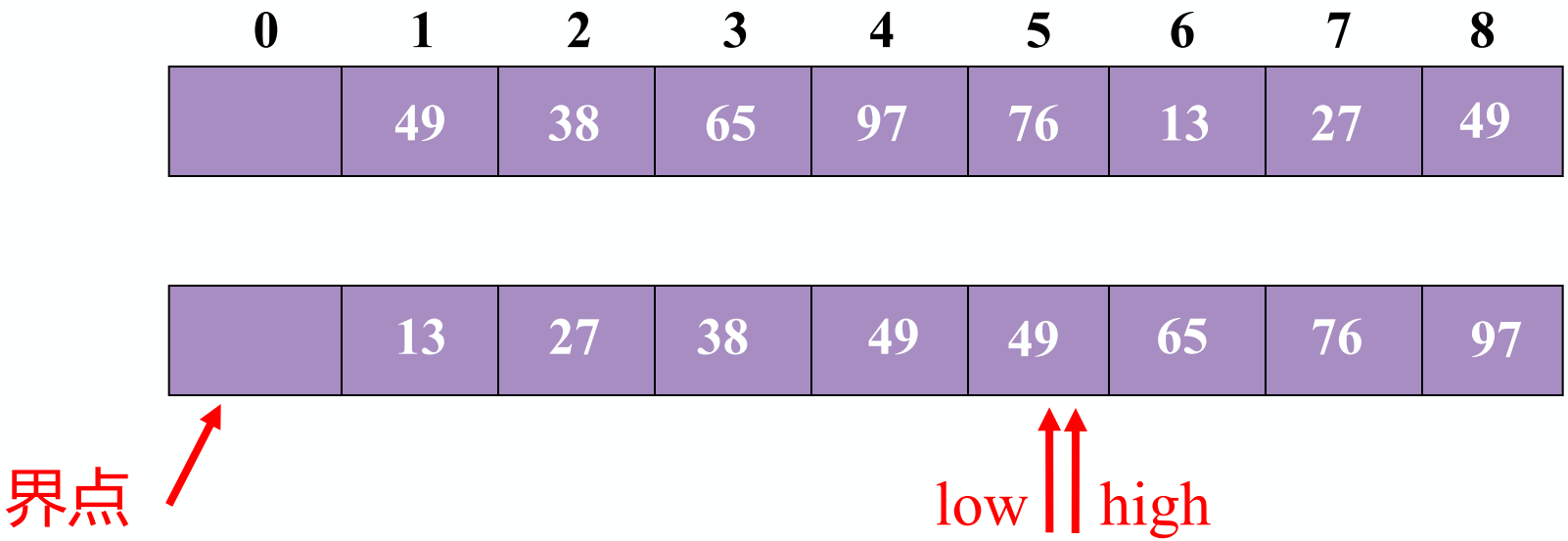
二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
13
27
38
49
65
76
97
界点
low
high

二、快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
49
13
27
38
49
65
76
97
界点
low
high

快速排序


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49
13
27
38
49
49
65
76
97
界点
low
high

二、快速排序

每一趟的子表的形成是采用从两头向中间交替式逼近法；

由于每趟中对各子表的操作都相似，可采用递归算法。

二、快速排序

```
void main ( )
 {   QSort ( L, 1, L.length ); }
```

```
void QSort ( SqList &L ， int low,  int  high )
{  if  ( low < high )
    {  pivotloc = Partition(L, low, high ) ;
       Qsort (L, low, pivotloc-1) ;
       Qsort (L, pivotloc+1, high )
     }
}
```

二、快速排序

```
int Partition ( SqList &L ，int low,  int  high )
{  L.r[0] = L.r[low];   pivotkey = L.r[low].key;
   while  ( low < high )
    { while ( low < high && L.r[high].key >= pivotkey )  --high;
                L.r[low] = L.r[high];
      while ( low < high && L.r[low].key <= pivotkey )  ++low;
                L.r[high] = L.r[low];
     }
    L.r[low]=L.r[0];
    return low;
}
```

二、快速排序

1.算法分析

可以证明，平均计算时间是$O(n\log_2 n)$。

实验结果表明：就平均计算时间而言，快速排序是我们所讨论的所有内排序方法中最好的一个。

快速排序是递归的，需要有一个栈存放每层递归调用时参数（新的low和high）。

最大递归调用层次数与递归树的深度一致，因此，要求存储开销为$O(\log_2 n)$。

二、快速排序

1.算法分析

最好

划分后，左侧右侧子序列的长度相同。

最坏

从小到大排好序，递归树成为单支树，每次划分只得到一个比上一次少一个对象的子序列，必须经过 n-1 趟才能把所有对象定位，而且第 i 趟需要经过 n-i 次关键码比较才能找到第 i 个对象的安放位置。

$$\sum_{i=1}^{n-1}(n-i)=\frac{1}{2}n(n-1)\approx\frac{n^2}{2}$$

二、快速排序

1.算法分析

1

时间效率：

$O(n\log_2 n)$ —每趟确定的元素呈指数增加；

2

空间效率：

$O(\log_2 n)$ —递归要用到栈空间；

3

稳 定 性：

不稳定 —可选任一元素为支点。

小结

1. 冒泡排序
2. 快速排序