



交换排序

- 基本思想：两两比较待排序数据的关键字的值，并交换哪些不满足顺序要求的偶对，直到全部满足顺序要求为止。
- 具体方法：**冒泡**排序，**快速**排序



冒泡排序方法

- 将待排序的数据元素的关键字**顺次****两两比较**，若为**逆序**则将两个数据元素**交换**。
- 将序列照此方法从头到尾处理一遍称作一趟冒泡排序，它将关键字值最大的数据元素交换到排序的最终位置。
- 若某一趟冒泡排序**没发生任何数据元素的交换**，则排序过程结束。
- 对含 n 个记录的文件排序最多需要 **$n-1$ 趟**冒泡排序。



冒泡排序

第一趟：第1个与第2个比较，大则交换；

第2个与第3个比较，大则交换，

.....,

关键字最大的数据元素交换到最后一个位置上.

第二趟：对前 $n-1$ 个数据元素进行同样的操作，关键字次

大的数据元素交换到第 $n-1$ 个位置上；

依次类推，则完成排序。



冒泡排序

■ 25, 56, 49, 78, 11, 65, 41, 36



冒泡排序-第一趟

■ 25, 56, 49, 78, 11, 65, 41, 36



冒泡排序-第一趟

■ 25, 56, 49, 78, 11, 65, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 78, 11, 65, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 78, 11, 65, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 78, 11, 65, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 11, 78, 65, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 11, 78, 65, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 11, 65, 78, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 11, 65, 78, 41, 36



冒泡排序-第一趟

■ 25, 49, 56, 11, 65, 41, 78, 36



冒泡排序-第一趟

■ 25, 49, 56, 11, 65, 41, 78, 36



冒泡排序-第一趟

■ 25, 49, 56, 11, 65, 41, 36, 78



冒泡排序-第一趟

■ 25, 49, 56, 11, 65, 41, 36, 78



冒泡排序-第二趟

■ 25, 49, 56, 11, 65, 41, 36, 78



冒泡排序-第二趟

■ 25, 49, 56, 11, 65, 41, 36, 78



冒泡排序-第二趟

■ 25, 49, 56, 11, 65, 41, 36, 78



冒泡排序-第二趟

■ 25, 49, 11, 56, 65, 41, 36, 78



冒泡排序-第二趟

■ 25, 49, 11, 56, 65, 41, 36, 78



冒泡排序-第二趟

■ 25, 49, 11, 56, 65, 41, 36, 78



冒泡排序-第二趟

■ 25, 49, 11, 56, 41, 65, 36, 78



冒泡排序-第二趟

■ 25, 49, 11, 56, 41, 65, 36, 78



冒泡排序-第二趟

■ 25, 49, 11, 56, 41, 36, 65, 78



冒泡排序-第二趟

■ 25, 49, 11, 56, 41, 36, 65, 78



冒泡排序-第三趟

■ 25, 49, 11, 56, 41, 36, 65, 78



冒泡排序-第三趟

■ 25, 49, 11, 56, 41, 36, 65, 78



冒泡排序-第三趟

■ 25, 11, 49, 56, 41, 36, 65, 78



冒泡排序-第三趟

■ 25, 11, 49, 56, 41, 36, 65, 78



冒泡排序-第三趟

■ 25, 11, 49, 56, 41, 36, 65, 78



冒泡排序-第三趟

■ 25, 11, 49, 41, 56, 36, 65, 78



冒泡排序-第三趟

■ 25, 11, 49, 41, 56, 36, 65, 78



冒泡排序-第三趟

■ 25, 11, 49, 41, 36, 56, 65, 78



冒泡排序-第三趟

■ 25, 11, 49, 41, 36, 56, 65, 78



冒泡排序-第四趟

■ 25, 11, 49, 41, 36, 56, 65, 78



冒泡排序-第四趟

■ 11, 25, 49, 41, 36, 56, 65, 78



冒泡排序-第四趟

■ 11, 25, 49, 41, 36, 56, 65, 78



冒泡排序-第四趟

■ 11, 25, 49, 41, 36, 56, 65, 78



冒泡排序-第四趟

■ 11, 25, 41, 49, 36, 56, 65, 78



冒泡排序-第四趟

■ 11, 25, 41, 49, 36, 56, 65, 78



冒泡排序-第四趟

■ 11, 25, 41, 36, 49, 56, 65, 78



冒泡排序-第四趟

■ 11, 25, 41, 36, 49, 56, 65, 78



冒泡排序-第五趟

■ 11, 25, 41, 36, 49, 56, 65, 78



冒泡排序-第五趟

■ 11, 25, 41, 36, 49, 56, 65, 78



冒泡排序-第五趟

■ 11, 25, 41, 36, 49, 56, 65, 78



冒泡排序-第五趟

■ 11, 25, 36, 41, 49, 56, 65, 78



冒泡排序-第五趟

■ 11, 25, 36, 41, 49, 56, 65, 78



冒泡排序-第六趟

■ 11, 25, 36, 41, 49, 56, 65, 78



冒泡排序-第六趟

■ 11, 25, 36, 41, 49, 56, 65, 78



冒泡排序-第六趟

- 11, 25, 36, 41, 49, 56, 65, 78
- 第六趟冒泡排序，一次数据移动也没发生，说所有数据已经有序，冒泡排序结束
- 冒泡排序的结束条件：
 - 做完 $n-1$ 趟
 - 或：某趟冒泡排序过程中一次数据移动也没发生，说所有数据已经有序，冒泡排序结束



冒泡排序

- 对 n 个数据元素排序最多需要 $n-1$ 趟冒泡排序。
- 最好情况： n 个数据元素， 1 趟冒泡排序， 0 次数据移动， $n-1$ 次比较。
- 最坏情况： n 个数据元素， $n-1$ 趟冒泡排序。
- 平均时间复杂度 $O(n^2)$
- 一个额外的辅助空间 $O(1)$ 。
- 冒泡排序是稳定的排序方法。



```
void qppx(SqList &L)
```

```
{
```

```
    int i,j,k;
```

```
    j=1;k=1;
```

```
    while((j<L.length) && (k>0))
```

```
    { k=0;
```

```
      for(i=1;i<=L.length-j; i++)
```

```
      if(L.r[i+1].key<L.r[i].key)
```

```
      { L.r[0]=L.r[i];
```

```
        L.r[i]=L.r[i+1];
```

```
        L.r[i+1]=L.r[0];
```

```
        k++;}
```

```
    j++;}
```

```
}
```



快速排序

- 在待排序的 n 个数据元素中任取一个数据元素（通常取第一个），以**该数据元素的关键字为基准**用交换的方法将所有数据元素分成三部分，所有键值比它小的安置在一部分，所有键值比它大的安置在另一部分，并把该数据元素放在这两部分的中间，这也是该数据元素排序后的**最终位置**这个过程称为**一趟快速排序**。
- 然后分别对所划分的前后两部分重复上述过程，一直重复到每部分只有一个数据元素为止，排序完成。

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]

	49	38	65	67	76	13	50
--	----	----	----	----	----	----	----

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	49	38	65	67	76	13	50
----	----	----	----	----	----	----	----

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]

49	49	38	65	67	76	13	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	49	38	65	67	76	13	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	49	38	65	67	76	13	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]

49	13	38	65	67	76	13	50
----	----	----	----	----	----	----	----

↑
l

↑
h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]

49	13	38	65	67	76	13	50
----	----	----	----	----	----	----	----

↑
l

↑
h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	65	67	76	13	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	65	67	76	13	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	65	67	76	13	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	65	67	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]

49	13	38	65	67	76	65	50
----	----	----	----	----	----	----	----

↑
l

↑
h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	65	67	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	65	67	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



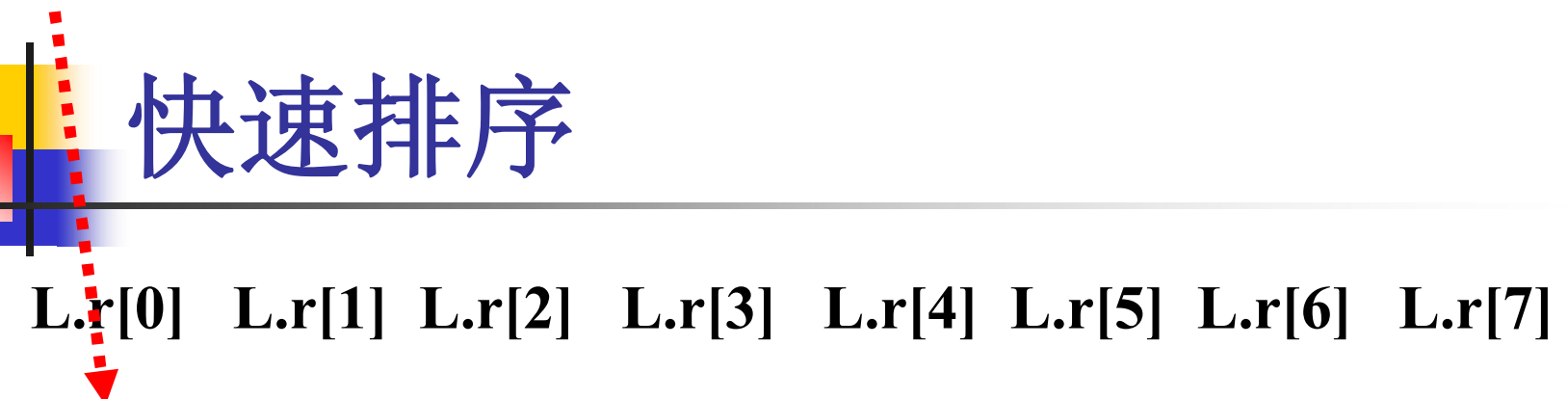
49	13	38	65	67	76	65	50
----	----	----	----	----	----	----	----

↑↑
l h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	49	67	76	65	50
----	----	----	----	----	----	----	----

↑↑
l h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



49	13	38	49	67	76	65	50
----	----	----	----	----	----	----	----

↑↑
l h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



13	13	38	49	67	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



13	13	38	49	67	76	65	50
----	----	----	----	----	----	----	----

l h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	67	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	67	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	76	65	50
----	----	----	----	----	----	----	----

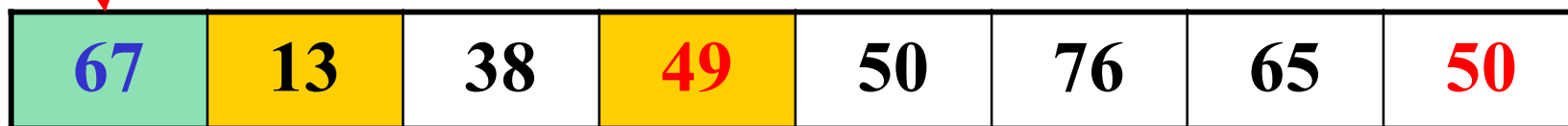
l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	76	65	50
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]

67	13	38	49	50	76	65	76
----	----	----	----	----	----	----	----

↑
l

↑
h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	76	65	76
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	76	65	76
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	65	65	76
----	----	----	----	----	----	----	----

↑
l

↑
h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	65	65	76
----	----	----	----	----	----	----	----

↑ ↑
l *h*

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



67	13	38	49	50	65	67	76
----	----	----	----	----	----	----	----

↑ ↑
l *h*

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



50	13	38	49	50	65	67	76
----	----	----	----	----	----	----	----

l

h

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



50	13	38	49	50	65	67	76
----	----	----	----	----	----	----	----

↑ ↑
l *h*

基准

快速排序

L.r[0] L.r[1] L.r[2] L.r[3] L.r[4] L.r[5] L.r[6] L.r[7]



50	13	38	49	50	65	67	76
----	----	----	----	----	----	----	----

l h



快速排序

- 与**基准**相同的数据元素的处理：放在基准的右侧
- **基准**的选取：**第一个数据元素**、最后一个数据元素、中间位置的数据元素



快速排序—算法步骤

- 设待排序的记录存放与数组 $r[l], r[l+1], \dots, r[h]$ 。
基准为 $r[l]$ ，将 $r[l]$ 保存于 $r[0]$ 。
 1. 从 h 所指位置向**左**搜索，直到找到一键值小于基准 $r[0].key$ ，将 $r[h]$ **写到** $r[l]$ ；
 2. 再从 l 所指位置向**右**搜索，直到找到一键值**大**于基准 $r[0].key$ ，将 $r[l]$ **写到** $r[h]$ ；
 3. 重复1、2直到 $l=h$ 。



快速排序

```
int partition( SqList L, int l, int h )
{  L.r[0]=L.r[l];//取基准为待排序范围的第一个数据，放入r[0]
  while(l<h)
  {
    while(( l < h ) && (L.r[h].key >=L.r[0].key) ) h--;
    if ( l < h ) {L.r[l]=L.r[h]; l++;}
    while( ( l < h ) && (L.r[l ].key <L.r[0].key )) l++;
    if( l < h ) {L.r[h]=L.r[l]; h--;}
  }
  L.r[l]=L.r[0];//将基准从r[0]处移到左右边界重合处，即：基准排序后的最终位置 (l=h)
  return l;
}
```



快速排序

```
void QSort (SqList &L,int l,int h)
{   int t;
    if (l<h)
    {
        t = partition( L, l, h);
        QSort (L, l, t-1);
        QSort (L, t+1, h);
    }
}
```



快速排序

- 稳定性? 快速排序是不稳定的。
- 时间复杂度为 $O(n\log_2 n)$ 。
- 最坏情况下快速排序的时间复杂度为 $O(n^2)$ 。
- 最坏情况待排序的记录基本有序。
-