

交换类排序

8-3-1 起泡排序

讲什么？



起泡排序的基本思想



起泡排序的算法



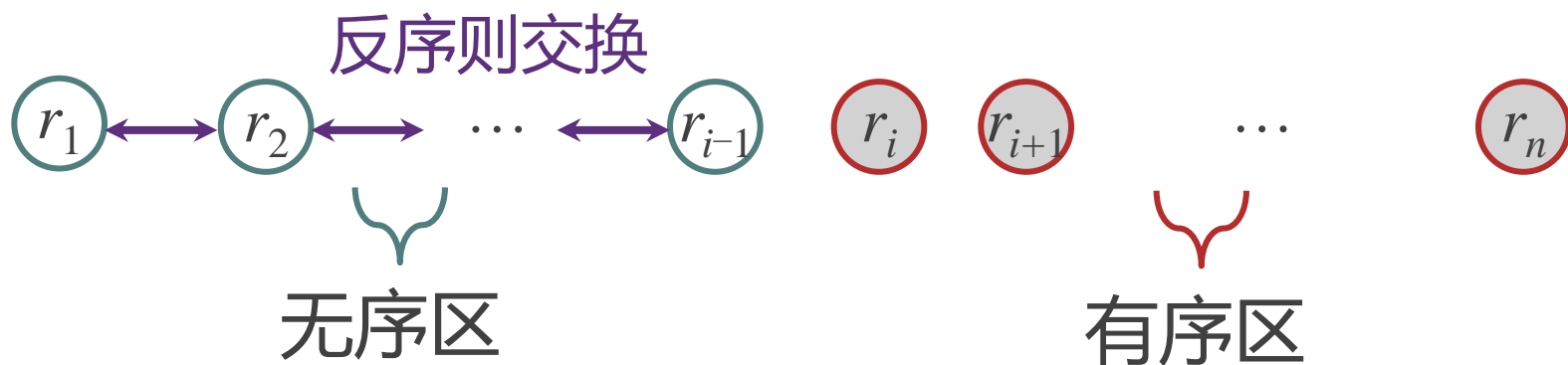
起泡排序的时空性能



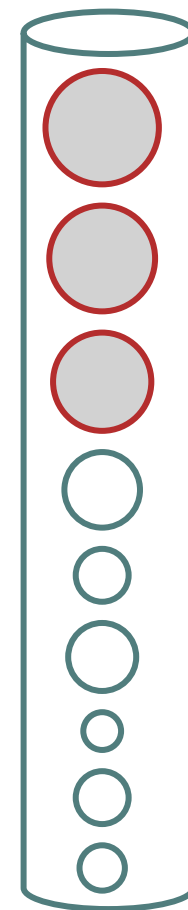
起泡排序的稳定性

基本思想

📌 起泡排序的基本思想：两两比较**相邻**记录，如果**反序**则交换，直到没有反序的记录为止。



类似水中的气泡，体积大的浮到上面，起泡排序因而得名



运行实例

待排序序列

8

12

20

10

18

24

第一趟排序结果

8

12

10

18

20

24

第二趟排序结果

8

10

12

18

20

24

第三趟排序结果

8

10

12

18

20

24

第四趟排序结果

8

10

12

18

20

24

第五趟排序结果

8

10

12

18

20

24

冒泡排序伪代码

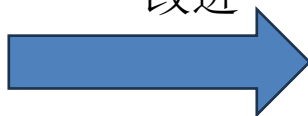
算法10-5 冒泡排序 BubbleSort(a, l, r)

输入：序列 a ，左端点下标 l ，右端点下标 r

输出：调整 a_l, a_{l+1}, \dots, a_r 元素顺序，使元素按照非递减顺序排列

```
1  for  $i \leftarrow l$  to  $r$  do
2    | for  $j \leftarrow l$  to  $r - i$  do
3      | | if  $a_j > a_{j+1}$  then
4        | | | Swap( $a_j, a_{j+1}$ ) //两个元素交换位置
5      | | end
6    | end
7  end
```

改进



运行实例

待排序序列

8

12

20

10

18

24

第一趟排序结果

8

12

10

18

20

24

第二趟排序结果

8

10

12

18

20

24

第三趟排序结果

8

10

12

18

20

24

第四趟排序结果

8

10

12

18

20

24

第五趟排序结果

8

10

12

18

20

24



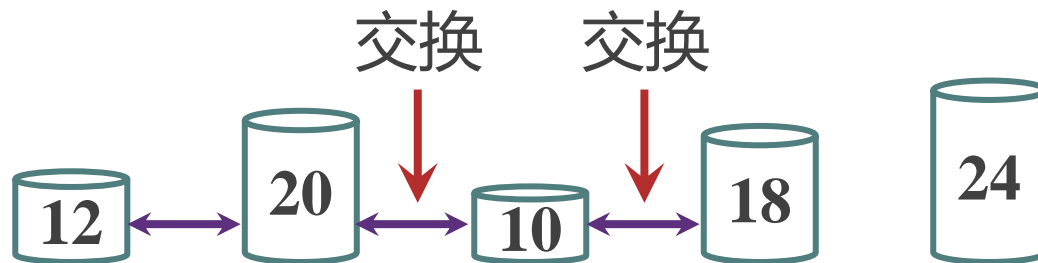
第二趟排序有必要吗？



第五趟排序有必要吗？

关键问题

待排序序列



第一趟排序结果



解决方法：设置变量`exchange`记载交换的位置，一趟排序后`exchange`记载的就是最后交换的位置，从`exchange`之后的记录不参加下一趟排序。



算法描述：

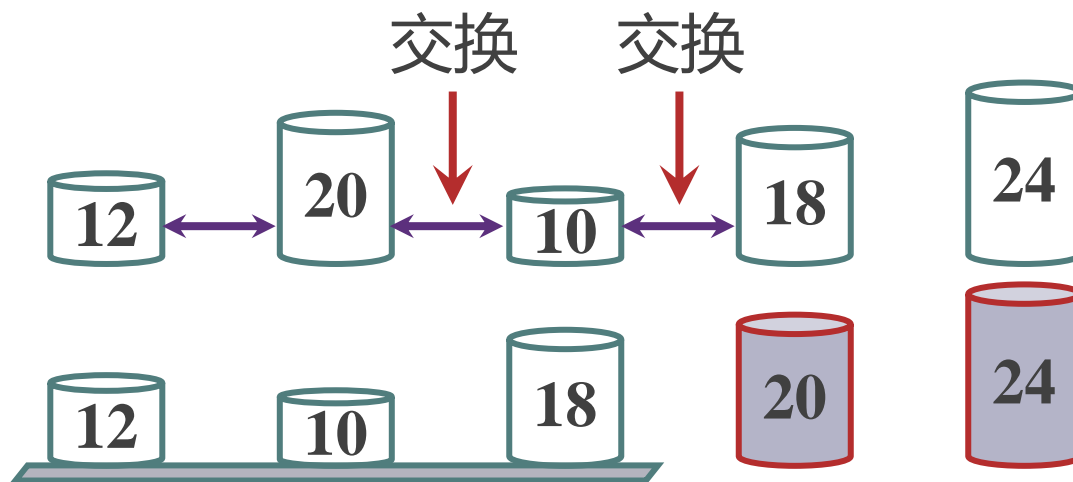
```
if (data[j] > data[j+1]){  
    temp = data[j]; data[j] = data[j+1]; data[j+1] = temp;  
    exchange = j;  
}
```



如果有多个记录位于最终位置，如何不参加下一趟排序？

关键问题

待排序序列



第一趟排序结果

✈️ **解决方法：** 设置变量bound表示一趟起泡排序的范围 $[1, \text{bound}]$ ，并且bound与上一趟起泡排序的最后交换的位置exchange之间的关系是 $\text{bound} = \text{exchange}$ 。

✈️ **算法描述：**

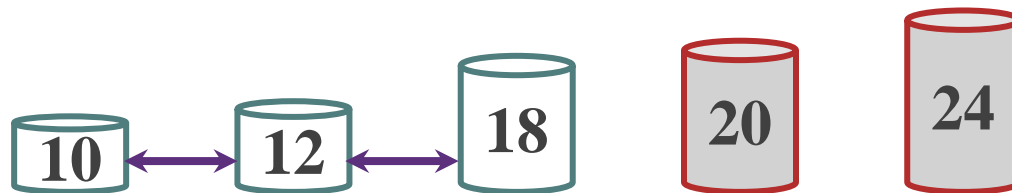
```
bound = exchange; exchange = 0;
for (j = 1; j < bound; j++)
    if (data[j] > data[j+1]){
        交换data[j]和data[j+1]; exchange = j;
    }
```



下一趟排序的范围是多少？

关键问题

某趟排序结果



下一趟排序结果



✈️ 解决方法：一趟排序没有交换，则表明整个序列已经有序

✈️ 算法描述：

```
while (exchange != 0)
{
    执行一趟起泡排序;
}
```

🕒 如何判别起泡排序的结束？

算法描述

```
void Sort :: BubbleSort( )
{
    int j, exchange, bound, temp;
    exchange = length - 1;
    while (exchange != 0)
    {
        bound = exchange; exchange = 0;
        for (j = 0; j < bound; j++)
            if (data[j] > data[j+1]) {
                temp = data[j];
                data[j] = data[j+1];
                data[j+1] = temp;
                exchange = j;
            }
    }
}
```

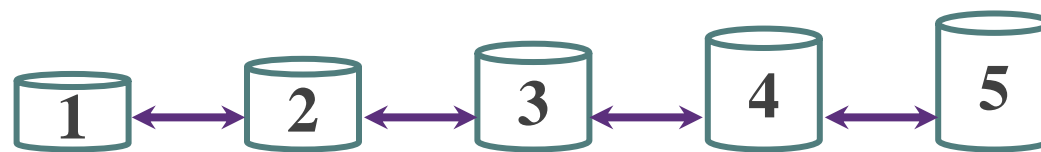
比较次数



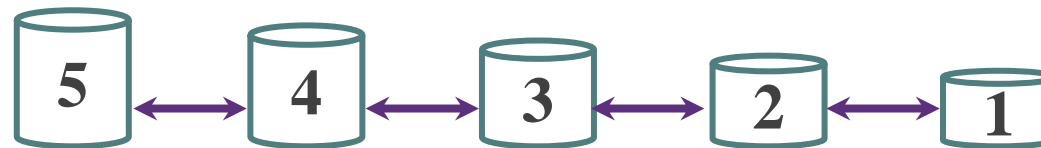
比较语句？ 执行次数？



最好情况： $n-1$ 次




最坏情况： $n-1 + n-2 + \dots + 1$ 次



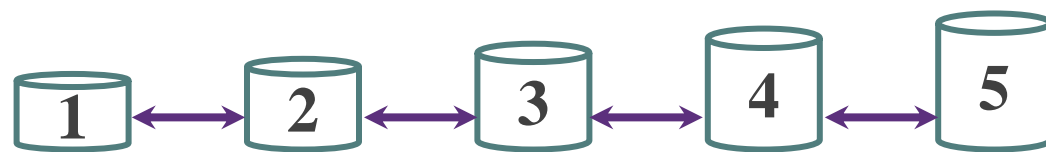
算法描述


```
void Sort :: BubbleSort( )
{
    int j, exchange, bound, temp;
    exchange = length - 1;
    while (exchange != 0)
    {
        bound = exchange; exchange = 0;
        for (j = 0; j < bound; j++)
            if (data[j] > data[j+1]) {
                temp = data[j];
                data[j] = data[j+1];
                data[j+1] = temp;
                exchange = j;
            }
    }
}
```

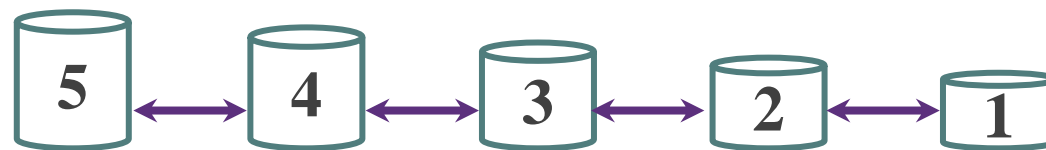
移动次数

 移动语句？ 执行次数？

 最好情况：0 次



 最坏情况：($n-1 + n-2 + \dots + 1$) 次



时间性能

📜 最好情况：正序 $O(n)$

📎 比较次数： $n-1$ 次

📎 移动次数：0次

📜 最坏情况：逆序 $O(n^2)$

📎 比较次数： $\sum_{i=1}^{n-1} n-i = \frac{n(n-1)}{2}$ 次

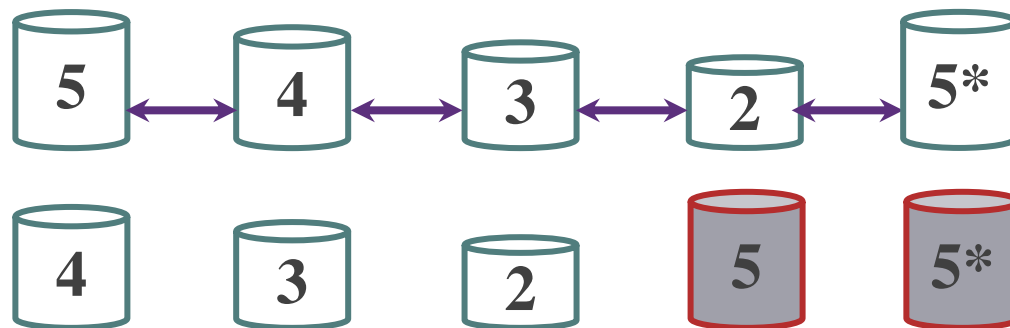
📎 移动次数： $\sum_{i=1}^{n-1} n-i = \frac{n(n-1)}{2}$ 次

📜 平均情况：随机排列， $O(n^2)$

空间性能

📜 空间性能： $O(1)$

📜 稳定性：稳定



```
if (data[j] > data[j+1]){  
    交换data[j]和data[j+1];  
    exchange = j;  
}
```

交换类排序

8-3-2 快速排序

改进的着眼点

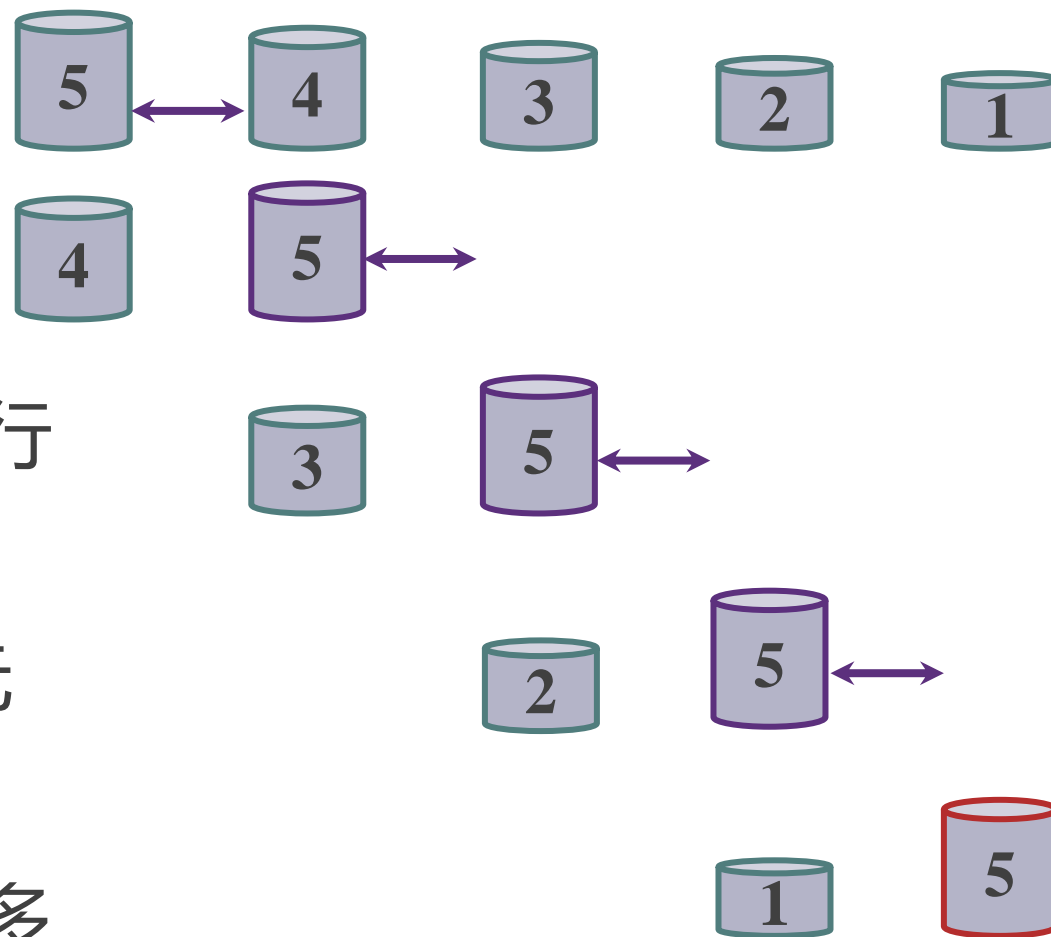
记录的比较在相邻单元中进行



每次交换只能右移一个单元



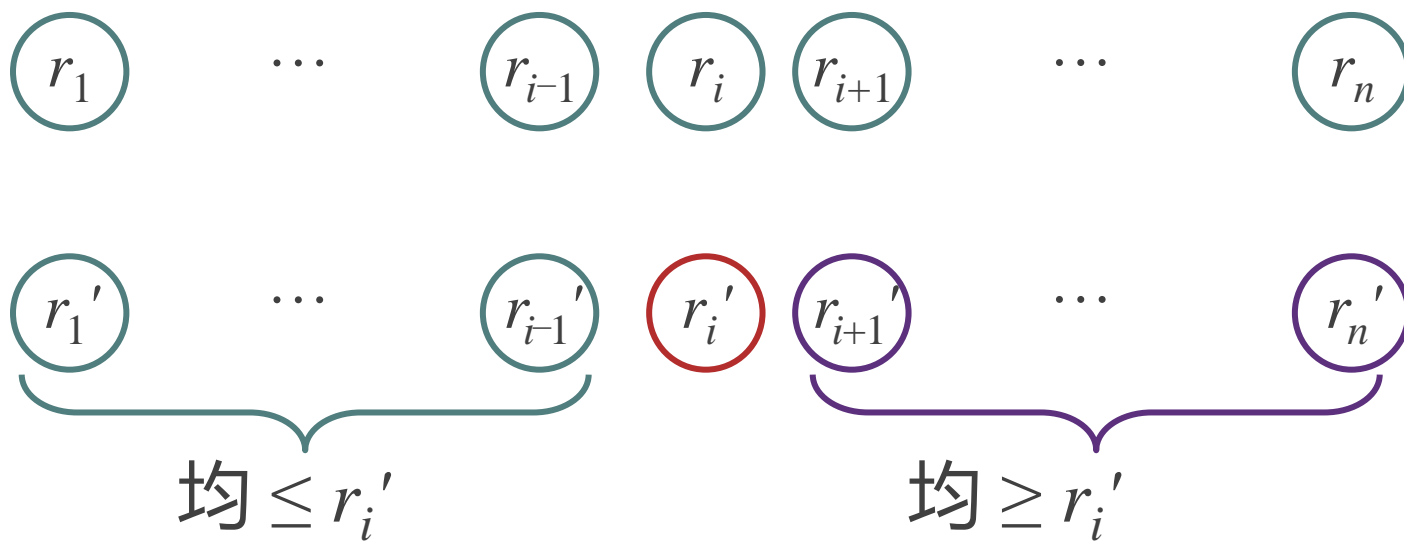
总的比较次数和移动次数较多



🕒 较大记录从前面直接移到后面，较小记录从后面直接移到前面？

基本思想

📎 快速排序的基本思想：选一个轴值，将待排序记录划分成两部分，左侧记录均小于或等于轴值，右侧记录均大于或等于轴值，然后分别对这两部分重复上述过程，直到整个序列有序。



运行实例

待排序序列



第一趟排序结果



第二趟排序结果



第三趟排序结果

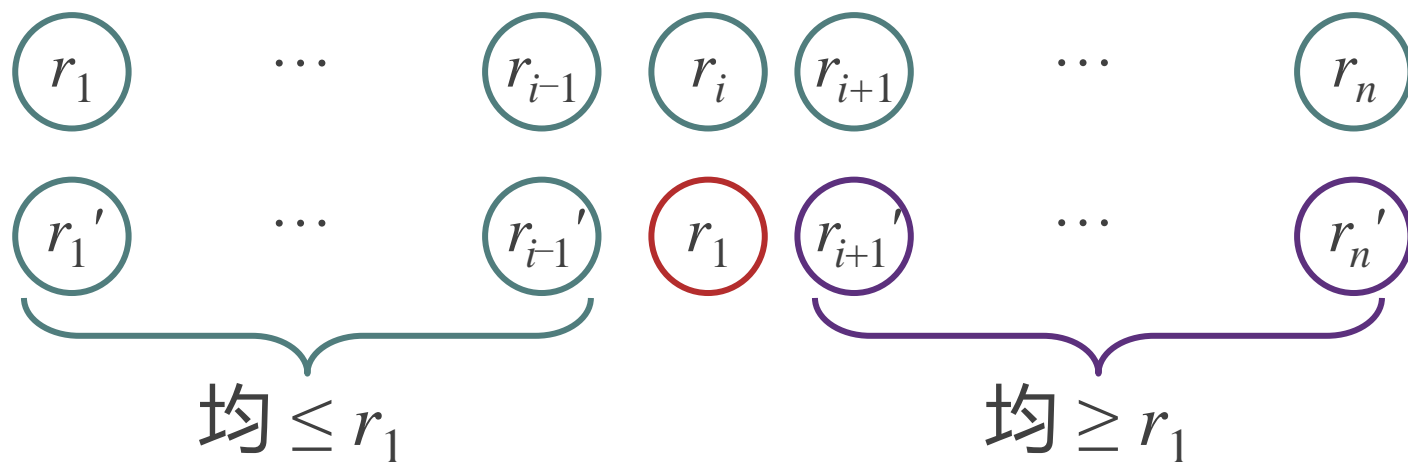


最终排序结果



一次划分

📌 一次划分：以轴值为基准将无序序列划分为两部分，即：



关键问题

待排序序列



一次划分结果



解决方法:

- (1) 第一个记录;
- (2) 随机选取;
- (3) 比较三个记录取值居中者;

决定排序的时间性能

决定两个子序列的长度

简单起见，取第一个记录作为轴值

🕒 如何选择轴值——比较的基准？选取不同轴值有什么后果？

关键问题

待排序序列



一次划分结果



减少了总的比较次数和移动次数

较小的记录一次就能从后面移到前面（较大的记录？）

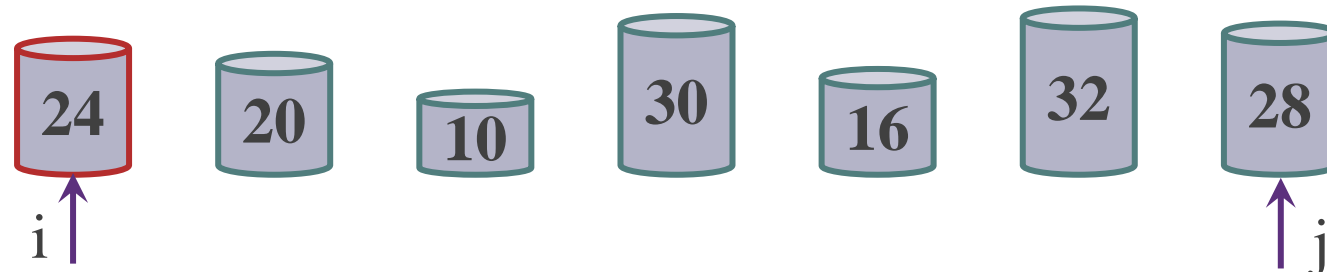
记录的比较和移动从两端向中间进行



如何实现一次划分——较大的记录移到后面，较小记录移到前面？

运行实例

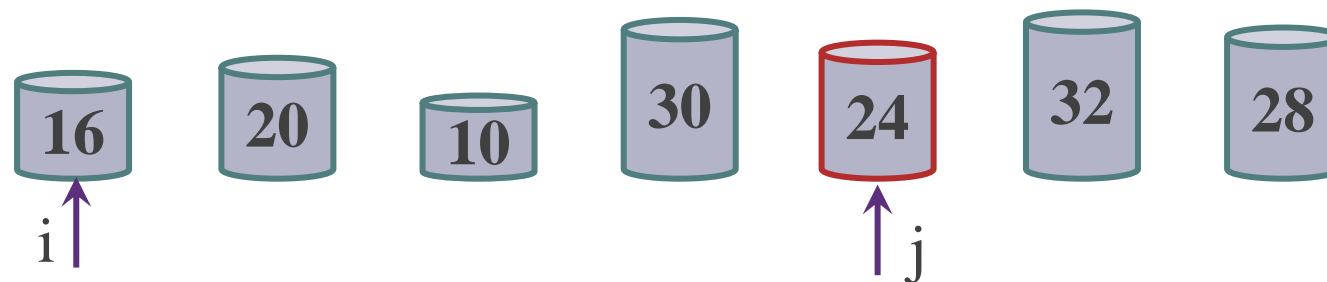
待排序序列



j 从后向前扫描
直到 $r[j] < r[i]$

交换 $r[j]$ 和 $r[i]$

$i++$



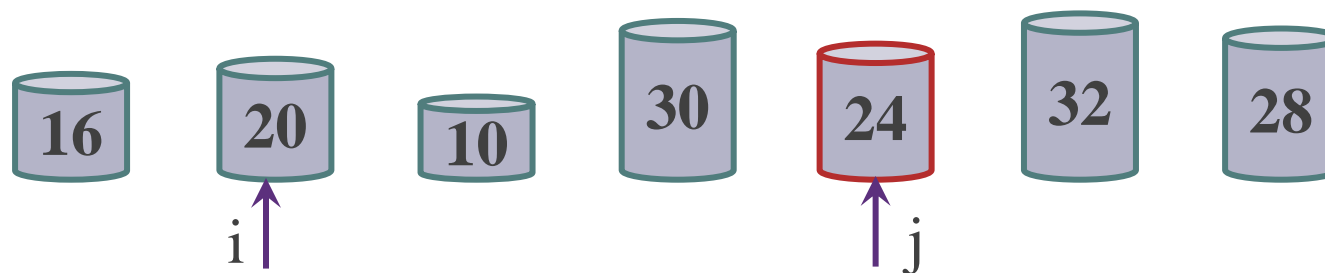
运行实例

待排序序列



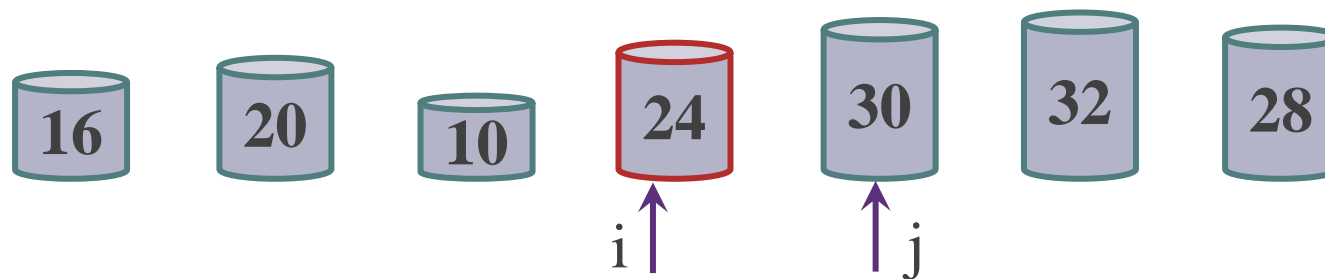
j 从后向前扫描
直到 $r[j] < r[i]$

交换 $r[j]$ 和 $r[i]$
 $i++$



i 从前向后扫描
直到 $r[j] < r[i]$

交换 $r[j]$ 和 $r[i]$
 $j--$



重复上述过程，直到 i 等于 j

算法描述

```
int Sort :: Partition(int first, int last)
```

```
{  
    int i = first, j = last, temp;
```

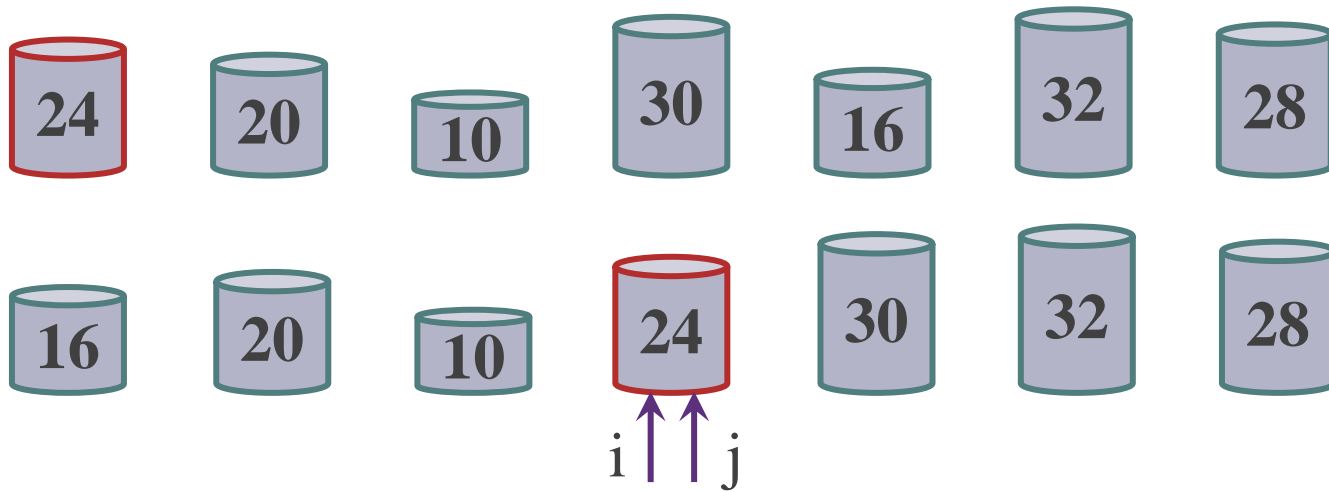


为什么设置形参first和last? 表示什么?

```
    while (i < j)
```

表示待划分区间[first, last], 是变化的

```
    {
```



```
    }
```

```
    return i;
```

```
}
```

/*i为轴值记录的最终位置*/

时间性能

```
int Sort :: Partition(int first, int last)
{
    int i = first, j = last, temp;
    while (i < j)
    {
        while (i < j && data[i] <= data[j]) j--;
        if (i < j) {
            temp = data[i]; data[i] = data[j]; data[j] = temp; i++;
        }
        while (i < j && data[i] <= data[j]) i++;
        if (i < j) {
            temp = data[i]; data[i] = data[j]; data[j] = temp; j--;
        }
    }
    return i;
}
```



时间复杂度是多少?

下标 i 和 j 共同将数组扫描一遍, $O(n)$

/*右侧扫描*/

/*左侧扫描*/

/*i为轴值记录的最终位置*/

关键问题

待排序序列



一次划分结果



解决方法:

递归执行快速排序

算法描述:

```
void Sort :: QuickSort (int first, int last )  
{  
    int pivot = Partition (first, last);  
    QuickSort (first, pivot-1);  
    QuickSort (pivot+1, last );  
}
```

🕒 如何处理一次划分得到的两个待排序子序列?

运行实例

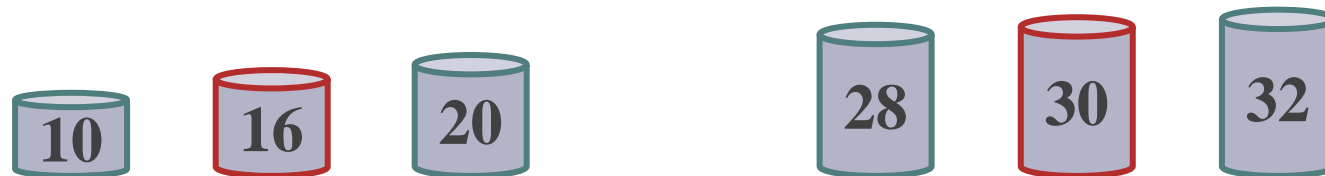
待排序序列



第一趟排序结果



第二趟排序结果



✈ 解决方法:

若待排序序列只有一个记录，即待划分区间长度为 1

`if (first == last) return;`



递归何时结束?

算法描述

```
void Sort :: QuickSort(int first, int last)
{
    if (first == last) return;           /*区间长度为1，递归结束*/
    else {
        int pivot = Partition(first, last);
        QuickSort(first, pivot-1);
        QuickSort(pivot+1, last);
    }
}
```

时间性能

```
int Sort :: Partition(int first, int last)
```

```
{
```

```
    int i = first, j = last, temp;
```

```
    while (i < j)
```

```
    {
```

```
        while (i < j && data[i] <= data[j]) j--;
```

```
        if (i < j) {
```

```
            temp = data[i]; data[i] = data[j]; data[j] = temp; i++;
```

```
        }
```

```
        while (i < j && data[i] <= data[j]) i++;
```

```
        if (i < j) {
```

```
            temp = data[i]; data[i] = data[j]; data[j] = temp; j--;
```

```
        }
```

```
    }
```

```
    return i;
```

```
}
```



比较语句？ 执行次数？



移动语句？ 执行次数？



取决于待排序序列的初始状态

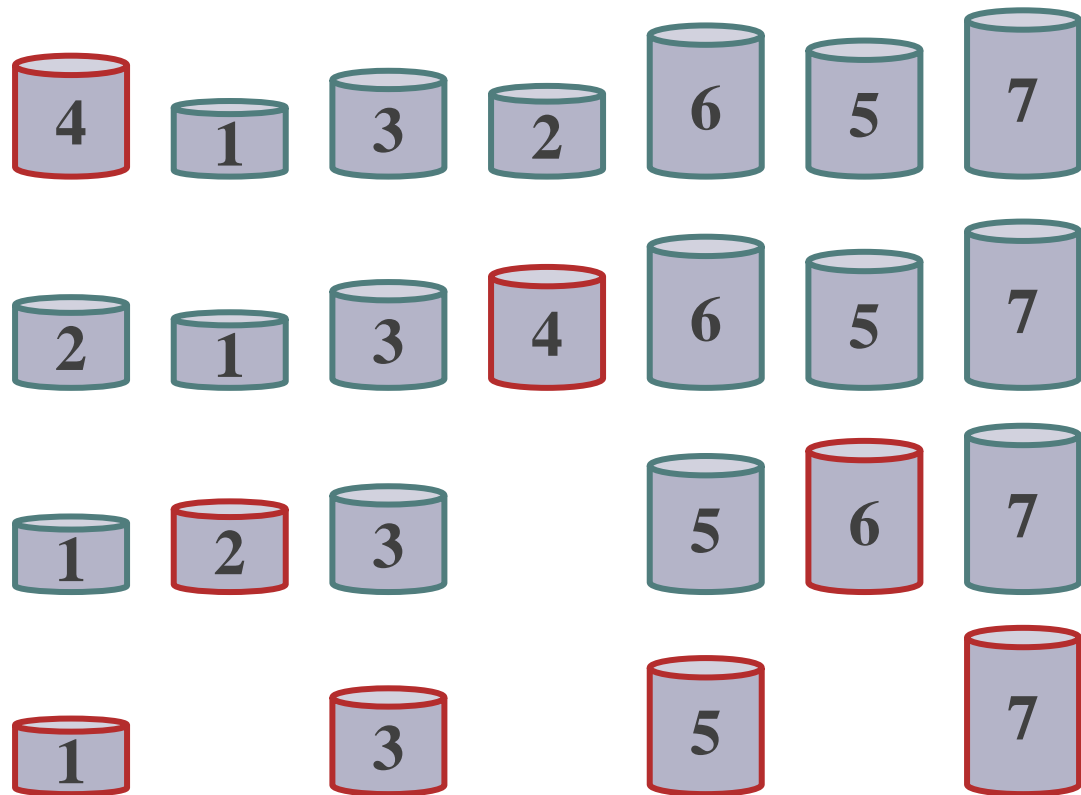
时间性能

📌 最好情况：每次划分的轴值均是中值

$O(n\log_2 n)$

📌 排序趟数： $\log_2 n$

📌 一趟排序： $O(n)$



时间性能

📜 最好情况：每次划分的轴值均是中值

$O(n\log_2 n)$

📎 排序趟数： $\log_2 n$

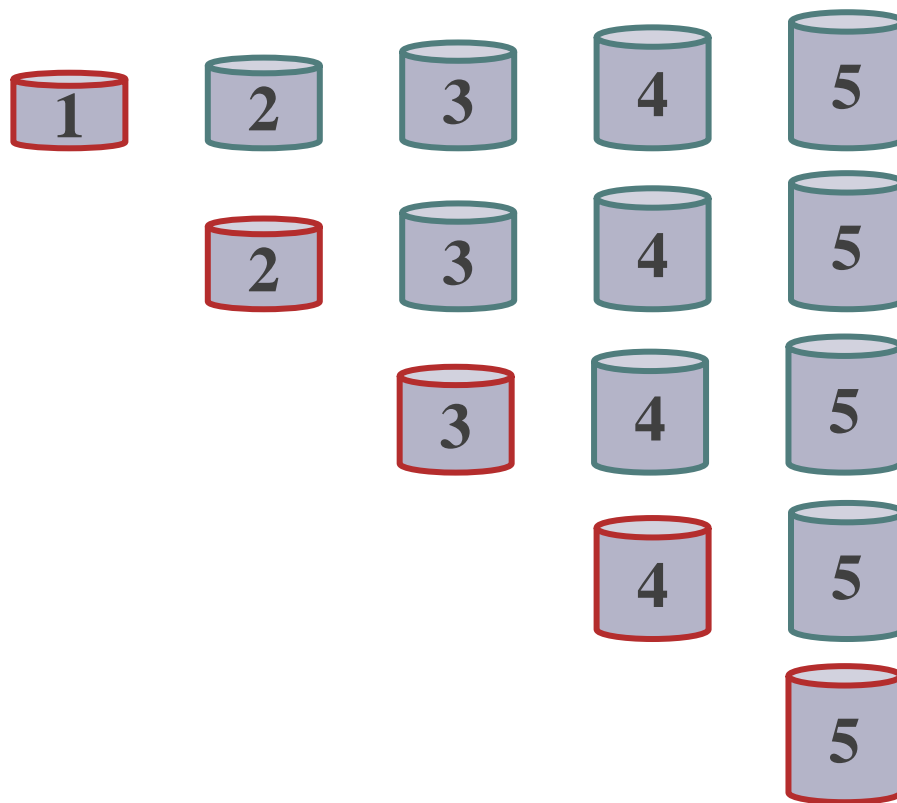
📎 一趟排序： $O(n)$

📜 最坏情况：正序、逆序 $O(n^2)$

📎 排序趟数： $n-1$

📎 一趟排序： $O(n)$

📜 平均情况： $O(n\log_2 n)$



空间性能

 空间性能: $O(\log_2 n) \sim O(n)$

 一次划分: $O(1)$

 递归深度: $O(\log_2 n) \sim O(n)$

 稳定性: 不稳定

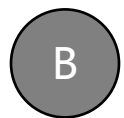
```
void Sort :: QuickSort (int first, int last)
{
    int pivot = Partition (first, last);
    QuickSort (first, pivot-1);
    QuickSort (pivot+1, last );
}
```

1. 在起泡排序过程中，交换记录在相邻单元中进行。



A

正确



B

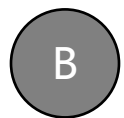
错误

提交

2. 起泡排序在最好情况下，没有发生交换记录的操作。



正确



错误

提交

3. 每一趟起泡排序只能确定一个记录的最终位置。

☐ A 正确

☒ B 错误

提交

4. 对于待排序序列{5, 4, 3, 2, 1}, 起泡排序移动记录的次数是 ()。

☐ A 10

☐ B 15

☐ C 20

☒ D 30

提交

5. 对于待排序记录序列{30, 25, 10, 12, 15, 20, 35}, 写出起泡排序每一趟的结果。

1. 对 n 个记录的集合进行快速排序，所需要的辅助空间是 $O(n)$ 。

☐ A 正确

☒ B 错误

提交

2. 快速排序每次只能确定一个记录的最终位置，因而时间性能较低。

☐ A 正确

☒ B 错误

提交

3. 当轴值是（ ）时，快速排序达到最好情况。

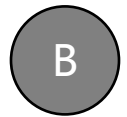
- ☐ A 第一个记录
- ☐ B 随机选取
- ☒ C 区间中值
- ☐ D 最后一个记录

提交

4. 快速排序一次划分的时间复杂度是 $O(n)$ 。



正确



错误

提交

5. 在快速排序的一次划分算法中，记录的比较次数取决于待排序的初始状态。

☐ A 正确

☒ B 错误

提交

6. 对于待排序记录序列{20, 15, 25, 18, 20*, 12, 30}, 写出一次划分的过程。