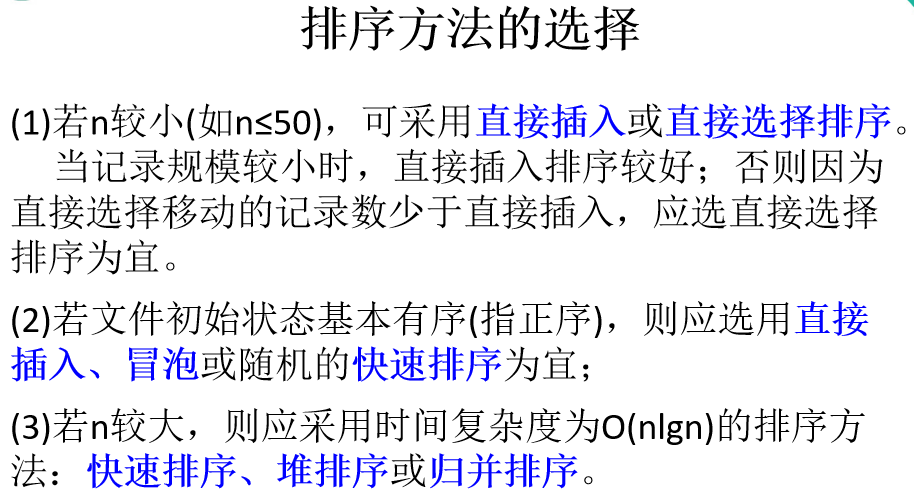
**数组排序**

* 插入排序
* 直接插入排序
* 折半插入排序
* Shell排序
* 交换排序
* 冒泡排序
* 快速排序（或分区交换排序）
* 选择排序
* 直接(简单)选择排序
* 堆排序
* 归并排序
* 基数排序



**1、冒泡排序**

* **排序思想：**

相邻两元素进行比较，如有需要则进行交换，每完成一次循环就将最大元素排在最后（如从小到大排序），下一次循环是将其它的数进行类似操作。

// 使用冒泡排序使数组元素从小到大排列  
// 循环的次数  
for (int i = 0; i < data.length - 1; i++) {  
 boolean flag = false;  
 // 比较的次数  
 for (int j = 0; j < data.length - 1 - i; j++) {  
 if (data[j].compareTo(data[j + 1]) > 0) {  
 DataWrap temp = data[j + 1];  
 data[j + 1] = data[j];  
 data[j] = temp;  
 flag = true;  
 }  
 }  
 System.***out***.println(java.util.Arrays.toString(data));  
 if (!flag) {  
 break;  
 }  
}

* **冒泡排序效率分析：**

算法的时间效率：从冒泡排序的算法可以看出，若待排序的元素为正序，则只需进行一趟排序，比较次数为n-1次，移动元素次数为0；若待排序的元素为逆序，则需要进行**n-1趟排序**，**比较次数为(n2-n)/2**，**移动次数为3(n2-n)/2**，因此**时间复杂度为O(n2)**。

算法的空间效率：空间效率很高，只需要一个附加程序单元用于交换，其空间效率为O(1)。

算法的稳定性：稳定

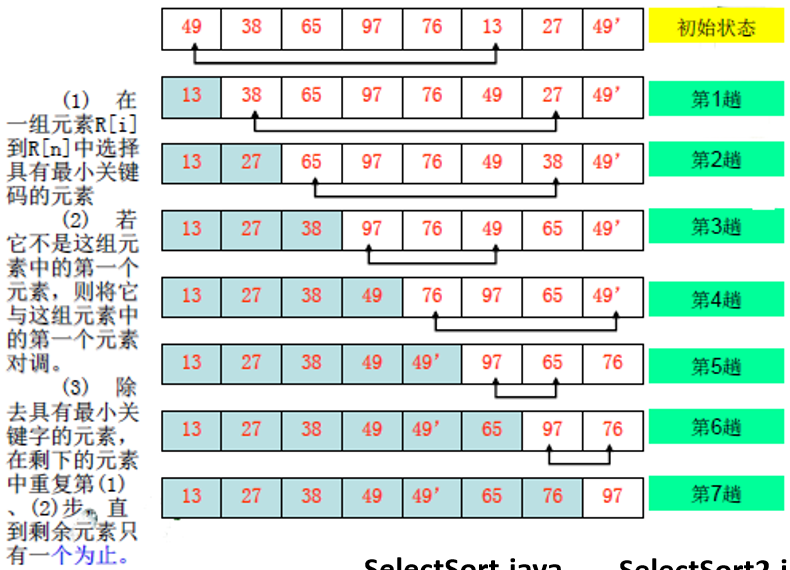
**2、直接选择排序**

* **选择排序基本原理：**

将待排序的元素分为已排序（初始为空）和未排序两种，依次将未排序的元素中值最小的元素放入已排序的组中。

直接选择排序简单直观，但性能略差；堆排序是一种较为高效的选择排序方法，但实现起来略微复杂。

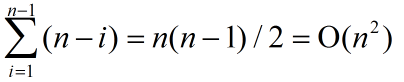
* **基本过程：**
* 在一组元素R[i]到R[n]中选择具有最小关键码的元素。
* 若它不是这组元素中的第一个元素，则将它与这组元素中的第一个元素对调。
* 除去具有最小关键字的元素，在剩下的元素中重复第（1）、（2）步，直到剩余元素只有一个为止。



// 使用直接选择排序使数组元素从小到大排列  
 // 循环次数  
 for (int i = 0; i < data.length - 1; i++) {  
 // 比较次数  
 for (int j = i + 1; j < data.length; j++) {  
 if (data[i].compareTo(data[j]) > 0) {  
 DataWrap temp = data[i];  
 data[i] = data[j];  
 data[j] = temp;  
 }  
 }  
 System.***out***.println(java.util.Arrays.toString(data));  
 }  
}

* **直接选择排序效率分析：**

算法的时间效率：无论初始状态如何，在第i趟排序中选择最小关键码，需做n-i次比较，因此总的比较次数为：



算法的空间效率：空间效率很高，只需要一个附加程序单元用于交换，其空间效率为O(1)。

算法的稳定性：不稳定。