

**比较类排序**：通过比较来决定元素间的相对次序，由于其时间复杂度不能突破O(nlogn)，因此也称为非线性时间比较类排序。

**非比较类排序**：不通过比较来决定元素间的相对次序，它可以突破基于比较排序的时间下界，以线性时间运行，因此也称为线性时间非比较类排序。

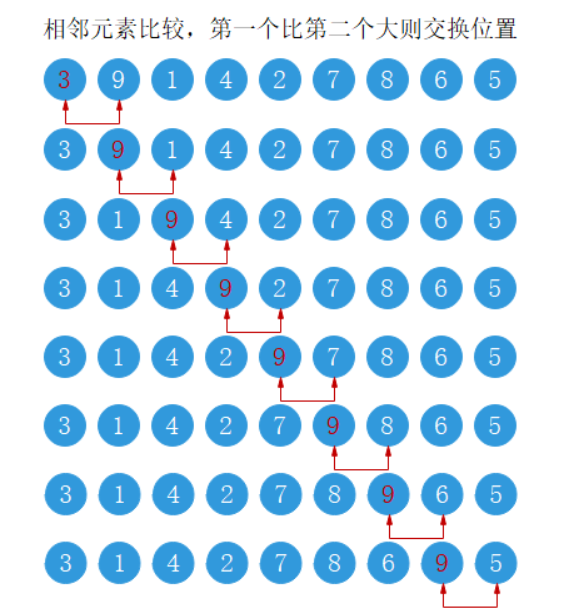
## 交换排序

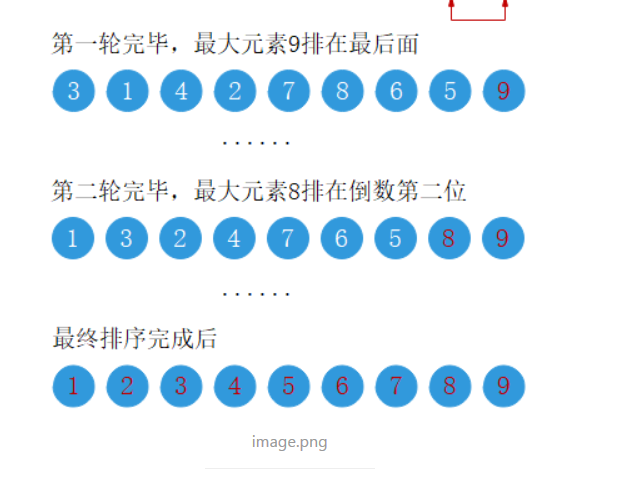
### 冒泡排序

原理：如同冒泡排序的名字一样，像汽水中的气泡会浮到顶端一样，此排序的每一次过程都会将最大元素浮到顶端。(从大到小)

实现过程：

1. 比较相邻元素，如果第一个比第二个大就交换位置。
2. 每一轮对每一对相邻的元素做同样的操作，直到最大值到达顶端
3. N个数两两相互比较每一轮并把最大值放在顶端因此只需要比较n-1轮
4. 由于每一轮比较的时候都会把最大的放在最顶端，因此下一轮比较只要到达其前一个就停止。因此每一轮比较个数n-i(i是第几轮)





//伪算法

Public function($data){

$len=count($data);

For($i=1;$i<len;$i++){//进行n-1轮冒泡

For($j=0;$j<len-$i;$j++)//剩余乱序的数

{

If($data[$j]>$data[$j+1])

{

$temp=$data[$j];

$data[$j]=$data[$j+1];

$data[$j+1]=$temp;

}

}

Return $data;

}

冒泡算法最好的情况正序

只需要进行n-1次比对

时间复杂度O(n)

最坏情况反序

执行n-1趟和n-i此比对

平均时间复杂度==O（N\*\*）

时间复杂度，空间复杂度计算过程：<https://blog.csdn.net/jsjwk/article/details/84315770>

果n无限大的时候，T(n) =  time(1+2n)中的常量1就没有意义了，倍数2也意义不大。因此直接简化为T(n) =  O(n) 就可以了。

时间复杂度计算：T(n) = O( f(n) )，其中f(n) 表示每行代码执行次数之和

两层循环，第1次遍历n次(n个元素)，第二次遍历n-1次，... 依次类推

N+(n-1)+(n-2)+…+1=n\*(n+1)/2=O(n\*\*);

空间复杂度： O（1）

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的一个量度，

如果没有new 数组都是O(1)

临时New 长度为n的数组 空间复杂度O(n)

### 快速排序

快速排序与冒泡排序的不同

快速排序：**在每一轮挑选一个基准元素，并让其他比它大的元素移动到数列一边，比它小的元素移动到数列的另一边，**

冒泡排序在每一轮只把一个元素冒泡到数列的一端

思路：递归+分治法

**两种实现方法**

1. **挖坑法：**

实现过程与思路:

1. 选定基准元素 pivot 该位置为index---为坑的位置
2. 设置两个指针i,j分别指向最左边和最右边
3. 第一轮从右边开始如果arr[j]大于等于基准元素，j前移即j--
4. 如果arr[j]小于基准元素，arr[i]=arr[j]

i指针后移即i++, index指向j,下一轮从左侧开始

1. arr[j]小于等于基准元素，i后移即i++
2. 如果arr[i]大于基准元素， arr[j]= a[i]

j指针前移即j--,index指向i,下一轮从右侧开始

7.i和j重复的时候arr[index]=基准元素

i,j重复时跳出大循环

递归思路

Quicksort（）

*//递归结束条件***if**($first>=$end){  
 **return** ;  
}**else**{  
 *//获取基准元素* $pivot=$this->parttition($arr,$first,$end);  
 $this->quicksort($arr,$first,$pivot-1);  
 $this->quicksort($arr,$pivot+1,$end);

}

}

*//寻找基准元素***public function** parttition($arr,$first,$end){  
 $pivot=$arr[$first];  
 $i=$first;  
 $j=$end;  
 *//坑的位置* $index=$first;  
 *//外层循环在左右指针重合或交错时结束* **while**($i<$j){  
 *//右侧扫描* **while**($i<$j){  
 *//右侧指针值大于等于基准元素，右侧指针前移* **if**($arr[$j]>=$pivot){  
 $j--;  
 }  
 **else**{*//右侧指针值小于基准元素，* $arr[$i]=$arr[$j];  
 $index=$j;  
 $i++;*//左侧指针后移* **break**;  
 }  
 }  
 *//左侧扫描* **while**($i<$j){  
 *//左侧指针值小于等于基准元素，左侧指针后移* **if**($arr[$i]<=$pivot){  
 $i++;  
 }**else**{*//左侧指针值小于基准元素，* $arr[$j]=$arr[$i];  
 $index=$i;  
 $j--;  
 **break**;  
 }  
 }  
 }  
 $arr[$index]=$pivot;  
 **return** $index;  
}

1. 指针交换法
2. 选定基准元素 pivot 该位置为index-
3. 设置两个指针i,j分别指向最左边和最右边
4. 每一轮从右侧开始，如果arr[j]>=基准元素j前移即j--，如果arr[j]<基准元素，就跳到左指针，如果arr[i]<=基准元素i后移即i++，如果arr[i]<基准元素则交换a[i],a[j]的值------简单地说即使每一轮从又开始右边找到第一个小于基准元素的值，左边找到第一个大于基准元素的值然后交换
5. 当i==j的时候将基准元素和arr[start]（arr的第一个元素）交换
6. *//指针交换法***public function** quicksort2(&$arr,$first,$end){  
    *//递归结束条件* **if**($first>=$end){  
    **return** ;  
    }**else**{  
    $pivot=$this->parttition2($arr,$first,$end);  
    $this->quicksort2($arr,$first,$pivot-1);  
    $this->quicksort2($arr,$pivot+1,$end);  
    }  
   }  
   **public function** parttition2($arr,$first,$end){  
    $pivot=$arr[$first];  
    $i=$first;  
    $j=$end;  
    *//外层循环在左右指针重复的时候结束* **while**($i!=$j){  
    *//右侧扫描* **while**($i<$j&&$arr[$j]>=$pivot){  
    $j--;  
    }  
    *//左侧扫描* **while**($i<$j&&$arr[$i]<=$pivot){  
    $i++;  
    }  
    *//交换left和right指向的元素* **if**($i<$j){  
    $temp=$arr[$i];  
    $arr[$i]=$arr[$j];  
    $arr[$j]=$temp;  
    }  
    }  
    *//左右指针重合 替换基准元素* $temp=$arr[$i];  
    $arr[$i]=$arr[$first];  
    $arr[$first]=$temp;  
    **return** $i;  
   }

参考 ：<http://www.sohu.com/a/246785807_684445>

<http://www.xinyueseo.com/algorithm/209.html>

## 插入排序

### 直接插入排序

如同打扑克牌时整理扑克牌一样，将扑克牌分成两个不同部分，一边是已经排好序的(3,5,8)，一边没有排好序(4,6,9)，我们将现将未排序的第一个元素4放入已排好序的进行比较找到比他小的元素并将4放在他后面一次类推。

对于一个无序数组，一开始我们将数组第一个元素视为已排好序，后面的为无序部分。用无序部分第一个元素在以排序好的部分从后往前进行比较，如果已排序好的元素大于无需元素，则以排序好的元素向后移动，无序元素插入到原来有序元素位置。

*//直接插入排序***public function** insertsort($arr){  
 **for**($i=1;$i<*count*($arr);$i++){*//无序部分，默认第一个是有序的  
 //备份待插入元素* $waitInsert=$arr[$i];  
 $j=$i-1;  
 **for**($j;$j>=0&&$arr[$j]>$waitInsert;$j--){*//有序元素部分大于待插入元素，有序元素后移* $arr[$j+1]=$arr[$j];  
 }  
 $arr[$j+1]=$waitInsert;  
 }  
 **return** $arr;  
}

参考：https://www.itcodemonkey.com/article/3290.html

### 希尔排序：

以增量d(d=length/2)对arr数组进行分组，然后对每组进行直接插入排序，然后增量d=d/2,再次对每组进行直接插入排序直至d=1.

*//希尔排序***public function** shellsort($arr,$n){  
 **for**($d=$n/2;$d>=1;$d=$d/2){  
 **for**($i=$d;$i<$n;$i++){  
 $waitInsert=$arr[$i];  
 $j=$i-1;  
 **for**($j;$j>=0&&$arr[$j]>$waitInsert;$j--){  
 $arr[$j+1]=$arr[$j];  
 }  
 $arr[$j+1]=$waitInsert;  
 }  
 }  
 **return** $arr;  
}

时间复杂度：O(n\*)~O(nlog2n)

## 选择排序

### 选择排序

每趟排序选出最小的元素放在最前面

*//选择排序***public function** selectsort($arr){  
 **for**($i=0;$i<*count*($arr)-1;$i++){*//排序所需趟数* $min=$i;  
 **for**($j=$i+1;$j<*count*($arr);$j++){  
 **if**($arr[$j]<$arr[$min]){  
 $min=$j;  
 }  
 }  
 *//每趟和待排序列第一个交换最小值* $temp=$arr[$i];  
 $arr[$i]=$arr[$min];  
 $arr[$min]=$temp;  
  
 }  
 **return** $arr;  
}

### 堆排序