一、冒泡排序（Bubble Sort）

它重复地走访过要排序的数列，依次比较两个元素，如果他们的顺序错误就把他们交换过来；走访数列的工作是重复地进行直到没有再需要交换；这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端

与前面的比较，交换与否

1、步骤

（1）比较相邻的元素：如果第一个比第二个大，就交换他们两个

（2）对每一对相邻元素作同样的工作：从开始第一对到结尾的最后一对，最后的元素应该会是最大的数

（3）针对所有的元素重复以上的步骤，除了最后一个

（4）持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较

2、代码

```php
//冒泡排序
function bubbleSort($arr){
    $len = count($arr);
    //该层循环控制 需要冒泡的轮数
    for($i=1;$i<$len;$i++){
        //该层循环用来控制每轮 冒出一个数 需要比较的次数
        for($k=0;$k<($len-$i);$k++){
            if($arr[$k] > $arr[$k+1]){
                $tmp = $arr[$k+1]; //声明一个临时变量
                $arr[$k+1] = $arr[$k];
                $arr[$k] = $tmp;
            }
        }
        //var_dump($arr);
    }
    return $arr;
}
```

二、快速排序

快速排序是由东尼·霍尔所发展的一种排序算法。在平均状况下，排序 n 个项目要 O (n log n)次比较。在最坏状况下则需要 O (n2)次比较，但这种状况并不常见。事实上，快速排序通常明显比其他 O (n log n) 算法更快，因为它的内部循环（inner loop）可以在大部分的架构上很有效率地被实现出来，且在大部分真实世界的数据，可以决定设计的选择，减少所需时间的二次方项之可能性

重点是基准值，递归

1、步骤

（1）从数列中挑出一个元素，称为 “基准”（pivot）

（2）重新排序数列：所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）；在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作

（3）递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序

2、代码

```php
//快速排序
function quick_sort($arr){
    //判断参数是否是一个数组
    if(!is_array($arr)){
        return false;
    }
    //递归出口:数组长度为1，直接返回数组
    $length = count($arr);
    if($length<=1){
        return $arr;
    }
    //数组元素有多个,则定义两个空数组
    $left = $right = array();
    //使用for循环进行遍历，把第一个元素当做比较的对象
    for($i=1;$i<$length;$i++){
        //判断当前元素的大小
        if($arr[$i] < $arr[0]){
            $left[]= $arr[$i];
        }else{
            $right[]= $arr[$i];
        }
    }
    //递归调用
    $left= quick_sort($left);
    $right = quick_sort($right);
    //将所有的结果合并
    return array_merge($left,array($arr[0]),$right);
}
```

三、选择排序(Selection sort)

实现思路：双重循环完成，外层控制轮数，当前的最小值；内层控制的比较次数

重点在最小值

1、步骤

（1）首先在未排序序列中找到最小元素，存放到排序序列的起始位置

（2）再从剩余未排序元素中继续寻找最小元素，然后放到排序序列末尾

（3）以此类推，直到所有元素均排序完毕

2、代码

```php
//选择排序
function select_sort($arr){
    //$i：当前最小值的位置，需要参与比较的元素
    $len = count($arr);
    for($i=0;$i<($len-1);$i++){
        //先假设最小的值的位置
        $p = $i;
        //$j：当前都需要和哪些元素比较，$i后边的
        for($j=$i+1;$j<$len; $j++){
            //$arr[$p]：当前已知的最小值
            if($arr[$p] > $arr[$j]){
                //比较发现更小的,记录下最小值的位置；并且在下次比较时，应该采用已知的最小值进行比较
                $p = $j;
            }
        }
        //上述已经确定了当前的最小值的位置，保存到$p中
        //如果发现最小值的位置与当前假设的位置$i不同，则位置互换即可
        if($p != $i){
            $tmp = $arr[$p];
            $arr[$p] = $arr[$i];
            $arr[$i] = $tmp;
        }
    }
    return $arr;
}
```

四、插入排序（Insertion Sort）

工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

插入排序在实现上，通常采用 in-place 排序（即只需用到 O(1)的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间

遍历元素与前面的元素比较，根据结果执行交换与否，直到遇到否的情况则终止内部遍历

1、步骤

（1）从第一个元素开始，该元素可以认为已经被排序

（2）取出下一个元素，在已经排序的元素序列中从后向前扫描

（3）如果该元素（已排序）大于新元素，将该元素移到下一位置

（4）重复步骤 3，直到找到已排序的元素小于或者等于新元素的位置

（5）将新元素插入到该位置中

（6）重复步骤 2

2、代码

```php
//插入排序
function insert_sort($arr){
    $len=count($arr);
    for($i=1;$i<$len;$i++){
        //获得当前需要比较的元素值
        $tmp = $arr[$i];
        //内层循环控制比较并插入
        for($j=$i-1;$j>=0;$j--){
            //$arr[$i]：需要插入的元素 $arr[$j]：需要比较的元素
            if($tmp < $arr[$j]){
                //发现插入的元素要小，交换位置 将后边的元素与前面的元素互换
                $arr[$j+1] = $arr[$j];
                //将前面的数设置为 当前需要交换的数
                $arr[$j] = $tmp;
            }else{
                //如果碰到不需要移动的元素,由于是已经排序好是数组，则前面的就不需要再次比较了
                break;
            }
        }
    }
    return $arr;
}
```

五、其他