前提：分别用冒泡排序法，快速排序法，选择排序法，插入排序法、二叉树、迷宫算法将下面数组中的值按照从小到大的顺序进行排序。

**$arr(1,43,54,62,21,66,32,78,36,76,39);**

1. 冒泡排序 思路分析：在要排序的一组数中，对当前还未排好的序列，从前往后对相邻的两个数依次进行比较和调整，让较大的数往下沉，较小的往上冒。即，每当两相邻的数比较后发现它们的排序与排序要求相反时，就将它们互换。

代码实现：

$arr=**array**(1,43,54,62,21,66,32,78,36,76,39);

**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];

$arr[$k+1]=$arr[$k];

$arr[$k]=$tmp;

}

}

}

**return** $arr;

}

1. 选择排序 思路分析：在要排序的一组数中，选出最小的一个数与第一个位置的数交换。然后在剩下的数当中再找最小的与第二个位置的数交换，如此循环到倒数第二个数和最后一个数比较为止。

代码实现：

**function** **selectSort**($arr) {

*//双重循环完成，外层控制轮数，内层控制比较次数*

$len=count($arr);

**for**($i=0; $i<$len-1; $i++) {

*//先假设最小的值的位置*

$p = $i;

**for**($j=$i+1; $j<$len; $j++) {

*//$arr[$p] 是当前已知的最小值*

**if**($arr[$p] > $arr[$j]) {

*//比较，发现更小的,记录下最小值的位置；并且在下次比较时采用已知的最小值进行比较。*

$p = $j;

}

}

*//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;

}

}

*//返回最终结果*

**return** $arr;

}

3.插入排序 思路分析：在要排序的一组数中，假设前面的数已经是排好顺序的，现在要把第n个数插到前面的有序数中，使得这n个数也是排好顺序的。如此反复循环，直到全部排好顺序。

代码实现：

**function** **selectSort**($arr) {

*//双重循环完成，外层控制轮数，内层控制比较次数*

$len=count($arr);

**for**($i=0; $i<$len-1; $i++) {

*//先假设最小的值的位置*

$p = $i;

**for**($j=$i+1; $j<$len; $j++) {

*//$arr[$p] 是当前已知的最小值*

**if**($arr[$p] > $arr[$j]) {

*//比较，发现更小的,记录下最小值的位置；并且在下次比较时采用已知的最小值进行比较。*

$p = $j;

}

}

*//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;

}

}

*//返回最终结果*

**return** $arr;

}

*//插入排序法(小到大排序) 效率又比 选择排序法要高一些*

**function** **insertSort**(&$arr){

*//先默认下标为0的这个数已经是有序*

**for**($i=1;$i<count($arr);$i++){

*//$insertVal是准备插入的数*

$insertVal=$arr[$i];

*//准备先和谁下标为$inserIndex的比较*

$inserIndex=$i-1;

*//如果这个条件满足，说明我们还没有找到适当的位置*

**while**($inserIndex >= 0 && $insertVal < $arr[$inserIndex]){

*//同时把数后移*

$arr[$inserIndex+1] = $arr[$inserIndex];

$inserIndex--;

}

*//插入（这时就给$inserIndex找到适当的位置）*

$arr[$inserIndex+1] = $insertVal;

}

}

4.快速排序  
思路分析：选择一个基准元素，通常选择第一个元素或者最后一个元素。通过一趟扫描，将待排序列分成两部分，一部分比基准元素小，一部分大于等于基准元素。此时基准元素在其排好序后的正确位置，然后再用同样的方法递归地排序划分的两部分。

代码实现：

**function** **quickSort**($arr) {

*//先判断是否需要继续进行*

$length = count($arr);

**if**($length <= 1) {

**return** $arr;

}

*//选择第一个元素作为基准*

$base\_num = $arr[0];

*//遍历除了标尺外的所有元素，按照大小关系放入两个数组内*

*//初始化两个数组*

$left\_array = **array**(); *//小于基准的*

$right\_array = **array**(); *//大于基准的*

**for**($i=1; $i<$length; $i++) {

**if**($base\_num > $arr[$i]) {

*//放入左边数组*

$left\_array[] = $arr[$i];

} **else** {

*//放入右边*

$right\_array[] = $arr[$i];

}

}

*//再分别对左边和右边的数组进行相同的排序处理方式递归调用这个函数*

$left\_array = quick\_sort($left\_array);

$right\_array = quick\_sort($right\_array);

*//合并*

**return** array\_merge($left\_array, **array**($base\_num), $right\_array); *// 注意这里的索引键会重新排列。*

}