1. **冒泡排序**

基本思路：数组中定义需要排的区域A,那么可知A的尾指针初始为arr.length-1, 每次都从0开始遍历区域A，不包括该区域的最后一个点（因为对比的是当前数和其下一个数，因此遍历到B区域倒数第二个位置就行）。将当前数和其下一个数对比，如果小就不变，指针往后一位，如果大则交换，指针往后一位，这样保证大的数被一直推到末尾。然后A区域缩小一位（尾指针自减1）。一直循环直到尾指针到达1的位置（说明还剩两个数），进行最后一次排序。

1. **选择排序**

基本思路：数组中定义需要排序区域A,定义A区域头指针i，将i作为存放数组最小值的地方，遍历i+1开始的所有数(到arr.length-1)与i相比，有小的则换过来直到数组结束。然后A区域缩小，即i++,进入下一次循环。当i到达数组倒数第二个数的时候说明数组剩下两个数，进行最后一次排序即可。

1. **插入排序**

基本思路：将数组分为两个部分，前部分为有序区域，后部分为无序区域设为A.初始的情况是有序区只有一个数（单个数可以认为必定有序）。定义Aq区域头指针i ，i-1就是有序区域了。让i和有序区域的值进行对比，如果比起小就交换，知道有序区的第一个数，中间遇到比其大的情况就停止遍历，将i++，缩小无序区域，直到i=arr.length-1说明到无序的只剩一个数了，进行最后一操作。

1. **归并排序**
   1. merge函数（外排函数）

给定一个数组和其分界点，将其分为左右两个部分，这两个部分都是各自有序了，要求将整个数组排好序。

思路：新建一个辅助数组长度和该数组长度一样，定义两个指针，分别指向左边子数组和右边子数组头部，对比指针所对应的数，哪个指针指的数小就将其放入辅助数组中，然后指针向后移动一位（注意另一个指着不动），然后进行下一次对比，一直到其中有一个数组已经走完（此时另一个数组必定还没走完，因为每次只动一个指针），这时候就将没走完的数组的剩余部分依次放入辅助数组中。最后将辅助数组复制回原数组，排序完成。

* 1. 利用分治的思想

引例：求一个数组的最大值。利用分治的思想来做：就是将一个数组不断地一分为二，直到不能再分。具体是找到数组的中点将其分为左右两个部分，求其各自的最大值，然后对比找出两个较大者即为整个数组最大值。而求其左右部分的最大值时候明显可以重复该过程（利用递归函数来做），一直分下去，终止条件就是数组不能再分下去了，只剩下一个数了直接返回该数（剩下一个数可以认为就是最大值了）。

回到排序的题目中，也是先找到数组的中点，分为左右两个部分，将其各自排序，然后得到左右都各自排序的数组，利用merge函数可以将整个排好序。而左右部分排序也是重复这样分割的过程，直到剩下一个数就不能在分了，此时只有一个值也可以视为有序了，直接返回（不是返回该数，而是函数返回）。返回到上一层函数中用merger函数的整合。

3、归并函数应用：小和问题和逆序对  
 **求小和问题**：在随机元素，随机数组大小的数组中，找出左边比右边元素小的所有元素之和。

思路：利用分治的思想，将数组分为左右两个部分，分别求出左右部分各自的小和，然后自出合并后新增的小和，将这三部分加起来就能得到整个数组的小和。（注意分治到最后剩下一个数的时候没有小和，返回0）

求小和的过程可以这样理解：就是对数组中的每一个数i,找到其后面比他大的数，假设个数为N,则i产生的小和就是i\*N,整个数组的小和就是每个元素的小和累加。因此在merge的过程中，保持原来排序操作不变的情况下加上小和累加过程即可。就是当发现左指针的数小于右指针的数的时候(这个时候左右数组已经都是排好序了，因此认为右指针及其后面所有数都大于左指针的数，该数乘以个数可以得到其小和，累加到变量中)。

**逆序对问题：**和小和问题一样，只是对数组的每一个元素，找其后面比他小的所有数分别组成逆序对。同样在merge函数中进行就行。只是把排序换成从大到小排就行

**总的来说这两个问题就是在归并排序的merge过程中加入相应的处理，其他都不变。**

1. **快速排序**
   1. partition函数（荷兰国旗问题）  
       对于一个数组我们定义小于区域和大于区域（指针分别为less和more）和用来区分的数num,例如对于L-R的数组，less和more的初始值为L-1和R+1代表了这两个区域为空，定义cur指针遍历数组，如果cur上的数小于num就和小于区域的下一个位置上的数交换，小于区域扩大一位，然后遍历的指针找下一个即cur++，如果等于num就不作处理，遍历的指针找下一个cur++,如果大于就与大于区域的前一个数交换，大于区域扩大一个数（注意此时cur不指向下一个，一位从大于区域前一个数换过来的不知道其大小情况，应该对该数进行判断）。最后的终止条件就是cur>=more。这样数组最终就变成了小于区域在左边，等于区域在中间，大于区域在右边，返回等于区域的左右边界（用来确定小于区域和大于区域）

**应用：利用partition的思路能解决剑指offer中：对数组操作是的奇数在偶数之前的问题。**

* 1. 经典快排：经典快速排序主要利用了partition函数。对于一个数组，把自己的最后一个元素作为区分的依据，（就是充当partition函数中的num参数），然后进行了一次partition后就分成上述的三个区域，等于区域就不用动了，对小于区域和大于区域进行相同的操作，是一个递归过程，终止条件是当要操作的区域只有一个数或者不存在，直接返回。
  2. 随机快排：经典快排受到数据状况的限制，随机快排就是在经典快排的基础上对区分的依据（原来都是最后一个数来区分）进行改进，在每次partition之前，利用随机函数在数组长度内生成随机的指针，然后将这个指针的数和最后一个元素交换，这样数组的最后一个元素就有可能是数组中任意位置的数，然后进行经典快排。

1. **堆排序**
   1. 堆结构：堆其实就是一个完全二叉树，在程序中以数组的形式存在，其关系为：设当前节点指针i,则其父节点为（i-1）/2,左子节点为（2\*i+1）,右子节点为（2\*i+2），还有重要的一点是堆的大小为size,当指针i**<size**的时候才认为是堆中的元素。
   2. 大根堆和小根堆：整个树以及其子树中，父节点都大于其左右子节点为大根堆，同理如果都父节点小于其两个子节点为小根堆
   3. HeapInsert函数（在堆中插入新的元素）  
       对每个新加入的元素判断，如果插入的元素大于其父节点，就和父节点交换，一直循环操作指导堆顶或者小于其父节点  
       因此，遍历一个数组，将其每个元素都通过heapInsert函数加入堆中，每加入一个元素就让size++,就能创建一个大根堆。
   4. Heapify函数（如果堆中某个位置变小了，就可能不是大根堆了，该函数 将其恢复到大根堆）。  
       思路：设该位置为index,则首先找其左节点left（如果右节点存在，那么左节点必定存在，根据完全二叉树的特性得知，因此先找左）,判断left在堆中吗（和size对比），如果不在说明该位置已经是叶节点了，不必进行任何操作，已经是大根堆了。如果存在，left+1找到其右节点判断是不是存在right,存在的话就取这两个的最大值和index的数进行对比，看index是不是比子节点的最大值大，是的话跳出循环，已经是大根堆，不是的话就和子节点中的较大节点互换位置，更新index的值，进入下一次循环。总的来说相当于大根堆某个位置变小了就进行判断让其不断下沉。
   5. 堆排序：首先遍历数组，让整个数组形成大根堆（此时最大值为堆顶，索引为0的位置），然后进入循环，将堆顶元素和最后一个位置的元素互换，缩小堆大小size,这样数组中最大的元素就出现在最末端，而且不属于堆中（就是把堆顶弹出），而此时的大根堆结构可能被破坏，因为堆顶位置换了数，比原来小了，因此进行heapify函数将其重新整理成大根堆。接下来就进入第二次循环。。。最终到了堆只剩下一个元素的时候循环终止，得到已经排序好的数组。
2. **六种基本排序的总结**
3. **时间复杂度和空间复杂度:**

冒泡排序：时间复杂度O(N^2)，空间复杂度O(1)

选择排序：时间复杂度O(N^2)，空间复杂度O(1)

插入排序：时间复杂度O(N^2)，空间复杂度O(1)（与前两种不一样的是还受到数据状况的影响，如果已经是有序的就是O(N),如果是逆序的就是O(N^2)。

归并排序：时间复杂度O(N\*logN),空间复杂度O(N)

随机快排：时间复杂度O(N\*logN),空间复杂度O(logN)（空间主要体现在记录等于区域的两个边界上）

堆排序：时间复杂度O(N\*logN),空间复杂度O(1)

1. **排序的稳定性:**

排序稳定性指的就是排序的时候对相同的元素会不会改变其相对位置，如果改变了就是稳定的，如果没改变就是稳定的

上述排序中稳定的有：冒泡排序，插入排序，归并排序，不稳定：选择排序，快速排序，堆排序

1. **工程上的选择：**

一般工程上是多种排序算法的结合，比如插入排序在数据量少的情况下是首选，因为数据量少的时候时间复杂度就不会差太多，拼的就是常数项，插入排序的常数项更小。而大数据量情况下如果是要求稳定的就选归并排序，如果不要求稳定的首选快排。

**补充说明：**归并排序的空间复杂度可以做到O(1),叫内部缓存法，快速排序可以做到稳定排序，这些都是论文级别的，不需要深究。

1. **基于非比较的排序：**

桶排序，计数排序，基数排序，这些排序和数据状况有关，一般不常用。

桶排序的思路：假设所要排序的数据的范围是0-N,就准备N个桶（其实就是一个数组），每个桶对应一0-N中的一个数，遍历要排序的数组，如果和桶的编号一致，那么桶的计数变量加一，一直到遍历完，那么就可以通过桶的编号及其各自的计数信息还原出排序好的数组。

桶排序的时间复杂度为O(N)，好于上述六个排序

重要题目：基于桶排序思路的应用。给定一个未排序的数组，求如果数组排序之后，相邻数的最大差值。要求时间复杂度为O(n),且要求不能用非基于比较的排序。

思路：最容易想到的就是先排序，然后逐个比较差值，但是时间复杂度O(N),上述六种基本排序都无法做到，如果用桶排序是可以做到，但是不能使用。

利用桶排序的思路，若数组中有N个数，定义N+1个桶。首先求出数组的最大值max和最小值min，将max-min这个范围的数分成N+1分，代表了每个桶所存数据的宽度，桶中定义三个变量：最大值maxs,最小值mins和是否有数了hasNum(boolean类型)。遍历整个数组，判断其值处于哪个桶中，然后对相应的桶进行判断，如果hasNum为false说明之前是个空桶，就让maxs=mins=该值，hasNum更新状态为ture说明有数进来了。进入下一个循环，如果判断到该桶有数了，就拿当前数和桶中的最大和最小值比较，更新桶的最大值和最小值。这样知道数组遍历完成。

此时由于只有N个数，但是有N+1个桶，因此必定有一个空桶，可以做空桶两边的差值是大于桶长度的，而桶内部的差值必定小于桶长度，因此最大间隙智能处于桶与桶之间，不可能存在于桶内部。于是从第0号桶开始判断，将其最大值于下一个非空的桶进行比较，得到两者差值，依照这种方法遍历下去，就可以找到最大差值了。