1. Counting sort计数排序
2. Bucket sort桶排序
3. Radix sort 基数排序

共性：都是非比较排序算法。也就是说，它们并不是通过比较元素之间的大小关系来进行排序的。

计数排序

O(n)时间复杂度

O(k)空间复杂度，每一个元素都是整数，并且位于0~k-1之间，即在[0,k-1]均匀分布。

是一种稳定排序

思想：对每个输入元素，计算/统计小于它的元素个数。例如有9个元素小于它，那么它就应该放在第10位置上。那么可以用序列的index值代表元素值，用序列的值代表这个元素的重复次数。

算法：

假设：

有一个待排序列xi x∈[0,10), i = 10

既然xi是整数，那么建立一个计数序列counting\_list[10]=[0]\*10。

现在开始统计待排序列中的元素出现次数：例如当xi=4出现时，counting\_list[4]=counting\_list[4]+1。重复这个过程，最终得到一个计数序列，这个序列中的index为待排序列的值，这个序列中的值为待排序列中的某个元素出现的次数。

此时只需要按顺序输出计数序列的index，就可以得到排好的序列。当counting\_list[i]==0时，表示待排序列中没有等于i的元素，那么跳过。当counting\_list[i]=a,a>1时，表示待排序列中等于i的元素有a个，那么便需要连续输出a次i。

桶排序

O(n)时间复杂度，最坏情况O(n\*n)或者O(n\*logn)

O(n)空间复杂度

通过某个函数的一对一映射关系，把待排序列的元素划分进不同的区间中，每一个区间就是一个桶。每一个区间的长度相同，即桶的大小相同，例如[0,1),[1,2),[2,3)...。若桶的大小都是1，此时与计数排序就很相似了，唯一的不同之处是，在桶中存储的是元素本身，而不是它们的计数。

算法：

假设：

有一个待排序列 xi x∈[0,100), i = 10

既然xi是0~100之间的随机数，那么可以设置5个桶，即每相邻的20个数所在的区间为一个桶。那么映射关系的函数可以为f(x)=x/10，即通过十位的值进行分桶。

例如f(xi)∈[0,2)，则将xi的值添加进区间为[0,2)的桶中。重复这个过程后，最终会把待排序列中的所有元素放入了对应的桶中。

接下来只需要在每个桶中进行一次排序，然后按桶的顺序输出所有元素，就可以得到最终排序好的序列了。

而桶中的排序效率决定了整个排序的效率。最优情况下，元素均匀分布在各个桶中，此时时间复杂度为O(n)，最坏情况下，所有元素都落入同一个桶中，此时时间复杂度等于在桶中应用的排序算法。这个排序算法可以时计数排序，也可以时其他的比较排序。

基数排序

O(dn)(d次调用桶排序)

O(k)空间复杂度

ki ∈[0,k-1]，k是键(key)的基(base)。如果k=10，则键就是一个十进制数。

思想：对于一组数据，按照每一位对他们进行排序。即先从最小位开始排序。例如一组十位数，那么先根据这些数据的个位值进行排序，得到一个中间序列，再对这个中间序列的十位进行排序，此时便得到了有序序列。

算法：

假设：

有一个待排序列 xi x∈[0,100), i = 10，xi是整数。

建立一个基数序列radix[10][]，这个基数序列中包含i=10个桶，每个桶的index表示十位或个位的值。

首先对待排序列的元素的个位数进行排序，将个位=i的元素存入radix[i][]。重复这个过程，最终得到一个含有待排序列中所有的元素的基数序列1。

在每个桶内进行一次排序，使桶内元素有序。

按顺序提取这些元素使之合并为一个中间序列。然后清空原来的计数序列radix[10][]。

然后对中间序列的元素的十位进行排序，将十位=i的元素存入radix[i][]。重复这个过程，最终得到一个含有中间序列所有元素的基数序列2。

在每个桶内进行一次排序，使桶内元素有序。

按顺序提取这些元素使之合并为一个新序列，此时这个新序列就是通过基数排序得到的有序序列。