计数排序（Counting Sort）

- 之前学习的冒泡、选择、插入、归并、快速、希尔、堆排序，都是基于比较的排序
  - 平均时间复杂度目前最低是 O(nlogn)

- 计数排序、桶排序、基数排序，都不是基于比较的排序
  - 它们是典型的用空间换时间，在某些时候，平均时间复杂度可以比 O(nlogn) 更低

- 计数排序于1954年由Harold H. Seward提出，适合对一定范围内的整数进行排序

- 计数排序的核心思想
  - 统计每个整数在序列中出现的次数，进而推导出每个整数在有序序列中的索引

计数排序 – 最简单的实现

7	3	5	8	6	7	4	5

	存放所有整数出现的次数								
索引	0	1	2	3	4	5	6	7	8
次数				1	1	2	1	2	1

3	4	5	5	6	7	7	8

- 这个版本的实现存在以下问题
  - 无法对负整数进行排序
  - 极其浪费内存空间
  - 是个不稳定的排序
  - ……

```
int max = array[0]; // 最大值
for (int i = 1; i < array.length; i++) {
    if (array[i] > max) {
        max = array[i];
    }
}
// 统计元素出现的次数
int[] counts = new int[max + 1];
for (int i = 0; i < array.length; i++) {
    counts[array[i]]++;
}
// 按顺序赋值
int index = 0;
for (int i = 0; i < counts.length; i++) {
    while (counts[i]-- > 0) {
        array[index++] = i;
    }
}
```

计数排序 – 改进思路

array

7	3	5	8	6	7	4	5

	从索引0开始依次存放3~8出现的次数					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	1	2	1	2	1

counts

	每个次数累加上其前面的所有次数 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	2	4	5	7	8

0	1	2	3	4	5	6	7
3	4	5	5	6	7	7	8

- 假设array中的最小值是 min
- array中的元素 k 对应的 counts 索引是 k – min
- array中的元素 k 在有序序列中的索引
  - counts[k – min] – p
  - p 代表着是倒数第几个 k

- 比如元素 8 在有序序列中的索引
  - counts[8 – 3] – 1，结果为 7

- 倒数第 1 个元素 7 在有序序列中的索引
  - counts[7 – 3] – 1，结果为 6

- 倒数第 2 个元素 7 在有序序列中的索引
  - counts[7 – 3] – 2，结果为 5

计数排序 – 改进思路

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	2	4	5	7	8

0	1	2	3	4	5	6	7

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	2	3	5	7	8

0	1	2	3	4	5	6	7
			5				

计数排序 – 改进思路

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	1	3	5	7	8

0	1	2	3	4	5	6	7
	4		5				

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	1	3	5	6	8

0	1	2	3	4	5	6	7
	4		5			7	

计数排序 – 改进思路

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	1	3	4	6	8

0	1	2	3	4	5	6	7
	4		5	6		7	

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	1	3	4	6	7

0	1	2	3	4	5	6	7
	4		5	6		7	8

计数排序 – 改进思路

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	1	1	2	4	6	7

0	1	2	3	4	5	6	7
	4	5	5	6		7	8

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	0	1	2	4	6	7

0	1	2	3	4	5	6	7
3	4	5	5	6		7	8

计数排序 – 改进思路

7	3	5	8	6	7	4	5

	每个元素累加上其前面的所有元素 得到的就是元素在有序序列中的位置信息					
元素	3	4	5	6	7	8
索引	0	1	2	3	4	5
次数	0	1	2	4	5	7

0	1	2	3	4	5	6	7
3	4	5	5	6	7	7	8

计数排序 – 改进实现

```
int max = array[0]; // 最大值
int min = array[0]; // 最小值
for (int i = 1; i < array.length; i++) {
    if (array[i] > max) {
        max = array[i];
    }
    if (array[i] < min) {
        min = array[i];
    }
}
```

```
// 用于计数
int[] counts = new int[max - min + 1];
for (int i = 0; i < array.length; i++) {
    counts[array[i] - min]++;
}
for (int i = 1; i < counts.length; i++) {
    counts[i] += counts[i - 1];
}
```

```
// 用于存放排好序的数据
int[] output = new int[array.length];
for (int i = array.length - 1; i >= 0; i--){
    output[--counts[array[i] - min]] = array[i];
}

for (int i = 0; i < array.length; i++){
    array[i] = output[i];
}
```

- 最好、最坏、平均时间复杂度：$O(n + k)$
- 空间复杂度：$O(n + k)$
- k 是整数的取值范围
- 属于稳定排序

计数排序 – 对自定义对象进行排序

- 如果自定义对象可以提供用以排序的整数类型，依然可以使用计数排序

```
private static class Person {
    int age;
    String name;
    Person(int age, String name) {
        this.age = age;
        this.name = name;
    }
    @Override
    public String toString() {
        return "Person [age=" + age
                + ", name=" + name + "]";
    }
}
```

```
Person[] persons = new Person[] {
        new Person(20, "A"),
        new Person(-13, "B"),
        new Person(17, "C"),
        new Person(12, "D"),
        new Person(-13, "E"),
        new Person(20, "F")
};
```

```
int max = persons[0].age;
int min = persons[0].age;
for (int i = 1; i < persons.length; i++) {
    if (persons[i].age > max) {
        max = persons[i].age;
    }
    if (persons[i].age < min) {
        min = persons[i].age;
    }
}
```

计数排序 – 对自定义对象进行排序

```
// 用于计数
int[] counts = new int[max - min + 1];
for (int i = 0; i < persons.length; i++) {
    counts[persons[i].age - min]++;
}
for (int i = 1; i < counts.length; i++) {
    counts[i] += counts[i - 1];
}

// 用于存放排好序的数据
Person[] output = new Person[persons.length];
for (int i = persons.length - 1; i >= 0; i--){
    output[--counts[persons[i].age - min]] = persons[i];
}
```

- 排序之后的结果

① Person [age=-13, name=B]
② Person [age=-13, name=E]
③ Person [age=12, name=D]
④ Person [age=17, name=C]
⑤ Person [age=20, name=A]
⑥ Person [age=20, name=F]