本文内容是对计数排序的梳理和总结，本文内容包括：

排序算法十大经典方法

1. 冒泡排序：比较相邻元素并交换，每一轮将最大的元素移动到最后。
2. 选择排序：每一轮选出未排序部分中最小的元素，并将其放在已排序部分的末尾。
3. 插入排序：将未排序部分的第一个元素插入到已排序部分的合适位置
4. 希尔排序：改进的插入排序，将数据分组排序，最终合并排序
5. 归并排序：将序列拆分成子序列，分别排序后合并，递归完成
6. 快速排序：选定一个基准值，将小于基准值的元素放在左边，大于基准值的元素放在右边，递归排序
7. 堆排序：将序列构建成一个堆，然后一次将堆顶元素取出并调整堆
8. **计数排序：统计每个元素出现的次数，再按照元素大小输出**
9. 桶排序：将数据分到一个或多个桶中，对每个桶进行排序，最后输出
10. 基数排序：按照元素的位数从低到高进行排序，每次排序只考虑一位

1 计数排序(Counting Sort)

- 计数排序（Counting sort）是一种非基于比较的排序算法，其核心在于将输入的数据值转化为键存储在额外开辟的数组空间中以达到排序的效果

1.1 算法步骤

1. 找出待排序的数组中最大和最小的元素
2. 然后创建一个长度为 max - min + 1 的计数数组

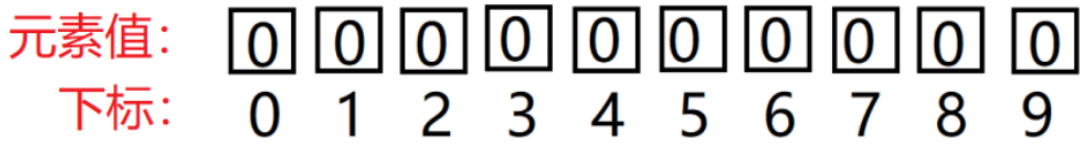
3. 扫描一遍初始数组，以当前值 - min 作为下标，将该下标的计数器加1
4. 反向填充目标数组：将每个元素i放在新数组的第C(i)项，每放一个元素就将C(i)减去1

1.2 举例说明

需求描述：现在有一组待排序的数字（如下图所示），要求通过计数排序升序排序。

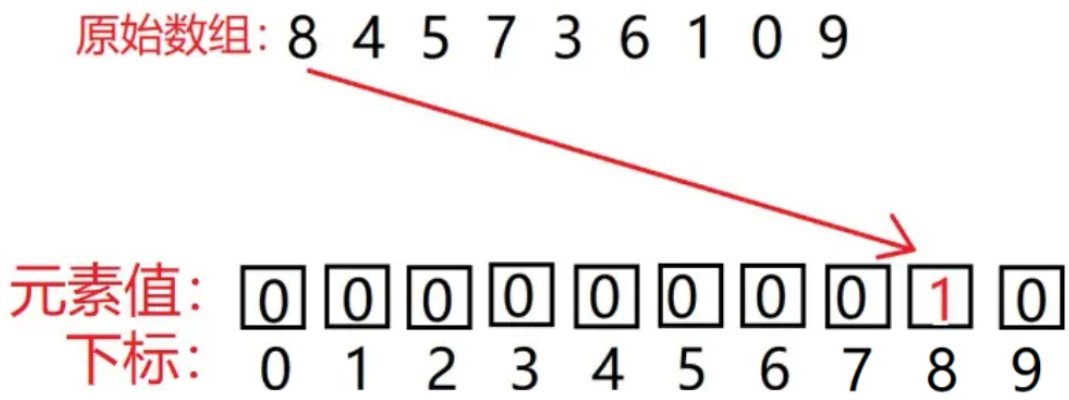
第一步：创建新数组

- 数组里的元素的最大值是9，最小值是0，于是我们可以创建一个长度为 （9 - 0 + 1）的数组，元素的初始值都为0，如图下所示

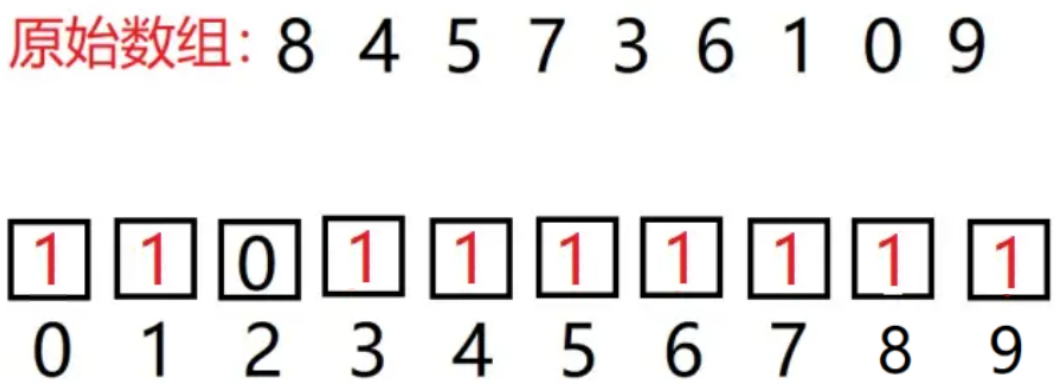


第二步：计数

- 遍历原始数组，让每一个整数按照值对号入座，对应数组下标的元素加1，第一个元素为8，就在数组下标为（8 - 0）的元素加1，如下图所示



- 按照此方法，依次遍历后面的元素，最后得到的结果如下图所示：

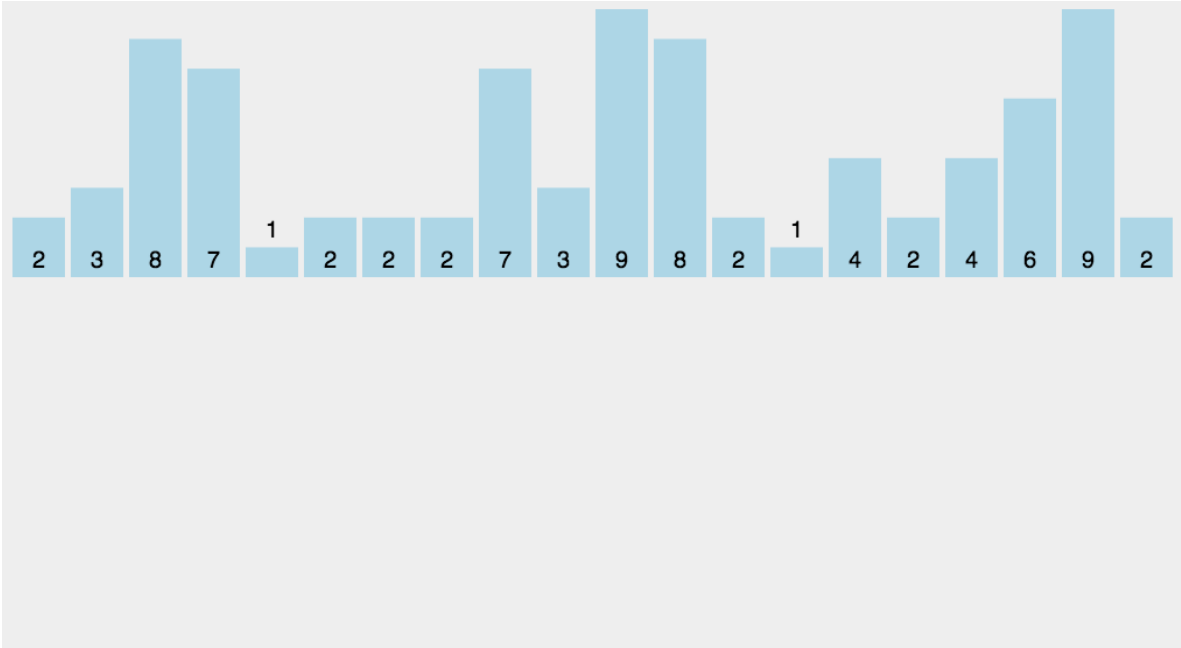


第三步：反向填充

- 最后，按照下标的先后顺序依次将下标输出，该下标的元素是几，就将下标输出几次，最后就会得到一个顺序为升序的数组，如下图所示

1.3 图解算法

如果下图不动，点击这里查看在线的图解



1.4 代码实现

```java
public static void main(String[] args) {
    int a[] = {8, 4, 5, 7, 1, 6, 3, 0, 9};
    int b[] = countSort(a);
    System.out.println(Arrays.toString(b));
}
public static int[] countSort(int[] num) {
    // 找最大值和最小值
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < num.length; i++) {
        if (num[i] > max) {
            max = num[i];
        }
        if (num[i] < min) {
            min = num[i];
        }
    }

    // 创建计数数组
    int k = max - min + 1;
    int[] count = new int[k];

    // 开始计数
    for (int i = 0; i < num.length; i++) {
        count[num[i] - min] ++;
    }

    // 反向填充
    int index = 0;
    for (int i = 0; i < k; i++) {
        while (count[i] > 0) {
            num[index ++] = i + min;
            count[i] --;
        }
    }
    return num;
}
```

1.5 算法分析

- **稳定性**： 稳定
- **时间复杂度**： 最佳：O(n+k)， 最差：O(n+k)，平均：O(n+k)
- **空间复杂度**： O(k)