本文内容是对基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 基数排序(Radix Sort)

- 基数排序是一种非比较型整数排序算法，其原理是将数据按位数切割成不同的数字，然后按每个位数分别比较。

1.1 算法步骤

1. 事先准备10个数组(10个桶)， 0-9 分别对应位数的 0-9
2. 第一轮按照个位大小放入到对应的桶中

3. 然后从 0-9 个桶，依次按照加入元素的先后顺序取出，放回原数组中
4. 第二轮按照十位排序，将各个数，按照十位大小放入到对应桶中
5. 然后从 0-9 个数组/桶，依次，按照加入元素的先后顺序取出，放回到原数组中
6. 重复上述炒作直至最大数位数为止

1.2 举例说明

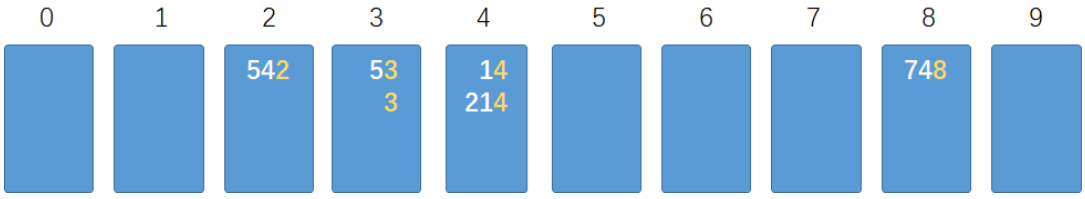
需求描述：现在有一组待排序的数字（如下图所示），要求通过基数排序排序。

第一步：创建10个桶

- 准备10个数组(10个桶)， 0-9 分别对应 位数的 0-9

第二步：按照个位排序

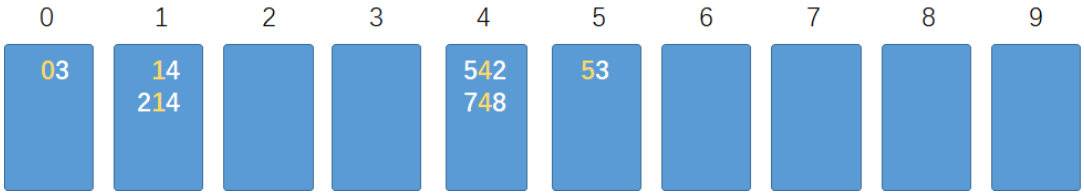
- 将每个数，按照个位大小放入到对应的桶中



- 然后从 0-9 个桶中依次按照加入元素的先后顺序取出，放回到原数组中

第三步：按照十位排序

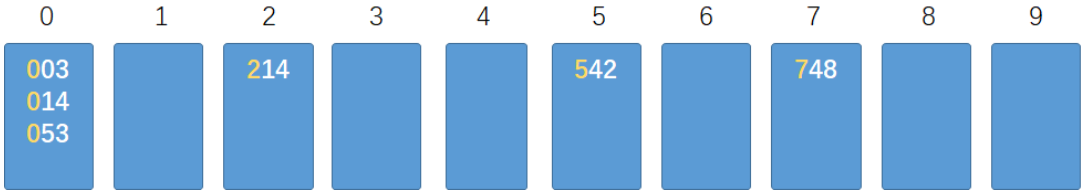
- 将每个数，按照十位大小放入到对应的桶中
- 十位上没有数值时，补0



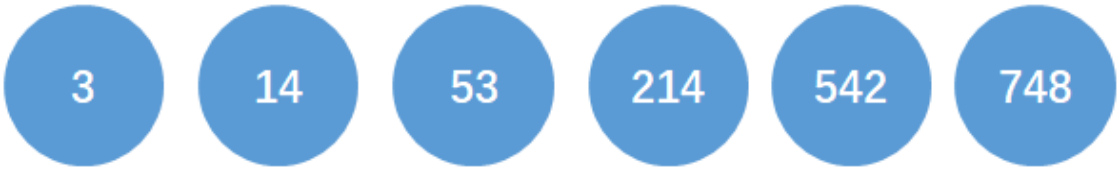
- 然后从 0-9 个桶中依次按照加入元素的先后顺序取出，放回到原数组中

第四步：按照百位排序

- 将每个数，按照百位大小放入到对应的桶中
- 百位上没有数值时，补0



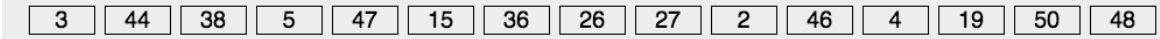
- 然后从 0-9 个桶中依次按照加入元素的先后顺序取出，放回到原数组中



此时，原数组中的数据已是有序的了

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

```
public static void radixSort(int[] arr)
{
    // 找最大位数
    int max = 0;
    for (int i = 0; i < arr.length; i++) {
        max = max < Integer.toString(arr[i]).length() ? Integer.toString(arr[i]).length() : max;
    }

    // 事先准备10个桶
    int[][] buckets = new int[10][arr.length];

    // order存储每个桶中存储数据的个数
    int[] order = new int[10];

    int k = 0;
    int n = 1;
    int m = 1; //控制排序依据是个位,还是十位,还是百位
    while (m <= max) {
        // 将数据放入桶中
        for (int i = 0; i < arr.length; i++) {
            int lsd = ((arr[i] / n) % 10);
            buckets[lsd][order[lsd]] = arr[i];
            order[lsd]++;
```

```
        }
        // 将桶中的数据放回到原数组中
        for (int i = 0; i < 10; i++) {
            if (order[i] != 0)
                for (int j = 0; j < order[i]; j++) {
                    arr[k] = buckets[i][j];
                    k++;
                }
            order[i] = 0;
        }
        n *= 10;
        k = 0;
        m++;
    }
}

public static void main(String[] args) {
    int[] arr = {53, 3, 542, 748, 14, 214};
    radixSort(arr);
    System.out.println(Arrays.toString(arr));
}
```

1.5 算法分析

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