数组排序之后相邻数的最大差值

【题目】

给定一个整型数组 arr，返回排序后的相邻两数的最大差值。

【举例】

arr=[9,3,1,10]。如果排序，结果为[1,3,9,10]，9 和 3 的差为最大差值，故返回 6。

arr=[5,5,5,5]。返回 0。

【要求】

如果 arr 的长度为 N，请做到时间复杂度为 $O(N)$。

【难度】

尉 ★★☆☆

【解答】

本题如果用排序法实现，其时间复杂度是 $O(N\log N)$，而如果利用桶排序的思想（不是直接进行桶排序），可以做到时间复杂度为 $O(N)$，额外空间复杂度为 $O(N)$。遍历 arr 找到最小值和最大值，分别记为 min 和 max。如果 arr 的长度为 N，那么我们准备 $N+1$ 个桶，把 max 单独放在第 $N+1$ 号桶里。arr 中在[min,max)范围上的数放在 1~N 号桶里，对于 1~N 号桶中的每一个桶来说，负责的区间大小为(max-min)/N。比如长度为 10 的数组 arr 中，最小值为 10，最大值为 110。那么就准备 11 个桶，arr 中等于 110 的数全部放在第 11 号桶里。区间[10,20)的数全部放在 1 号桶里，区间[20,30)的数全部放在 2 号桶里……，区间[100,110)的数全部放在 10 号桶里。那么如果一个数为 num，它应该分配进(num - min) × len / (max - min)号桶里。

arr 一共有 N 个数，min 一定会放进 1 号桶里，max 一定会放进最后的桶里，所以，如果把所有的数放入 $N+1$ 个桶中，必然有桶是空的。如果 arr 经过排序，相邻的数有可能此时在同一个桶中，也可能在不同的桶中。在同一个桶中的任何两个数的差值都不会大于区间值，而在空桶左右两边不空的桶里，相邻数的差值肯定大于区间值。所以产生最大差值的两个相邻数肯定来自不同的桶。所以只要计算桶之间数的间距就可以，也就是只用记录

每个桶的最大值和最小值，最大差值只可能来自某个非空桶的最小值减去前一个非空桶的最大值。

具体过程请参看如下代码中的 maxGap 方法。

```
public int maxGap(int[] nums) {
        if (nums == null || nums.length < 2) {
                return 0;
        }
        int len = nums.length;
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        for (int i = 0; i < len; i++) {
            min = Math.min(min, nums[i]);
            max = Math.max(max, nums[i]);
        }
        if (min == max) {
            return 0;
        }
        boolean[] hasNum = new boolean[len + 1];
        int[] maxs = new int[len + 1];
        int[] mins = new int[len + 1];
        int bid = 0;
        for (int i = 0; i < len; i++) {
            bid = bucket(nums[i], len, min, max); // 算出桶号
            mins[bid] = hasNum[bid] ? Math.min(mins[bid], nums[i]) : nums[i];
            maxs[bid] = hasNum[bid] ? Math.max(maxs[bid], nums[i]) : nums[i];
            hasNum[bid] = true;
        }
        int res = 0;
        int lastMax = 0;
        int i = 0;
        while (i <= len) {
            if (hasNum[i++]) { // 找到第一个不为空的桶
                lastMax = maxs[i - 1];
                break;
            }
        }
        for (; i <= len; i++) {
            if (hasNum[i]) {
                res = Math.max(res, mins[i] - lastMax);
                lastMax = maxs[i];
            }
        }
        return res;
}

// 使用 long 类型是为了防止相乘时溢出
public int bucket(long num, long len, long min, long max) {
        return (int) ((num - min) * len / (max - min));
}
```