912 排序数组

Label：数组、排序

给你一个整数数组 nums，请你将该数组升序排列。
输入：nums = [5,2,3,1]
输出：[1,2,3,5]
输入：nums = [5,1,1,2,0,0]
输出：[0,0,1,1,2,5]

- API

```
class Solution {
    public int[] sortArray(int[] nums) {
        Arrays.sort(nums);
        return nums;
    }
}
```

- 冒泡排序

```
class Solution {
    public int[] sortArray(int[] nums) {
       for (int i = 0; i < nums.length; i++) {
           for (int j = i+1; j < nums.length; j++) { // 第i轮过后，前i个肯定是最小有序的了
               if (nums[i] > nums[j]) {
                   int temp = nums[j];
                   nums[j] = nums[i];
                   nums[i] = temp;
               }
           }
       }
       return nums;
    }
}
```

- 堆排序

```
class Solution {
    public int[] sortArray(int[] nums) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for (int i = 0; i < nums.length; i++) {  // 建立小跟堆
            pq.offer(nums[i]);
        }
        for a(int i = 0; i < nums.length; i++) {  // 建立小跟堆
            nums[i] = pq.poll();
        }
        return nums;
    }
}
```

- 归并排序

```java
class Solution {
    int[] tmp;

    public int[] sortArray(int[] nums) {
        tmp = new int[nums.length];
        mergeSort(nums, 0, nums.length - 1);
        return nums;
    }

    public void mergeSort(int[] nums, int l, int r) {
        if (l >= r) return;

        int mid = (l + r) >> 1;
        mergeSort(nums, l, mid);
        mergeSort(nums, mid + 1, r);

        int i = l, j = mid + 1;
        int cnt = 0;

        while (i <= mid && j <= r) {
            if (nums[i] <= nums[j]) {
                tmp[cnt++] = nums[i++];
            } else {
                tmp[cnt++] = nums[j++];
            }
        }
        while (i <= mid) {
            tmp[cnt++] = nums[i++];
        }
        while (j <= r) {
            tmp[cnt++] = nums[j++];
        }

        for (int k = 0; k < r - l + 1; ++k) {  // ? ?
            nums[k + l] = tmp[k];
        }
    }
}
```

- 快排

```
class Solution {
    public int[] sortArray(int[] nums) {
        quickSort (nums, 0, nums.length-1);
        return nums;
    }

    public void quickSort (int[] nums, int low, int high) {
        if (low < high) {
            int index = partition(nums,low,high); // 首先找到数组中第一个元素应该所处
的位置
            quickSort(nums,low,index-1);
            quickSort(nums,index+1,high);
        }
    }

    public int partition (int[] nums, int low, int high) {
        int pivot = nums[low]; // 首元素为基准
        int start = low;

        while (low < high) {
            while (low < high && nums[high] >= pivot)
                high--;

            while (low < high && nums[low] <= pivot)
                low++;

            if (low >= high)
                break;
            swap(nums, low, high);
        }

        //基准值归位
        swap(nums,start,low);
        return low;
    }

    public void swap (int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
     }
}
```