

```

    }

    public void insertionSort(int[] arr, int begin, int end) {
        for (int i = begin + 1; i != end + 1; i++) {
            for (int j = i; j != begin; j--) {
                if (arr[j - 1] > arr[j]) {
                    swap(arr, j - 1, j);
                } else {
                    break;
                }
            }
        }
    }
}

```

需要排序的最短子数组长度

【题目】

给定一个无序数组 `arr`，求出需要排序的最短子数组长度。

例如：`arr = [1, 5, 3, 4, 2, 6, 7]`返回 4，因为只有`[5, 3, 4, 2]`需要排序。

【难度】

士 ★☆☆☆

【解答】

解决这个问题可以做到时间复杂度为 $O(N)$ 、额外空间复杂度为 $O(1)$ 。

初始化变量 `noMinIndex = -1`，从右向左遍历，遍历的过程中记录右侧出现过的数的最小值，记为 `min`。假设当前数为 `arr[i]`，如果 `arr[i] > min`，说明如果要整体有序，`min` 值必然会挪到 `arr[i]` 的左边。用 `noMinIndex` 记录最左边出现这种情况的位置。如果遍历完成后，`noMinIndex` 依然等于 -1，说明从右到左始终不升序，原数组本来就有顺序，直接返回 0，即完全不需要排序。

接下来从左向右遍历，遍历的过程中记录左侧出现过的数的最大值，记为 `max`。假设当前数为 `arr[i]`，如果 `arr[i] < max`，说明如果排序，`max` 值必然会挪到 `arr[i]` 的右边。用变量 `noMaxIndex` 记录最右边出现这种情况的位置。

遍历完成后，`arr[noMinIndex..noMaxIndex]` 是真正需要排序的部分，返回它的长度即可。

具体过程参看如下代码中的 `getMinLength` 方法。

```
public int getMinLength(int[] arr) {
    if (arr == null || arr.length < 2) {
        return 0;
    }
    int min = arr[arr.length - 1];
    int noMinIndex = -1;
    for (int i = arr.length - 2; i != -1; i--) {
        if (arr[i] > min) {
            noMinIndex = i;
        } else {
            min = Math.min(min, arr[i]);
        }
    }
    if (noMinIndex == -1) {
        return 0;
    }
    int max = arr[0];
    int noMaxIndex = -1;
    for (int i = 1; i != arr.length; i++) {
        if (arr[i] < max) {
            noMaxIndex = i;
        } else {
            max = Math.max(max, arr[i]);
        }
    }
    return noMaxIndex - noMinIndex + 1;
}
```

在数组中找到出现次数大于 N/K 的数

【题目】

给定一个整型数组 `arr`，打印其中出现次数大于一半的数，如果没有这样的数，打印提示信息。

【进阶】

给定一个整型数组 `arr`，再给定一个整数 K ，打印所有出现次数大于 N/K 的数，如果没有这样的数，打印提示信息。

【要求】

原问题要求时间复杂度为 $O(N)$ ，额外空间复杂度为 $O(1)$ 。进阶问题要求时间复杂度为 $O(N \times K)$ ，额外空间复杂度为 $O(K)$ 。