```
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 $O(K)$。

【难度】

校 ★★★☆

【解答】

无论是原问题还是进阶问题，都可以用哈希表记录每个数及其出现的次数，但是额外空间复杂度为 $O(N)$，不符合题目要求，所以本书不再详述这种简单的方法。本书提供方法的核心思路是，一次在数组中删掉 K 个不同的数，不停地删除，直到剩下数的种类不足 K 就停止删除，那么，如果一个数在数组中出现的次数大于 N/K，则这个数最后一定会被剩下来。

对于原问题，出现次数大于一半的数最多只会有一个，还可能不存在这样的数。具体的过程为，一次在数组中删掉两个不同的数，不停地删除，直到剩下的数只有一种，如果一个数出现次数大于一半，这个数最后一定会剩下来。如下代码中的 printHalfMajor 方法就是这种思路的具体实现，我们先列出代码，然后进行解释。

```
public void printHalfMajor(int[] arr) {
        int cand = 0;
        int times = 0;
        for (int i = 0; i != arr.length; i++) {
                if (times == 0) {
                        cand = arr[i];
                        times = 1;
                } else if (arr[i] == cand) {
                        times++;
                } else {
                        times--;
                }
        }
        times = 0;
        for (int i = 0; i != arr.length; i++) {
                if (arr[i] == cand) {
                        times++;
                }
        }
        if (times > arr.length / 2) {
                System.out.println(cand);
        } else {
                System.out.println("no such number.");
        }
}
```

printHalfMajor 方法中第一个 for 循环就是一次在数组中删掉两个不同的数的代码实现。

我们把变量 cand 叫作候选，times 叫作次数，读者先不用纠结这两个变量是什么意义，我们看在第一个 for 循环中发生了什么。

- times==0 时，表示当前没有候选，则把当前数 arr[i]设成候选，同时把 times 设置成 1。
- times!=0 时，表示当前有候选，如果当前的数 arr[i]与候选一样，就把 times 加 1；如果当前的数 arr[i]与候选不一样，就把 times 减 1，减到 0 则表示又没有候选了。

这具体是什么意思呢？当没有候选时，我们把当前的数作为候选，说明我们找到了两个不同的数中的第一个；当有候选且当前的数和候选一样时，说明目前没有找到两个不同的数中的另外一个，反而是同一种数反复出现了，那么就把 times++表示反复出现的数在累计自己的点数。当有候选且当前的数和候选不一样时，说明找全了两个不同的数，但是候选可能在之前多次出现，如果此时把候选完全换掉，候选的这个数相当于一下被删掉了多个，对吧？所以这时候选“付出”一个自己的点数，即 times 减 1，然后当前数也被删掉。这样还是相当于一次删掉了两个不同的数。当然，如果 times 被减到为 0，说明候选的点数完全被消耗完，那么又表示候选空缺，arr 中的下一个数(arr[i+1])就又被作为候选。

综上所述，第一个 for 循环的实质就是我们的核心解题思路，一次在数组中删掉两个不同的数，不停地删除，直到剩下的数只有一种，如果一个数出现次数大于一半，则这个数最后一定会被剩下来，也就是最后的 cand 值。

这里请注意一点，一个数出现次数虽然大于一半，它肯定会被剩下来，但那并不表示剩下来的数一定是符合条件的。例如，1，2，1。其中 1 符合出现次数超过了一半，所以 1 肯定会剩下来。再如 1，2，3，其中没有任何一个数出现的次数超过了一半，可 3 最后也剩下来了。所以 printHalfMajor 方法中第二个 for 循环的工作就是检验最后剩下来的那个数（即 cand）是否真的是出现次数大于一半的数。如果 cand 都不符合条件，那么其他的数也一定都不符合，说明 arr 中没有任何一个数出现了一半以上。

进阶问题解法核心也是类似的，一次在数组中删掉 K 个不同的数，不停地删除，直到剩下的数的种类不足 K，那么，如果某些数在数组中出现次数大于 N/K，则这些数最后一定会被剩下来。原问题中，我们解决了找到出现次数超过 $N/2$ 的数，解决的办法是立了 1 个候选 cand，以及这个候选的 times 统计。进阶问题具体的实现也类似，只要立 K-1 个候选，然后有 K-1 个 times 统计即可，具体过程如下。

遍历到 arr[i]时，看 arr[i]是否与已经被选出的某一个候选相同：

如果与某一个候选，就把属于那个候选的点数统计加 1。

如果与所有的候选都不相同，先看当前的候选是否选满了，K-1 就是满，否则就是不满：

- 如果不满，把 arr[i]作为一个新的候选，属于它的点数初始化为 1。
- 如果已满，说明此时发现了 K 个不同的数，arr[i]就是第 K 个。此时把每一个候选各自的点数全部减 1，表示每个候选“付出”一个自己的点数。如果某些候选的点数在减 1 之后等于 0，则还需要把这些候选都删除，候选又变成不满的状态。

在遍历过程结束后，再遍历一次 arr，验证被选出来的所有候选有哪些出现次数真的大于 N/K，符合条件的候选就打印。具体请参看如下代码中的 printKMajor 方法。

```
public void printKMajor(int[] arr, int K) {
        if (K < 2) {
                System.out.println("the value of K is invalid.");
                return;
        }
        HashMap<Integer, Integer> cands = new HashMap<Integer, Integer>();
        for (int i = 0; i != arr.length; i++) {
                if (cands.containsKey(arr[i])) {
                        cands.put(arr[i], cands.get(arr[i]) + 1);
                } else {
                        if (cands.size() == K - 1) {
                                allCandsMinusOne(cands);
                        } else {
                                cands.put(arr[i], 1);
                        }
                }
        }
        HashMap<Integer, Integer> reals = getReals(arr, cands);
        boolean hasPrint = false;
        for (Entry<Integer, Integer> set : cands.entrySet()) {
                Integer key = set.getKey();
                if (reals.get(key) > arr.length / K) {
                        hasPrint = true;
                        System.out.print(key + " ");
                }
        }
        System.out.println(hasPrint ? "" : "no such number.");
}

public void allCandsMinusOne(HashMap<Integer, Integer> map) {
        List<Integer> removeList = new LinkedList<Integer>();
        for (Entry<Integer, Integer> set : map.entrySet()) {
                Integer key = set.getKey();
                Integer value = set.getValue();
                if (value == 1) {
                        removeList.add(key);
                }
                map.put(key, value - 1);
        }
        for (Integer removeKey : removeList) {
                map.remove(removeKey);
```

```
        }
    }

    public HashMap<Integer, Integer> getReals(int[] arr,
                HashMap<Integer, Integer> cands) {
        HashMap<Integer, Integer> reals = new HashMap<Integer, Integer>();
        for (int i = 0; i != arr.length; i++) {
                int curNum = arr[i];
                if (cands.containsKey(curNum)) {
                        if (reals.containsKey(curNum)) {
                                reals.put(curNum, reals.get(curNum) + 1);
                        } else {
                                reals.put(curNum, 1);
                        }
                }
        }
        return reals;
    }
```

【扩展】

这种一次删掉 K 个不同的数的思想在面试中通常会变形之后反复出现。例如，下面这道面试真题：有一场投票，投票有效的条件是必须有一个候选人得票数超过半数，但是验票人员不能看到每张选票上选了谁，只能把任意两张选票放到一台机器上看这两张选票是否一样，若一样，则机器给出 true 的提醒，不一样则给出 false 的提醒。如果你作为验票的人员，怎么判断这场投票是有效的？

这道题目就是原问题的变形，但是“不能看到每张选票上选了谁”的这个限制实际上把用哈希表来解题的可能性完全堵死了。但本文的方法却可以满足题目的要求，因为我们实现的方法只需要当前数和候选数做比较，而不需要知道每个数的值。

在行列都排好序的矩阵中找数

【题目】

给定一个有 $N \times M$ 的整型矩阵 matrix 和一个整数 K，matrix 的每一行和每一列都是排好序的。实现一个函数，判断 K 是否在 matrix 中。

例如：

0　1　2　5

2　3　4　7