期中试题

第 1 题

a) 请证明调和级数满足：$1+\frac{1}{2}+\cdots+\frac{1}{n}=\Theta(\log n)$。

b) 请证明：$\log n!=\Theta(\mathrm{n}\log n)$。
（注：不可以使用 Stirling 公式。）

第 2 题

如果一个数组A[1..2n+1]，满足A[1] < A[2] > A[3] < A[4] > … < A[2n] > A[2n+1]，我们称之为“蛇形”的。给定数组B[1..2n+1]，其中元素各不相同，现在需要将它变成蛇形的。你只能通过元素间的大小比较来调整数组的形态。

1) 请设计一个O(n log n)的算法。
2) 请设计一个O(n)的算法。

(注：你需要阐述算法的正确性，并分析其代价)

第 3 题

考虑课上讲过的最坏情况O(n)的选择算法，记为SELECT5算法。你可能会有这样的疑惑：为什么元素应该分为5个一组，其它个数的分组是否可行。现在我们来分析一下3个元素一组的选择算法为什么不行。
现在假设我们按3个元素一组来进行选择，算法的其它设计与SELECT5算法一样。这一新的算法记为SELECT3算法。

a) 请分析并证明：对 SELECT3 算法而言，在任何情况下总是比 m*（median-of-median）小的元素的个数不超过 $\frac{n}{3}+3$。（注：请不要忽略 n 不是 3 的倍数等边界情况。）

b) 请分析并证明算法的最坏情况时间复杂度 T(n)可以用下面的递归方程来刻画：

$$\mathrm{T(n)} \geq \mathrm{T}\left(\left\lceil\frac{n}{3}\right\rceil\right)+\mathrm{T}\left(\frac{2n}{3}-3\right)+\Omega(\mathrm{n})\quad 。$$

c) 请用“prove by substitution”的方法证明$\mathrm{T(n)}=\Omega(n\log n)$。

第 4 题

给定n个各不相同的两两可比的元素$x_1, x_2, \ldots, x_n$，每个元素有正的权重值$w_1, w_2, \ldots, w_n$。记所有元素权重之和为W。定义这组元素中的1/3-median为满足下面条件的元素x_k：

$$\sum_{x_i<x_k} w_i<\frac{W}{3},\ \sum_{x_i>x_k} w_i \leq \frac{2W}{3}$$

a) 请设计一个O(nlogn)的算法，找出给定元素中的1/3-median。
b) 请设计一个O(n)的算法，找出给定元素中的1/3-median。

第 5 题

你在一个有 n 个代表的政治会议会场内，每个代表都隶属于一个政党，但是并不知道他们属于哪个政党。假设你直接询问一个代表，他会拒绝回答，但是你可以通过介绍两个代表认识来分辨他们是否属于同一个政党（因为同一政党的代表会礼貌地握手并给予对方微笑；不同政党的代表会怒视对方）。

a) 假设代表中的大多数（一半以上）来自同一政党（称之为主要政党）。请设计一个算法来判定每个代表是否属于这个主要政党。

b) 假设代表们来自 k 个政党，一个政党占多数当且仅当属于它的代表的数目比其他任何政党的代表都多。请设计一个算法找出一个来自占多数的政党的代表，或者返回不存在占多数的政党。

第 6 题

给定一个二维比特数组，它有n行k列，存放了所有可能的k比特串，仅仅有某一个k比特串被剔除，所以我们有$n=2^k-1$。例如图中k=3，n=7，唯一缺失的比特串是101。现在我们需要计算出缺失的那个k比特串，所能做的关键操作是"检查数组的某一位是0还是1"。

1) 请设计一个O(nk)的算法，找出缺失的比特串。

2) 请设计一个O(n)的算法，找出缺失的比特串。

(注：你需要阐述算法的正确性，并分析其代价)

1	1	0
1	1	1
0	0	1
0	1	0
0	0	0
0	1	1
1	0	0

题解不唯一，答案仅供参考，如有错误，欢迎指正。

1.

a) 令$x_n = 1+\frac{1}{2}+\cdots\frac{1}{n}$, $y_n = \log n$

解法 1：根据 Stolz 定理：

$$\lim_{n\to\infty}\frac{x_n}{y_n} = \lim_{n\to\infty}\frac{x_{n+1}-x_n}{y_{n+1}-y_n} = \lim_{n\to\infty}\frac{\frac{1}{n+1}}{\log\left(1+\frac{1}{n}\right)} = \lim_{n\to\infty}\frac{1}{\log\left(1+\frac{1}{n}\right)^{n+1}} = c\ (c<+\infty)$$

$$(\because \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e)$$

解法 2：根据积分缩放

$$x_n < 1+\int_1^n \frac{1}{x}dx = 1+\ln n$$

$$x_n > \int_1^{n+1}\frac{1}{x}dx = \ln(n+1)$$

根据夹逼法可得，$\lim_{n\to\infty}\frac{x_n}{y_n} = c\ (c<+\infty)$

b)

$$n! = 1*2*3*\ldots*(n-1)*n$$

(1) n为偶数时：

$$n! = (1*n)*[2*(n-1)]*\ldots*[\frac{n}{2}*(\frac{n}{2}+1)]$$

等式右侧$\frac{n}{2}$ 项从左往右依次递增，故：

$$(1*n)^{n/2} < n! < \left[\frac{n}{2}*\left(\frac{n}{2}+1\right)\right]^{n/2} < (n^2)^{n/2} = n^n$$

取 log:

$$\frac{n}{2}\log n < \log n! < n\log n$$

(2) n为奇数时：

$$n! = (1*n)*[2*(n-1)]*\ldots*\left[\frac{n-1}{2}*\left(\frac{n+1}{2}+1\right)\right]*(\frac{n+1}{2})$$

$$n! = \frac{1}{2}*(1*n)*[2*(n-1)]*\ldots*\left[\frac{n-1}{2}*\left(\frac{n+1}{2}+1\right)\right]*(n+1)$$

同理：

$$\frac{1}{2}(1*n)^{\frac{n+1}{2}} < n! < \frac{1}{2}*\left[\frac{n-1}{2}*\frac{n+3}{2}\right]^{\frac{n+1}{2}} < n^{n+1}$$

取 log:

$$\frac{n+1}{2}\log n < \log n! < (n+1)\log n$$

综上，

有$\lim_{n}\frac{\log n!}{nlog} = c\ \ (c<+\infty)$

2.

1）思路：对$A[2n+1]$数组进行排序（快排、归并等$O(nlogn)$时间复杂度的排序），排序后的数组记为$B[2n+1]$，其中前 $n+1$ 项元素 $B[1..n+1]$ 依次放到$A[1], A[3], \ldots, A[2n+1]$（即奇数下标的位置），后 n 项元素依次放到$A[2], A[4], \ldots, A[2n]$（即偶数下标的位置），即可完成蛇形排序。

伪代码：[语法细节不用在意，部分关键方法调用最好在考试中手写实现]

```
def snakeSort1(A[1..2*n+1]):
    sort(A)                        # 最好手写实现
    for i in [1, n+1]:
        B[2*i-1] = A[i]
    for i in [1, n]:
        B[2*i] = A[n+1+i]
    return B
```

2）思路：分析问题，可以发现只需要保证 $A[2k] > A[2k-1]\ and\ A[2k] > A[2k+1]$，即偶数下标对应的元素大于其相邻的奇数下标对应的元素，而$A[2], A[4], \ldots, A[2n]$ 和 $A[1], A[3], \ldots, A[2n+1]$ 内部之间并无排序需求。因此，我们可以考虑简化算法：只需要找到数组 A 中前 $n+1$ 小的元素以任意的次序放在$A[1], A[3], \ldots, A[2n+1]$中，剩下元素以任意的次序放在$A[2], A[4], \ldots, A[2n]$中即可。因此，使用线性时间的选择算法找到阶为 n+1 的元素，并对剩余元素进行 partition，之后过程同 1）

伪代码：[语法细节不用在意，部分关键方法调用最好在考试中手写实现]

```
def select(A, p, r, k):
    if p == r:
        return A[p]
    q = Partition(A, p, r)         # 最好手写实现
    x = q - p - 1
    if k == x:
        return A[q]
    elif k < x:
        return select(A, p, q-1, k)
    else:
        return select(A, q+1, r, k-x)

def snakeSort(A[1..2*n+1]):
    select(A, 1, 2*n+1, n+1)
    for i in [1, n+1]:
        B[2*i-1] = A[i]
    for i in [1, n]:
        B[2*i] = A[n+1+i]
    return B
```

3

a)

一共划分成$\lceil \frac{n}{3} \rceil$组，前$\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil \rceil$组每组共享2个比m*小的数，所以不超过：

$$\begin{aligned} 2\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil \rceil &\leq 2(\frac{1}{2} \lceil \frac{n}{3} \rceil + 1) \\ &= \lceil \frac{n}{3} \rceil + 2 \\ &\leq (\frac{n}{3} + 1) + 2 \\ &= \frac{n}{3} + 3 \end{aligned}$$

[这个界很宽，只需要合理证明小于$\frac{n}{3}+3$即可。]

b)

- 一共可以划分成$\lceil \frac{n}{3} \rceil$组对其进行递归求解m*，需要$T(\lceil \frac{n}{3} \rceil)$
- 根据a)的结果，不确定与m*关系的至少有$n - (\frac{n}{3} + 3) = \frac{2n}{3} - 3$个元素，对其递归求解，需要$T(\frac{2n}{3} - 3)$
- 还需要$\Omega(n)$的时间来分组和partition。

一共就是$T(n) \geq T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} - 3) + \Omega(n)$。

c)

假设$T(n) \geq cnlogn$，根据b)的结果有

$$\begin{aligned} T(n) &\geq T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} - 3) + \Omega(n) \\ &\geq c\lceil \frac{n}{3} \rceil log \lceil \frac{n}{3} \rceil + c(\frac{2n}{3} - 3) log(\frac{2n}{3} - 3) + c_1 n \\ &\geq c\frac{n}{3} log\frac{n}{3} + c\frac{2n}{3} log(\frac{2n}{3} - 3) - 3clog(\frac{2n}{3} - 3) + c_1 n \\ &= c\frac{n}{3} logn + c\frac{2n}{3} log(2n - 9) - cnlog3 - 3clog(\frac{2n}{3} - 3) + c_1 n \\ &\geq c\frac{n}{3} logn + c\frac{2n}{3} logn - cnlog3 - 3clog(\frac{2n}{3} - 3) + c_1 n \\ &\geq cnlogn + (c_1 n - cnlog3 - 3clog(\frac{2n}{3} - 3)) \\ &\geq cnlogn + (c_1 n - 2cn - 3cn) \\ &= cnlogn + (c_1 n - 5cn) \end{aligned}$$

此时需要$c_1 n - 5cn \geq 0$，即$c \leq \frac{c_1}{5}$，有$T(n) \geq cnlogn$，假设成立。

[需要求出c的范围，不能与n有关，缩放要合理]

4

a)

先对x_i排序，然后逐个统计每个x_i是否满足条件。

```
ALG4A(x[1..n], w[1..n])
用归并排序对x[1..n]排序，同时wi随着xi一起移动。
W = 0
for i = 1 to n do
    W += w[i]
wsum = 0
for i = 1 to n do
    if (wsum < W/3 && W-wsum-w[i] <= 2W/3) return x[i]
    wsum += w[i]
```

时间复杂度一共为$O(nlogn) + O(n) = O(nlogn)$

[要指出w_i随x_i移动]

b)

类似查找k阶元素，递归求解1/3-median。

```
SOLVE(x[1..n], w[1..n], lsum, rsum, W)
if (n == 1) return x[1]
用SELECT-WLINEAR找到x[1..n]的中位数m
根据m对x[1..n]进行partition，比m小的放左边，比m大的放右边，同时wi随着xi一起移动。
l = lsum, r = rsum
for i = 1 to n/2-1 do
    l += w[i]
for i = n downto n/2+1 do
    r += w[i]
if (l < W/3 && r <= 2W/3) return m
if (l >= W/3) return SOLVE(x[1..n/2-1], w[1..n/2-1], lsum, r+w[n/2], W)
return SOLVE(x[n/2+1..n], w[n/2+1..n], l+w[n/2], rsum, W)

ALG4B(x[1..n], w[1..n])
W = 0
for i = 1 to n do
    W += w[i]
return SOLVE(x[1..n], w[1..n], 0, 0, W)
```

$SOLVE$复杂度为$T(n) = T(n/2) + O(n)$，由Master定理可知$T(n) = O(n)$。

时间复杂度一共为$O(n)$。

[要指出w_i随着x_i移动，还有记住递归外的局部和$lsum$和$rsum$]

第5题：

题目源自教材第7章课后习题7.9，并且派生于习题7.7 *(常见元素)*

a） *(分点作答，逻辑清晰，避免全篇文字说明，只添加必要的证明或注释，下同)*

1. **问题抽象**：*(对实际问题建模，选择合适的数据结构，下同)*……………………………………………………1分

> 设第i个代表隶属的政党的编号为$A[i]$，则"主要政党"就是数组$A[1..n]$中出现次数超过一半($\lfloor \frac{n}{2} \rfloor + 1$)的元素(姑且称其为"主要元素"，记为$MainElem$)，限定的关键操作是比较数组元素$A[i]$和$A[j]$是否相等；

2. **算法思路**：*(包括算法原理和正确性的简要说明，下同)*……………………………………………………3分

> **Lemma1**: 若e是数组$A[1..n]$的$MainElem$，令$A_l = A[1..\lfloor \frac{n}{2} \rfloor], A_r = A[\lfloor \frac{n}{2} \rfloor + 1..n]$，
>
> 则e至少是A_l和A_r这两个子数组其中之一的$MainElem \Rightarrow$ **反证如下**：
>
> 若e既不是A_l也不是A_r的$MainElem$，则e在A中的出现次数：
>
> $$cnt_e = cnt_l + cnt_r \le \lfloor \frac{n_l}{2} \rfloor + \lfloor \frac{n_r}{2} \rfloor = \lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \rfloor + \lfloor \frac{\lceil \frac{n}{2} \rceil}{2} \rfloor \le \lfloor \frac{\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$$
>
> 故e亦不是A的$MainElem$，矛盾 $\Rightarrow$ 根据Lemma1，可设计**分治算法**：
>
> I. 递归找到左右子问题A_l和A_r中的$MainElem$；
>
> II. 遍历数组A分别检验二者是否为原问题的$MainElem$，检验成功则返回，否则返回无$MainElem$；
>
> III. 找到$MainElem$后，再将将每个元素$A[i]$与之比较，若相等则说明第i个代表属于"主要政党"，否则就不属于；

3. **算法伪码**：*(得分重难点，要求代码结构清晰, 变量/库函数命名直观, 关键点添加注释，下同)*………5分

```
Algorithm: IsMainElem(A[1..n])
// arr[i] == true: A[i] is a MainElem
arr := [false]; // all false at fisrt
// step1: Find the MainElem
idx := FindMainElem(A[1..n]);
// step2: compare with MainElem
for i := 1 to n do
  if A[i] == A[idx] then
    arr[i] = true;
return arr;
```

```
Algorithm: FindMainElem(A[l..r])
// base case: A only contains one elem
n := r-l+1;
if n == 1 then
  return l;
```

```
// recurse to left half and right half
m := (l+r)/2;
il := FindMainElem(A[l..m]);
ir := FindMainElem(A[m+1..r]);
// count the number of A[il] in A[l..r]
if Count(A[l..r],il) > n/2 then // found the MainElem in A[l..r]
  return il;
// count the number of A[ir] in A[l..r]
if Count(A[l..r],ir) > n/2 then // found the MainElem in A[l..r]
  return ir;
// both il and ir are not the MainElem
return 0;
```

```
Algorithm: Count(A[l..r],i) // count the equal elems to A[i] in A[l..r]
cnt := 0;
if i != 0 then
  for j:=l to r do
    if A[i] == A[j] then
      cnt := cnt + 1;
return cnt;
```

4. **算法分析**：*(时空复杂度分析，即使题目没要求，分析较简单时还是尽量写上，下同)……………………1分*

> 1. 时间复杂度：
>
>   ① 寻找$MainElem$的分治算法有代价递归式：$T(n)=2T(\frac{n}{2})+O(n)$，由$Master$定理得：$T(n)=O(n\log n)$；
>
>   ② 确定每个代表是否为$MainElem$：只需要遍历一遍原数组，为$O(n)$；
>
>   综合①②：总时间复杂度为$O(n\log n)$；
> 2. 空间复杂度：易知仅使用了递归栈的空间，复杂度相当于递归深度：$O(\log n)$；

---

b)

1. **问题抽象**：*1分*

> 要找出一个来自占多数的政党的代表，即：在一个仅由k个元素组成的数组$A[1..n]$中找到出现次数最多的元素，即统计学上的“众数”($Mode$)，或者返回该数组不存在$Mode$；

2. **算法思路**：*3分*

> **Lemma2**: 同a)理，若e是数组$A[1..n]$的$kMainElem$，令$A_l=A[1..\lfloor\frac{n}{2}\rfloor],A_r=A[\lfloor\frac{n}{2}\rfloor+1..n]$，则$e$至少是$A_l$和$A_r$这两个子数组其中之一的$kMainElem$；
>
> **Lemma3**: 数组A中的$kMainElem$少于k个，证明如下：设A中的所有$kMainElem$分别为：$e_1,e_2,..,e_m$，即$\forall e_i,cnt(e_i)\geq\lfloor\frac{n}{k}\rfloor+1>\frac{n}{k}$，则：

$$n \geq \sum_{i=1}^{m} cnt(e_i) \geq m \times (\lfloor \frac{n}{k} \rfloor + 1) > m \cdot \frac{n}{k} \Rightarrow m < \frac{n}{\frac{n}{k}} = k$$

> 根据Lemma2,3，可设计**分治算法**：
>
> I. 递归找到左右子问题A_l和A_r中所有$kMainElem$；
>
> II. 遍历数组A检验这些候选数是否为原问题的$kMainElem$，将检验成功的保存下来；
>
> III. 找到原问题所有$kMainElem$后，再分别统计其出现次数，选出$Mode$；或返回没有$Mode$；

3. **算法伪码**：*5分*

```
Algorithm: FindMode(A[1..n])
// step1: Find all the kMainElem
I,C := FindKMainElem(A[1..n]);
// step2: Find the Mode among kMainElem
idxArray := argmax(C);
if len(idxArray) == 1 then // unique max
  return idxArray[1];
return 0;
```

```
Algorithm: FindKMainElem(A[l..r],k) // find the kMainElem in A[l..r]
// base case: A only contains less than k elems, then all elems are kMainElem
n := r-l+1;
I := []; C := [];
if n < k then
  I := [l..r];
  C := [1,1,..,1]; // all-one array of size n
  return I,C;
// recurse to left half and right half
m := (l+r)/2;
Il,Cl := FindKMainElem(A[l..m],k);
Ir,Cr := FindKMainElem(A[m+1..r],k);
// count the number of A[il] in A[l..r] for each il in Il
for each il in Il do
  if Count(A[l..r],il) > n/k then // found one kMainElem in A[l..r]
    I.append(il); C.append(cnt);
// count the number of A[ir] in A[l..r] for each ir in Ir
for each ir in Ir do
  if Count(A[l..r],ir) > n/k then // found one kMainElem in A[l..r]
    I.append(ir); C.append(cnt);

return I,C;
```

4. **算法分析**：*1分*

> 1. 时间复杂度：
>
> ① 递归式：$T(n) = 2T(\frac{n}{2}) + O(kn)$，由$Master$定理得：$T(n) = O(kn \log n)$；

> ② 由于第①步已经顺便统计出来了C，故只需要找到$argmax\{C[i]\}$，为$O(k)$；
>
> 综合①②：总时间复杂度为$O(kn\log n)$；
>
> 2. 空间复杂度：易知I和C均可复用，且长度均不超过$2k$，加上递归栈，共：$O(k+\log n)$；

第6题：

题目源自教材第7章课后习题*7.12 (寻找缺失的比特串)*

1）*(记二维比特数组为$bitStr$，缺失比特串为$mStr$，下同)*；

1. **算法思路**：*3分*

> **Lemma1**: 当$k \geq 2$时，$bitStr$第j列的布尔和等于$mstr$的第j位比特值，证明如下：
>
> ① 若没有缺失比特串且$k \geq 2$，则$bitStr$任何一列上的布尔和等于偶数个1的布尔和，为0；
>
> ② 若$mstr[j]=0$，则$bitStr$第j列的布尔和仍等于偶数个1的布尔和 (*仅少了一个*0)，为0；
>
> ③ 若$mstr[j]=1$，则$bitStr$第j列的布尔和等于奇数个1的布尔和(*仅少了一个*1)，为1；
>
> 综合①~③：Lemma1得证 $\Rightarrow$ 根据Lemma1，可设计**朴素查找算法**如下：
>
> I. 若$k=1$，则$bitStr$仅含$0,1$两个单比特串，缺失一个，将剩余那个取反即可得$mstr$；
>
> II. 若$k \geq 2$，则对每一个$j \in [1,k]$，有：$mstr[j] = \sum_{i=1}^{n-1} bitStr[i][j]$；

2. **算法伪码**：*6分*

```
Algorithm FindMissingString1(bitStr[1..n-1][1..k]) // => O(nk)
// base case: k = 1, n-1 = 1, just return the flipped bitStr[1][1]
if k == 1 then
  return bitStr[1][1] ⊕ 1; // 1 => 0, 0 => 1
// bool sum for each bit in column j
mstr := [00..0]; // all-zero array of size k
for j:=1 to k do
  sum := 0;
  for i:=1 to n-1 do
    sum := sum ⊕ bitStr[i][j];
  mstr[j] := sum;
return mstr;
```

3. **算法分析**：*1分*

> 时间复杂度：易知对$bitStr$每一位仅访问1次 $\Rightarrow O(nk)$；
>
> 空间复杂度：易知只使用了常数额外空间 $\Rightarrow O(1)$；

2）

1. **算法思路**：*3分*

> **Lemma2**: 当$k \geq 3$时，若$mstr[j]=b$，则所有第j位为b的比特串，其第$j+1$列的布尔和等于$mstr$的第$j+1$位比特值，证明如下：
>
> ① 当$k \geq 3$时，第j位为b的比特串一定满足0和1的数量相等且为偶数，因此其布尔和为0；
>
> ② 故在求和第$j+1$列时，只需要考虑那些第j位为0的比特串，共$\lfloor \frac{n-1}{2} \rfloor$个；
>
> 根据Lemma2，可设计**分治查找算法**如下：
>
> I. 从$j=1$开始，当确定$mstr[j]=b$后，删除$bitStr$第j位不是b的一半比特串；
>
> II. 在剩下的一半比特串上计算$mstr[j+1]$，重复上述步骤直到比特串只剩下一个；
>
> III. 若比特串只剩下一个，显然此时$j=k$，直接将其第k位取反即得到$mstr[k]$；

2. **算法伪码**：*6分*

```
Algorithm FindMissingString2(bitStr[1..n][1..k],j,mstr) // => O(n)
// base case1: nj = 1, just return the flipped bitStr[1][j]
if n == 1 then
  mstr[j] := bitStr[1][j] ⊕ 1; // 1 => 0, 0 => 1
  return;
// bool sum for column j
sum := 0;
for i:=1 to n do
  sum := sum ⊕ bitStr[i][j];
mstr[j] := sum;
// get subBitStr containing those who hold the same jth bit with mstr
subBitStr := [][];
for i:=1 to n do
  if bitStr[i][j] == mstr[j] then
    subBitStr[i] = bitStr[i];
// recurse to subBitStr to find (j+1)th bit
FindMissingString2(subBitStr[1..n/2][1..k],j+1,mstr);
```

3. **算法分析**：*1分*

> 1. 时间代价递归式：$T(n)=T(\frac{n}{2})+O(n)$，由$Master$定理得：$T(n)=O(n)$；
> 2. 空间代价递归式：$T(n,k)=T(\frac{n}{2},k)+O(nk)$，由$Master$定理得：$T(n,k)=O(nk)$；

第5题a问补充：

本题有$O(n)$的解法，称为“**Boyer-Moore 投票法**”，俗称“PK法”：

1. 算法思路：

> 假设现在所有代表因政见不和开始”PK“，PK形式为1v1单挑，同政党代表不会PK，且不同政党代表PK总是”两败俱伤“后被抬出会场 $\Rightarrow$ 则对于”主要政党“，最坏情况就是所有其他政党都联合起来PK自己，但由于自己人数超过一半，因此最终还站在会场的一定是”主要政党“的代表（若其他政党彼此还互相PK，只会让”主要政党“的代表所剩人数更多）；

2. 算法流程：

> 根据上述形象的思路，我们可以对数组A的遍历也模拟出一个”PK“过程：
>
> I. 维护一个计数器cnt和索引idx，表示遍历途中，$A[idx]$所属政党目前带上了cnt个兄弟来PK，依次将$A[idx]$和$A[i]$进行比较：
>
> II. 若相等则说明$A[i]$也是该政党的兄弟，带上一起 $\Rightarrow cnt = cnt + 1$；
>
> 若不等则说明$A[i]$是敌对政党，则派一个人去跟他PK并抬出会场 $\Rightarrow cnt = cnt - 1$；
>
> III. 若此时$cnt = 0$，说明该政党目前带上的人已经全部牺牲，那么$A[i+1]$所属政党将”捡漏“成为目前还有人站着的政党$\Rightarrow cnt = 1, idx = i + 1$；
>
> IV. 遍历完后，idx和cnt记录的即是：$A[idx]$所属政党，即”主要政党“，还剩cnt个人站着；

3. 算法分析：

> 时间复杂度：易知只遍历了一遍数组，为$O(n)$；
>
> 空间复杂度：易知只使用了常数额外空间，为$O(1)$；
