

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:

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

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

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, $\lim_{n \rightarrow \infty} \frac{x_n}{y_n} = c \quad (c < +\infty)$

b)

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

(1) n 为偶数时:

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

等式右侧 $\frac{n}{2}$ 项从左往右依次递增, 故:

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

取 \log :

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

(2) n 为奇数时:

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

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

同理:

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

取 \log :

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

综上,

$$\text{有} \lim_n \frac{\log n!}{n \log} = c \quad (c < +\infty)$$

2.

1) 思路：对 $A[2n+1]$ 数组进行排序（快排、归并等 $O(n\log n)$ 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 $A[1], A[3], \dots, A[2n+1]$ （即奇数下标的位置），后 n 项元素依次放到 $A[2], A[4], \dots, A[2n]$ （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 $A[2], A[4], \dots, A[2n]$ 和 $A[1], A[3], \dots, A[2n+1]$ 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 $A[1], A[3], \dots, A[2n+1]$ 中，剩下元素以任意的次序放在 $A[2], A[4], \dots, A[2n]$ 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 $T(n) \geq cn \log n$ ，根据b)的结果有

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

此时需要 $c_1 n - 5cn \geq 0$ ，即 $c \leq \frac{c_1}{5}$ ，有 $T(n) \geq cn \log n$ ，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 $O(n\log n) + O(n) = O(n\log n)$

[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 $MainElem \Rightarrow$ 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:

$$cnt_e = cnt_l + cnt_r \leq \lfloor \frac{n_l}{2} \rfloor + \lfloor \frac{n_r}{2} \rfloor = \lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \rfloor + \lfloor \frac{\lceil \frac{n}{2} \rceil}{2} \rfloor \leq \lfloor \frac{\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$$

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

```
1  Algorithm: IsMainElem(A[1..n])
2  // arr[i] == true: A[i] is a MainElem
3  arr := [false]; // all false at first
4  // step1: Find the MainElem
5  idx := FindMainElem(A[1..n]);
6  // step2: compare with MainElem
7  for i := 1 to n do
8      if A[i] == A[idx] then
9          arr[i] = true;
10 return arr;
```

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

```

6 // recurse to left half and right half
7 m := (l+r)/2;
8 il := FindMainElem(A[l..m]);
9 ir := FindMainElem(A[m+1..r]);
10 // count the number of A[il] in A[l..r]
11 if Count(A[l..r],il) > n/2 then // found the MainElem in A[l..r]
12     return il;
13 // count the number of A[ir] in A[l..r]
14 if Count(A[l..r],ir) > n/2 then // found the MainElem in A[l..r]
15     return ir;
16 // both il and ir are not the MainElem
17 return 0;

```

```

1 Algorithm: Count(A[l..r],i) // count the equal elems to A[i] in A[l..r]
2 cnt := 0;
3 if i != 0 then
4     for j:=l to r do
5         if A[i] == A[j] then
6             cnt := cnt + 1;
7 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, \dots, e_m , 即 $\forall e_i, cnt(e_i) \geq \lfloor \frac{n}{k} \rfloor + 1 > \frac{n}{k}$, 则:

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

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

```

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

```

1 Algorithm: FindKMainElem(A[l..r], k) // find the kMainElem in A[l..r]
2 // base case: A only contains less than k elems, then all elems are kMainElem
3 n := r-l+1;
4 I := []; C := [];
5 if n < k then
6     I := [l..r];
7     C := [1, 1, ..., 1]; // all-one array of size n
8     return I, C;
9 // recurse to left half and right half
10 m := (l+r)/2;
11 Il, Cl := FindKMainElem(A[l..m], k);
12 Ir, Cr := FindKMainElem(A[m+1..r], k);
13 // count the number of A[il] in A[l..r] for each il in Il
14 for each il in Il do
15     if Count(A[l..r], il) > n/k then // found one kMainElem in A[l..r]
16         I.append(il); C.append(cnt);
17 // count the number of A[ir] in A[l..r] for each ir in Ir
18 for each ir in Ir do
19     if Count(A[l..r], ir) > n/k then // found one kMainElem in A[l..r]
20         I.append(ir); C.append(cnt);
21
22 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 $\operatorname{argmax}\{C[i]\}$ ，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 $\text{mstr}[j] = 0$ ，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 $\text{mstr}[j] = 1$ ，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：
$$\text{mstr}[j] = \sum_{i=1}^{n-1} \text{bitStr}[i][j];$$

2. 算法伪码：6分

```
1  Algorithm FindMissingString1(bitStr[1..n-1][1..k]) // => O(nk)
2  // base case: k = 1, n-1 = 1, just return the flipped bitStr[1][1]
3  if k == 1 then
4      return bitStr[1][1] ⊕ 1; // 1 => 0, 0 => 1
5  // bool sum for each bit in column j
6  mstr := [00..0]; // all-zero array of size k
7  for j:=1 to k do
8      sum := 0;
9      for i:=1 to n-1 do
10         sum := sum ⊕ bitStr[i][j];
11     mstr[j] := sum;
12 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

```
1 Algorithm FindMissingString2(bitStr[1..n][1..k], j, mstr) // => O(n)
2 // base case1: nj = 1, just return the flipped bitStr[1][j]
3 if n == 1 then
4     mstr[j] := bitStr[1][j] ⊕ 1; // 1 => 0, 0 => 1
5     return;
6 // bool sum for column j
7 sum := 0;
8 for i:=1 to n do
9     sum := sum ⊕ bitStr[i][j];
10 mstr[j] := sum;
11 // get subBitStr containing those who hold the same jth bit with mstr
12 subBitStr := [];
13 for i:=1 to n do
14     if bitStr[i][j] == mstr[j] then
15         subBitStr[i] = bitStr[i];
16 // recurse to subBitStr to find (j+1)th bit
17 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 $O(1)$;
