

2025 年秋季学期算法基础期中考试（答案）

学号 _____ 姓名 _____

主定理： 令 $a \geq 1$ 和 $b > 1$ 是常数， $f(n)$ 是一个函数， $T(n)$ 是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将 n/b 解释为 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$ 。那么 $T(n)$ 有如下渐进界：

1. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = O(n^{\log_b a - \varepsilon})$ ，则 $T(n) = \Theta(n^{\log_b a})$ 。
2. 若对整数 $k \geq 0$ 有 $f(n) = \Theta(n^{\log_b a} \lg^k n)$ ，则 $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ 。
3. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$ ，且对某个常数 $c < 1$ 和所有足够大的 n 有 $af(n/b) \leq cf(n)$ ，则 $T(n) = \Theta(f(n))$ 。

一、判断题（根据表述判断正误，并简要说明理由；每题 5 分，共 30 分；每一题判断答案正确得 2 分，解释理由正确得 3 分。）

1. (T, F) 递归式 $T(n) = 2T(\frac{2n}{3}) + n \log n$ 的解满足 $T(n) = \Theta(n^2)$ 。

F 错误。

由主定理可得，对于 $\varepsilon = \frac{1}{2} \log_{\frac{3}{2}} \frac{4}{3}$ 有 $f(n) = O(n^{\log_{\frac{3}{2}} \frac{4}{3} - \varepsilon}) = O(n^{1 + \log_{\frac{3}{2}} \frac{4}{3} - \varepsilon})$ ，则 $T(n) = \Theta(n^{\log_{\frac{3}{2}} \frac{4}{3}})$ 。

2. (T, F) 记 $\sigma(n)$ 表示数字 n 的因子个数，例如 $\sigma(6) = 4$ ，因为 1, 2, 3, 6 都是 6 的因子。那么 $\sum_{i=1}^n \sigma(i)$ 是 $\Theta(n\sqrt{n})$ 的。

F 错误。

对于每个数字 i ，包含 i 这个因子的数字有 $i, 2i, 3i, \dots, \lfloor \frac{n}{i} \rfloor i$ ，一共 $\lfloor \frac{n}{i} \rfloor$ 个，求和就是：

$$\sum_{i=1}^n \frac{n}{i} = \Theta(n \log n)$$

等价于调和级数。

3. (T, F) 归并排序是稳定的，其时间复杂度在任何情况下都一定是 $O(n \log n)$ 。

T 正确。

均是归并排序的基本结论。

4. (T, F) 计数排序 (Counting Sort) 的空间复杂度是 $O(1)$ 。

F 错误。计数排序需要 $O(n+k)$ 的空间。

5. (T, F) 某二叉树的前序、中序、后序遍历中，任意给定两项，都可以唯一确定第三项。

F 错误。

只有前序和后序时无法确定中序遍历，例如前序是 1, 2，后序是 2, 1，此时无法确定 2 是左儿子还是右儿子。

6. (T, F) 对于一棵黑高为 h 的红黑树，其内部节点（也就是有键值的点）的数量至多是 $2^{h+1} - 1$ 。

F 错误。

至多为 $2^{2h} - 1$ ，构造从根开始黑红交替的满二叉树即可。

二、简答题（根据题目要求写出解答过程；每题 10 分，共 40 分）。

1. 对于递归式 $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log_2 n}$ ，请给出渐近确界。

解：

$$T(n) = 2T(n/2) + n/\log n$$

由于 $n^{\log_2 2} = n$ ，而 $f(n) = n/\log n$ ，此时主定理不适用。考虑递归树分析：

$$\begin{aligned} T(n) &= 2T(n/2) + \frac{n}{\log n} \\ &= 4T(n/4) + \frac{n}{\log n} + \frac{n}{\log \frac{n}{2}} \\ &= n \sum_{i=0}^{\log n - 1} \frac{1}{\log \frac{n}{2^i}} \end{aligned}$$

假设存在整数 k 使得 $2^k \leq n < 2^{k+1}$ 。这意味着 $k \leq \log_2 n < k+1$ ，由于 $\log_2 n \geq k$ ，我们有 $\log_2 n - i \geq k - i$ 。我们来分析求和部分 $S(n) = \sum_{i=0}^{k-1} \frac{1}{\log_2 n - i}$ 的上界：

$$S(n) = \sum_{i=0}^{k-1} \frac{1}{\log_2 n - i} \leq \sum_{i=0}^{k-1} \frac{1}{k - i} = \sum_{j=1}^k \frac{1}{j} = H_k$$

其中 H_k 是调和级数，我们知道 $H_k = \Theta(\log k)$ 。所以， $T(n) \leq n \cdot \Theta(\log k) = O(n \log k)$ 。而另一侧的下界同理，可以得到 $T(n) = \Omega(n \log k)$ 。综合得到 $T(n) = \Theta(n \log k)$ 。

使用 Akra-Bazzi 方法也可以求出该解，具体解法可参照课本。

2. 有一排共 n 桶染料，每桶染料的颜色用一个 $[1, n]$ 之间的数字表示。画家小 S 想要使用至少 k 种颜色作画，但是他只能使用**连续的**若干桶染料（也就是若干桶相邻的染料）。现在请你求出：他最少需要用多少桶染料，才能够满足至少 k 种颜色的要求。给出伪代码，并说明算法的工作原理。你的算法时间复杂度需要是 $O(n)$ 的，超出会酌情扣分。

注意，你的算法不需要考虑无解的情况。

解：

考虑使用双指针 + 桶计数。我们使用 $left$ 和 $right$ 维护一个滑动的窗口，也就是一个连续子数组 $[left \dots right]$ ，同时用桶维护当前子数组中不同的颜色数量 $total$ 。我们的目标是最小化这个区间的长度，也就是 $right - left + 1$ 。

首先初始化 $left = 1, right = 0, total = 0, answer = n, cnt[] = 0$ ，然后每一步中：

- 不断向右扩展 $right$ ，直到当前窗口包含 k 种不同的颜色
 - $cnt[color[r]] \leftarrow cnt[color[r]] + 1$ ，如果此时 $cnt[color[r]]$ 从 0 变成 1，说明出现了一种新的颜色，那么 $total \leftarrow total + 1$
 - $r \leftarrow r + 1$ ，如果 $r > n$ ，则退出循环
- 更新答案 $answer \leftarrow \min(answer, right - left + 1)$
- $cnt[color[l]] \leftarrow cnt[color[l]] - 1$ ，如果此时 $cnt[color[l]]$ 从 1 变成 0，说明将当前窗口右移一位后，少出现了一种颜色，那么 $total \leftarrow total - 1$
- $left \leftarrow left + 1$ ，表示窗口的左端点向右移动一步。

Algorithm 1 最短连续 k 色子数组

Require: 颜色数组 $colors[1\dots n]$, 目标颜色数 k

```
1:  $n \leftarrow \text{length of } colors$ 
2:  $counts \leftarrow \text{new array of size } (n+1) \text{ initialized to } 0$ 
3:  $left \leftarrow 1$ 
4:  $right \leftarrow 0$ 
5:  $total \leftarrow 0$ 
6:  $answer \leftarrow n$ 
7: while  $left \leq n$  do
8:   while  $total < k$  and  $right < n$  do
9:      $right \leftarrow right + 1$ 
10:     $c_{right} \leftarrow colors[right]$ 
11:    if  $counts[c_{right}] = 0$  then
12:       $total \leftarrow total + 1$ 
13:     $counts[c_{right}] \leftarrow counts[c_{right}] + 1$ 
14:  if  $total < k$  then
15:    break
16:   $answer \leftarrow \min(answer, right - left + 1)$ 
17:   $c_{left} \leftarrow colors[left]$ 
18:   $counts[c_{left}] \leftarrow counts[c_{left}] - 1$ 
19:  if  $counts[c_{left}] = 0$  then
20:     $total \leftarrow total - 1$ 
21:   $left \leftarrow left + 1$ 
22: return  $answer$ 
```

3. 已知在一个区间集合中, 被最多数区间覆盖的点被称为最大重叠点, 且它一定可以是某个区间的端点。请你简述一个数据结构的设计思路, 使它支持动态查询一个动态区间集合中的最大重叠点 (任意一个符合要求的端点即可), 并分析各操作的时间复杂度。

解:

利用扩展红黑树, 维护区间集合内的区间端点。其中左端点权重为 $+1$, 右端点权重为 -1 , 任意位置的覆盖数量可以表示为从最左端点到该位置的权重前缀和。最大重叠点即为整个序列中权重前缀和的最大值所对应的端点。

在红黑树中, 每个节点存储以下三个辅助信息:

- 子树权重和 $\text{sum}(x)$: 以该节点为根的子树中所有权重之和;
- 最大前缀和 $\text{max_prefix}(x)$: 该子树内从最左端点出发可达到的最大前缀权重和;
- 最大前缀位置 $\text{argmax}(x)$: 达到该最大前缀和的端点位置。

当红黑树结构发生变化 (如插入、删除或旋转) 时, 需要在回溯过程中重新计算每个受影响节点的辅助信息, 即:

首先计算该节点的子树权重和: 等于左子树权重和 + 当前节点权重 + 右子树权重。

然后计算最大前缀和: 取三种可能的情形中的最大者:

- 左子树内部的最大前缀和
- 经过当前节点 (左子树权重和 + 当前节点权重)
- 跨过当前节点并延伸到右子树的最大前缀 (左子树权重和 + 当前节点权重 + 右子树的最大前缀和)。

与最大前缀和对应的位置 $\text{argmax}(x)$ 同步更新为产生该最大值的端点。

因此, 每当插入或删除一个区间的端点时, 只需在回溯更新过程中依次执行上述“重新计算节点信息”的过程, 复杂度为 $O(\log n)$

最终，红黑树根节点的 argmax 即为当前区间集合的最大重叠点，查询时间为 $O(1)$

4. 对于图一所给的斐波那契堆，黑色节点表示结点的 mark 属性为 true :
 - 1) 抽取最小结点，画出操作后最后的结果（需要考虑 CONSOLIDATE(合并) 操作,）。
 - 2) 在 1) 的基础上，把关键字为 30 的节点的值降低为 4，画出操作后的最后的结果（需考虑 CASCADING-CUT(级联切断) 操作）。

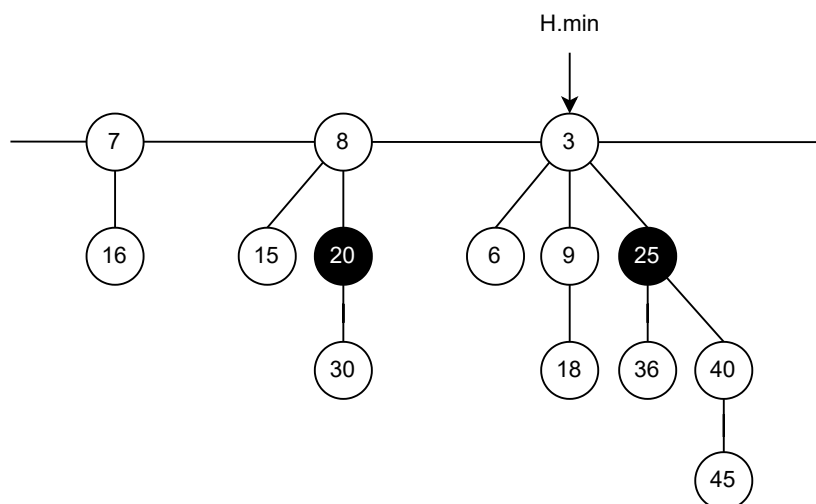
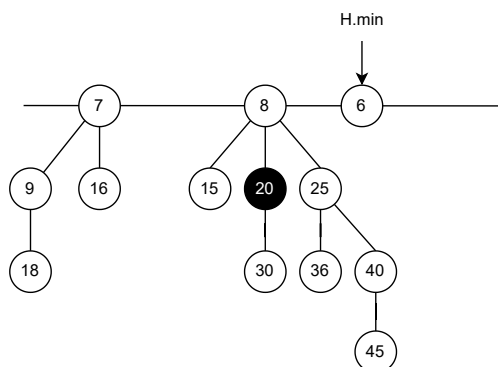
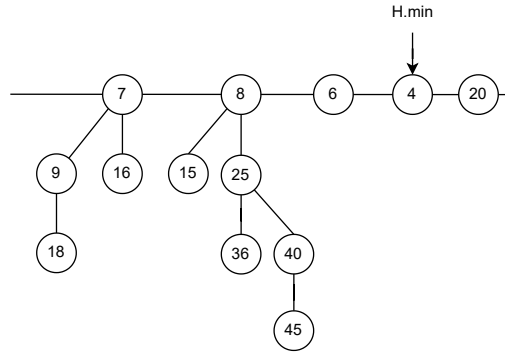


图 1: 斐波那契堆

1)



2)



三、综合题（根据题目要求写出解答过程；每题 15 分，共 30 分）。

1. 线性排序算法

(a) 基数排序 (Radix Sort) 的正确性依赖于一个稳定的子排序算法。假设我们使用计数排序 (Counting Sort) 作为 RADIX-SORT 所需的稳定子排序算法 STABLE-SORT，请写出 COUNTING-SORT(A, d) 的伪代码，该算法用于对一个包含 n 个元素的数组 A 进行排序，其中每个元素都是不大于 m 的非负整数。

解：

Algorithm 2 COUNTING-SORT(A, m)

```

1: let  $B[0..n], C[0..m]$  be a new array
2: for  $i = 0$  to  $m$  do
3:    $C[i] = 0$ 
4: for  $j = 1$  to  $A.length$  do
5:    $C[A[j]] = C[A[j]] + 1$ 
6: for  $i = 1$  to  $m$  do
7:    $C[i] = C[i] + C[i - 1]$ 
8: for  $j = A.length$  to  $1$  do
9:    $B[C[A[j]]] = A[j]$ 
10:   $C[A[j]] = C[A[j]] - 1$ 
11: Return  $B$ 

```

(b) 已知待排序的 n 个元素中的每一个都有 d 位数字，且每一位数字都可以取 0 到 $k-1$ 中的任意一个值（即最多有 k 个可能的值），请推导出 RADIX-SORT 的总时间复杂度。

解：

1. RADIX-SORT 的核心是一个从 1 到 d 的 for 循环。
2. 循环体内部调用一次 STABLE-SORT（即计数排序），其时间复杂度为 $\Theta(n+k)$ 。由于循环执行 d 次，总时间复杂度是 d 次 $\Theta(n+k)$ 的总和，即 $\Theta(d(n+k))$ 。

(c) 在实际应用中，我们经常处理 b 位的二进制数。一种高效的策略是不按 1 位排序，而是将 b 位分成 d 组，每组 r 位（其中 $r \leq b$ ）。我们对这 $d = \lceil b/r \rceil$ 组分别使用基数排序（同样使用计数排序作为子算法）。此时，RADIX-SORT 的总时间复杂度 $T(n, b, r)$ 是多少？

解：

- 每次计数排序的时间复杂度变为 $\Theta(n+k) = \Theta(n+2^r)$ 。
- 总共需要排序的“位数” $d = \lceil b/r \rceil$ 。
- 因此，总时间复杂度 $T(n, b, r) = \Theta(d(n+2^r)) = \Theta(\lceil b/r \rceil (n+2^r))$ 。

2. 在标准二叉搜索树中，我们可以高效支持 SEARCH、INSERT 和 DELETE 等操作，但无法高效处理顺序统计问题，如“第 i 小的关键字是什么？”或者“关键字 x 的排名是多少？”

为解决此问题，我们引入顺序统计树 (Order-Statistic Tree, OST)：它在每个结点 x 中额外维护一个属性 $x.size$ ，表示以 x 为根的子树中包含的结点总数（包括 x 自身）。空指针（记为 NIL）的 $size$ 定义为 0。利用 $size$ 信息可实现 OS-SELECT(x, i)，返回以 x 为根的子树中第 i 小的关键字对应的结点 ($1 \leq i \leq x.size$)。

(a) 在一棵 OST 中，某结点 x 满足 $x.left.size = 3$, $x.right.size = 5$ 。

1. 计算 $x.size$ 。

2. 若调用 OS-SELECT($x, 4$)，将返回哪个结点？简要说明理由。

(b) 用伪代码实现函数 OS-SELECT(x, i)，其中 x 是一棵 OST 的根结点， i 为正整数。若 $1 \leq i \leq x.size$ ，返回对应结点；否则返回 NIL。

(c) 假设 OST 由关键字 $1, 2, \dots, n$ 按均匀随机排列插入构建。令 D_i 表示第 i 小关键字对应结点的深度（根深度为 0）。

1. 证明：对任意 $1 \leq i \leq n$,

$$\mathbb{E}[D_i] = H_i + H_{n-i+1} - 2,$$

(提示： $\ln(k+1) \leq H_k \leq 1 + \ln k$ 对所有整数 $k \geq 1$ 成立，其中 $H_k = \sum_{j=1}^k \frac{1}{j}$ 为第 k 个调和数。)

2. 利用 (1) 的结论，推导 OS-SELECT($T.root, i$) 的时间复杂度。

解：(a)

1. 由 $size$ 定义：

$$x.size = x.left.size + x.right.size + 1 = 3 + 5 + 1 = \boxed{9}.$$

2. 设 $r = x.left.size + 1 = 3 + 1 = 4$ 。根据 OST 性质， x 是其子树中第 $r = 4$ 小的元素。因此，OS-SELECT($x, 4$) 返回结点 x 自身。

(b)

Algorithm 3 OS-SELECT(x, i)

Require: A node x and an integer i

```

1: if  $x = \text{NIL}$  or  $i < 1$  or  $i > x.size$  then
2:   return NIL
3: end if
4: if  $x.left = \text{NIL}$  then
5:    $r \leftarrow 1$ 
6: else
7:    $r \leftarrow x.left.size + 1$ 
8: end if
9: if  $i = r$  then
10:  return  $x$ 
11: else if  $i < r$  then
12:  return OS-SELECT( $x.left, i$ )
13: else
14:  return OS-SELECT( $x.right, i - r$ )
15: end if

```

(c)

1. 证明：在随机 BST 中，结点 $j \neq i$ 是结点 i 的祖先当且仅当在区间 $[\min(i, j), \max(i, j)]$ 中 j 最先插入，故

$$\Pr[j \text{ 是 } i \text{ 的祖先}] = \frac{1}{|i - j| + 1}.$$

深度 D_i 等于祖先个数，因此

$$\mathbb{E}[D_i] = \sum_{j=1}^{i-1} \frac{1}{i-j+1} + \sum_{j=i+1}^n \frac{1}{j-i+1} = \sum_{k=2}^i \frac{1}{k} + \sum_{k=2}^{n-i+1} \frac{1}{k} = (H_i - 1) + (H_{n-i+1} - 1) = H_i + H_{n-i+1} - 2.$$

2. OS-SELECT($T.\text{root}, i$) 访问 $D_i + 1$ 个结点（深度为 D_i ，路径含 $D_i + 1$ 个结点），故运行时间 $= D_i + 1$ 。取期望：

$$\mathbb{E}[\text{time}] = \mathbb{E}[D_i] + 1 = H_i + H_{n-i+1} + O(1).$$

由于 $H_k \leq \ln k + 1$ ，有

$$\mathbb{E}[D_i] + 1 \leq 2(\ln i + \ln(n - i + 1)) + O(1) \leq 4 \ln n + O(1) = O(\log n).$$

因此，时间复杂度为 $O(\log n)$ 。

四、附加题（根据题目要求写出解答过程，共 10 分）。

你得到了 n 块按顺序排成一排的「记忆碎片」，第 i 块碎片上记录的记忆发生时间为 t_i 。已知 $t_1, t_2, \dots, t_n$ 是 $1 \sim n$ 的一个排列，也就是说，这 n 块碎片本来应该是按 $1, 2, \dots, n$ 的时间顺序出现的，但现在被打乱了。你希望通过若干次交换操作，把这些碎片恢复成严格单调递增的顺序：

$$(t_1, t_2, \dots, t_n) \longrightarrow (1, 2, \dots, n).$$

你拥有一个「回忆能力值」 k 。一次操作中，你可以任选两块碎片，设它们记忆的时间分别是 x 和 y ，当且仅当 $|x - y| \leq k$ 时，你可以交换它们现在所在的位置。交换的体力消耗规则如下：

- 若 $|x - y| = 1$ ，也就是这两段记忆发生在相邻的两个时间点上，由于发生时间过于靠近的两块碎片很难回想，这次交换需要消耗你 **1 点** 精力；
- 若 $2 \leq |x - y| \leq k$ ，则这次交换**不需要**消耗精力，代价为 0。

你可以进行任意多次满足条件的交换操作。请你计算：为了将碎片恢复成 $(1, 2, \dots, n)$ ，你最少需要消耗多少点精力。

设计时间复杂度尽可能小的算法。（输入： $n, [t_1, t_2, \dots, t_n], k$ ；输出：最少需要消耗的精力值）

解：下面分情况讨论。

一、 $n < 4$ 的情况 排列的状态总数不超过 $3! = 6$ 个。直接写出 6 种排列的最优代价表。不再赘述。

二、 $n \geq 4$ 的情况

(1) $k = 1$ 这是经典的“只用相邻数值换位”把一个排列排成升序，所需的最少此类交换次数等于这个排列的逆序对个数。形式化地，定义

$$\text{inv}(t) = \#\{(i, j) \mid 1 \leq i < j \leq n, t_i > t_j\}.$$

那么当 $k = 1$ 时，答案就是

$$\boxed{\text{ans} = \text{inv}(t)}.$$

可以用归并排序维护 $\text{inv}(t)$ 计数器，可以在 $O(n \log n)$ 时间内统计。下面是算法：

1. 若当前这段只剩下一个元素，则它本身已经有序，逆序对数为 0。
2. 否则，把当前这段分成长度尽量相等的左右两半，递归地分别对左半段、右半段做同样的处理，这一步会得到“左半段内部的逆序对数”和“右半段内部的逆序对数”，同时也会得到“排好序的左半段”和“排好序的右半段”。
3. 接下来要把两个已经排好序的子段“像归并排序那样”合成一段有序的数组。合并时，左半段和右半段都从头开始往后看：

- 如果左半段当前的数 $\leq$ 右半段当前的数，就先把左半段的这个数放进结果里，然后左半段指针后移；
 - 否则（也就是左半段当前的数 $>$ 右半段当前的数），说明右半段这个数应该排在当前左半段这个数以及左半段后面所有还没放进去的数的前面。由于左半段目前还没放进去的元素都是“比这个右半段元素大”的，所以这一时刻就一口气产生了“左半段当前还剩多少个元素”这么多个逆序对；把这个数量加到总答案里，再把右半段当前的数放进结果，然后右半段指针后移。
4. 两边有一边先用完了，就把另一边剩下的所有元素按顺序接到结果后面；这一部分不会再产生新的逆序对。
5. 当前这段的逆序对数 = “左半段内部的逆序对数” + “右半段内部的逆序对数” + “合并时数到的那些跨左右的逆序对数”。

(2) $k = 2$ 当 $|x - y| = 2$ 时交换免费；而差为 2 的两个数一定同奇偶，于是：

- 所有奇数记忆 (1, 3, 5, ...) 之间可以免费任意重排；
- 所有偶数记忆 (2, 4, 6, ...) 之间也可以免费任意重排；
- 想要把一个“奇数位置上却放了偶数”的错误修正，就必须和某个“偶数位置上却放了奇数”的错误做一次 $|x - y| = 1$ 的付费交换。

目标序列 (1, 2, ..., n) 在位置 i 上的奇偶为 $i \bmod 2$ 。统计原序列中奇偶不匹配的位置数：

$$\text{mism} = \#\{i \mid (t_i \bmod 2) \neq (i \bmod 2)\}.$$

一条付费交换（交换一对差为 1 的数，例如 5 和 6）会同时把一个“奇放偶位”的错误和一个“偶放奇位”的错误一起消掉，因此所需的付费次数正好是

$$\boxed{\text{ans} = \frac{\text{mism}}{2}}.$$

在 $n \geq 4$ 的前提下，奇偶不匹配的个数一定是偶数，上式必为整数。该步骤只需线性时间 $O(n)$ 。

(3) $k \geq 3$ 当能力值至少为 3 时，我们既可以免费交换差为 2 的一对（保持奇偶），也可以免费交换差为 3 的一对（改变奇偶）；当 $n \geq 4$ 时，可以证明任意一个数字都能通过若干次免费交换移动到任意位置，因此不需要做任何一次差为 1 的付费交换，答案为

$$\boxed{\text{ans} = 0}.$$