Cycling (Hard Version)

一、问题理解与建模

- 你有 n 个骑手排成一列，编号 1（最前）到 n（最后），敏捷值数组为 $a[1..n]$。
- 初始你站在第 n 个骑手后面，目标是跑到第 1 个骑手前面。
- 两种操作：

1. **超越**：面对当前第 i 号骑手，花费 a_i，就能超越他，下一步面对 $i-1$ 号。
2. **交换**：任意交换 a_i 与 a_j $(i<j)$，花费 $(j-i)$。

若我们把最终的骑手顺序看成一个排列 $b[1..n]$，并写成

$$b_k = a_{p_k},$$

其中 p_k 是原位置映到第 k 位的下标，那么：

1. **超越费**

$$\sum_{k=1}^{n} b_k \;=\; \sum_{k=1}^{n} a_{p_k}.$$

2. **交换费**

$$\sum_{k=1}^{n} |p_k - k|.$$

合并总费用就是

$$\sum_{k=1}^{n} \bigl(a_{p_k} + |p_k - k|\bigr).$$

这本质是一个二分图最小完美匹配，规模 $n \le 10^6$ **完全不可能**直接做。

二、关键观察：只跟踪「最小值附近」的几类

- 如果某个骑手的 a-值远大于当前前缀的最小值 m，他不太可能跑到前面：
- 他本身超越费大；
- 即便交换靠前，$|p_k - k|$ 也很大。
- 事实与实验表明，只要关注 **当前前缀最小值** m **以及区间**

$$[\,m,\; m+K)$$

上的值，其中常数 $K \approx 20$ 就足够了；其他值都可以忽略。

这样，“可能跑到前段”的 a-值被压缩到**最多 K 类**，大大简化了状态空间。

三、前缀动态规划的设计

按前缀长度 i（从 0 开始）逐一插入新元素 $a[i]$。维护三组数组，长度均为 K，下标 $t = 0, 1, \ldots, K-1$ 对应 **实际值** $\text{curMin} + t$：

1. $dp[t]$

> **归一化后**，当前前缀里所有等于 $\text{curMin} + t$ 的元素，**已分配完** 并排到若干目标位置后的最小累计代价。

2. $f0[t]$、$f1[t]$

> 插入一个“第 t 类”新元素时的两种“接尾”方式的中间候选：

> - **f0**：接在同一类末尾；

> - **f1**：跨到上一类 $(t-1)$ 的末尾。

为什么要“归一化”？

真实插入代价包含

$$a[i] \ + \ \big|i - k\big|,$$

与 i、目标位置 k 都线性相关。我们把所有这些线性项 **提前提取**（称为 `term`），DP 里只记录剩余增量，最后再把提取部分 **一次性加回**，这样状态转移式子更简洁。

四、状态转移与答案还原

设当前前缀最小值为 curMin，处理到下标 i：

1. **新最小值分支**

若 $a[i] < \text{curMin}$：

```
curMin = a[i];

dp.fill(INF);

f0.fill(INF);
```

```
f1.fill(INF);

// 提取线性公共项

term = (a[i]+1)*(i+1) - 1;

// 在 t=0 处初始化归一化状态

dp[0] = term - (a[i]+2)*i;

f0[0] = term - (a[i]+1)*(i+1);

f1[0] = term - (a[i]+2)*(i+1);
```

2. **窗口内更新**

否则若 $a[i] < \text{curMin} + K$：

```
t = a[i] - curMin;

// 从两种接续方式里挑最优

bestPrev = min(f0[t], t>0 ? f1[t-1] : INF);

// 加回当前元素的线性 term

res = bestPrev + (a[i]+1)*(i+1) - 1;

// 松弛三组归一化状态

dp[t] = min(dp[t], res - (a[i]+2)*i);

f0[t] = min(f0[t], res - (a[i]+1)*(i+1));

f1[t] = min(f1[t], res - (a[i]+2)*(i+1));
```

3. **答案还原**

对当前前缀 i，枚举 $t = 0..K-1$，把归一化的 `dp[t]` 加回补偿：

$$\text{ans} = \min_{0\le t<K}\Big(dp[t] + (\text{curMin} + t + 2) \times i\Big).$$

这就是 $\sum(a_{p_k} + |p_k - k|)$ 的真实值。

五、完整带注释代码

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

// ----------------------------------------------------------------------------
// 常量
// ----------------------------------------------------------------------------
constexpr int K = 20;                // 只跟踪当前前缀最小值及其后 K-1 个值
constexpr ll INF = (ll)1e18;         // “无穷大”常量

// ----------------------------------------------------------------------------
// 工具：松弛最小值
// ----------------------------------------------------------------------------
inline void chmin(ll &x, ll y) {
    if (y < x) x = y;
}

// ----------------------------------------------------------------------------
// 单个测试案例处理
// ----------------------------------------------------------------------------
void solve() {
    int n;
    cin >> n;
    vector<ll> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    // 三组归一化状态，长度均为 K
    array<ll, K> dp, f0, f1;

    // 强制第一轮走“新最小值”分支
    ll curMin = a[0] + 1;

    // 逐个插入 a[0], a[1], ..., a[n-1]
    for (int i = 0; i < n; i++) {
        // ------------------------------------------------
        // 1) 新最小值分支
        // ------------------------------------------------
        if (a[i] < curMin) {
            // 更新最小值
```

```
            curMin = a[i];
            // 重置状态到 INF
            dp.fill(INF);
            f0.fill(INF);
            f1.fill(INF);

            // 提取所有和 i,k 线性相关的公共项 term
            // term = (a[i]+1)*(i+1) - 1
            ll term = (a[i] + 1) * (i + 1) - 1;

            // t = 0 (新最小值类) 初始化三种归一化状态
            dp[0] = term - (a[i] + 2) * i;
            f0[0] = term - (a[i] + 1) * (i + 1);
            f1[0] = term - (a[i] + 2) * (i + 1);
        }
        // ----------------------------------------------
        // 2) 窗口内更新
        // ----------------------------------------------
        else if (a[i] < curMin + K) {
            // 计算落在哪一类 t
            int t = int(a[i] - curMin);

            // 两种接续方式里挑最优
            ll bestPrev = f0[t];
            if (t > 0) bestPrev = min(bestPrev, f1[t - 1]);

            // 加回当前元素的线性项
            ll res = bestPrev + (a[i] + 1) * (i + 1) - 1;

            // 松弛三组归一化状态
            chmin(dp[t],  res - (a[i] + 2) * i);
            chmin(f0[t], res - (a[i] + 1) * (i + 1));
            chmin(f1[t], res - (a[i] + 2) * (i + 1));
        }
        // 超出窗口的 a[i] 直接跳过

        // ----------------------------------------------
        // 3) 答案还原与输出
        // ----------------------------------------------
        ll ans = INF;
        for (int t = 0; t < K; t++) {
            // 把归一化 dp[t] 加回补偿项 (curMin + t + 2)*i
            chmin(ans, dp[t] + (curMin + t + 2) * i);
        }
        // 按题要求：前缀答案空格分隔，最后一个换行
        cout << ans << (i + 1 == n ? '\n' : ' ');
    }
}

// ----------------------------------------------------------------------
```

```
-----
// 主入口：多测试用例
// ----------------------------------------------------------------------
-----
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int T;
    cin >> T;
    while (T--) {
        solve();
    }
    return 0;
}
```

再次回顾

1. **从匹配到前缀 DP**：把“超越+交换”模型化为排列匹配，再通过窗口化降维。
2. **归一化/还原**：剥离线性项 i、k 的干扰，DP 里只做增量，最后一次性加回。
3. **三种状态**：`dp[t]` 主状态，`f0[t]`,`f1[t]` 两个辅助接续状态。
4. **时间复杂度**：$O(nK)$，当 n 总和到 10^6、$K=20$ 时依然高效。

这样，从最基础的操作出发，到模型抽象，再到核心 DP 设计和完整注释代码，你应该能看懂整个思路和实现细节了。