INCREASING

Statement:

You're given a sequence a_0, a_1, ..., a_{N-1} of positive integers. Processing it left to right, each element must go to exactly one of two people, Boris or Ihor. Each person's received elements, in the order received, must form a strictly increasing sequence.

For every K from 0 to N (each an independent query), decide whether a valid split exists in which Boris receives exactly K elements.

Constraints:

- $2 \leqslant N \leqslant 4 \cdot 10^5$
- $1 \leqslant a_i \leqslant 10^9$ for each $0 \leqslant i < N$

INCREASING

Global observations:

- **No split exists at all (for any K) if a contains a non-increasing subsequence of length 3** — i.e. three indices i < j < k with a_i ⩾ a_j ⩾ a_k (equal values count too, not just strict decreases!). With only two strictly increasing chains available, three such elements can never be validly distributed.
- **Symmetry:** the answer for K equals the answer for N − K — just swap the roles of Boris and Ihor.

INCREASING

Under **subtask 4's** condition, the permutation always looks like a sequence of decreasing runs, each run's starting value being a new record maximum:

4 3 2 1 | 6 5 | 9 8 7 | 10

- If any run has length ⩾ 3, that's already a non-increasing subsequence of length 3 → answer is all-false.

INCREASING

Under **subtask 4's** condition, the permutation always looks like a sequence of decreasing runs, each run's starting value being a new record maximum:

4 3 2 1 | 6 5 | 9 8 7 | 10

- Otherwise every run has length 1 or 2.
  - Length-2 run: the two elements can never sit in the same chain, so exactly one goes to Boris and one to Ihor, and either way works — this contributes +1 guaranteed to both group sizes.
  - Length-1 run: the single element can go to either group freely, with no effect on future decisions.

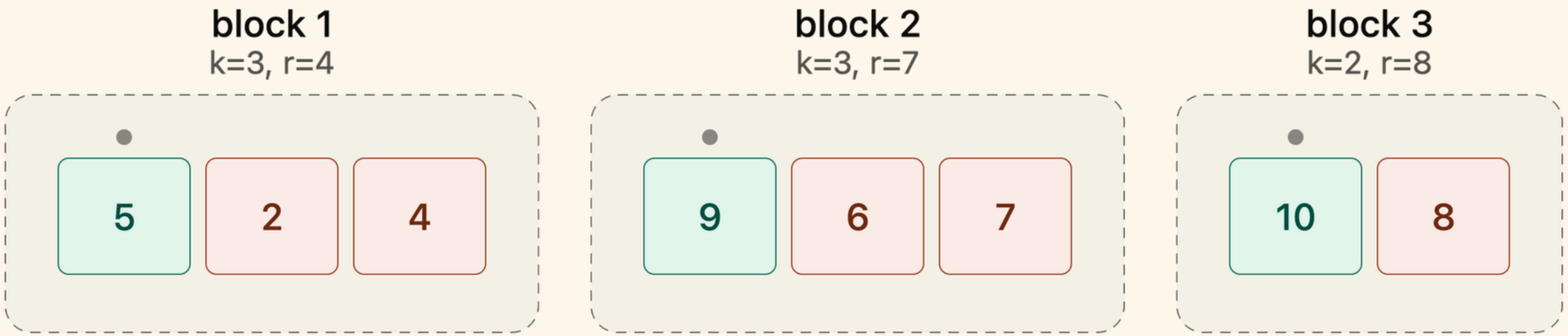
Let b = number of length-2 runs, f = number of length-1 runs. The valid values of K form exactly the contiguous range [b, b+f].

INCREASING

Notation: let p_1 < p_2 < ... < p_t be the positions of the **prefix maxima** (record highs) of a, and let mx_i = a[p_i]. Define **block i** as the elements from position p_i up to (not including) p_{i+1} — so **block i** is mx_i followed by a (possibly empty) increasing run. Let k_i = size of block i, and let r_i denote the value of the last element of block i. We will also consider the case when the run is empty separately.

Structure of a block: the run inside a block must itself be strictly increasing — otherwise it forms a non-increasing triple together with mx_i.

INCREASING



INCREASING

Key deduction: say mx_i is placed as the current last element of chain A. Since the rest of the block is ⩽ mx_i, none of it can extend A — so the entire run after mx_i must go to the other chain, B, in order.
So after processing block i, the two chains' last elements are always mx_i (chain A) and r_i (chain B).
Transition to the next block: mx_{i+1} is a new global record, so it can always extend whichever chain we like. The real question is where the run following mx_{i+1} (call its first element f) can go.
Check first (before anything else): if f ⩽ r_i, no valid split exists at all — neither assignment works.

INCREASING

Otherwise (f > r_i), two cases based on comparing f to mx_i:

1. f > mx_i: both assignments are valid → **free choice**.
2. r_i < f ≤ mx_i: only one assignment works — mx_{i+1} must join chain A (extending the same chain that held mx_i), and the run joins chain B → **forced**.

INCREASING

DP formulation. Let dp[i][j] = true if, after processing i blocks, chain A (the one ending in mx_i) has exactly j elements.

- Forced case: dp[i][j] → dp[i+1][j+1] only.
- Free case (block i+1 has k elements total, so its run has k−1 elements): dp[i][j] → both dp[i+1][j+1] and dp[i+1][j+(k−1)].

Reformulating as knapsack. Every block guarantees a +1. On top of that, a free block optionally adds (k−1) − 1 = k−2 more. Sum up all the guaranteed 1's separately, then run a 0/1 knapsack on just the optional values (k_i − 2), and add the guaranteed constant back at the end. This is $O(N^2)$ and already passes several subtasks.

INCREASING

Speed up to O(N$\sqrt{}$N). The trick: in a 0/1 knapsack, if a value x appears ⩾ 3 times among the items, replace three copies of x with one 2x and one x — every reachable sum is preserved, but with fewer items. Doing this for all values leaves each distinct value appearing at most twice, and since the items still sum to ⩽ N, only O($\sqrt{}$N) items remain. Running the knapsack on these gives O(N$\sqrt{}$N) total (a bitset shaves off another constant factor).