Bubble Sort Machine (Solution)

Subtask 1: At most 10 presses of Button 1

We can simulate Button 1 as defined, in $O(N)$ time per press. By precomputing a prefix sum table after each simulation, we can answer each query involving Button 2 in $O(1)$ time.

Subtask 2: $L_j = R_j = 1$

In this editorial, to reduce the number of edge cases, we assume $A_{N+1} = A_{N+2} = \cdots = \infty$.

Let k be the number of times Button 1 is pressed. The answer to the query is $\min(A_1, \ldots, A_{1+k})$. We prove this below.

Let $A_i = \min(A_1, \ldots, A_{1+k})$, and among such i, choose the smallest. This A_i continues moving left until it reaches the beginning of the array (i.e., position a_1), which happens after pressing Button 1 exactly $i - 1$ times. After reaching a_1, it will not be swapped with a_2 because no elements from $A_{2+k}, A_{3+k}, \ldots$ can reach a_2 by the k-th press.

To answer each query efficiently, we can precompute a prefix min table, allowing us to solve the problem in $O(N + Q)$ time.

Subtask 3: $1 \leq A_i \leq 2$

Analyzing the movement of 1s when Button 1 is pressed, we observe:

- If there is at least one 2 to the left at the time of the press, the 1 moves one step to the left.
- Otherwise, it stays in place.

From this, we can easily compute the final position of each 1 after k presses of Button 1. Using binary search, we can compute the number of 1s in a given range $[l, r]$ in $O(\log N)$ time per query.

With further analysis, and by discovering properties similar to those shown in Subtasks 5 and 6, it is also possible to answer each query in $O(1)$ time.

Subtask 4: $L_j = R_j$

We consider solving this using binary search on the answer. To do this, it suffices to determine whether the answer is at least x.

We transform the array by replacing values less than x with 1, and values at least x with 2. This reduces the problem to the same situation as in Subtask 3, and we only need to compute a_i in that setting.

A 1 reaches position a_i in the following two cases:

- It moved left exactly k times.
- It moved left fewer than k times and then stayed there.

These can be checked by conditions such as whether $A_{i+k} = 1$, or whether there are at least i 1s among $A_1, \ldots, A_{i+k}$.

If we process queries in increasing order of $i + k$, we can answer each in $O(\log N)$ time.

Subtasks 5 and 6

To compute the sum of elements from position L_j to R_j, we can perform two queries of the form " sum of the first n_j elements. "

So we focus on understanding the configuration of the first n_j elements after k presses of Button 1.

We can prove the following: For any n and k, the following two multisets are identical:

- $a_1, \ldots, a_n$ after pressing Button 1 k times.
- The first n elements when $A_1, \ldots, A_{n+k}$ are sorted in ascending order.

To prove this, it suffices to show that the number of elements less than any value x is the same in both multisets. It is enough to show this for the case where $A_i \in \{1, 2\}$. This can be shown using the observations in Subtask 3 and reasoning similar to Subtask 2.

Ultimately, the problem reduces to computing the sum of the smallest n elements in $A_1, \ldots, A_{n+k}$.

There are various ways to handle this type of query. For example, if we sort queries by increasing $n + k$, then we can use a segment tree or Fenwick tree to answer each query in $O(\log N)$ time.

Some solutions may result in $O(\log^2 N)$ time per query, but with a small enough constant factor, they are fast enough to score full marks or at least pass Subtask 5.