Triple Jump (Solution)

For any i and j, we consider when they are possibly used as $a = i$ and $b = j$ (in any one of the queries).

First, if there exists k satisfying $i < k < j$ and $A_k \geq A_j$, we do not need to consider the solution with $a = i$ and $b = j$ because if we put $b = k$, we get a solution which is not worse than it.

Second, if $A_i \leq A_j$ and there exists k satisfying $i < k < j$ and $A_i \leq A_k \leq A_j$, we do not need to consider the solution with $a = i$ and $b = j$ because if we put $a = k$, we get a solution which is not worse than it.

By the above two reasons, the number of pairs i, j we need only consider is $O(N)$. We can list all such pairs if for each i from N to 1, we keep the set of candidate j's. We get a list of pairs i, j which are candidates of a, b.

Then, we process the queries. For each t from N to 1, we keep a sequence v. Here the x-th value of the sequence v is defined as the maximum of the sum $A_a + A_b + A_c$ for a, b, c $(b - a \leq c - b)$ satisfying $t \leq a < b < c = x$.

Once we can keep the sequence v, it is easy to answer the queries.

Let us consider how the sequence v varies if we change t from $s + 1$ to s. For each pair i, j of a candidate of a, b as above satisfying $s = i$, for every k with $2j - i \leq k$, we have

$$v_k = \max(v_k, A_i + A_j + A_k)$$

Using Segment Tree, we can update a candidate in $O(\log N)$ time. We can calculate the answer for each query in $O(\log N)$ time. Therefore, we can solve this task in $O((N + Q) \log N)$ time.