Fully Retroactive Dynamic Read-only Array  Describe and analyze a fully retroactive data structure for storing a list of elements that you can access like an array. Starting with an empty list your data structure should support $\text{Insert}(t, \text{update})$ and $\text{Delete}(t, \text{update})$ where $t$ denotes the time of the operation and $\text{update}$ is one of the following four update operations:

- $\text{addL}(x)$: Add element $x$ to the left of the list.
- $\text{addR}(x)$: Add element $x$ to the right of the list.
- $\text{remL}()$: Remove the leftmost element from the list.
- $\text{remR}()$: Remove the rightmost element from the list.

In addition your data structure should support $\text{Query}(t, \text{query})$ where $t$ denotes the time of the operation and $\text{query}$ is one of following two query operations:

- $\text{size}()$: Return the number of elements in the list.
- $\text{get}(i)$: Counting from the left, return the $i$th element in the list.

Your data structure should support retroactive updates, $\text{Insert}(t, \text{update})$ and $\text{Delete}(t, \text{update})$ where $\text{update} \in \{\text{addL}(x), \text{addR}(x), \text{remL}(), \text{remR}()\}$, and retroactive queries, $\text{Query}(t, \text{query})$ where $\text{query} \in \{\text{size}(), \text{get}(i)\}$, in $O(\log m)$ time per operation, where $m$ denotes the number of update operations (i.e. addL, addR, remL, and remR) in the current timeline of the data structure. The space usage should be $O(m)$, though partial credit will be given for a solution using $O(m \log m)$ space.