Constant time:

__len__: The linked list class methods that add or remove nodes also edit a variable that keeps track of how many nodes are in the linked list. self.__size is increased by 1 or decreased by 1 every time a node is added or removed, respectively. This function only returns that variable, always 1 step, hence is O(1).

append.element: This function creates and adds a node at the tail of the linked list. Because a sentinel node is used, we can go straight to the end of the list to add the node, instead of having to walk from the start to the end to add the node. Therefore, this function takes the same number of steps every time, hence O(1).

rotate_left: This function retrieves and stores the data of the first node of the linked list into a variable (is constant time because we are only retrieving the first element after the header sentinel, which does not change in steps despite the input). Then we append to the linked list with that stored variable (which is constant time as explained previously). Then the first node is removed, making the node next of the sentinel header the second node. (This is constant time because we are removing the first node, which does not change in steps with different inputs). Since all the steps are constant time, rotate_left is also O(1)

__iter__ and __next__: both are constant time, the number of steps does not depend on the input, hence O(1)

Linear Time

get_element_at: is linear because the current pointer has to walk through every node to get to the node at the index, so as n increases, the number of steps increase by N*k (a constant), therefore it is linear time.

insert_element_at: is linear because the current pointer has to walk through every node up to the index, so as n increases, the number of steps increase by n*k (a constant) The insertion itself is O(1). This makes this function linear time.

remove_element_at: is linear because the current pointer has to walk through every node up to the node at the index, so as n increases, the number of steps increase by n*k (a constant). The removal itself is O(1). This makes the function linear time.

__str__: is linear because it has to walk through every node and add them to a list that gets returned, which means that if the list is n length it would have to go through n steps, hence it is linear time.

__reversed__: is linear because starting from the trailer node, it walks through every node until the header so as n increases, the number of steps increase by n*k (a constant), and appends each of these nodes to the reversed linked list object. The appending of the element is constant time as explained previously, so the function is still linear time.

Quadratic Time

None of the methods are quadratic time.