In the Linked_List class, __len__(), append_element(), rotate_left(), __iter__(), __next__() are all can be operated in constant time, O(1). insert_element_at(), get_element_at(), __str__(), __reserved__(), remove_element_at() are all can be operated in linear time, O(n). There's no method need to be operated in quadratic time O()

__len__(): It is O(1) because for this method, we only need to return the size of the Linked_List but not need to visit every node to check or find something. The time it takes to run the method does not depend on the size of the method.

append_element(): It is O(1) because for this method, we only need to link the new node before the tail and after the last Node in the Linked List. We do not need to visit every node. The time it takes to append_element on the end does not depend on the size of the method.

rotate_left(): It is O(1) because to rotate the left, we only need to link the self._head to the second node of the linked list, and append the first node before self._tail and after the last Node. through this process, we don't need to know the size of the Linked_List and don't need to go over every node in the Linked_list

__iter__(): It is O(1) because every node has constant time to walk through. And to iterate the list, it is independent of the size of the linked list.

__next__(): It is O(1) because every node has constant time to walk through. To go from the first node to the next node and to iterate through does not need to know the size of the linked list.

insert_element_at(): It is O(n) because to insert a node into the right index in the linked list, we need to run the linked list and find the position of it in the linked list. On each node, it performs a constant number of steps.

get_element_at(): It is O(n) because to find the element in the linked list, we need to run the linked list and find its position(index) of the linked list. On each node, it performs a constant number of steps. And it does depend on the size of the linked_list.

__str__(): It is O(n) because to print the whole linked list, we need to run all the nodes in the list and after all we can print it. On each node, it performs a constant number of steps. And it does depend on the size of the linked_list

__reversed__(): It is O(n) because to reserve the whole linked list, we need to find out all nodes in the list and then reorder it into a new linked list. To get a new reverse linked list, we need to run all nodes in the linked_list. On each node, it performs a constant number of steps.

Remove_element_at(): It is O(n) because to remove the element on the certain index, we need to run the linked list and find the position of the Node and save the value of it to the value variable, then delete it. On each node, it performs a constant number of steps.