Node Class:

-init: O(1). (Every time this runs, it does the same three steps each time. Even in the worst case, this applies.)

Linked_List Class:

-init: O(1). (Every time this runs, it does the same five steps each time. Even in the worst case, this applies.)

-len: O(1). (Whenever called, it just returns one thing. Does not depend on anything else. Even in the worst case, this applies.)

-append: O(1). (Creates a new node and sticks it to the linked list. Basically drawing arrows. Size of the linked list does not affect how many steps this function will take. Worst case is the same as the best case.)

-current_walk: O(n). (The for loop makes this linear. So in the worst case, it would take a pretty long time.)

-insert_element_at: O(n). (This is very similar to the append function, but it relies on the current walk. Hence, linear. Worst case is not the same as best case here.)

-remove_element_at: O(n). (Again, similar to the append in that we are essentially drawing arrows, but it also relies on the current walk which makes it linear. Worst case will take a while.)

-get_element_at: O(n). (Relies entirely on current walk so linear. Worst case will take a while.)

-rotate_left: O(n). (A combination of a constant function - append - and a linear function - remove - so it is linear. Worst case makes it worse, unlike constant time.)

-str: O(n). (This is linear because of the while loop. Worst case will take a while.)

-iter: O(1). (The function only does one thing. It does not depend on the length of the linked list. Hence, constant. Worst case is no worse than best case.)

-next: O(1). (Same as iter.)

-reversed: O(n). (This is linear because of the while loop. It might look quadratic, but the function that is called in the while loop - append - is constant, so it doesn't do anything. It does take longer if you take the worst case, but it would not be as bad as if it were quadratic.)