Both **constructors**, **__len__**, **__iter__**, **__next__**, **append_element**, and **rotate_left** operate at constant time. Their operations only include instantiation, assignments, getting and returning a value, an if-else statement, and/or raising an exception. They have no loops or any form of iteration through a list, including with special cases, so factors like list size does not affect runtime in any significant way.

The methods **insert_element_at**, **remove_element_at**, and **get_element_at** operate at constant time only if the specified index is out of bounds, in which case the methods immediately raise an exception. Otherwise, they run at linear time since they must run through the list from the header until it reaches the specified index, which is necessary because it is impossible to access the corresponding node directly. As a result, the longer the list and the larger the index (that is within bounds), the longer the runtime will become. Additionally, the operations inside their loops are all in constant time, so the methods do not run in quadratic time.

The **__reversed__** method always runs through the entire list in all cases, so a longer list means more elements to run through, and thus a longer runtime. And since the operations inside its loop only include assignment and element retrieval, which are constant time, the method does not operate in quadratic time, so the method always operates on linear time.

Similarly, the **__str__** method also always runs through the entire list in all cases (except for when the list is empty, in which case it simply returns a string), so a longer list means more elements to run through, and thus a longer runtime. The operations inside the loop involve retrieving values, assignment, instantiation, and the append method, which are all constant time. The join method is used at the end to create the returning string instead of concatenation is to avoid quadratic time.