TEST CASES:

Part 1: This first Linked List test case, I used to see if my append element function was working properly. After every append I checked to see if the str method printed out a string form of my list and also checked to see if the len method updated properly to give the correct size of the Linked List. I tried this with various of inputs for values and each time, all the outputs were just as planned. The length method also gave the correct length each time it was called upon and was updated properly.

Part 2: I first created a linked list object that would create an empty linked list. I tried to insert a method at any index and the index error was handled as planned and printed the statement I had given it in my try and except block in the function. I then created a Linked List by appending various of values and then once again checked to see my string and len functions were working properly and printed the linked list as a string and the correct length. I then attempted to insert a node at a specific index in the range 0 to the size -1, I again tested the str and len method to see that the node was placed in the correct place and that the function updated properly. I used the insert element method again twice: one for negative indices and one for indices greater than or equal to the size of the Linked List. Both time the index error was raised, and the proper statement was printed without changing any aspect of the list and remaining the length the same as before by calling on the functions again. I tried various of idices for each situation and no problem arouse each time I attempted them.

Part 3: I created a Linked List object that would create an empty linked list. Like in part 2, I tried to remove an element from the empty and an exception was raised that printed out the statement explaining that the index is not possible since of course there are no indices in the list when it is empty. I again created a linked list by adding various

elements via the append element function. I then checked to see if the str and len method was working properly. I used two invalid index cases again: one that checked for all negative indices and the other that checked for any index that was equal to or greater than the size of the list. Nothing happens to the Linked List and length when an invalid index is called. The last method call was to a valid index that would properly remove the value for the indicated index and then return the value that was removed and also show the updated length properly shown.

Part 4: This linked list object is filled with many nodes. I used this approach to check to see that my string and length function was working properly for any size. Also, I used this to check various of different indexes to see if the get element at function was working properly and not disrupting anything in the Linked List but just saving the value of the node in a variable. For get element I had two calls that would check any position that would have a valid index and the other two checked both negative indices print out the except and also indices that have an equal to or greater value than that of the size of the Linked List.

Performance Characteristics of the Functions:

Append_element: O(1): The node is added from one away the self.trailer position every time, therefore it adds a value in the same amount of time every single time the function is called.
Insert_element_at: O(n): The function uses one loop to iterate through the Linked List to get to the specified index.
Remove_element_at: O(n): The function uses one loop to iterate through the Linked List to get to the specified index.
Get_element_at: O(n): The function uses one loop to iterate through the Linked List to get to the specified index.

Performance Characteristics of Josephus:

In order to create the list with n numbers, I create a for loop that runs from 1 to n+1 in order to add n nodes to the Linked List. This is accomplished in O(n) since it goes in a single loop at a linear time frame.

In the actual Josephus function, there is only one while loop that will iterate n – 1 times. The rotate left function is constant (O(1)) performance and remove element at is also O(1) performance since it constantly removes at the start of the list rather than having to go through a loop. Therefore, the function as a whole runs in a O(n) performance.