What is the worst-case performance of every method in Array_Deque and Linked_List_Deque? In Array_Deque, how do you distinguish between an empty deque and a deque with one entry? Why does the grow method double the size of the array instead of just increase it by one cell? Which test cases are designed to test these two structures? Make an argument that those test cases are complete; do they cover all expected possibilities when a user programs with your implementation?

In Array_Deque, __str__ and __grow are both linear time performance, O(n), which is the worst case. This is because the larger the input, more steps are done. Push_front and Push_back are also linear time because when self.__size == self.__capacity, the grow method is called which runs in linear time, so both push methods are also linear time. All the other Array_Deque methods are constant time O(1) because they only involve adding or removing or getting element at the head or tail, which does not require a walk and can be accessed with the same number of steps every time.

In Linked_List_Deque, every method is constant time O(1), except __str__ (which runs in linear time since it has to walk through every element). For all the other methods, they are constant time because the insertions, removals, and peeks, all only occur at the head or tail of the list. Because of this, no walk is needed, and the functions take the same number of steps regardless of the size of the input.

In both Deques, we can distinguish between an empty deque and a deque with one entry with self.__size. If self.__size is 0, it is empty, if self.__size is 1, then the deque has 1 item.

The grow method doubles in size instead because if it increases by one cell every insertion, then we would have to call that function many times which would take a huge load when dealing with very large inputs. When doubling, we don’t have to call the function as many times, which would help with performance.

The tests under ### DEQUE TESTS fully test the program. I made sure to involve tests that check the functionality of all 6 deque functions at an empty deque, a deque with 1 item, 2 items, and 3 items. This way push_front and push_back, pop_front and pop_back, and peek_front and peek_back were functional in all of these scenarios. I made sure to involve 3 items to test the grow method. The length method was tested manually on different lengths, and the str function was used in the other test functions and was functional, so that would be used as a test in its own form.