Performance:

In array deque, we store the front and back of the array, and update it. Due to the nature of a deque, operations will only be performed on the front and back of the deque. Since we store the front and back locations, we don't not have to traverse the entire array when we wish to push, pop, or peek at either end. This causes us to have constant time performance instead of linear time. The linked list deque is similar to the array deque, but instead of front and back we have header.next and trailer.previous (in the original linked list class), which allow us to quickly access the first and last data node position. With a little modification to the linked list class, we can push, pop, and peek, at both ends of the linked list deque in constant time instead of linear time. The len methods in both LL deque and array deque are constant as they just return a value. In array deque, both grow and str are linear time. As the input increases in size these two methods take more steps to accomplish their respective goals. The str method in LL deque is also linear. It returns the value from the linked list str method which is linear because as the input increases the number of times the while loop iterates increases, causing more steps.

Array deque specifics:

The way I differentiated between an empty array deque and an array deque with one entry was mainly the size. An array deque with one entry has a size of one and capacity of one; an empty array deque has a size of zero and capacity of one. This allows for an insertion into an empty array deque because the capacity is one, but also means that an empty array deque is truly empty, because there are no values i.e., the size is zero.

The grow function doubles the capacity of the array because if you just increased by one every time it would be drastically less efficient. This is because whenever you need to grow you have to copy the array to a new array. This task is extremely costly, so doing it as little as possible is ideal. Doubling the size decreases the number of times you must copy the array, that is why it is preferred to incrementing the capacity by one.

Test Cases:

The test cases which test decks are in lines 19-300. I thought the most important things to test were the basic functions of both array deque and linked list, and the stress cases. The basic functions are easy enough to check. Testing the base cases includes, pushing front and back, popping, and peeking one or multiple values. The more difficult cases to test are the stress cases. These special cases include things like growing, removing until things are empty, and making sure the attributes of the deque are correct when it becomes empty. In my test cases I strived to ensure my deques worked as intended by extensively testing the stress cases. For instance, testing growing multiple times, emptying an array after growing, and ensuring the attributes of an empty deque are what they need to be. In this way I think my tests are complete as they not only extensively test the stress cases, but through the testing of the base cases and stress cases, the basic functions of the deque are tested thoroughly.