Deque Performance:

For Array_Deque, the str and grow methods perform in linear time O(n) in the worst case. This is because of the while loop. The rest of the methods perform in constant time O(1) (except for the two push functions which also perform in linear time since they call the grow function on certain conditions).

For Linked_List_Deque, we are mainly calling functions from the Linked_List we created in project 2. Besides the insert/get/remove_element_at functions, all the functions we are calling run in constant time O(1) in our original Linked_List, so they also run in constant time here in our Linked_List_Deque.. However, this time around, we updated our Linked_List's current walk function so that it starts at either the head or tail node depending on which is closer to the specified index. This actually makes the insert/get/remove_element_at functions all run in constant time O(1) instead of the previous linear time O(n) since we are only allowed to do operations at the ends of the deque. Hence, the entire Linked_List_Deque runs in O(1) in the worst case. The exception is the str method which still runs in linear time O(n) since it still has a while loop.

In the Array-Deque, we can distinguish between these two cases by checking the size. If the self.__size=0, then we have an empty deque. If self.__size=1, then we have a deque with one entry.

We chose to make the grow method double the size of the array instead of just increasing it by one cell so that we don't have to call that method as many times. It runs in linear time O(n) so it could be costly to call it many times.

All the tests under "#deque tests" and above "#stack tests" are designed to test both the array and linked_list deques. The tests should cover all possibilities. In the very first test, I tested the string function on an empty list. For the rest of the tests, I devoted most of the code to testing the 6 main functions. I basically checked these 6 functions' functionality on deques of size 0,1,2 to check for looping/modding. I did use the len and str functions to check their functionality so these two methods are also being tested at the same time.

Stack and Queue Performance:

In both the Array and Linked_List versions of the stack and queue, the str method runs in linear time O(n) since it relies on a while loop. Otherwise, all the other methods in both versions of both the stack and the queue run in constant time O(1). The one exception to this rule is the push/enqueue method of the stack/queue when using the Array_Deque, since this still relies on the grow function which makes it run in linear time O(n).

Personally, I would have liked to raise exceptions. I would have done this mostly on the peek/pop (sometimes dequeue) methods when used on empty arrays/deques/stacks/queues. I think this makes more logical sense than, say, returning None since removing something from nothing is pretty much illogical and should get an error instead of a None.

The test cases for stacks and queues are all the ones under the "#stacks tests" and "#queue tests" headers, respectively. I tested the str function on an empty str and I also tested each of the 3 main functions on a list of length 0,1,2 for a total of 10 tests for the stack and 10 tests for the queue. I believe that this is a conclusive test as I am able to check the functionality of each method in a variety of settings that covers modding and other tricky scenarios. I did use the len and str functions to check the functionality of the main functions so these two methods are also being tested at the same time.