What is the worst-case performance of every method in Stack and Queue? Include in your analysis discussions of both Array_Deque-based Stacks and Queues and Linked_List_Deque-based Stacks and Queues. What is your opinion of our design decision not to raise exceptions in any methods? Does this limit functionality in any way? Present a reasoned argument to justify your opinion. 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 both Array_Deque and Linked_List_Deque, the __str__ method runs in linear time O(n) since it takes more steps the longer the input as it has to get every item in the Deque. All the other methods are in constant time O(1) because they all involve adding, removing, or peeking at the front or back of the deque, which none require a walk so they take the same number of steps, except for Array_Deque based push and enqueue. In the Array_Deque based push and enqueue, the __grow method is called when self.__size == self.__capacity so they perform in linear time O(n).

Not raising an exception made sense for this program. There are no instances where something out of index can be called so there would be no index errors. Additionally, popping an empty stack or queue would return ‘None’ which makes sense because it shows that there is nothing in the queue or stack. This is much better than raising an error because technically the pop when empty can be used as a check and it does not modify the queue or stack.

The test for queues and stacks are under #QUEUE TEST and #STACK TEST, respectively. The tests were complete and cover all possibilities because I included tests that test the functionality of the enqueue/push and dequeue/pop on an empty queue, a queue with 1 item, 2 items, and 3 items. I also included tests for the peek and length at these different lengths. The __str__ function was tested by being used in the test cases so those were used as a test in itself. Since every possible scenario that a user can do has been accounted for, the tests are definitely complete.