1. **Array_Deque-based Stacks and Queues:**
   Worst-case performance is O(1):
   __init__() (of course); __len__() which simply returns the value of self.size; pop() and peek() because they simply get the element indexed at front (pop() removes it and move the front by one); dequeue() because it calls pop_front() of Deque class, which takes constant time.
   Worst-case performance is O(n):
   enqueue() and push() because they invoke __grow() from Array_Deque while the size reaches the capacity, which is the worst case.

   Worst-case performance is O(n^2):

   __str__() because it loops through the array from front to back and get every element into one string. Every time the string is concatenated it initiates a new string and copy the original string to the new one, so it's linear time inside linear time, which is quadratic time.
   Linked_List-based Stacks and Queues:
   Worst-case performance is O(1):
   Every methods except __str__(). Linked_List-based stacks and queues don't have __grow() method. Push(), pop(), enqueue(), dequeue(), and peek() are all of constant time performance because they just need to manipulate the first or last node of the linked list.
   Worst-case performance is O(n^2):
   __str__(), because it loops through the linked list from front to back and gets every value of the node and concatenate it to the string. Every time the string is concatenated it initiates a new string and copy the original string to the new one, so it's linear time inside linear time, which is quadratic time.
2. I think the design decision not to raise exceptions in any methods is alright and doesn't limit functionality in any way. Instead, for example, when popping an empty stack, "None" is returned instead of exceptions. It also makes sense because it doesn't affect the functionality and tells the client that the stack is empty.
3. In my DSQ_Test.py, almost all the functions test all of deque, stack and queue (for brevity) together because they share some similar attributes. I test the cases that: print the empty stack and queue; print the length of empty stack and queue; push one element into the stack; enqueue one element into the queue; print the length of stack and queue after pushing/enqueueing one element; peek the stack and queue of one entry; push/enqueue 3 elements and output the length; push/enqueue 3 elements and output its string representation; push/enqueue 3 elements and then popping/dequeuing 1 element and output its string representation; peek the stack and queue of 3 elements; pop/dequeue the empty stack/queue. They invoke every methods implemented in the Stack and Queue classes including __grow() and thus I think they cover all expected possibilities.