# Exercise 5

1. A) For a singly linked list in a stack, insertion adds the newly inserted data at the head of the stack because a stack is a Last In First Out (LIFO) data structure. This means when a new list is added to the stack, it is added to the head. It is added to the head because an insertion at the head of the stack always has a constant time complexity, O(1). It does not matter how large the stack is because the list is being added to the front.  
   B) No, we cannot insert a list to the end of a stack. This is because a stack is LIFO. If we inserted a singly linked list to the end of a stack, the order of the stack would change.  
   C) Since a stack works as a LIFO data structure, the time complexity for both pushing and popping data is a constant time complexity, O(1), so there would be very little difference, if any at all, in the difference of operation time.
2. A) For a singly linked list in a queue, we add a new pointer that points to the tail of the linked list because a queue is a First In First Out (FIFO) data structure. This means that the first item to be inserted, is also the first item to be deleted. This means that any new data will be added to the tail of the queue, and since it is being added directly to the tail, it is added in constant time complexity, O(1).  
   B) Yes, we can implement the queue without the tail.  
   C) We know that the queue is a FIFO data structure, and without the tail, the list to be added will have to traverse the entire linked list from the beginning to the end. This results in a linear time complexity, O(n). The dequeuing of a singly linked list will continue to happen at the head, so it will have a constant time complexity, O(1). The difference in operation time depends on the amount of data in the linked list since O(n) will grow linearly with how much data there is. So the difference is O(n) – O(1) which approximates to an operation time of O(n) for enqueuing and dequeuing the queue.  
   D) We could change the behaviour of the enqueuing and dequeuing where we enqueue at the head and dequeue at the tail. This is **not** a good idea. While enqueuing would be constant time complexity, O(1), dequeuing would not be constant because the tail pointer of the list before the list to be dequeued would have to be removed. Since these are singly linked list, the list before the list to be removed would have to be found. So, it would take a linear time complexity, O(n), to find the list which needs to remove its pointer from the list that needs to be dequeued. This is inefficient and a waste of time.