Question 5

Exercise 5 Stacks and Queues are a special form of linked lists with some modifications that makes operation better. For parts 1 and 2 assume we are using a singly linked list

1. In stacks, insertion (push) adds the newly inserted data at head a. why? (0.5 pts) b. Can we insert data at the end of the linked list? (0.5 pts) c. If yes, then what is the difference in operation time (if any) for pushing and popping data from the stack? (1 pt)
2. In Queues, we added a new pointer that points to the tail of the linked list a. why? (0.5 pts) b. Can we implement the Queue without the tail? (0.5 pts) c. If yes, then what is the difference in operation time (if any) for enqueuing and dequeuing data from the stack? (1 pt) d. Can we change the behavior of the enqueue and dequeue where we enqueue at head and dequeue at tail? Do you think it is a good idea? (1 pt)
3. Revisit your answers for part 1 and 2 but now with the assumption that we are using circular doubly linked list (5 pts)

1.

1. To save time and code. Since the 'head' node is already accessibly it is much easier to insert data at the head rather than traverse through the whole list to insert data at the end. Stacks also inherit "last-in first-out" for applications of them, so being pushed in at the head makes this true.
2. Yes, to insert data at the end of a (singly) linked list we would need to traverse to the final node in the list and then link the new data.
3. It would be longer to push and pop data from the end of a stack, since traversing the list is needed O(n). The pointer to the start of the list already exists and makes it much quicker to push and pop data from the list O(1), whereas the end of the list requires a traversal.

2.

1. A pointer to the end of the linked list exists so that the end of the linked list can easily be found, not requiring a traversal of the list. It is not needed, but without it the list would have to be iterated through till the end was found everytime a new item was added to the queue.
2. Yes, but the queue will take longer to enqueue data, as a full iteration of the queue is needed to get to where new data is to be enqueued.
3. Since operation with the tail pointer is one operation to find the end of the queue, the complexity is O(1). We would go from O(1) to traversing the queue until we find the tail item, which means n amount of iterations resulting in O(n) complexity.
4. Yes, it is possible to do so; however, it would be inefficient. Deqeueing at the tail would make it necessary to first traverse the linked list to find the element that is linked to the tail, and then relink the tail to this node. This almost makes the tail obsolete. Enqeueing at the tail is much more efficient, as no traversal is required. Ultimately, reversing the roles of head and tail is not a good idea.

3.

1. (circular doubly linked list)

1. To keep the "last-in first-out" nature of stacks true.

2. Yes, since the list is circular, the head and tail are connected resulting in a very quick way to find the end of the list.
3. Since we no longer have to traverse the list and just follow the head's pointer to the tail node, the time is now constant, which means there is no difference in time. The issue with doing this is not keeping the nature of a stack being "last-in first-out".

2. (circular doubly linked list)

1. By definition, the pointer already exists in a circular doubly linked list.
2. A circular doubly linked list already references its tail from its head, so if you remove this reference it is no longer circular. This is possible but would result in inefficent time and no longer a circular doubly linked list.
3. O(n) is going to be the complexity due to a traversal of the list being required without the reference to the tail, this is because enqueing data requires the tail of the list, and now since there isnt a reference we must traverse each element of the list till we reach the tail O(n).
4. Yes, it is possible to do so. Since we are dealing with a circular doubly linked list, the tasks involved with deqeueing and enqeueing would not change, they would just happen at different places within the list. Consequently, reversal of head and tail would not effect the functionality of the queue and would not necessarily be a bad idea. However, for the sake of nomenclature, it is best to keep the roles as is.