Section 1: CS Fundamentals
Queue

Runtime Analysis of enqueue() and dequeue() methods:
enqueue() - O(1) per operation
dequeue() - Amortized O(1) per operation

enqueue():
A new element to be inserted in the queue is always pushed on top of pushStack. We know pushing an element in a Stack is O(1) operation. Hence, enqueue() method is O(1) in Time Complexity. Space Complexity of enqueue() is O(n) as we need to maintain a stack of n elements.

dequeue():
The Time Complexity of dequeue() is Amortized O(1) with a Worse Case of O(n). The worst case happens when popStack is empty and for a dequeue() call, we need to pop all the n elements from pushStack and push it to popStack. So, this means we have to do 2*n operations, which is O(n). When popStack is not empty, dequeue() is a simple O(1) operation because we just pop and popping from a non-empty Stack is O(1) operation and also, the Space complexity is O(1).

Amortized Analysis:
The main idea in Amortized Analysis is that a worst case operation can alter the Data Structure in such a way that the worst case operation does not happen again, for a long time. So, it amortizes the cost. Let us consider we start with an empty Queue and perform the following operations:
enqueue()_1,enqueue()_2,...,enqueue()_n,dequeue()_1,dequeue_2,...,dequeue_n
Obviously, the worst case Time complexity of a single dequeue() operation is O(n). With n dequeue() operations, the worst case per operation analysis means a time of O(n^2). But, the worst case does not happen in each and every dequeue() operation. A few of them may be expensive while others could be really cheap. From the sequence of operations above, we can see that, the number of dequeue() calls is limited by the number of enqueue() calls made before it and so a single dequeue() is expensive only once per n items where n is the size of the Queue. When popStack is empty, we need to pop all the elements from pushStack and pop it into popStack. The Time Complexity of this process is n(for push operation) + 2*n(for first pop operation) + n-1(for pop operation) which is O(2*n). Hence, the average time per operation is O(2n/2n)=O(1).