CSSE230: Stacks and Queues

# Name(s): Jordan Hazelrigg and Yilun Wu

# Analysis

**Table 1:** Big-Theta runtimes of enqueue and dequeue for 4 implementations of the Queue ADT:

|  |  |  |
| --- | --- | --- |
| Implementation | Enqueue runtime | Dequeue |
| LinkedList | Θ( 1 ) | Θ( 1 ) |
| ArrayList | Amortized Θ (1);Worst case Θ (n) | Θ( N ) |
| Two stacks | Θ( 1 ) | Amortized Θ (1);Worst case Θ (n) |
| Growable circular array | Amortized Θ (1);Worst case Θ (n) | Θ( 1 ) |

# Part 2: Discussion

Justify each of the runtimes in Table 1, as described in the specification:

**LinkedList**

enqueue:

Assuming there is a pointer to the head and the tail of the list, adding an item to the tail of queue can be done by changing around one pointer no matter how long the list itself is

dequeue:

Once again assuming there is a pointer to the last item in the list, only the pointer from the dummy node “head” has to be changed in order to add remove the element from the LinkedList.

**ArrayList**

enqueue:

Adding elements to an ArrayList based queue is amortized Θ( 1 ) because, assuming there is a pointer to the last element in the list, a simple .add could be called to add the element to the end of the list. However this changes when the ArrayList needs to grow to hold more items and all of the items in the list must be copied to the new list.

dequeue:

Removing elements is done in Θ( N ) time because when the first element is removed from the ArrayList, all other elements must be shifted down one index which requires all elements to be accessed which is in linear time.

**Two** **stacks**

enqueue:

For this problem, we use two stacks to save data. In our algorithm, we just need to push the element into the first stack. So it is Θ( 1 )

dequeue:

If we want to dequeuer the first element of the queue, we just pop the first element of the second stack so is Amortized Θ (1). However, when it is empty, we need to pop all elements in the first stack and push them into second stack, so the worst case is Θ(N).

**Growable circular array**

enqueue:

A growable circular array is Θ( 1 ) for adding elements because an item can just be added to the end using a pointer that points to the last element in the list. It is amortized Θ( 1 ) because the array needs to grow once it has been completely filled with objects which requires accessing all indices to copy the list to the new, longer list. When it happened, the worst case is Θ(N).

dequeue:

Removing elements is Θ( 1 ) because an item can be effectively removed without shifting all other items in the list. This is done by relocating the point to the beginning of the list to the index + 1 or “wrapping” the index to the starting index of the entire list. If the element is not contained between the start and end pointers than it can simply be overwritten when enqueuer is called.