CSSE230: Stacks and Queues

# Name(s):

# 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(1) |
| Two stacks | Θ(1) | Θ(n) |
| Growable circular array | Amortized Θ(1)  Worst case Θ(n) | Θ(n) |

# Part 2: Discussion

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

**LinkedList**

enqueue: LinkedList have tail node, you are able to add a new tail node and connect to the old tail node

dequeue: LinkedList have head node, you are able to remove the head node and make the next node the new head node.

**ArrayList**

enqueue: to add a new element at the end of arraylist

dequeue: based on the requirement of this assignment that we don’t have to shrink the size when we remove an element from the arraylist. So dequeuer runtime is just the runtime of remove an element from arraylist.

**Two** **stacks**

enqueue: push the new element to the top of the stack1.

dequeue: you have to push all element from stack1 to stack2, pop the first element from stack2, and then push all element from stack2 back to stack1.

**Growable circular array**

enqueue: add the element to the end of the array, so the runtime is add an element to array. Worst case is ϴ(N) because when you have to grow the array you have to copy all of the element and copy into the new array.

dequeue: When you want to remove an element from growable circular array, you need to loop through the array until you find the first element that is not null, and then remove it.