CSSE230: Stacks and Queues

# Name(s): Naziia Raitova, Ruth Hammond

# Analysis

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

|  |  |  |
| --- | --- | --- |
| Implementation | Enqueue runtime | Dequeue |
| LinkedList | O(1) | O(1) |
| ArrayList | Amortized O(1), worst-case O(N) | O(N) |
| Two stacks | O(1) | O(N^2) |
| Growable circular array | O(N) | O(1) |

# Part 2: Discussion

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

**LinkedList**

enqueue: The LinkedList has pointer to the head, so there is no need to search the

the entire array making the operation O(1)

dequeue: The LinkedList has pointer to the tail, so there is no need to search the entire array – making the operation O(1)

**ArrayList**

enqueue: It is worst-case O(N), because enqueuing at the beginning array would cause every element to shift over.

dequeue: At the beginning of the ArrayList, we have to shift everything by one, making the operation worst-case O(N)

**Two** **stacks**

enqueue: We push to the stack which is an O(1) operation

dequeue: We transfer every element in 1 stack to the other stack and back again, which makes it an O(N^2) operation.

**Growable circular array**

enqueue: If we resize an array, we have to access every element to copy it over to the new array.

dequeue: We only access the last element of the array.