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 | Θ(1) | Θ(N) |
| Two stacks | Θ(1) | Θ(N) |
| Growable circular array | Θ(1) | Θ(1) |

# Part 2: Discussion

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

**LinkedList**

Assuming our implementation keeps track of both the first (head) and last (tail) node’s position as field variables.

enqueue():

The enqueue method accepts the new data value and places it at the end of the queue. In this implementation this is done be creating a new node object and attaching it to the node presently pointed to by the tail variable. This node is now the tail node so the variable is updated accordingly. If the queue is empty then both the head and tail variables point to the new node. It is efficient because all that need to be done is the creation of the new node object and the updating of the tail variable’s value.

dequeue():

The dequeue method removes the head node and returns its data. This will be done by simply reading the data in the node pointed to be the head variable, updating the head variable to point to what used to be the second node in the Linked list, then returning the data. It is efficient because all that must be done is reading the old head’s data and then changing the head variable to point to the second node in the list.

**ArrayList**

enqueue():

The Big-Theta runtime of ArrayList’s enqueue method is typically Θ(1).

enqueue() is done using ArrayList’s add(Object) method. This method appends the specified element to the end of the list; is rather efficient, simply checking to see if there is enough room, then inserting the new data into 1 + the size. The size is stored in a field variable, so no loops are needed. The only time its runtime is not Θ(1) is when the ArrayList has to resize to accommodate the new element.

ArrayList’s add(Object) method’s source code:

**public** **boolean** [http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)add(E e) {[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)

         ensureCapacity([size](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java" \l "ArrayList.0size" \o "int size) + 1);

[elementData](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java" \l "ArrayList.0elementData" \o "Object[] elementData)[[size](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java#ArrayList.0size)++] = e;

**return** **true**;

     }

The only time the enqueue method will take more than 1 operation is when the ArrayList needs to resize to accommodate the new element.

dequeue():

dequeue(): is done using ArrayList’s remove(int index) method. This method removes the element at the specified position in this list, and then shifts any subsequent elements to the left (subtracts one from their indices). The int will be 0 in this case, as it is the tail of the Arraylist. Clearly the remove method is the fundamental inefficiency in this implementation: requiring a for loop to shift every element in the Arraylist to the left by one (O(N)).[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)

ArrayList’s remove(int index) method’s source code:

**public** E [http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)remove(**int** index) {

         rangeCheck(index);

[modCount](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/AbstractList.java" \l "AbstractList.0modCount" \o "int modCount)++;[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)

         E oldValue = elementData(index);

**int** numMoved = [size](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java#ArrayList.0size) - index - 1;[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)

**if** (numMoved > 0)

             System.arraycopy([elementData](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java" \l "ArrayList.0elementData" \o "Object[] elementData), index+1, [elementData](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java" \l "ArrayList.0elementData" \o "Object[] elementData),

index, numMoved);

[elementData](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java" \l "ArrayList.0elementData" \o "Object[] elementData)[--[size](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java#ArrayList.0size)] = **null**; // Let gc do its work

**return** oldValue;[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)

      }

**Two** **stacks**

enqueue**:** This queue implementation has two stacks, inStack and outStack. To add an element to the queue, simply push the element onto the inStack. This is Big theta of one.

dequeue: To dequeue an element with this queue implementation, first check to see if outStack is empty. If it is not pop an element off outstack, else copy all elements to outStack from inStack, then pop off element from outStack. Thus, dequeueing from the stack has big theta of N.

**Growable circular array**

enqueue: Enqueue simply adds an element to the array after end point marker which is big theta of 1. Resizing might be required, but this has an amortized cost of big theta of 1.

dequeue: returns element at start point and then increments the start marker. All dequeue is doing is reading an element from an array and incrementing a counter which is big theta of 1.