CSSE230: Stacks and Queues

# Name(s): Bo Peng and Zach Hull

# 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(N) | θ(1) |

# Part 2: Discussion

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

**LinkedList**

enqueue: When using a linkedlist to enqueue, the program only need to change the tail of the linked list, which could be stored as “this.tail”. We could directly use “this.tail.next = newtail” and “this.tail = this.tail.next” to implement enqueue. All those statements above don’t use any “while” or “for” loop which gives a Big-theta of θ(1).

dequeue: Similarly, when using a linked list to implement dequeue, we could use the “this.head.data” in the linked list, which gives the value of the first node. Then, we could use “this.head = this.head.next” to set the new head. Those statements above don’t need to use any “for” or “while” loop, which gives a Big-theta of θ(1).

**ArrayList**

enqueue: When using an arraylist to implement enqueue, everytime the array that stores the value runs out of space, it needs to call the resize function to create a new array which could contain those objects. The exact run-time depends on how resize() worked. If the arraylist double its space when it calls the resize(), then the big-theta of enqueue is θ(1) since the resize function is not called for so many times.

dequeue: When the ArrayList needs to dequeue, after it gets the first value, it needs to create a new list and put every elements of the old list to the new one from the beginning of the list. Thus, it will take about the amount of the elements in the array times to dequeue which will give a Big-theta of θ(N).

**Two** **stacks**

enqueue: When using two stacks to do the enqueue, we just need to push the value to the first queue as always which would only use one statement :“first.push(data);” which gives the Big- theta as θ(1).

dequeue: When using two stacks to implement dequeue, we need to use a while loop to go though the first stack and pop the value to push it into the second stack until the first stack is empty. Then we pop the first value in the second stack which is the first value that was put int the first queue. Then, that will give the Big-theta of θ(N).

**Growable circular array**

enqueue: When using the Growable circular array to implement the enqueue, we use a “header” which is the index of the start value in the array. When we try to add one value into the growable circular array, we need to use one while loop. The loop goes from the headed until it reaches the “end” of the circular array. It will detect whether the present “space” is null. If so, the value will be put here and end the whole enqueue method. If not, it will start from the index 0. If there is “space” that is null, the value will be put into the array. If not, the resize() method will be called, which is similar to the resize method that is used in the arraylist. Since the resize method doubles the length of the array will also pulls all elements from the old array to the new one. Also, since it is hardly to be called, we could get that the Big-theta of calling resize() is θ(1). However, it actually use one while loop which could go though the array, which gives the Big- theta which is θ(N). Thus, in summery, the Big-theta is θ(N) for the enqueue.

dequeue: In contrast of the enqueue, since the header was recorded. Thus, we could directly get the first value of the circular list. Also, since this circular array is “circular” in contrast of regular array and uses a header to point the first, the dequeue function doesn’t need to create a new array and put those elements from old array to the new array. Thus, the Big-theta is θ(1) instead of θ(N) in contrast of the arraylist.