# CSCE 221-200: Honors Data Structures and Algorithms
# Assignment Cover Page
# Spring 2021

| | |
|---|---|
| **Name:** | Alexander Akomer |
| **Email:** | alexakomer@tamu.edu |
| **Assignment:** | PA 1 |
| **Grade (filled in by grader):** | |

Please list below all sources (people, books, webpages, etc) consulted regarding this assignment (use the back if necessary):

| CSCE 221 Students | Other People | Printed Material | Web Material (give URL) | Other Sources |
|---|---|---|---|---|
| 1. | 1. | 1. | 1. http://www1bpt.bridgeport.edu/~jelee/courses/CS102_W17/notes/Ch3.%20Queues.pdf | 1. |
| 2. | 2. | 2. | 2. | 2. |
| 3. | 3. | 3. | 3. | 3. |

Recall that TAMU Student Rules define academic misconduct to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. *Disciplinary actions range from grade penalty to expulsion.*

"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment."

| | |
|---|---|
| **Signature:** | Alexander Akomer |
| **Date:** | 02/09/2021 |

Alexander Akomer

# PA 1 Report

## 1. *Introduction:*

This assignment is an early exercise in calculating efficiency of code. We were tasked with creating two different programs that used a queue implementation – one following the design of a circular array and the other that of a linked list. To compare the time and memory efficiency of the two, the number of elements was increased until memory ran out and the time for operations was recorded in each iteration of the elements increasing. The results are as follows.

## 2. *Theoretical analysis:*

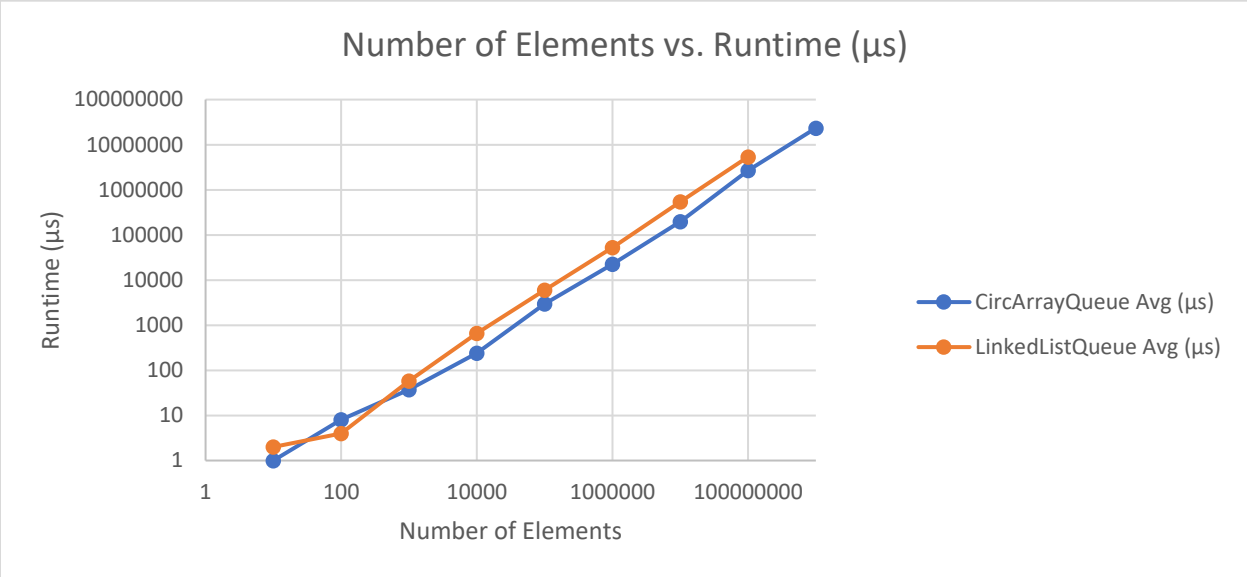
The theoretical asymptotic analysis of the circular array queue enqueue operation is O(1) (Repeated n times, the graph should create a linear function). This is true because the space to enqueue into is already allocated and, if not, then space is reallocated making the operation O(n) in the worst case, but these cases become increasingly few and far between as size increases (this case occurs every log N cases).

The theoretical asymptotic analysis of the doubly linked list queue enqueue operation is also O(1) (Repeated n times, the graph should create a linear function). This is true because insertion into a doubly linked list simply requires the rearranging of surrounding pointers, as there are no indices to shift. However, because this requires the rearranging of multiple pointers as well as the addition of new data, there is a multiplicative constant that makes the operation slower than a typical circular array enqueue.

3. *Experimental Setup:*

The machine the program was run on for my trials was a Surface Book 2 with 16 GB of RAM, an i7-8650U @ 1.90 GHz, and 109 GB / 475 GB free on the C: drive. My timing mechanism was the code attached on Canvas, which included the chrono package. This package allowed me to run a high-resolution clock and output the runtime in microseconds for each trial. To ensure accuracy of my results, I included three trials of each iteration for both the circular array and linked list implementations of the queue.

4. *Experimental Results:*




After testing the enqueue operation of both the circular array and the linked list, the runtime data led to the following conclusions:

Circular Array Enqueueing: O(1) (Repeated n times, the graph should create a linear function).

Linked List Enqueueing: O(1) (Repeated n times, the graph should create a linear function).

Although the linked list is slower as evidenced by the runtime data, both data structures' enqueue operations grew at a linear rate and are therefore bounded by the same linear function. This is comparable to the theoretical complexity I estimated previously. Because the circular array is an O(1) function repeated n times, it follows a linear growth rate. Because the resizing function is O(n) and is repeated every log n calls, the growth is O(n + (n/2^n)), which still simplifies to O(n) but explains the dip in efficiency that the circular array experiences early on. For instance, there are four calls to the resize() function between 10 and 100 elements, which makes the efficiency drop below that of the linked list implementation early in the iterations. These results agree almost exactly with the theoretical.