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

Name:	Priyanshu Barnwal
Email:	Priyanshu@tamu.edu
Assignment:	Programming Assignment 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
			https://www.geeksforgeeks.org/doubly-circular-linked-list-set-1-introduction-and-insertion/	My old code for linked lists from 121
			https://www.cplusplus.com/reference/chrono/ (and many subpages)	
			https://stackoverflow.com/questions/6547602/expression-must-have-class-type	
			https://stackoverflow.com/questions/21807658/check-if-the-input-is-a-number-or-string-in-c	

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:	Priyanshu Barnwal
Date:	2/9/2021

Introduction:

Runtimes are very important in the world of programming. No one likes a slow website, a stuttering game, or a staticky signal. In this programming assignment, I test two different Data Structures for their runtimes to see which is more efficient. I implemented both a circular array and a doubly linked circular list with different magnitudes of inputs to demonstrate which is faster.

Theoretical Analysis:

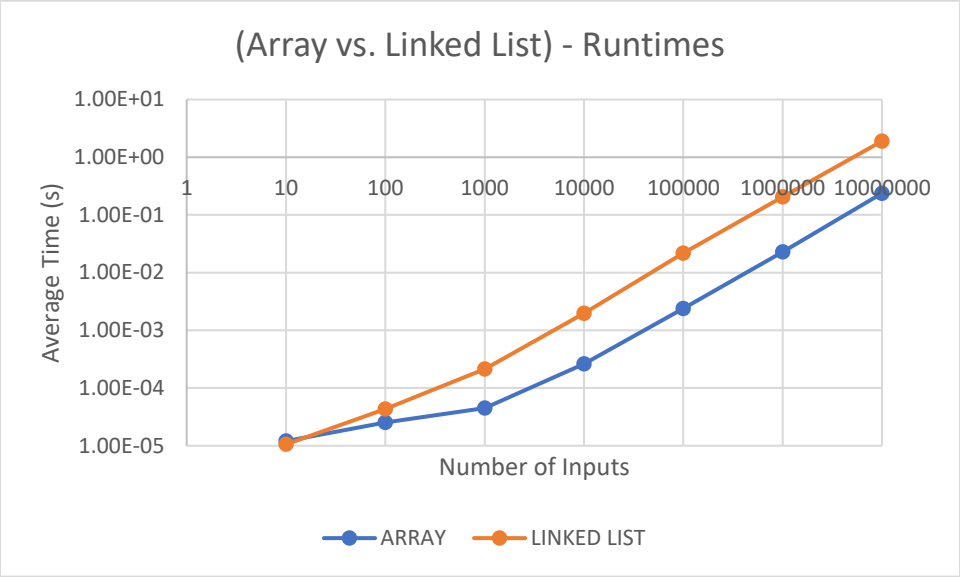
The programs both should run on O(n) to put it simply, because they perform the same function in a very similar way. The array should run at this speed because we are adding values at the end and array insertion is an algorithm that runs on O(n). If we were adding values at the beginning, this would be a very different problem, of course because all the memory addresses would have to be moved over. The linked list should also run at this speed because we are adding values to the end of it and pulling from the end as well. Linked lists do not care where you add a value as long as you aren't adding a value based on another value's position. Thus, this algorithm, too, should run on O(n).

Experimental Setup:

I am currently running this on a computer with 32GB RAM, however, Visual Studio only allocates 2 GB of processing memory for any given project. I am using Visual Studio 2019, which runs C++17 with experimental features of C++20. For my timing mechanism I used the high-resolution clock inbuilt function to accurately time how fast my program ran. I repeated each experiment five times, increasing the number of inputs by a factor of 10 each time until my program ran out of memory.

Experimental Results:

The graph shown below was made in excel using the runtimes of my code:



This is graphed on a log-log scale, increasing by a factor of ten on both the x and y axes. As clearly indicated by the data, it seems that Linked List starts out faster than the Array but as it increases in size, it becomes clear that the array can handle larger numbers of inputs far better than the linked list can. It seems that all the pointer switches to insert a node into a linked list really bogged it down, heavily increasing the runtime for larger operations. It seems that my implementation of a circular doubly linked list was not as efficient as possible, and I am sure that It could be more efficient. I believe that it is possible to get the algorithm's complexity down to O(n), even though at the moment it seems to be at O(2n). I cannot seem to remove any links without the algorithm collapsing, however, so it remains to be seen whether or not it is possible to make a linked list with fewer links.