**CSCE 221 Cover Page**

**Programming Assignment #1**

**Due Date: Sunday October 6, 2019**

**Submit this cover page along with your report**

**First Name:** Andrew **Last Name**: Han **UIN: 227009495**

**Any assignment turned in without a fully completed cover page will NOT BE GRADED.**

Please list all below all sources (people, books, webpages, etc) consulted regarding this assignment:

CSCE 221 Students: none

Other People:

1. Akash Tyagi 2. Yahui Sun

Printed Material :

1. Lecture slides for sets 4-6

Web Material (URL):

1. <https://www.geeksforgeeks.org/memory-leak-in-c-and-how-to-avoid-it/>

2. <http://www.algolist.net/Data_structures/Singly-linked_list/Removal>

3. <https://www.geeksforgeeks.org/vector-in-cpp-stl/>

4. <http://www.fredosaurus.com/notes-cpp/misc/random.html>

Other:

1. My previous CSCE 121 projects including implementation for linked lists.

Recall that University Regulations, Section 42, define scholastic dishonesty 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 penalties to expulsion. Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

**I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.**

**Today’s Date: 10/4/19**

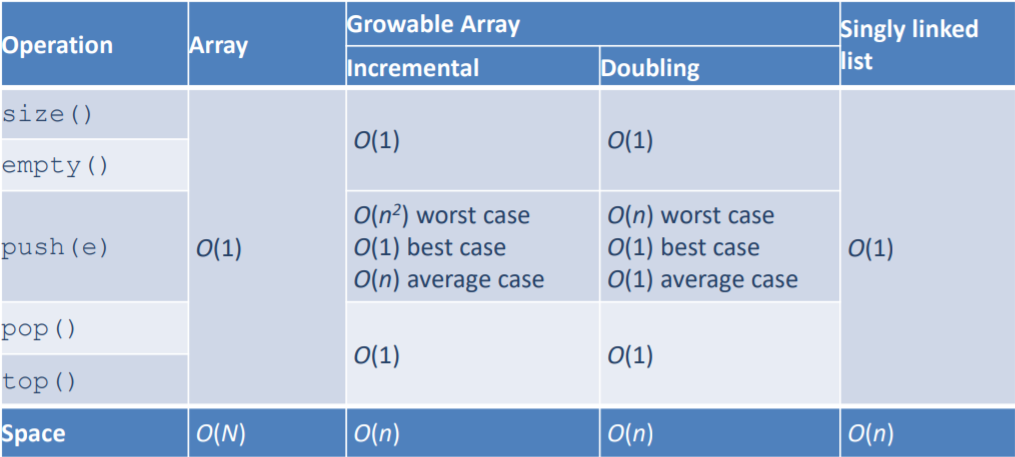
**Printed Name (in lieu of a signature): Andrew Han**

1. **Introduction**

The objective of this assignment is to compare the runtimes of a certain number of push operations for three different stack implementations. A stack is a data structure that has a LIFO (last in, first out) property regarding adding and removing elements from it. This project will be testing stack implementations that are different in the way they adjust their size if the stack becomes full. Two implementations will be array-based starting at a given array size, one being an array that will increase by a fixed interval when it becomes full and the other will be an array that doubles in size when it becomes full. The third implementation will be a linked list that will create a new tail pointer when a new element is added. Each implementation will be given a certain number of random elements that it will push, and each implementation will be timed to see how long it takes to complete all the push operations. Once timed, the three stack implementations will be compared to each other and see if the results agree with theoretical runtime based on the algorithm complexities of each. The runtime of a vector data type will additionally be tested along with the three stack implementations listed. A vector acts like an array but is dynamic in size and can re-size whenever it needs to add more elements.

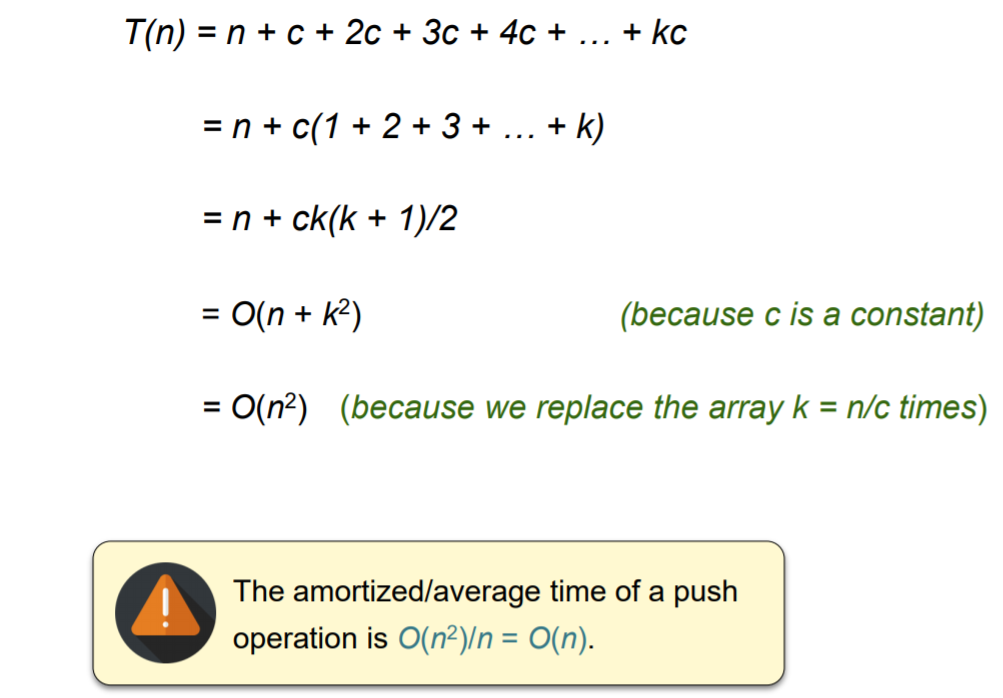
1. **Theoretical Analysis**

A push operation adds an element on the top of a stack and makes it the first element to be removed. For the two array implementations, a push operation will add an element to the end of the array. If the array is full, a new array of increased will be created and all the elements will have to be copied over before pushing the next element. For linked lists, a push operation will create a new node and the tail will be directed to point at this new node that is pushed to the top of the stack. The following table provided in professor Tyagi’s SET7 slides compares the complexities of different stack functions for these three implementations.



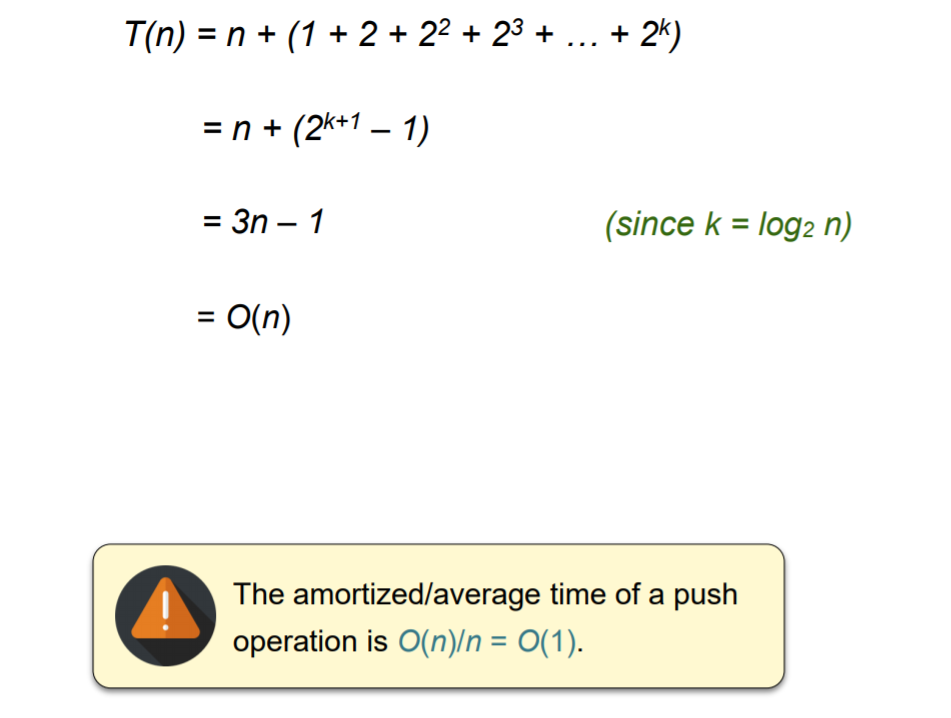
*Courtesy of Akash Tyagi’s SET7 lecture slides*

Here we can see that the array implementations have different worst, best, and average cases while the linked list implementation will always have a complexity of O(1). The worst case of the incremental array will be O(n^2) because array is resized (number of pushes)/(interval), the nested for loop in this case causes the complexity to be O(n^2).The average runtime will be O(n^2/n) = O(n) in this case. This can be visualized in Tyagi’s slides below.



*Courtesy of Akash Tyagi’s SET6 lecture slides.*

For the doubling array, the array will be replaced by a factor of log(n) times. The time needed to complete the push operation will be n + (2^k+1 – 1) because the operation will increase the array by a factor of 2^i as i goes from 0 to k. Because k = log(n), T(n) will simplify to 3n-1, which means T(n) = O(n) for the worst case scenario. The average case will be O(n)/n = O(1). This is visualized below in Tyagi’s slides.

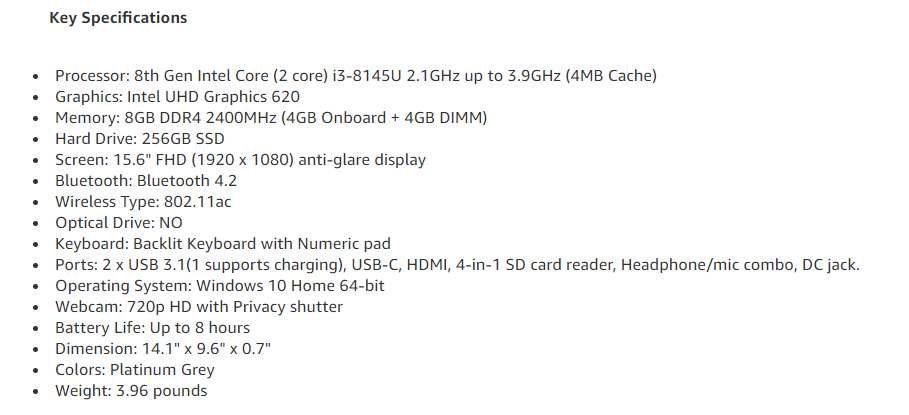


*Courtesy of Akash Tyagi’s SET6 lecture slides*

A linked list will always have a complexity of O(1) because no matter what the size is, the list will not have to be recreated and you will not have to traverse through the entire array again to copy its elements. For a push, all that needs to be done is create a new node and assign the tail to this new node to show that this is at the top of the stack now. For vectors, the complexity should be the same as the linked list, O(1). This is because whenever an element is pushed, the vector can just add the element to the end of the array without having to create a new vector and copy the elements.

1. **Experimental Setup**

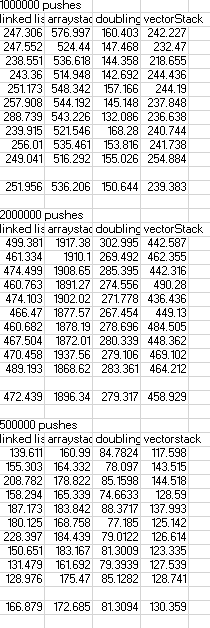
The machine I used to run my code and time the push operations for each implementation is a Lenovo IdeaPad s340 with the following specs



For test inputs, I decided to run the experiment with 3 separate number of push operations: 500000, 1000000, 2000000. I chose these 3 numbers so that I could observe the runtime for half and double the amount of the required 1000000 pushes given in the instructions. For the arrayStack implementation, I kept the starting size of the array at a constant 25000 for each trial and kept the interval at 1000 for each trial. Changing these through the different amount of push operations would affect the results. For each number of push operations, I timed the 4 different implementations 10 different times and recorded them to get an average of the runtimes to make sure the data was consistent and there were no anomalies. After gathering these averages, I plotted four different graphs to show how each implementation ran as the number of push operations increased.

1. **Experiment Results**

Below are the results of the 4 different implementations tested. The time is in ms and the average for each is listed below the 10 trials.



Below are the four graphs of how each stack implementation ran as the push operations increased.

After conducting the experiment with the 4 implementations, the data shows that the doublingArrayStack structure performed the best for each of the different number of pushes. Originally, I thought that the linked list and vector implementations of the stack would be superior in terms of runtime because they would not need to resize and copy elements. However, it looks like the doublingArrayStack resizes so few times that the runtime is faster than the linked list, which has to create new nodes and pointers each push, and the vector, which can dynamically increase its own array size. The complexities of the doubleArrayStack, linked list, and vector are theoretically the same, so it is plausible that the doubleArrayStack outperformed the other two in this experiment. It was expected that the arrayStack would perform the worst as it has the largest complexity and it has to recreate itself more often than any other stack implementation has to. One question I have is if the doubleArrayStack will always outperform the linked list and vector implementations of the stack. If the latter implementations eventually outperform the doubleArrayStack, then those could be more suitable implementations for this experiment. However, the trials I ran show that the doubleArrayStack will complete its push operations in the shortest amount of time.