CSCE 2221 Cover Page
Programing Assignment #1

First Name: Yuri Last Name: Castro UIN: 626003324

Any assignment turned in without a fully completed coverpage will receive ZERO POINTS. Please list all below all sources (people, books, webpages, etc) consulted regarding this assignment:

CSCE 221 Students:
- Questions and answers posted on Piazza

Other People:
- Peer teachers at peer teacher central

Printed Material
--

Web Material:
- Stack Overflow
- Cplusplus.com
- YouTube

Other
--

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: February 14, 2018

Printed Name (in lieu of a signature): Yuri A. Castro

Yuri Castro
CSCE 221-200
Programing Assignment 1
February 14, 2018

Introduction

In this programming assignment, three different implementations of a stack are built in order to be compared in runtime. The first implementation is ArrayStack in which the array size rows by a constant amount. The second implementation is DoublingArrayStack is an array that grows by doubling its size each time. The third implementation is a stacked LinkedList which uses nodes in a list. Running these implementations, we time them to see how each type of stack may perform differently.

Theoretical Analysis

The overall purpose of a push() function in a stack is to insert an element to the top of the stack. In order for the push() function to work, there needs to be enough space for a new element to be added to the stack. For all the standard stack operations, the complexity for push() in the worse-case scenario can be O(1). However, as we see later, a stack can have different complexities depending on the type of implementation used for it. They are said to be O(1) since they should only work with the top of the stack. A stack created by a array increasing at a constant rate has the advantage of having a constant growth rate without running into risk of allocation too much memory for the stack. The disadvantage of it is that it takes a lot of time copying over each element in the stack to a larger one each time it reaches maximum capacity. The stack with a doubling array also has this dis advantage but it has to resize itself less often. This is because it is changing its array size by twice itself each time. In having to resize itself less often this makes the runtime much shorter than a constant increasing array. A linked list stack array is different. In this experiment, I implemented a double linked list to make it easier to use the pop() and top() functions. In using a linked stack, it does not to be resized since there is no maximum number of elements in a linked list. However, this does take a lot of memory in having several nodes, pointers everywhere and such.

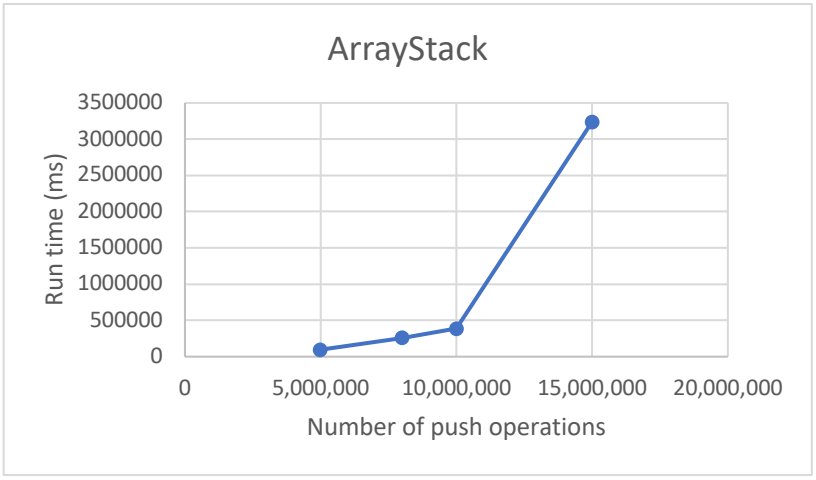
Experimental Setup

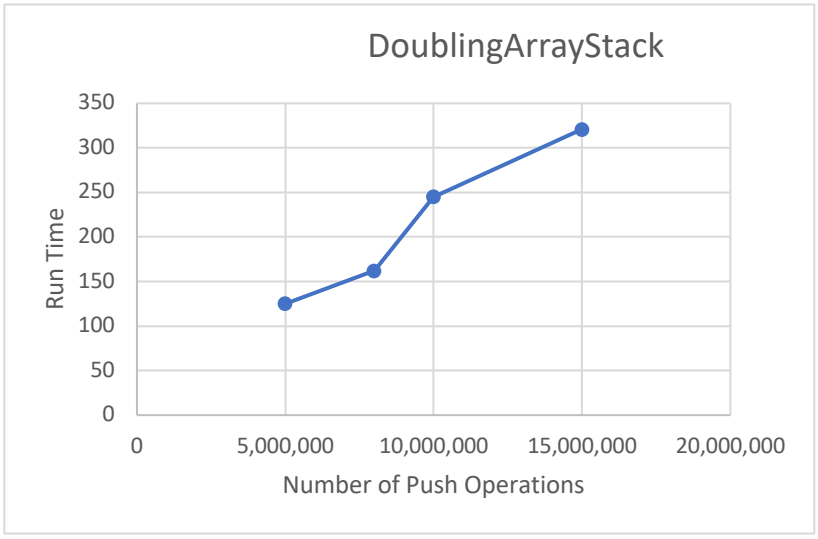
Machine Specification: MacBook Pro with 2.9 GHz Intel Core i5

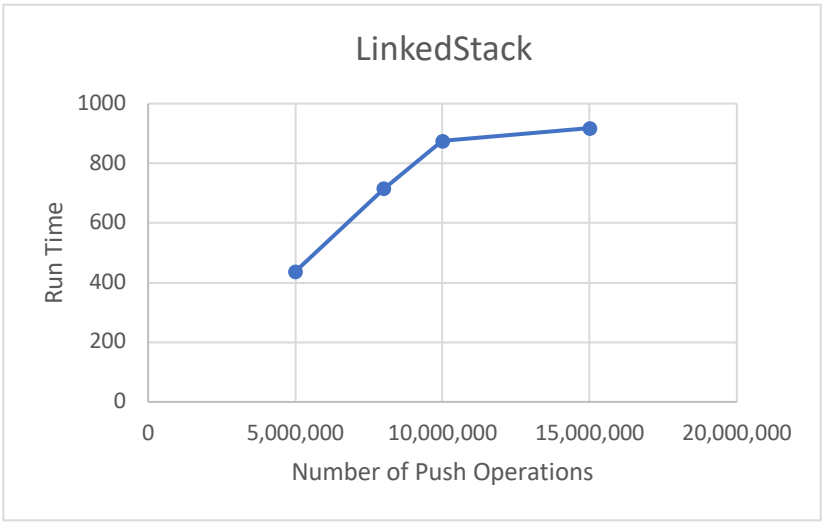
In order to generate test inputs, I implemented a random number generator which will give a random double number between 1 and 10 for each push. In testing the length of each type of stack, three different sizes of inputs were given to each stack(1,000,000 , 5,000,000 , and 10,000,000). Five trials of each size was ran through each type of stack. I decided in the end that a constant increment of 1000 in the constant array stack. The graph down below is a side-by-side comp

Experimental Results

Graphs:


ArrayStack
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
Run time (ms)
0
5,000,000
10,000,000
15,000,000
20,000,000
Number of push operations


DoublingArrayStack
350
300
250
200
150
100
50
0
Run Time
0
5,000,000
10,000,000
15,000,000
20,000,000
Number of Push Operations


LinkedStack
1000
800
600
400
200
0
Run Time
0
5,000,000
10,000,000
15,000,000
20,000,000
Number of Push Operations

Observing the experimental results, one can conclude that the Doubling Linked List implementation works the best. With the doubling linked list implementation, the push() operator depends on the input but makes the size of the array much larger. The stack with a constant growing array has the slowest run time. As seen in the graphs, it takes several times longer than the other stack implementations. It made sense in the end why the linked list took a while longer than the doubling array. With the push() function, it just goes through O(1) in an array. However, with the linked list, it has to create pointers and update a lot of variables for each push. If I had more time, changing the constant amount of how much the Array stack would be interesting to compare to the doubling array stack. In the end, the best way to do it is with doubling the array size, allowing for a fast growth rate and dealing with easier functions and implementations.