**CSCE 221 505**

**Programming Assignment #1**

**Report**

Vidhur Potluri

UIN: 626007235

**Introduction**

The aim of the assignment is to determine the run time complexity and efficiency of the different implementations of the stack ADT. The assignment includes pushing random numbers on to three different implementations of the stack - a growable array stack that increases its size by a constant amount (ArrayStack), a growable array stack that doubles its size when it’s at capacity (DoublingArrayStack), and a linked list (LinkedList) – and comparing the time it takes for each implementation.

**Theoretical analysis**

A push operation for a stack is the only way to increase the size of the stack and add elements to the top of it. Each individual push operation of all three implementations has a worst-case run time complexity of since each time only 1 number is pushed on to the stack. Since pushing *n* numbers on to the stack involves executing the push operation *n* number of times, and the linked list doesn’t employ any loops in its push operation, its average, worst-case and best case run time complexity is

Run time complexity of *n* push operations for the array-based stacks are evaluated below: (i = incrementor, n = number of push operations, sz = size of the array)

1. ArrayStack:

(I’m assuming the array starts with an initial capacity of 0.)

Therefore, the worst-case runtime complexity of the ArrayStack is if *i* is significantly smaller than *n*. The best case is O(1), and the average case is

1. DoublingArrayStack:

(I’m assuming the array has an initial capacity of 1 and that n is a power of 2.)

Therefore, the worst-case runtime complexity of the DoublingArrayStack is . Its best case runtime complexity is O(1). The average case is

StackArray time function grows the quickest and therefore, will be the slowest to perform a large number of push operations. LinkedList and DoublingArrayStack have an average runtime complexity of , and will therefore, grow at a rate similar to each other and have approximately the same run time. However, LinkedList has a worst case run time complexity of O(1) and DoublingArrayStack does not. Therefore, the linked list might perform better than the doubling array in certain cases.

**Experimental setup**

1. *Machine specifications:*

Computer model: HP Spectre x360

OS: Windows 10 Home

Processor: Intel® Core™ i5-8250U CPU @ 1.60 GHz 1.80 GHz

Installed Memory: 8.00 GB (7.84 GB Usable)

1. Procedure and testing: The input numbers for the different stacks were generated randomly using the rand() function from the ‘stdlib.h’ library. I pushed the same number of random numbers on to all of the different implementations to facilitate run time complexity evaluation and comparison. Each of the arrays were tested twice. The following are the classes I created, and the methods I used to test their ability to handle inputs of different sizes.
   * 1. AbstractStack: The AbstractStack class is the abstract class whose member functions are derived by both the ArrayStack and DoublingArrayStack classes. The class is constructed with an initial capacity for its array, and has an iterator that keeps track of the top element of the array
     2. ArrayStack: The class that increases its size by a constant number every time it reaches its capacity. I tested it by pushing 1,000,000 randomly generated numbers twice on to it with the same initial capacity of 1000, but different incremental amounts – 100 and 1000.
     3. DoublingArrayStack: The class that doubles its size every time it reaches its capacity. I tested it by pushing 1,000,000 randomly generated numbers on to it with an initial capacity of 1,000.
     4. LinkedList: The linked list - based stack contains elements that point to the next element in the stack, and its member functions are the same as the AbstractStack member function. I tested it by pushing 1,000,000 randomly generated numbers on to the stack. This class does not require an initial capacity since it isn’t array based.

**Experimental Results**

I couldn’t use the same plot for the different implementations since the ArrayStack took a significantly longer time. The following are the runtimes of the push operations for the different implementations of the stack. I used the same machine specifications for all tests.

**Fig 1 - ArrayStack with size increments of 100 runtime**

**Fig 2 - ArrayStack with size increments of 1000 runtime**

**Fig 3 – DoublingArrayStack runtime**

**Fig 3 – LinkedList runtime**

|  |  |  |  |
| --- | --- | --- | --- |
| Run time (s) | ArrayStack (i = 100) | DoublingArrayStack | LinkedList |
| Trial 1 | 23.49 | 1.33 | 1.25 |
| Trial 2 | 23.32 | 1.13 | 1.14 |

Discussion: Although the DoublingArrayStack and LinkedList have similar runtimes when one million numbers were pushed on to them, the LinkedList performs slightly better and took lesser time than the DoublingArrayStack when a higher number of push operations were performed. The ArrayStack is very slow compared to the other two. I also observed a major decrease in runtime when the incremental value was increased from 100 to 1000. The run time complexities prove that the theoretical analysis and comparison of the different implementations evaluated earlier were true. I believe the slight discrepancies observed between the trials and across the different implementations might be because of the changing frequency of the processor of the computer.

Although the linked list implementation and doubling array-based implementation of the stack proved to be more efficient, the needs of a user (client) and their resources (processor, power, money, etc) budget might lead the user to prefer a different implementation of the stack.