Siman Shrestha
Programming Assignment 1
Dr. Nakamura
UIN: 725002204

Introduction:

The purpose of this assignment was to implement 3 different forms of a Stack to learn how different algorithms can effect efficiency. Two of these implementations were array-based. The initial size of the array for each of these implementations are predefined. The only difference between the constant array and the doubling array is how they adjust the size of their arrays when they have been filled. In the constant array, the size will increase by a constant amount every time, and for the doubling array, the size will double each time. The last stack implementation is a linked list.

Theoretical Analysis:

For each implementation, the push operation takes a constant time, and the size of the stack does not change this property. The worst-case time complexity for a push operation is O(1) if the size of the array doesn't need to change.

The doubling implementation is very effective for stacks that will end up obtaining a large size. There are cases where the array's initial size can have a big impact on total runtime. If it is set to have a size of 1, the array will need to double the size of its array many times at the beginning, which is very time consuming, and ends up increasing its average run time. The average complexity of a push operation for this stack is O(1), with the worst case being O(n).

When the array is full for the stack that increases its size by a constant number it is different compared to the doubling array. The constant stack is effective for arrays that have a much smaller smaller size, but if a lot of data is being poured into the stack and the incremental value is not very large, it would have to copy itself into the bigger array many times. The average complexity and worst case of a push operation in this specific experiment for the constant stack is O(n).

For the singly linked list implementation, a new node is created every time an item is pushed to the stack; the pointer that was pointing to the prior top of the stack points to the newly acquired node, which ends up being time consuming. Each push operation will take a constant time for each set of data entered, and unlike the other two stacks it does not need to be copied into a larger array when it is full. The average and worse case for the singly linked list is O(1).

Experimental Setup:

Implementations of these Stacks were made in Xcode on a mid 2014 MacBook Pro, 2.8 GHz Intel Core i5 with 8 GB RAM. Ten thousand and then ten million test inputs were generated by using a for loop and the rand() function which generates a random number between 0 and RAND_MAX (2147483647). Most of the numbers generated were over 8 digits. I first tested importing 10,000 numbers into the stack to get quick results to compare to a much larger input. For this experiment I set the initial size for both versions of the arrays to 100. For the constant size array, I increased the size by a constant of 50. Time was recorded every 10 iterations in this trial.

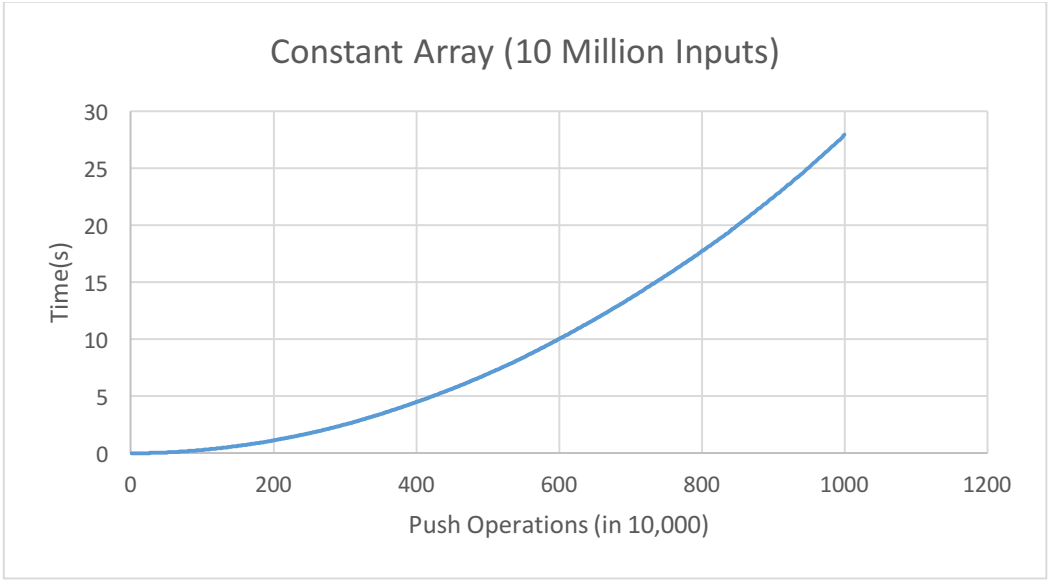
I then used 10 million inputs to find out more about the mathematical complexities of the algorithms, which yielded the desirable results I wanted. Parameters of the constant array stack consisted of an initial size of 100 and an incremental value of 10,000, meaning that the stack increased in capacity by 10,000 spaces each time it was filled. The doubling array was also initialized with a size of 100. Time was recorded every 10,000 iterations in this trial.

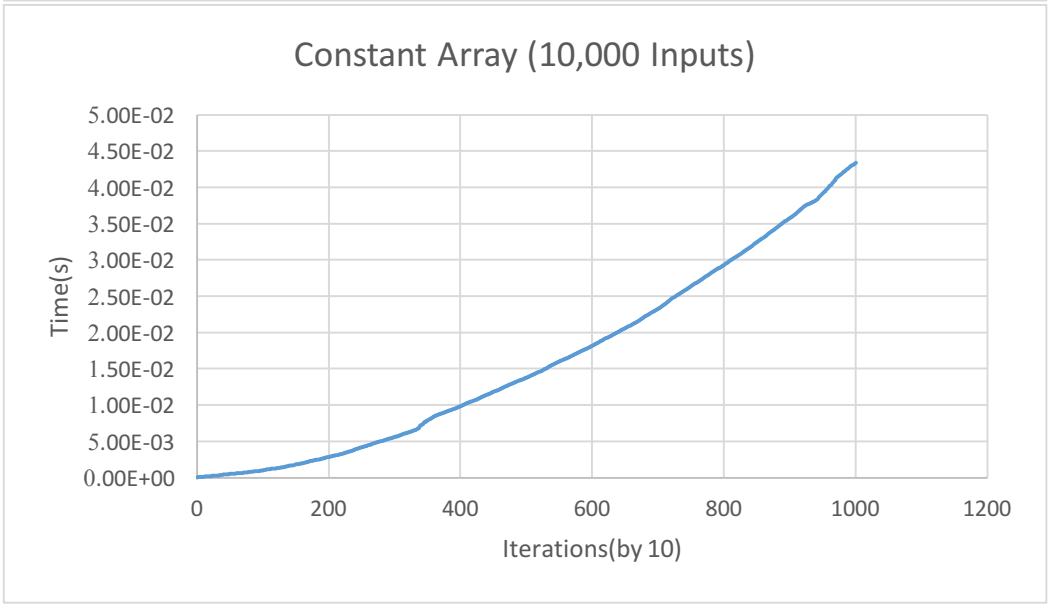
I repeated all experiments 10 times to make sure the results outputted similar conclusions and graphed the results for both input sizes (10,000 and 10 million).

Experimental Results:

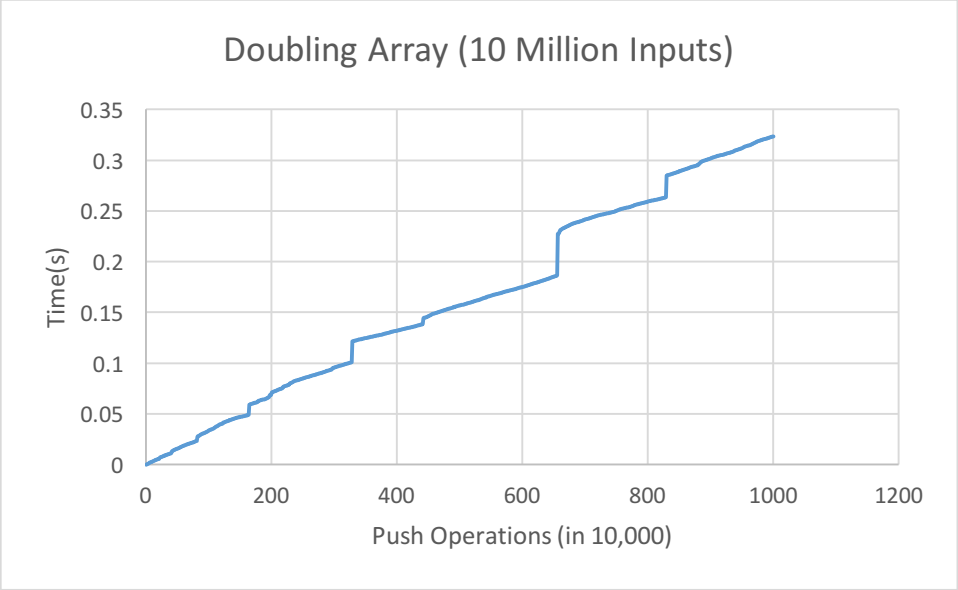
As seen in the graphs below, it is clear that the doubling array implementation of a stack outperformed its counterparts. The doubling array was able to complete 10 million push operations in about 0.325 seconds. It took the linked list about 0.8 seconds to complete the same task, putting in second place, and it took the constant array about 28 seconds. Although the constant incremental stack shows the slowest performance, it can be very fast when input size is small. The singly linked stack is very consistent for all sizes of data. For the most part my theoretical analysis of the implementations is correct; however, I didn't expect the doubling array to be so much quicker than the incremental implementation. This data shows strong correlation in number of times needed to increase the size of the array and time consumption.

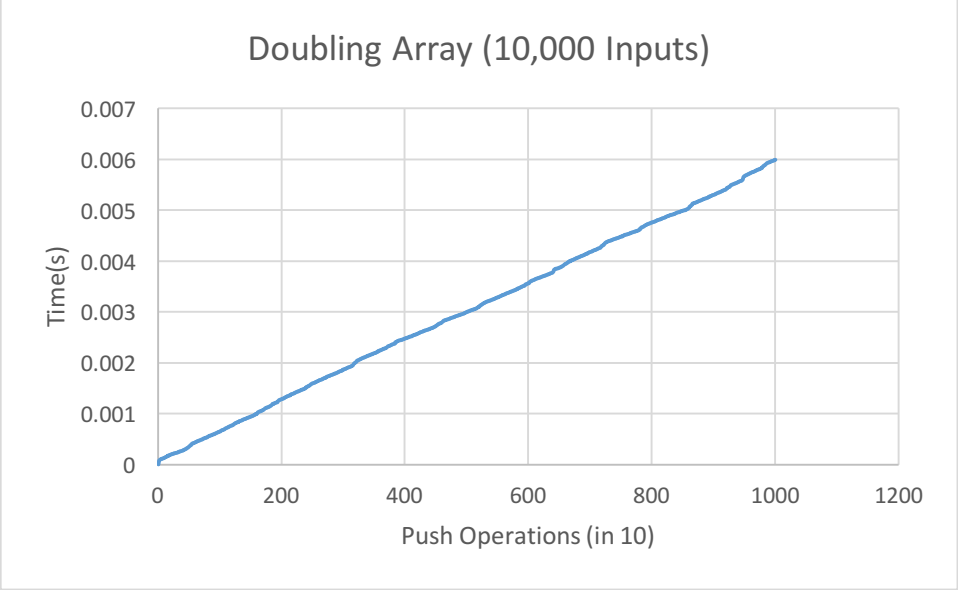
Siman Shrestha
Programming Assignment 1
Dr. Nakamura
UIN: 725002204





Siman Shrestha
Programming Assignment 1
Dr. Nakamura
UIN: 725002204





Siman Shrestha
Programming Assignment 1
Dr. Nakamura
UIN: 725002204