PA1 Report

Introduction:

In this assignment, the task was to implement the data structure of a stack using 3 different strategies. General functions of a stack (such as pop, push, top, etc) were implemented for the three strategies but the biggest difference was how push was implemented in each of the methods. The first two strategies were were based on a one dimensional dynamic array, but in the push function, the first strategy doubles the size of the stack when there is no space to push an object into the stack and the second strategy linearly increases the size of the stack by 10 when there is no space to push an object. The third strategy is quite different as we use a linked-list to create the stack data structure and the size of the stack increases as objects are pushed into the stack.

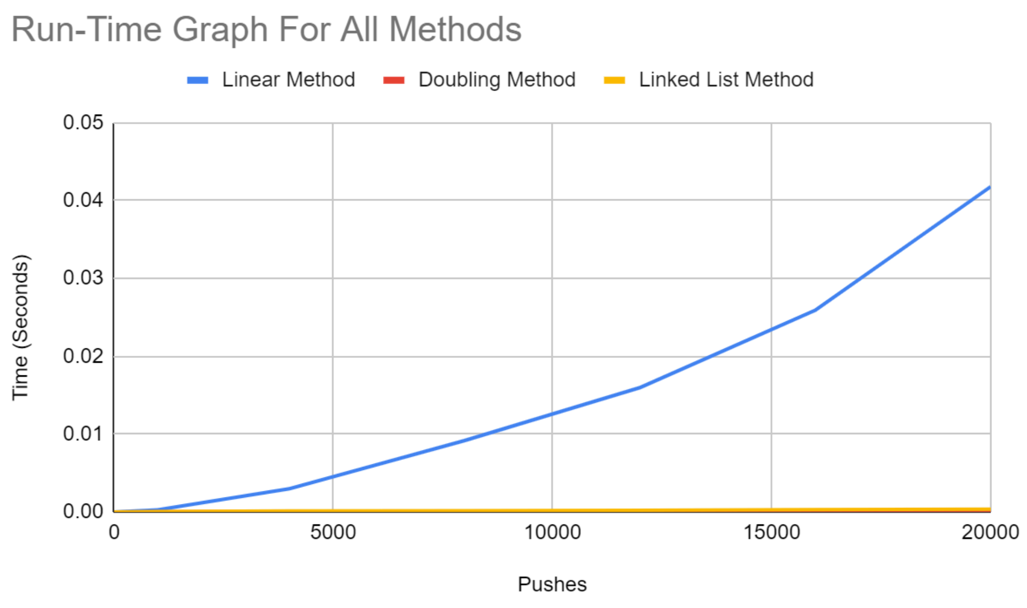
Theoretical Analysis:

All of the three strategies above have different time complexities for their push functions. The linear method has a worst case runtime complexity of O(n^2), a best case runtime complexity of O(1), and an average runtime complexity of O(n). The doubling method has a worst case complexity runtime complexity of O(n^2), a best case runtime complexity of O(1), and an average runtime complexity of O(1). Finally, the linked-list has a general run time complexity of O(1). Therefore, we can say that the consistently fastest push function is that of the linked-list method and the reason for this being the scalability and flexibility of a linked-list unlike a dynamic array, but there are also a few disadvantages to this method. It requires excess storage as each node in the stack needs to be stored independently on the heap and has a much more complicated implementation. The dynamic array strategies require less storage on the heap and accessing is much easier in terms of implementation and has a lower accessing runtime complexity, but resizing the dynamic array, as required by the push function, has a greater complexity when compared to linked-list especially when the size of the array is incremented linearly. Hence, I believe that the implementation of doubling the array gives the best result in terms of easier implementation and shorter run-time complexity, even though it has its own disadvantages.

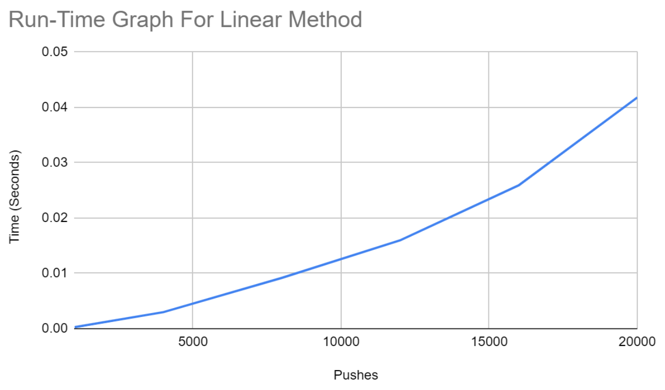
Experimental Results:

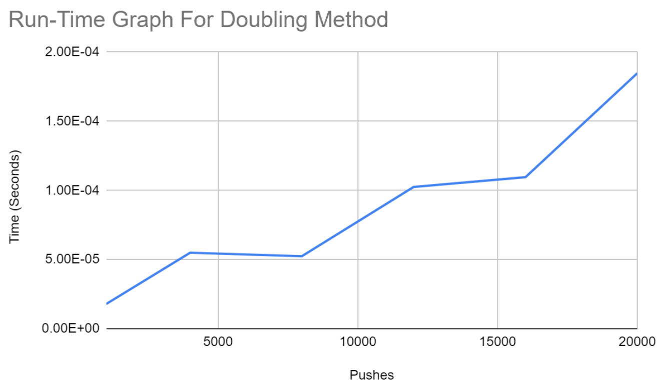
This experiment was conducted to find out the push function run-times of the three implementations mentioned above. About 20000 elements were pushed into a stack from each of the implementations and the time between intervals of 4000 was recorded and shown on the console. The graph of the same are as follows:

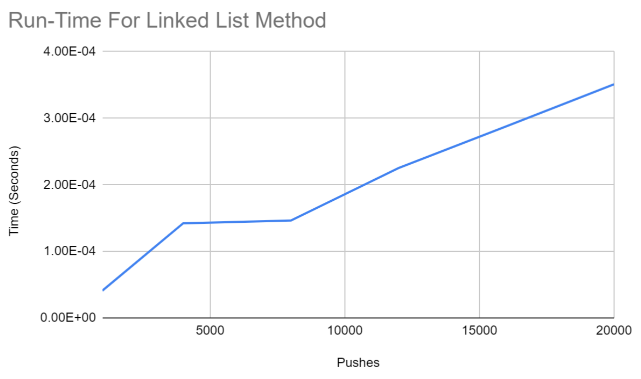
Graphs:



Graph Of The Run-Time For All Methods







Individual Graphs Of The Run-Times Of Each Method

Result Table:

Pushes	Linear	Double	Linked List
0	0	0	0
1000	0.000277432	1.78E-05	4.10E-05
4000	0.00297726	5.49E-05	0.000142061
8000	0.00917666	5.24E-05	0.000146322
12000	0.0159779	0.000102482	0.000225233
16000	0.0259267	0.000109528	0.000287794
20000	0.0417802	0.000184649	0.000351021

Analysis:
After the experiment was conducted, it was found that the doubling method has the lowest run-time when compared to the other 2 methods. This experiment also shows that the number of pushes does not have an effect on the run-time complexity when dealing with thousands of pushes. As expected from the theoretical analysis, increasing the stack size linearly in the push function did increase the run-time significantly when compared to the other 2 implementations and this can be seen very clearly on the first graph above. This might be the case because the stack is having to resize itself every 10 pushes which notably increases the run-time. Now, comparing the other 2 methods, the doubling method always had a lower run-time for each interval when compared to the linked-list, although theoretically I believed that the linked list would have a lower run-time because it has a general run-time complexity of O(1). This might happen because of the time expense of allocating new memory each time a new node is created. But this discovery further strengthens my point from the theoretical analysis above that the best way to implement a stack data structure is to double the size of the stack(dynamic array) in the push function when extra space is required.