Cody Wu
CSCE 221-594

Programming Assignment 1 Report

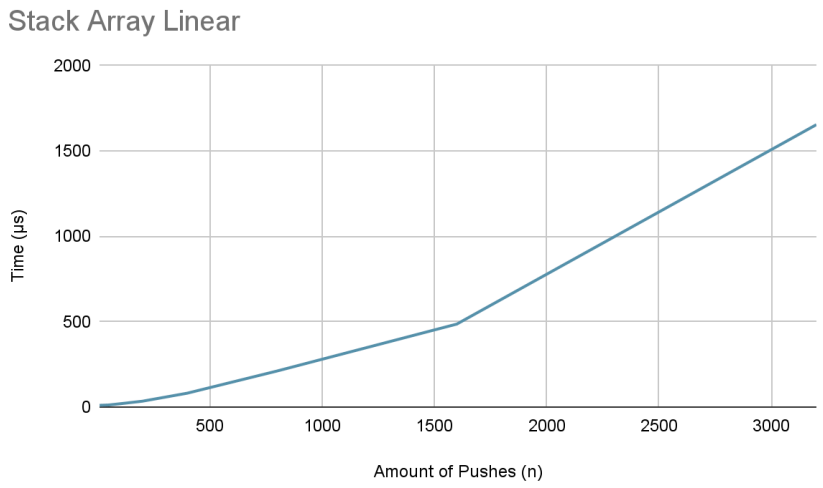
Introduction:

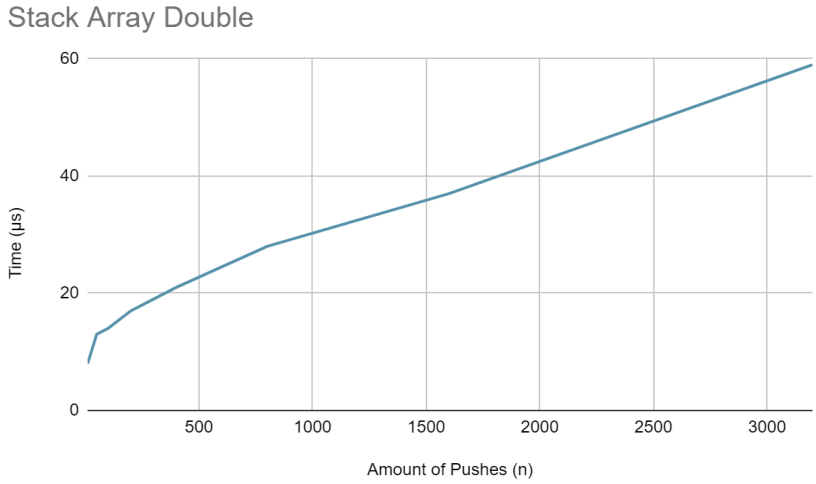
For this programming assignment, a stack will be created in three different ways. The objective is to implement the stack with all of its functions in different mediums to display understanding of what a stack is and how it operates. It is also a good way to review past C++ concepts. The first two ways are array-based. These arrays differ in the way that they expand when full. One is increased linearly while the other is doubled. The third implementation is through a linked list.

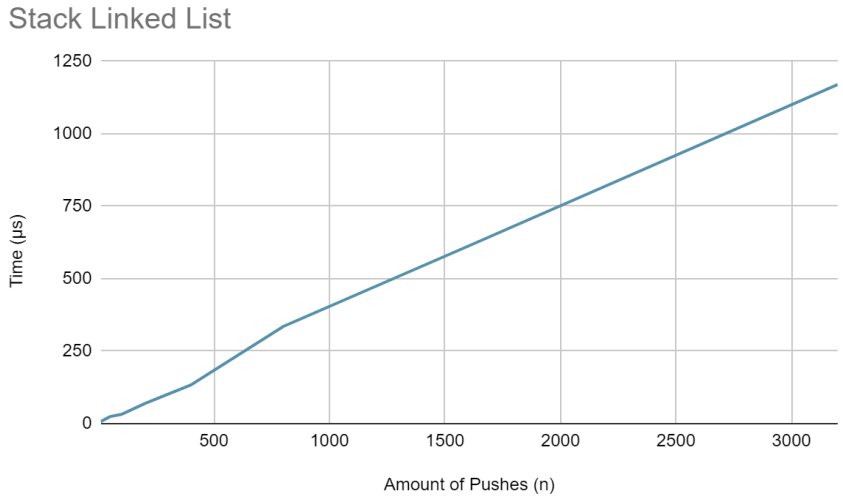
Theoretical Analysis:

A stack works by inserting and removing elements from a medium (array or linked list). The push operation inserts the element to the top of the stack. Each style has its own pros and cons. The linear and double growth array share the same advantages. They are less complicated and easier to implement. They are more accessible as elements within the array can easily be changed. However, they also have disadvantages. Everytime the stack is full, the array must be resized. This can be problematic if elements are continuously being pushed in. This disadvantage is worse for linear arrays as they can only increase by a certain amount. The doubling array can increase size a lot quicker than the linear array. The linked list covers the array's disadvantages. It does not need to be resized and can theoretically support any number of pushes without issue. The biggest linked list disadvantage is its complexity. It is a lot harder to code the linked list compared to the array. It also isn't as accessible as there is no easy way to single out a position in the stack/list. However, these problems only apply to less proficient programmers. All of the implementations require the same amount of space of O(n). The linked list is the better option as it is fundamentally better and doesn't require resizing. The incremental array's complexity has an average case of O(n) and a worst case of $O(n^2)$. The doubling array's complexity has an average case of O(1) and a worst case of O(n). The linked list's complexity is always O(1). The arrays all have a case of at least O(n). This is due to their requirement of resizing that makes them slightly less optimal compared to the linked list which is O(1) no matter what. By comparing the complexity of all the implementations, they should be ranked from most to least efficient as: linked list, doubling array, and incremental array.

Experimental Results:







Within the three plots, it can be seen that the doubling array performs the best by a significant margin. I believe that the performance does not depend on input but is most likely due to memory. The linked list is a more memory based structure. It depends on pointers that the arrays do not have to deal with. I believe that the process of going through pointers causes the linked list to be slower. The doubling array is also clearly faster than the linear array as well. This is due to the fact that the doubling array does not have to resize as often as the linear array. The theoretical analysis above does not completely agree with my experimental results. The linear array was correctly predicted to perform the worst. However, the doubling array unexpectedly performed the best.