Daniel Nguyen
831003833
PA1

Introduction

This assignment aims to provide experience implementing essential functions to mimic different ways of using a stack via dynamically allocated arrays, and linked lists to understand their real-world applications and time-complexities by observing how they are run and programmed differently.

Theoretical Analysis

The push operation adds an element, in this case to the top of a stack. Dynamic arrays that increse by a constant amount, allow for predictable memory increases in situations where memory is limited, however can lead to frequent resizing/copying when dealing with large data that increases the runtime. This issue is solved with arrays that double in capacity, lowering the frequency of resizing at the risk of allocating more memory than needed and falls short in situations where the stack size is fluctuating frequently. This can be addressed with linked lists that avoids the need for resizing and allows for the ability to traverse backwards through the data in the stack, without compromising efficient memory usage, however, may be at a cost of runtime. On average these implementations follow an O(1) time complexity

Experimental Results

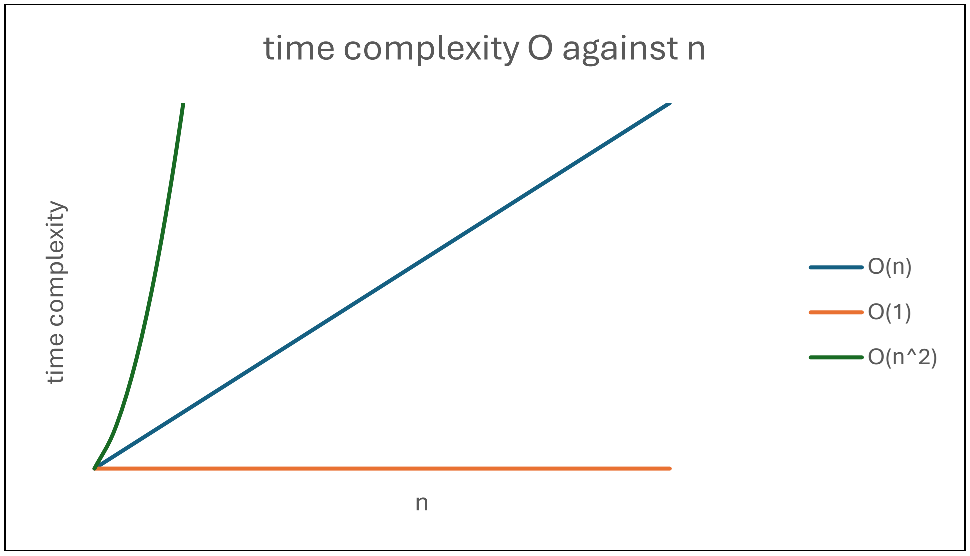

Figure 1. Worst case scenarios for push time complexities of incremental array O(n^2), doubling array O(n), and linked list O(1) implementations

The theoretical analysis aligns with the experiemental results, as linked list performs the best in terms of worst case scenario for it's push implementation's time complexity, which does not increase in extremely high data samples, whereas doubling arrays increase by O(n), and incremental arrays increase by O(n^2). This analysis extends only to the push time complexity in extremely high volume of input data, the most efficient implementation may change given smaller volumes of data, or different frequencies of capcity changes and less need for high efficient memory usage.