CSCE 221 Cover Page

Programming Assignment #1

First Name: Anthony Last Name: Shao UIN: 725004846

**Any assignment turned in without a fully completed cover page will receive ZERO POINTS.**

Please list all below all sources (people, books, webpages, etc.) consulted regarding this assignment:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CSCE 221 Students | Other People | Printed Material | Web Material (URL) | Other |
| Vishaal Makani | Jason Zuang |  | Stack Overflow | CSCE 221 PowerPoint Slides |
| Suvedh Srikanth |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Recall that University Regulations, Section 42, define scholastic dishonesty to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion. Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Today’s Date: 3/2/2018

Printed Name (in lieu of a signature): **Anthony Shao**

**Introduction**

The objective of this assignment is to implement three ways of a stack. First, one way is to increment the size of an array-based stack by a constant amount. Second, another way is to double the size. Third is to implement a (singly) linked-list-based stack.

**Theoretical Analysis**

For all three stack implementations (incremental, doubling, and list-linking), the push operation consistently had a complexity of O(1) in best, average, and worst cases.

The first and second strategies are implementing dynamic or growable arrays. Pushing an element onto the stack takes in the best case O(1) time cost and in the worst case O(n) time cost (for n insertions). Most push operations take O(1) time by allocating extra space (creating a new array that is either incremented or doubled—depending on strategy—of the old array’s capacity) and then lazily copying the elements over. An advantage to dynamic arrays is when the array is full, only O(1) time cost is required to push. Since new allocations are less frequent (compared to linked list allocations), dynamic arrays are *usually* faster than linked lists. However, a disadvantage is in the worst case that there may be O(n) time cost allocating unused elements of the dynamic array. Furthermore, in the long-run, the incremental strategy (by a constant number) is much slower than the doubling strategy; the incremental strategy allocates new space linearly whereas the doubling strategy allocates new space quadratically.

The third implementation uses a singly linked list. Allocating a new node, having its pointer to point to the old head of the list, and updating the head’s pointer to the new node all take a cost value of O(1). Deleting the first node of the list also has the cost value of O(1), which is updating the head’s pointer to point to the next node in the list and deleting the node. This frees up memory for the old head. Altogether, the advantage is that linked lists are more dynamic and its size can easily be manipulated. However, the disadvantage for this implementation is that each new node added requires new allocation every time. This can be expensive in cost value compared to other simpler operations.

**Experimental Setup**

Regarding machine specification, I used a Lenovo Yoga Pad 2 with Windows 10 running Microsoft’s Visual Studio 2017 to implement, code, and run the three types of stack implementations for this assignment.

Inputs into the stacks were random integers between 0 and 9. A for loop inside of a for loop was used to call the push operation to push the integer into the stack. Test sizes were in degrees of base 10 (1 thousand, 1 million, and 10 million). This was to test the efficiency of each algorithm over longer periods of time. The initial stack for each implementation was chosen to be 100. In the case of the incremental array-based stack, 10 thousand was used as a constant increment

Each experiment was tested at least 10 times.

**Experimental Results**

In the long-run, the rank of the best performing stack implementation is doubling the size of the array-based stack. Second is linked list implementation. And last is incrementing the size of the array-based stack by a constant number. However, at a smaller input (short-run) the ranks slightly change: the best performing stack implementation is still doubling the size of the array-based stack, but the incremental strategy comes in second (because the increment was temporarily still larger than the doubled increment) and the linked list strategy comes in third. In most of the experiments, the long-run ranks are used as presented in the results. The theoretical analysis strongly agrees with the experimental results.