Time Efficiency Analysis

Time efficiency to Push n elements:

Here we have a **Stack (i.e Last in First out (LIFO))** implemented by employing a singly-headed singly-linked list. Another specification is that the **Stack must have its top at the back of the linked list** as opposed to the Front (Head). Now as we know for a stack, a new element is always inserted (Push) at the top of the stack. (Figure 1)

Moreover, here as the top of the stack is located at the back of the singly-headed singly-linked list, thus in order **to add an element to the top we must traverse** through the linked list using the declared single-head (Head) at the front **i.e O(n)**. Furthermore, **to add an element a new link is created** by pointing the current last node to the newly created node and this can be done in **O(1) time complexity.** Thus the **total time complexity for inserting an element** at the top of stack (i.e the end of linked list here) will be **O(n)**. (Figure 2)

Finally, when this is **conducted for n elements i.e while inserting (pushing) n elements** in the stack, we would repeat the insertion procedure with time complexity O(n) for n times and thus the resultant **overall time complexity will be O(n^2).**

[Best case would be when we insert the first element as the list is yet empty and thus no traversal and hence it would be inserted with O(1) time complexity. Moving ahead for inserting second element we need to traverse 1 element, for third element we would traverse 2 elements and so on for nth element insertion we traverse n-1 elements and thus generally the resultant overall time complexity will be O(n^2).]

Time efficiency to Pop n elements:

Similar to above, here we have a **Stack (i.e Last in First out (LIFO))** implemented by employing a singly-headed singly-linked list. Another specification is that the **Stack must have its top at the back of the linked list** as opposed to the Front (Head). Now as we know for a stack, an element is always removed (Pop) from the top of the stack. (Figure 1)

Moreover, here as the top of the stack is located at the end of the singly-headed singly-linked list, thus in order **to remove an element from the top we must traverse** through the linked list using the declared single-head (Head) at the front **i.e O(n)**. Furthermore, **to remove an element a link is deleted** by pointing the current second-last node to the null pointer and this can be done in **O(1) time complexity**. Thus the **total time complexity for removing an element** from the top of stack (i.e the end of linked list here) will be **O(n)**. (Figure 2)

Finally, when this is **conducted for n elements i.e while removing (popping) n elements** from the stack, we would repeat the removal procedure with time complexity O(n) for n times and thus the resultant **overall time complexity will be O(n^2).**

[Best case would be when we remove the last remaining element as there is no traversal and hence it would be removed with O(1) time complexity. Moreover as elements are removed we need to traverse less elements as we travel n-1 elements to remove nth element while we just traverse one element to remove the second-last remaining element. Thus however generally the resultant overall time complexity will be O(n^2).]

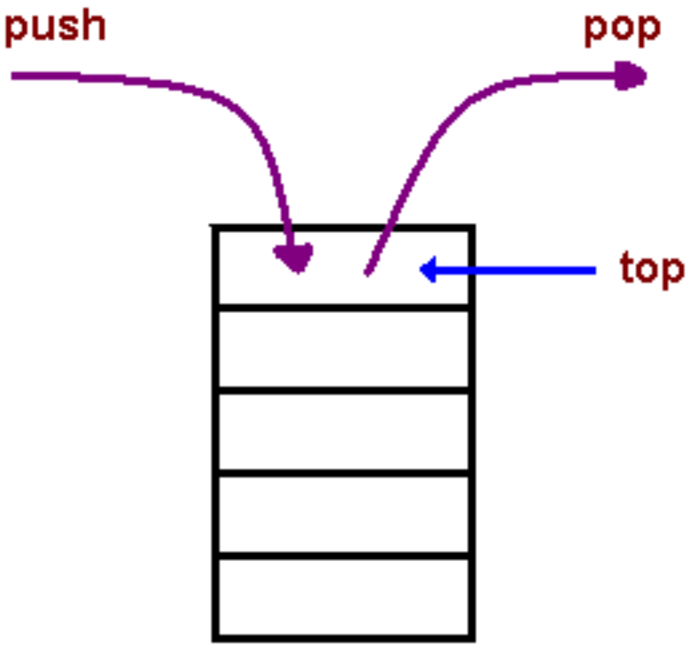

FIG. 1

(A General Stack design)

FIG. 2

(Singly-headed Singly-linked list)

Author: Vansh Bhatt
Student number: 301471598
CMPT 225 - D-100 - Assignment-2 - Solution to Q.2