

Author: Fardin Abdulla

Date: 16/02/2024

Time Efficiency Analysis

1.push():

```
39 // Push an element onto the stack
40 void Stack::push(int value)
41 {
42     // newNode creating
43     StackNode *newNode = new StackNode{value, nullptr};  $\rightarrow O(1)$ 
44
45     if (isEmpty())  $\rightarrow O(1)$ 
46     {
47         bottom = newNode; // because its empty so newNode is now the first node  $\rightarrow O(1)$ 
48     }
49     else
50     {
51         StackNode *current = bottom;  $\rightarrow O(1)$ 
52         while (current->next != nullptr)  $\rightarrow O(n)$ 
53         {
54             current = current->next;  $\rightarrow O(1)$ 
55         }
56         current->next = newNode;  $\rightarrow O(1)$ 
57     }
58 }
```

Overall time efficiency of Algorithm pop(): $\max[O(1), O(1), O(1), O(n), O(1)] = O(n)$

The push operation involves adding a new element to the back of the singly linked list. Since the top of the stack is located at the back of the list, we need to traverse the entire list to find the last node and then add a new node after it. But, as we are analyzing the total time required to push n elements to the Stack, it would be $= [O(n) * O(n)] = O(n^2)$

2. pop():

```

47 // Pop an element from the stack
48 int Stack::pop()
49 {
50     if (isEmpty())  $\rightarrow O(1)$ 
51     {
52         // Handle underflow (throw an exception)
53         // For simplicity, let's just print an error message and terminate the program
54         std::cerr << "Error: Stack underflow!" << std::endl;
55         std::exit(EXIT_FAILURE);
56     }
57
58     StackNode *current = bottom;
59     StackNode *prev = nullptr;
60
61     while (current->next != nullptr)  $\rightarrow O(n)$ 
62     {
63         prev = current;  $\rightarrow O(1)$ 
64         current = current->next;  $\rightarrow O(1)$ 
65     }
66
67     int poppedValue = current->data;  $\rightarrow O(1)$ 
68
69     if (prev != nullptr) // More than one node in the stack  $\rightarrow O(1)$ 
70     {
71         prev->next = nullptr; // Remove the last node which means remove the current node
72     }
73     else  $\rightarrow O(1)$ 
74     {
75         bottom = nullptr;  $\rightarrow O(1)$ 
76     }
77
78     delete current;  $\rightarrow O(1)$ 
79     return poppedValue;  $\rightarrow O(1)$ 
80 }

```

Overall time efficiency of Algorithm pop(): $\max[O(1), O(n), O(1), O(1), O(1), O(1), O(1)] = O(n)$

The pop operation involves removing the last element from the back of the SHSL. So, the time complexity is $O(n)$. But, as we are analyzing the total time required to pop those n elements from the Stack, it would be $= [O(n) * O(n)] = O(n^2)$