Asheesh Yadav
Eshaan Virk
CMPT 225
February 16, 2024

Push(): Time Complexity is O(n^2)

Reason: In this method, we are adding a new node at the end of the linked list. To do this, we must use a while loop and traverse the linked list, starting from the head and working our way to the end of the list. The time it takes to perform this depends on the size/number of elements in the list, hence the time complexity is O(n). Since the time complexity for pushing a single element is O(n), then the time complexity for pushing n elements will be O(n), since each push action traverses the linked list in O(n) time. But because you are calling n amount of elements and n amount of times we can see it as a sort of nested loop where instead of O(n) we get O(n^2) since pushing 1 element is n, pushing n elements n amount of times is essential n*n or n^2 where we get O(n^2)

Proof for O(n) for 1 element:

```
// Push operation
void Stack::push(int value) {
    // Create a new node
    StackNode* newNode = new StackNode; // (1) -> O(1)
    newNode->data = value; // (2) -> O(1)
    newNode->next = nullptr; // (3) -> O(1)

    // If the stack is empty, set the new node as the top
    if (isEmpty()) { // (4) -> O(1)
        top = newNode; // (5) -> O(1)
    } else { // (6) -> O(1)

        // Otherwise, add the new node to the back of the linked list
        StackNode* temp = top;// (7) -> O(1)

        while (temp->next != nullptr) { // (8) -> O(n)
            temp = temp->next; // (9) -> O(1)
        }
        temp->next = newNode;// (10) -> O(1)
    }
} // Total time complexity: O(n) since MAX[1,2,3,4,5,6,7,8,9,10] = O(n) [n = number of elements in the stack]
```

Pop(): Time Complexity is O(n^2)

Reason: In this method, you're removing a node from the end of the linked list. Similar to the push() operation, you need to traverse the entire list to find the last node and the node before the last node. Therefore, the time complexity is also O(n). This time complexity stays the same for popping n elements because each pop action requires us to traverse the linked list in O(n) time, and so the time complexity for popping n elements is O(n). And this would be if we are popping 1 element not n elements. Similar to our push() we can assume since popping 1 element n times takes a time complexity of O(n) so popping n elements n amount of times would result in O(n^2) since its n*n.

Proof for O(n) for 1 element:

```cpp
// Pop operation
int Stack::pop() {
    if (isEmpty()) {// (1) -> O(1)


        return -1; // (2) -> O(1)
    }

    // If there's only one element, update top to nullptr
    if (top->next == nullptr) { // (3) -> O(1)
        int data = top->data; // (4) -> O(1)
        delete top; // (5) -> O(1)
        top = nullptr; // (6) -> O(1)
        return data; //(7) -> O(1)
    }

    // If there are multiple elements, remove the last node
    StackNode* temp = top; // (8) -> O(1)
    while (temp->next->next != nullptr) { // (9) -> O(n)
        temp = temp->next; // (10) -> O(1)
    }
    int data = temp->next->data; // (11) -> O(1)
    delete temp->next; // (12) -> O(1)
    temp->next = nullptr; // (13) -> O(1)
    return data; // (14) -> O(1)
} // Total time complexity: O(n) since MAX[1,2,3,4,5,6,7,8,9,10,11,12,13,14] = O(n) [n = number of elements in the stack]
```