Q2: Assignment 2 CMPT-225

Authors: Izaz Zubayer (301386899) and Zoe Stanley (301427509)
Modified on: February 2024

Code included for reference, and showing of work.
N = Total number of inputs (in this case, elements).
Thus, O(N) = executes N times (linear), and O(1) = constant execution.

PUSH

```
bool Stack::push(int & data) {
        StackNode * newNode = new StackNode(data);  O(1)
                                                    IF-CLAUSE 1
        if (head == nullptr) {                      O(1)
        head = newNode;                             O(1)
        elementCount++;                             O(1)
        return true; // end function                O(1)

        } else {                                    IF-CLAUSE 2
        StackNode * current = head;                 O(1)
        while (current->next != nullptr) {  O(N) - makes N comparisons based on # elements
        current = current->next;                    O(1) * N - occurs N-1 times, head -> end
        }
        current->next = newNode;                    O(1)
        elementCount++;                             O(1)
        }
        return true;                                O(1)
}
```

FINAL ANALYSIS (Pushing N Elements):
O(1) if the list is empty: O(1) + O(1) * 4 = O(1), constant time.
O(N) if the list is full, as comparisons start at the head, through to the last element (checking until ->next == nullptr): O(1) + O(1) + O(1) * O(N) + O(1)*3 = O(N).
BEST CASE SCENARIO (empty list): O(1)
WORST + AVERAGE CASE SCENARIO (non-empty list): O(N).

POPPING N ELEMENTS

```
int Stack::pop() {                                                    IF-CLAUSE 1
        if (elementCount == 0) {                                      O(1)
        throw std::out_of_range("Stack is empty, nothing to pop here."); O(1)
        }                                         IF-CLAUSE 2
```

```
    if (head->next == nullptr) {              O(1)
    int value = head->data;                   O(1)
    delete head;                              O(1)
    head = nullptr;                           O(1)
    elementCount--;                           O(1)
    return value;                             O(1)
    }
    StackNode * current = head;               O(1)
    while (current->next->next != nullptr) {  O(1) * N-1 // only goes to second last from head
    current = current->next;                  O(1) * N-1
    }
    int value = current->next->data;          O(1)
    delete current->next;                     O(1)
    current->next = nullptr;                  O(1)
    elementCount--;                           O(1)
    return value;                             O(1)
}
```

So, in order to pop the N elements we pushed onto the Stack, we need to execute the 1st or 2nd IF-CLAUSE above.

FINAL ANALYSIS (Popping N Elements):
If the head of the original list was empty, push was completed in O(1), and to pop this single element we need only O(1) * 6 + O(1) statements, 7*O(1) which leads to popping in O(1).
If the list was populated, then in order to pop the first of N elements we must execute first the initial checks, O(1) for exception handling and O(1) * 6 for constant statements. Further elements now have a list of N-1, N-2, etc. So the final analysis looks something like:
O(N) + O(N-1) + O(N-2) + … + O(1) for the final (best-case) single element pop. Thus the sum of series formula gives (N(N+1))/2, and thus (N^2 + N)/2. Dropping lower order terms gives N^2/2, which is the same as O(N^2).

BEST-CASE SCENARIO: O(1), constant time.
WORST- AND AVERAGE-CASE SCENARIO: O(N^2), quadratic time.