Shrey Joshi

Net ID: SXJ210081

1) The loop iterates N times and the .add method to the List class is an $O(N)$ operation, so the Big-O running time is $O(N^2)$

a) If an ArrayList is passed for the two, insertion to the front is still $O(N)$ so the Big-O running time is still $O(N^2)$.

b) If a LinkedList is passed, insertion to the front becomes $O(N)$ since we are just rearranging pointers, so the Big-O running time becomes $O(N)$.

2) The iteration is $O(N)$ and the remove operator is also $O(N)$, so the total time complexity is $O(N^2)$. a) Still $O(N)$ since the remove operator is still $O(N)$ and there is an iteration loop of $O(N)$. b) $O(N)$ because now removing is $O(1)$ so we simply iterate.

3) There's a nested for loop so the complexity is $O(N * M)$ where N and M are the lengths of lst1 and lst2.

a) Still $O(N * M)$ since there are still two nested for loops, each iteration taking $O(N)$ and $O(M)$ time respectively

b) Still $O(N * M)$ because two nested for loops.

4) $O(N)$ since the loop is iterating N times and the .get function is $O(1)$. a) Still $O(N)$ since there is a loop that is $O(N)$ and the .get function is still $O(1)$. b) $O(N^2)$ since the loop contributes $O(N)$ but the indexing function is now $O(N)$ due to the nature of a linkedlist.

5) $O(N^2)$ for a List¡Integer¿. Each remove from the front is $O(N)$, and there are N such remove operations. Thus populating the stack is $O(N^2)$. Then inserting each item at the end is $O(N)$. Thus the total time complexity is $O(N^2)$

a) If an ArrayList is passed, none of the reasoning above changes so the time complexity is still $O(N^2)$

b) If a LinkedList is passed, removal is now $O(1)$ and so the time complexity becomes $O(N)$ since there are N iterations in populating the stack and N iterations in repopulating the linked list.

6. Show each step of converting a+b*c+(d-e) from infix to postfix notation, using the algorithm described in the textbook that uses a stack.

1) Encounter a: add a to output output: [a] stack: []

2) Encounter +: add + to stack output: [a] stack: [+]

3) Encounter b: add b to output output: [ab] stack: [+]

4) Encounter *: add * to stack output: [ab] stack: [+*]

5) Encounter c: add to output: output: [abc] stack: [+*]

6) Encounter +: pop * and add + to stack output: [abc*+] stack: [+]

7) Encounter (: add to stack output: [abc*+] stack: [+(]

8) Encounter d: add to output output: [abc*+d] stack: [+(]

8) Encounter -: add to stack output: [abc*+d] stack: [+(-]

9) Encounter e: add to output output: [abc*+de] stack: [+(-]

10) Encounter (: pop - and (from stack output: [abc*+de-] stack: [+]

11) End of input: pop everything output: [abc*+de-+] stack: []