Complexity Analysis (Project 3)

Fatima Rahman

NOTE: I assumed I should calculate worst case complexity. Of course, in a best or medium case, these answers may be very different, but at max this is the time complexity of these functions.

add() function — O(1). You are only performing one action, and it takes a constant amount of time, no matter how large the data is. In this case, you're setting the first item in the list equal to the item you input. (Both best and worst case.)

append() function — O(n). This is the worst case complexity, i.e. the list isn't empty, so you have to go to the else statement and cycle through the while loop. You have to cycle through the entire linked list, i.e. the length of n, to reach the end to append the item. (Best case, list is empty, O(1). Only have to traverse if statement.)

concatenate() function — O(n). You must cycle through the entirety of the other linked list in a for loop up until the end (i.e. length n) in order to fully append an entire list to the end of the primary list.

delete() function — O(n). This is the worst case complexity, i.e. the list isn't empty, so you have to cycle through a while loop to find something equal to the index (linear search). In the worst case, the index is either at the very end of the loop or not even existent, so you must loop through length n amount of times. (Best case: O(1). List is empty, so you can just return out of the method.)

get() function — O(n). This is the worst case complexity, i.e. you have to insert anywhere that isn't the very first element of the list, so you have to cycle through a while loop to find something equal to the index (linear search). In the worst case, the index is either at the very end of the loop or not even existent, so you must loop through length n amount of times. (Best case: O(1), i.e. you want to return the first item in the list, which only requires you to look at the very first if statement.)

insert() function — O(n). This is the worst case complexity, i.e. the list isn't empty, so you have to cycle through a while loop to find something equal to the index (linear search). In the worst case, the index is either at the very end of the loop or not even existent, so you must loop through length n amount of times in order to insert something. (Best case: O(1). This is if you want to insert an item at index 0, and thus all you have to do is call the add() function)

length() function — O(n). This is the worst case complexity, i.e. the list isn't empty. You have to loop through the entire linked list, of length n, to tally up the length. (Best case: O(1). This is if the linked list is empty, and thus the first element after head is null, therefore you only have to test the first if statement, which proves to be true.)

print() function — O(n). This is the worst case complexity, and similarly to the length() function, you have to loop through the entire linked list in order to fully execute the function, in this case, printing out the items in the linked list. (Best case: O(1), meaning the list is empty.)

remove() function — O(n^2). This is the worst case complexity, i.e. the list isn't empty, so you have to cycle through a for loop to find something equal to the item input (linear search). In the worst case, the item is either at the very end of the loop or not even existent, so you must loop through length n (of the list) amount of times. Then you must call the delete function, with time complexity of O(n). Because the linear search's complexity was O(n), n*n = n^2. (Best case, you want to remove the first element — O(1). Only one operation done.)

sort() function — O(n^2). Because this function is modeled off the bubble sort, which has a complexity of O(n^2), this function has the same time complexity. To give further detail, because we have two nested for loops, both looping through the list (the other one n amount of times, the inner n-i times. n*(n-i) = n^2 - ni. We take n^2 and get the big O of the function!). (Best case: O(n).)