Data Structures and Algorithms - Homework 1

HK Rho

January 31, 2020

1.

I would implement list concatenation by appending. The run-time for appending using the big O analysis is $O(n)$, or linear, since appending one element is $O(1)$ and appending n elements would make the run-time to be $O(n)$ according to the equation below.

$$n * O(1) = O(n) \tag{1}$$

If the list surpasses the allocated memory, the resize constant will be necessary. Given a resize constant of a, the time complexity would then be $O(an)$, which is still linear. Because what we care is the most dominant behavior, the time complexity $O(an)$ can be simplified to $O(n)$.

Insights After Going to NINJA Hours
If we are taking an amortized analysis on list concatenation, the big O analysis would be $O(1)$, for amortized analysis is done over the general/average behavior of the operation. Since the ratio of the number of times we append ($O(1)$) is far greater than the number of times we have to resize ($O(n)$) generally, the time complexity would be $O(1)$ for an amortized analysis.

2-(a).

Declare two empty lists - a currently-storing list, an output list
for i in range(length of input list - 1):
    Always add the first item of the input list to the currently storing list.
    Do this by checking the length of the currently storing list.

    Check if the next item from the input list > current item.
        If so, add the next item to the currently storing list.

    Check if the next item from the input list is not greater than the current item.
        Check if the len(current list) > len(output list).
            If so, update the output as current and empty the current.
        Check if the len(current list) < len(output list).
            If so, keep the output list and empty the current list.

2-(b).

The run-time of the pseudocode above would be $O(n)$, because it will go through all the elements of the given input list once each. This can be seen as there is one for loop that goes through all of the elements of the input list.