

CSC148 Worksheet 14 Solution

Hyungmo Gu

April 24, 2020

Question 1

a.

Operation	Running time
Insert at the front of the list	$\mathcal{O}(n)$
Insert at the end of the list	$\mathcal{O}(1)$
Look up the element at index i , where $0 \leq i < n$	$\mathcal{O}(n)$

Correct Solution:

Operation	Running time
Insert at the front of the list	$\mathcal{O}(n)$
Insert at the end of the list	$\mathcal{O}(1)$
Look up the element at index i , where $0 \leq i < n$	$\mathcal{O}(1)$

b. The inserting of an element at position i requires $n - i$ elements to be shifted to right.

Using this fact, we can write the Big-Oh expression for inserting an item at index i is $\mathcal{O}(n - i)$.

Question 2

a.

Operation	Running time
Insert at the front of the linked list	$\mathcal{O}(1)$
Insert at the end of the linked list	$\mathcal{O}(n)$
Look up the element at index i , where $0 \leq i < n$	$\mathcal{O}(n)$

Correct Solution:

Operation	Running time
Insert at the front of the linked list	$\mathcal{O}(1)$
Insert at the end of the linked list	$\mathcal{O}(n)$
Look up the element at index i , where $0 \leq i < n$	$\mathcal{O}(i)$

b. Without the traversal, the running time of inserting is $\mathcal{O}(1)$.

With the traversal, the running time of inserting is $\mathcal{O}(i)$.

Question 3

- Unlike linked lists that store node at different memory location, array-based lists store elements in memory immediately one after another.

Assuming it's easier for memory to find and perform operations on elements located right after another, I believe it's significantly faster for array-based lists to insert an element at position i .

Correct Solution:

Since $n - i = 1,000,000 - 500,000 = 500,000$, we can write $\mathcal{O}(n - i) \approx \mathcal{O}(i)$

Using this fact, we can conclude the speed of linked lists and array-based lists are roughly about the same.

Notes:

- Noticed that professor compared the performance of linked lists and array-based list in terms of Big-Oh.

Question 4

- a. When $n = 1$, the total number of nodes traversed is 0. This is because we are only replacing None in `self._first` with `_Node(item)`.

When $n = 2$, the total number of nodes traversed is 0. This is because after adding the first element, we start at `self._first`, and add `_Node(item)` to `self._first.next`.

When $n > 2$, the number of nodes traversed increases by 1 per item added starting with the 3rd element, and this continues until $n - 1$ (where it represents the last item in a list). So in this case, the total number of nodes traversed is

$$\sum_{i=2}^{n-1} (i - 1) = \sum_{i'=1}^{n-2} i' \quad (1)$$

$$= \frac{(n - 2)(n - 1)}{2} \quad (2)$$

Correct Solution:

The code for *append* method tells us

```

1      class LinkedList:
2          ...
3          def append(self, item: Any) -> None:
4              """Append <item> to the end of this list.
5              """
6              if self._first is None:
7                  self._first = _Node(item)
8              else:
9                  curr = self._first
10                 while curr.next is not None:
11                     curr = curr.next
12                 curr.next = _Node(item)
13

```

Listing 1: linked_list.py

When $n = 1$, the total number of nodes traversed is 0. This is because we are only replacing None in *self._first* with *_Node(item)*.

When $n > 1$, we know the number of nodes traversal required to add an item increases by 1 starting with the 2nd element and this continues until $n - 1$ (where it represents the last item in a list). So in this case, the total number of nodes traversed is

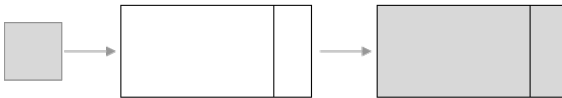
$$\sum_{i=1}^{n-1} i = \frac{n(n - 1)}{2} \quad (3)$$

Notes:

- I am feeling blue. I feel csc 148 worksheets are designed to be used in class, with details and expectations to questions learned as students are interacting with professor.
- Learned that the number of node traversed means the number of nodes that needs to be traveled to get to the closest None.



Node traversed : 0



Node traversed : 1



Node traversed : 2

- b. Since we know from previous problem that the running time of this operation is $\frac{n(n-1)}{2}$, we can conclude its Big-Oh expression is $\mathcal{O}(n^2)$.

Question 5

- It would be possible to return *lst.__contains__(148)* after visiting just a single node if the first node of the linked list contains value 148.
- It would be necessary to visit all nodes in the linked list when no nodes contain value 148.