CSC148 Worksheet 11 Solution

Hyungmo Gu

April 22, 2020

Question 1

a. Here, the constant time means the running time of accessing and assigning element by index doesn't depend on the length of the list.

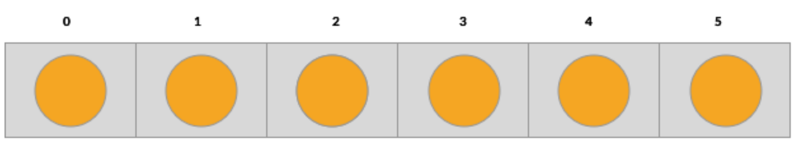
b.

$$n - i$$

many elements need to be shifted to right.

<u>**Notes:**</u>

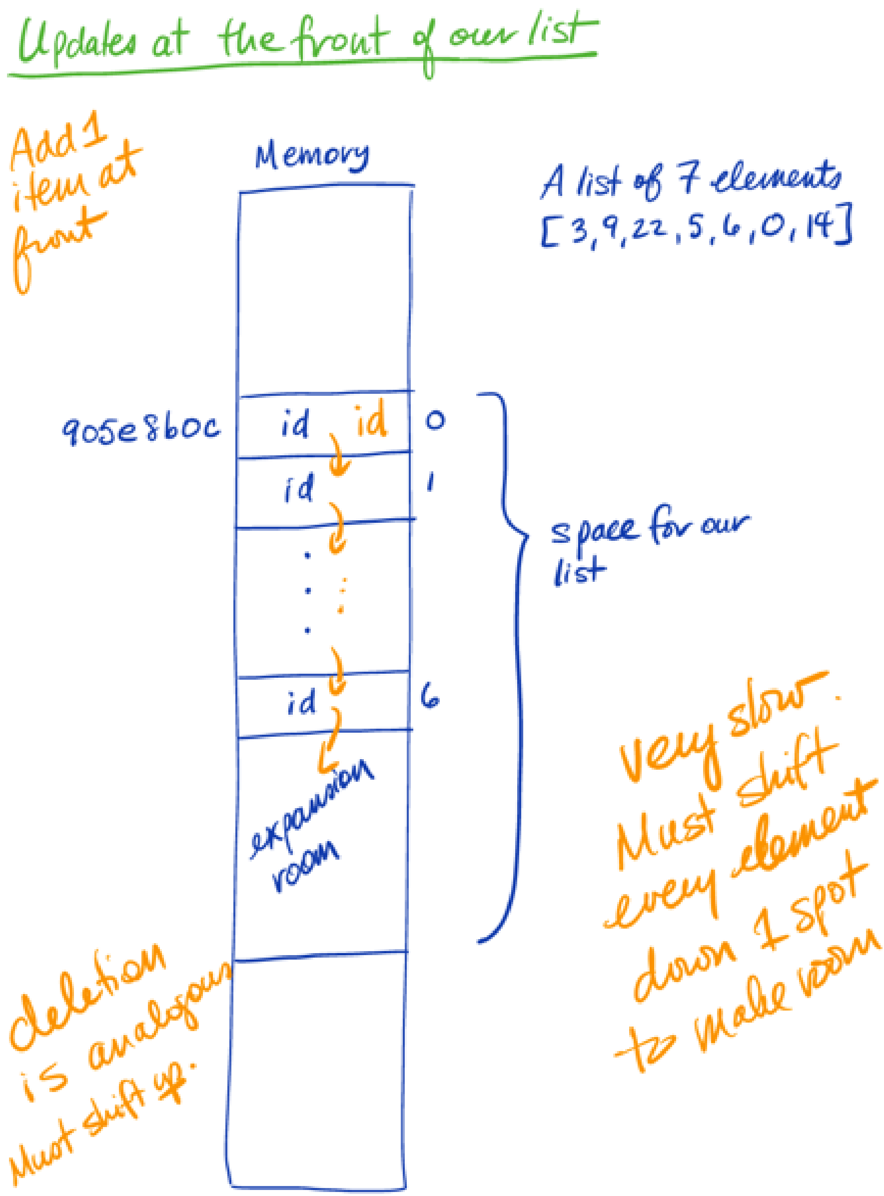
- The following example tells us



  to position an element at index $i = 3$ of the list, $n - i = 6 - 3 = 3$ elements must be moved over.

  Using this fact, we can generalize that to position an element at index i of the list, $n - i$ many elements must be shifted.

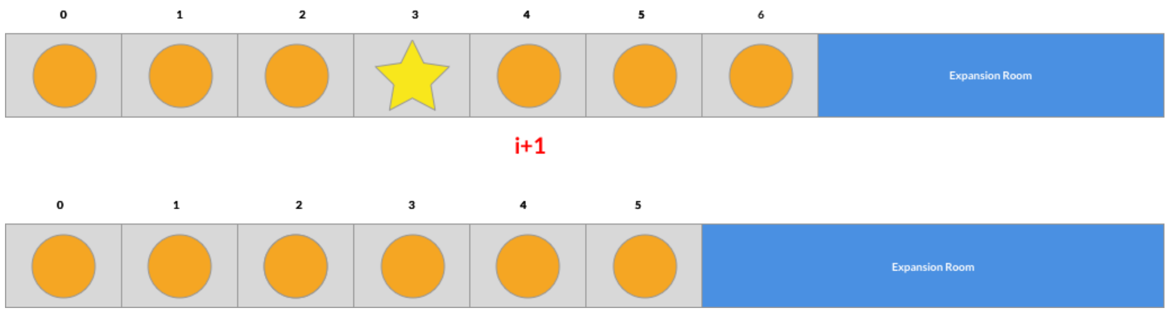
- Learned that when items shifts, it shifts into the expansion room.



c. Because we know the list size stays as is when an element is removed, we can conclude 0 many list elements must be moved.

Correct Solution:

The following example tells us



when an element at index $i = 3$ is removed from the list $n - (i + 1) = 7 - (3 - 1) = 3$ many elements must be moved.

Using this fact, we can generalize that when an element is removed, $n-(i+1) = n-i-1$ many elements must be shifted to left.

d. i. A solution is *LIST.remove(...)*.

The answer to question 1.d tells us when an element is removed, $n - i$ must be shifted to left.

Using this fact, we can write a list of smaller size needs to shift elements less.

Then, it follows from this fact that $n = 100$ works faster than $n = 1,000,000$.

ii. A solution is *LIST.append(...)*

The definition of append tells us that upon call, an element is added to the end of a list, it takes a constant time to add an element as long as the expansion room is not filled.

It follows from this fact that $n = 100$ and $n = 1,000,000$ takes roughly the same amount of time.

e. The definition of *Queue* tells us *Queue* is FIFO. That is, the first element inserted is the first to come out.

Since we know the front of *Queue* is the front of the list, we can conclude *LIST.insert(...)* and *LIST.pop(0)* are used to support *QUEUE.enqueue* and *QUEUE.dequeue*, respectively.

Since we know *LIST.insert(...)* requires shifting of elements by $n - i = n - 0 = n$ and *LIST.pop(0)* requires shifting of $n - i - 1 = n - 1$ many elements, we can conclude both *QUEUE.enqueue* and *QUEUE.dequeue* takes longer time as size increases.

Correct Solution:

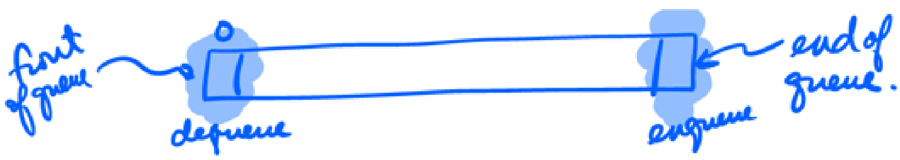
The definition of *Queue* tells us *Queue* is FIFO. That is, the first element inserted is the first to come out.

Since we know the front of *Queue* is the front of the list, we can conclude *LIST.append(...)* and *LIST.pop(0)* are used to support *QUEUE.enqueue* and *QUEUE.dequeue*, respectively.

Since we know *LIST.append(...)* requires the shifting of elements by $n - i = n - n = 0$ and *LIST.pop(0)* requires shifting of $n - i - 1 = n - 1$ many elements, we can conclude *QUEUE.dequeue* takes longer time as size increases.

Notes:

- Learned that the **front of queue** is where *QUEUE.dequeue* occurs.



Question 2

a. **Stack 1:**

$$\begin{aligned} \text{Number of steps for } s.push(1) + \text{Number of steps for } s.pop() &= 1 + 1 \\ &= 2 \end{aligned}$$

Stack 2:

$$\begin{aligned} \text{Number of steps for } s.push(1) + \text{Number of steps for } s.pop() &= (n+1) + (n+1) \\ &= 2n + 1 \end{aligned}$$

b. **Stack 1:**

We need to determine the total number of steps taken in the code using *Stack 1*.

First, we need to find the number of steps taken per iteration.

The code tells us there is only *s.push(i)* in for loop.

Since we know the *push* operation in *Stack 1* takes 1 step, we can conclude 1 step is taken per iteration.

Finally, we need to determine the total number of steps taken.

Because we know the loop performs 5 iterations and each iteration takes 1 step, we can conclude the code takes total of

$$5 \cdot 1 = 5 \tag{1}$$

steps.

Stack 2:

We need to determine the total number of steps taken in the code using *Stack 2*.

First, we need to find the number of steps taken per iteration.

The code tells us there is only *s.push(i)* in for loop.

Since we know the list in s starts empty, and the size of list grows to i at i^{th} iteration, we can conclude the size of list n is i.

Since we know the *push* operation in *Stack 2* takes $n+1$ step, and since we know $n = i$, we can conclude $i+1$ step is taken per iteration.

Finally, we need to determine the total number of steps taken.

Because we know the loop starts at $i = 0$, ends at $i = 4$ with each iteration taking $i+1$ steps, we can conclude the code has total of

$$\begin{align} \sum_{i=0}^{4}(i+1) &= \sum_{i'=1}^{5} i' \tag{1} \\ &= \frac{5(5+1)}{2} \tag{2} \\ &= \frac{30}{2} \tag{3} \\ &= 15 \tag{4} \end{align}$$

steps.

c. **<u>Stack 1:</u>**

We need to determine the total number of steps taken in the code using *Stack 1*.

First, we need to find the number of steps taken per iteration.

The code tells us there is only *s.push(i)* in for loop.

Since we know the *push* operation in *Stack 1* takes 1 step, we can conclude 1 step is taken per iteration.

Finally, we need to determine the total number of steps taken.

Because we know the loop performs $(k-1)-0+1=k$ iterations and each iteration takes 1 step, we can conclude the code takes total of

$$k \cdot 1 = k \tag{5}$$

steps.

<u>Stack 2:</u>

We need to determine the total number of steps taken in the code using *Stack 2*.

First, we need to find the number of steps taken per iteration.

The code tells us there is only *s.push(i)* in for loop.

Since we know the list in s starts empty, and the size of list grows to i at i^{th} iteration, we can conclude the size of list n is i.

Since we know the *push* operation in *Stack 2* takes $n+1$ step, and since we know $n=i$, we can conclude $i+1$ step is taken per iteration.

Finally, we need to determine the total number of steps taken.

Because we know the loop starts at $i=0$, ends at $i=k-1$ with each iteration taking $i+1$ steps, we can conclude the code has total of

$$\begin{aligned} \sum_{i=0}^{k-1}(i+1) &= \sum_{i'=1}^{k} i' && (1) \\ &= \frac{k(k+1)}{2} && (2) \end{aligned}$$

steps.

Correct Solution:

Stack 1:

We need to determine the total number of steps taken in the code using *Stack 1*.

First, we need to find the number of steps taken per iteration.

The code tells us there is only *s.push(i)* in for loop.

Since we know the *push* operation in *Stack 1* takes 1 step, we can conclude 1 step is taken per iteration.

Finally, we need to determine the total number of steps taken.

Because we know the loop performs $(k-1)-0+1=k$ iterations and each iteration takes 1 step, we can conclude the code takes total of

$$k \cdot 1 = k \tag{1}$$

steps.

Stack 2:

We need to determine the total number of steps taken in the code using *Stack 2*.

First, we need to find the number of steps taken per iteration.

The code tells us there is only *s.push(i)* in for loop.

Since we know the list in s starts empty, and the size of list grows to i at i^{th} iteration, we can conclude the size of list n is i.

Since we know the *push* operation in *Stack 2* takes $n+1$ step, and since we know $n=i$, we can conclude $i+1$ step is taken per iteration.

Finally, we need to determine the total number of steps taken.

Because we know the loop starts at $i=0$, ends at $i=k-1$ with each iteration taking $i+1$ steps, we can conclude the code has total of

$$\sum_{i=0}^{k-1}(i+1) = \sum_{i'=1}^{k} i' \quad (1)$$

$$= \frac{k(k+1)}{2} \quad (2)$$

steps.

d. <u>**Stack 1:**</u>

We need to determine the total number of steps taken in the code using *Stack 1*.

First, we need to determine the number of steps taken by *s2.push(s1.pop())*.

The problem tells us both *s2.push(...)* and *s1.pop()* takes 1 step.

Using this fact, we can conclude *s2.push(s1.pop())* takes 1 step.

Second, we need to determine the total number of steps taken by loop 1.

The code tells us that for loop 1, an element popped from *s1* is inserted to *s2*, and this is repeated until no elements are left in *s1*.

Since we know *s1* starts with n many elements, and its size decreases by 1 until its length is 0, we can write that with k representing k^{th} iteration in loop 1, the terminating condition is reached when

$$n - k \leq 0 \quad (1)$$

$$n \leq k \quad (2)$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 1 has

$$\lceil n \rceil = n \quad (3)$$

many iterations.

Since we know each iteration in loop 1 takes 1 step, we can conclude loop 1 takes total of

$$n \cdot 1 = n \tag{4}$$

steps.

Third, we need to show that the second loop takes total of 0 steps.

The code tells us *s2* starts a stack of size 0.

Since loop 2 terminates when the size of *s2* is 0, we can conclude loop 2 has 0 iterations.

Using this fact, we can conclude loop 2 has total of 0 steps.

Finally, we need to determine the total number of steps taken in this code.

Since we know loop 1 takes total of n steps and loop 2 takes total of 0 step, we can conclude the total number of steps taken by the code is

$$n + 0 = n \tag{5}$$

steps.

Rough Work:

We need to determine the total number of steps taken in the code using *Stack 1*.

1. Determine the number of steps taken by *s2.push(s1.pop())*

   First, we need to determine the number of steps taken by *s2.push(s1.pop())*.

   > First, we need to determine the number of steps taken by *s2.push(s1.pop())*.
   >
   > The problem tells us both *s2.push(...)* and *s1.pop()* takes 1 step.
   >
   > Using this fact, we can conclude *s2.push(s1.pop())* takes 1 step.

2. Determine the total number of steps taken by loop 1.

   Second, we need to determine the total number of steps taken by loop 1.

Second, we need to determine the total number of steps taken by loop 1.

The code tells us that for loop 1, an element popped from *s1* is inserted to *s2*, and this is repeated until no elements are left in *s1*.

Since we know *s1* starts with n many elements, and its size decreases by 1 until its length is 0, we can write that with k representing k^{th} iteration in loop 1, the terminating condition is reached when

$$n - k \leq 0 \tag{6}$$
$$n \leq k \tag{7}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 1 has

$$\lceil n \rceil = n \tag{8}$$

many iterations.

Since we know each iteration in loop 1 takes 1 step, we can conclude loop 1 takes total of

$$n \cdot 1 = n \tag{9}$$

steps.

3. Show that the second loop takes total of 0 steps.

   Third, we need to show that the second loop takes total of 0 steps.

Third, we need to show that the second loop takes total of 0 steps.

The code tells us *s2* starts a stack of size 0.

Since loop 2 terminates when the size of *s2* is 0, we can conclude loop 2 has 0 iterations.

Using this fact, we can conclude loop 2 has total of 0 steps.

4. Calculate the total number of steps.

Finally, we need to determine the total number of steps taken in this code.

Since we know loop 1 takes total of n steps and loop 2 takes total of 0 step, we can conclude the total number of steps taken by the code is

$$n + 0 = n \tag{10}$$

steps.

Stack 1:

We need to determine the total number of steps taken in the code using *Stack 1*.

First, we need to determine the number of steps taken by *s2.push(s1.pop())*.

The problem tells us both *s2.push(...)* and *s1.pop()* takes 1 step.

Using this fact, we can conclude *s2.push(s1.pop())* takes 1 step.

Second, we need to determine the total number of steps taken by loop 1.

The code tells us that for loop 1, an element popped from *s1* is inserted to *s2*, and this is repeated until no elements are left in *s1*.

Since we know *s1* starts with n many elements, and its size decreases by 1 until its length is 0, we can write that with k representing k^{th} iteration in loop 1, the terminating condition is reached when

$$n - k \leq 0 \tag{11}$$
$$n \leq k \tag{12}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 1 has

$$\lceil n \rceil = n \tag{13}$$

many iterations.

Since we know each iteration in loop 1 takes 1 step, we can conclude loop 1 takes total of

$$n \cdot 1 = n \tag{14}$$

steps.

Third, we need to show that the second loop takes total of 0 steps.

The code tells us *s2* starts a stack of size 0.

Since loop 2 terminates when the size of *s2* is 0, we can conclude loop 2 has 0 iterations.

Stack 2:

Rough Work:

We need to determine the total number of steps taken in the code using *Stack 2*.

1. Determine the number of steps taken by *s2.push(s1.pop())* in loop 1.

   First, we need to determine the number of steps taken by *s2.push(s1.pop())* in loop 1.

   1. State that *s1.pop()* takes $n_1 + 1$ steps and *s2.push(...)* takes $n_2 + 1$.

      Let n_1 and n_2 be the size of stack for *s1* and *s2*, respectively.

      The code tells us that *s1.pop()* operation takes $n_1 + 1$ steps and *s2.push(...)* takes $n_2 + 1$ steps, where

   2. Show that *s2.push(s1.pop())* takes $i + 2$ steps.

      Since we know $s2$ starts as an empty stack, and $s1$ starts as a stack with the size of n, and since we know *s2.push()* and *s1.pop(...)* causes the stack size of *s2* and *s1* to increase and decrease by 1 per iteration, respectively, we can conclude that at k^{th} iteration, *s2.push(s1.pop())* takes total of

      $$(n - k + 1) + (k + 1) = n + 2 \tag{16}$$

      steps.

2. Determine the total number of steps taken by loop 1.

   Second, we need to determine the total number of steps taken by loop 1.

   1. State that for loop 1, an element popped from *s1* is inserted to *s2*, and this is repeated until no elements are left in *s1*.

      The code tells us that for loop 1, an element popped from *s1* is inserted to *s2*, and this is repeated until no elements are left in *s1*.

   2. Show loop 1 takes n iterations.

Since we know *s1* starts with n many elements, and its size decreases by 1 until its length is 0, we can write that with k representing k^{th} iteration in loop 1, the terminating condition is reached when

$$n - k \leq 0 \tag{17}$$
$$n \leq k \tag{18}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 1 has

$$\lceil n \rceil = n \tag{19}$$

many iterations.

3. Show loop 1 takes total of $n(n+2)$ steps.

Since we know each iteration in loop 1 takes $n+2$ step, we can conclude loop 1 takes total of

$$n \cdot (n+2) = n(n+2) \tag{20}$$

steps.

Second, we need to determine the total number of steps taken by loop 1.

The code tells us that for loop 1, an element popped from *s1* is inserted to *s2*, and this is repeated until no elements are left in *s1*.

Since we know *s1* starts with n many elements, and its size decreases by 1 until its length is 0, we can write that with k representing k^{th} iteration in loop 1, the terminating condition is reached when

$$n - k \leq 0 \tag{21}$$
$$n \leq k \tag{22}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 1 has

$$\lceil n \rceil = n \tag{23}$$

many iterations.

Since we know each iteration in loop 1 takes $n + 2$ step, we can conclude loop 1 takes total of

$$n \cdot (n + 2) = n(n + 2) \tag{24}$$

steps.

3. Show that the total number of steps taken by loop 2 is 0.

Third, we need to show that the second loop takes total of 0 steps.

The code tells us *s2* starts a stack of size 0.

Since loop 2 terminates when the size of *s2* is 0, we can conclude loop 2 has 0 iterations.

Using this fact, we can conclude loop 2 has total of 0 steps.

4. Conclude by calculating the total number of steps taken in this code.

Finally, we need to determine the total number of steps taken in this code.

Since we know loop 1 takes total of n steps and loop 2 takes total of 0 step, we can conclude the total number of steps taken by the code is

$$n(n+2) + 0 = n(n+2) \tag{25}$$

steps.