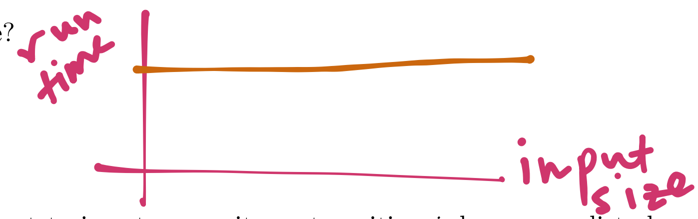
CSC148 - Running time efficiency: Lists, Stacks, and Queues

We have now seen that Python lists have the following running times for key operations:

- Accessing or assigning to any element by index takes *constant time*.
- Inserting or removing an item at a given index takes time *proportional to the number of items after the index*.

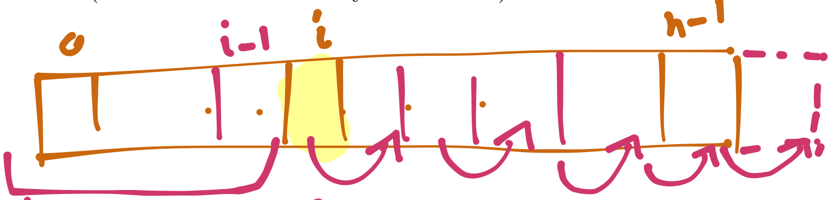
1. Answer the following questions to make sure you understand the key concepts before moving on.

   (a) What do we mean by "constant time" above?

   doesn't depend on input size

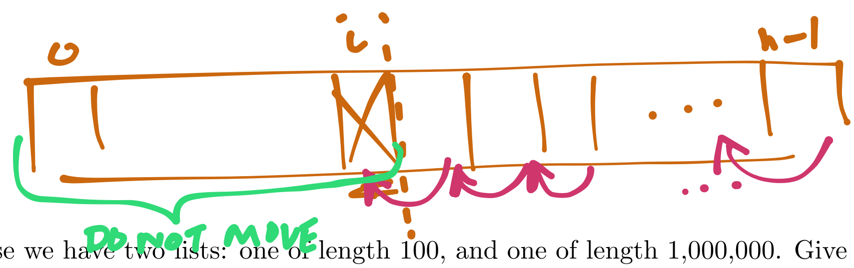
   (b) Suppose we have a list of length n. If we want to insert a new item at position i, how many list elements must be moved? (Be careful about off-by-one errors!)

   Total n elements
   i don't move
   $n-i$ do move

   i items do NOT move

   (c) Suppose we have a list of length n. If we want to remove the existing item at position i, how many list elements must be moved? Do not include the item being removed.

   $i+1$ items do NOT move
   $n-(i+1)$ do move

   (d) Suppose we have two lists: one of length 100, and one of length 1,000,000. Give an example of each of the following:

   (i) An operation that would be faster on the smaller list than the larger list.

   - Inserting new element to the beginning of the list

   (ii) An operation that would take roughly the same amount of time on the two lists.

   - Appending to the list
   - Accessing an index

   (e) Suppose we implement a Queue using a Python list, where the front of the list represents the front of the queue. Which operation do we expect to take longer as the queue size increases: `Queue.enqueue` or `Queue.dequeue`?

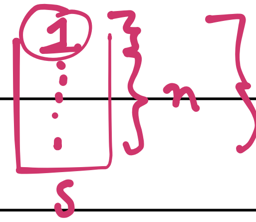
   add to end   remove the front

   * SLOWER

2. Now let's look at some code. Suppose we have two implementations of the Stack ADT: `Stack1` has `push` and `pop` operations that take 1 step, regardless of stack size, while `Stack2` has `push` and `pop` operations that take $n + 1$ steps, where n is the number of items currently on the stack. (You might argue it's "n steps", but that difference doesn't matter here.)

   For each of the code snippets on the next page, calculate the number of steps taken in total by all `push` and `pop` operations that are performed by the code. Do each calculation twice: once assuming we use the `Stack1` implementation, and once assuming we use the `Stack2` implementation. Ignore all other operations for this exercise—you're only counting steps for `push` and `pop` here.

(a)
```
# s starts as a stack of size n
s.push(1)
s.pop()
```



Stack1

push – 1 step
pop – 1 step
TOTAL = 2 steps

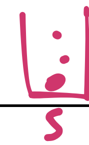
Stack2

push – n+1 steps
pop – (n+1)+1 steps

TOTAL: 2n+3

(b)
```
# s starts as an empty stack
for i in range(5)
    s.push(i)
```

loops 5 times



Stack1

push – 1 step × 5 loop iterations
TOTAL = 5 steps

Stack2

1st push = 0 +1 steps = 1
2nd push = 1 +1 steps = 2
TOTAL: $\underbrace{1+2+3+4+5}_{\text{5 loop iterations}}$ = 15 steps

(c)
```
# s starts as an empty stack, k is a positive integer.
# Calculate the number of steps in terms of k.
# Hint: 1 + 2 + 3 + ... + k = k * (k + 1) / 2
for i in range(k)
    s.push(i)
```

loops k times

Stack1

TOTAL = k steps

push = 1 step × k loop iterations

Stack2

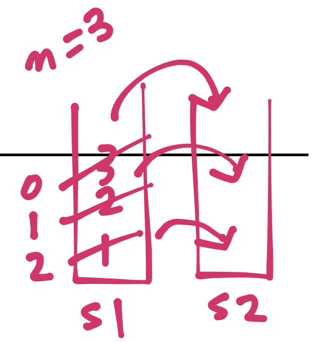
TOTAL: 1 + 2 + 3 + ... + k

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

(d)
```
# s1 starts as a stack of size n, and s2 starts as an empty stack
while not s1.is_empty():
    s2.push(s1.pop())

while not s2.is_empty():
    s1.push(s2.pop())
```

n iterations
2steps
n iterations
2steps



Stack1

2n for each loop

TOTAL = 4n

Stack2

1. Pushes
a) 1st loop

s2 has 0 1 2 .... (n-1)

1 step + 2 steps + 3 steps + ... + n

= n(n+1)/2 steps

continued

2. Pops

	1st pop	2nd pop	3rd pop		last pop
stack has	n	$n-1$	$n-2$	...	1
	$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$

$$\underset{\text{steps}}{n+1} + \underbrace{n + n-1 \quad +2+1}$$

$$= \sum_{i=1}^{n} i + (n+\cancel{1}) - \cancel{1}$$

$$= \frac{n(n+1)}{2} + n$$

b) second loop: another $\frac{n(n+1)}{2} + n$ steps

TOTAL pops: $2 \times \left[\frac{n(n+1)}{2} + n\right]$

$$= n(n+1) + 2n$$

$$= n^2 + n + 2n = n^2 + 3n$$

b) 2nd loop: another $n(n+1)/2$ steps for the pushes

TOTAL PUSHES $\cancel{2} \times \left[\frac{n(n+1)}{\cancel{2}}\right]$

$$= n(n+1)$$

$$= n^2 + n$$

GRAND TOTAL

$$(n^2+n) + (n^2+3n) = 2n^2 + 4n$$

$$= O(n^2)$$