- This lab will cover lists, list methods, and dynamic arrays.
- It is assumed that you have reviewed chapters 5 of the textbook. You may want to refer to the text and your lecture notes during lab as you solve the problems.
- When approaching the problems, think before you code. Doing so is good practice and can help you lay out possible solutions.
- Think of any possible test cases that can potentially cause your solution to fail!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (maximum 45 minutes)

---

1. Give the **worst case** run-time for each of the following list methods. Write your answer in terms of n, the length of the list. Provide an appropriate summation for multiple calls.

Given `lst = [ 1, 2, 3, 4, … ,n]` and `len(lst)` is `n`.

Have to worry about resizing and shifting

Method	Single (`1`) Call	Reason	Multiple (`n`) Calls `for i in range(n): ...`	Reason
`insert`	O(n)		What if lst starts empty and you're inserting n numbers?	n^2 + n(n+1)/2
`remove`	O(1)			
`append`	O(n)		What if lst starts empty and you're appending n numbers?	
`pop`	O(n)			

2. Given the following mystery functions:
   - **i.** Replace mystery with an appropriate name
   - **ii.** Determine the function's worst-case runtime and extra space usage with respect to the input size.

a.
```
def mystery(lst):
    for i in range(len(lst):
        val = lst.pop()
        lst.insert(i, val)
```

Reverse
O = n^2 = n+(n-1)+(n-2)+…+ = n(n-1)/2
Extra space = O(1)

b.
```
def mystery(n):
    for i in range(1, n+1):
        total = sum([num for num in range(i)])
        yield total
```

Sum_to
n(n+1)/2
Time: O(n^2)
Extra space O(n)

c.
```
def mystery(lst):
    lst2 = lst.copy()
    lst2.reverse()
    if (lst == lst2):
        return True
    return False
```

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code.

Download the **ArrayList.py** file found under Resources//Lectures on NYU Classes

Extend the ArrayList class implemented during lecture with the following methods:

a. Implement the `__repr__` method for the ArrayList class, which will allow us to display our ArrayList object like the Python list when calling the print function. The output is a sequence of elements enclosed in [ ] with each element separated by a space and a comma.

ex) `arrlst1` is an `ArrayList` with `[1, 2, 3]`
→ `print(arrlst1)` outputs `[1, 2, 3]`

Note: Your implementation should create the string in O(n), where n = len(arrlst).

b. Implement the `__add__` method for the ArrayList class, so that the expression arrlst1 + arrlst2 is evaluated to a **new** ArrayList object representing the concatenation of these two lists.

ex) `arrlst1` is an `ArrayList` with `[1, 2, 3]`
`arrlst2` is an `ArrayList` with `[4, 5, 6]`
→ `arrlst3 = arrlst1 + arrlst2`
`arrlst3` is a new `ArrayList` with `[1, 2, 3, 4, 5, 6]`.

Note: If n_1 is the size of arrlst1, and n_2 is the size of arrlst2, then __add__ should run in $\Theta(n_1 + n_2)$

c. Implement the `__iadd__` method for the ArrayList class, so that the expression arrlst1 += arrlst2 **mutates** arrlst1 to contain the concatenation of these two lists. You may remember that this operation functions similarly to the **extend method**.

Your implementation should return *self*, which is the object being mutated.

ex) `arrlst1` is an `ArrayList` with `[1, 2, 3]`
`arrlst2` is an `ArrayList` with `[4, 5, 6]`
→ `arrlst1 += arrlst2`
`arrlst1` is mutated and now has `[1, 2, 3, 4, 5, 6]`.

Note: If n_1 is the size of arrlst1, and n_2 is the size of arrlst2, then __iadd__ should run in $\Theta(n_1 + n_2)$. It's not n_2 because we have to take array resizing into account.

d. Modify the `__getitem__` and `__setitem__` methods implemented in class to also support **negative** indices. The position a negative index refers to is the same as in the Python list class. That is -1 is the index of the last element, -2 is the index of the second last, and so on.

ex) `arrlst1` is an `ArrayList` with `[1, 2, 3]`
→ `print(arrlst1[-1])` outputs `3`
→ `arrlst1[-1] = 5`
`print(arrlst1[-1])` outputs `5` now

Note: Your method should also raise an IndexError in any case the index (positive or negative) is out of range.

e. Implement the `__mul__` method for the ArrayList class, so that the expression arrlst1 * k (where k is a positive integer) creates a **new** ArrayList object, which contains k copies of the elements in mylist1.

ex) `arrlst1` is an `ArrayList` with `[1, 2, 3]`
→ `arrlst2 = arrlst1 * 2`
`arrlst2` is a new `ArrayList` with `[1, 2, 3, 1, 2, 3]`.

Note: If *n* is the size of arrlst1 and k is the int, then __mul__ should run in $\Theta(k * n)$.

f. Implement the `__rmul__` method to also allow the expression n * arrlst1. The behavior of n * arrlst1 should be equivalent to the behaviour of arrlst1 * n.

(You've done this before for the Vector problem in homework 1)