CS-UY 1134 **Lab 4** February 16, 2018

- This lab will cover Dynamic Arrays.
- It is assumed that you have reviewed chapter 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. It is good practice and generally helpful to lay out possible solutions for yourself.
- You should write test code to try out your solutions.
- You must stay for the duration of the lab. If you finish early, you can help other students to complete the lab. If you don't finish by the end of the lab, it is recommended that you complete the lab on your own time.
- Your TAs are available to answer your questions in lab, during office hours, and on Piazza.

Vitamins (maximum 30 minutes)

---

1. Given the following functions:
   - **i.** Trace the execution of each function with the given parameters:

     `lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
   - **ii.** Determine the function's asymptotic worst-case runtime.

a.

```
def doublst(lst):
      newlst = []
      for elem in lst:
            newlst.append(elem)
      lst.extend(newlst)
```

`Worst case` ______________________________

b.

```
def inslst(lst):
      length = 0
      origlen = len(lst)
      while length < (origlen * 2):
            square = lst[length]**2
            length += 1
            print(length, square)
            lst.insert(length, square)
            length += 1
```

`Worst case` ______________________________ c.

```
def popeven(lst):
      counter = len(lst)-1
      while counter > 0:
            if lst[counter] % 2 == 0:
                  lst.pop(counter)
                  counter -= 1
            counter -= 1
```

Worst case ____________________________

Coding

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

Extend the `MyList` class implemented during lecture with the following methods:

a. Implement the `__repr__` method for the `MyList` class, so that when calling the `repr` method on a `MyList` object it returns a representation of the list that follows the same representation as the builtin Python list. That is, a sequence of elements enclosed in `[ ]`, separated by a space.

   For example, if `mylist1` is an instance of `MyList` containing 1, 2, 3, then the call `str(mylist1)` should return "`[1, 2, 3]`".

   Note: Your implementation should run in a linear time. That is, proportional to the length of the created string.

b. Implement the `__add__` method for the `MyList` class, so that the expression `mylist1 + mylist2` is evaluated to a **new** `MyList` object representing the concatenation of these two lists.

   For example, if `mylist1` is a list containing `[1, 2, 3]` and if `mylist2` is a list containing `[4, 5, 6]`, and if we evaluate the expression `mylist3 = mylist1 + mylist2`, then `mylist3` will be a new `MyList` object containing `[1, 2, 3, 4, 5, 6]`.

   Note: if n_1 is the number of items in `mylist1`, and n_2 is the number of items in `mylist2`, then `mylist1 + mylist2` should run in $\Theta(n_1 + n_2)$

c. Implement the `__iadd__` method for the `MyList` class, so that the expression `mylist1 += mylist2` **mutates** `mylist1` to contain the concatenation of these two lists.

   For example, if `mylist1` is a list containing `[1, 2, 3]` and `mylist2` is a list containing `[4, 5, 6]`, and we evaluate the expression `mylist1 += mylist2`, then `mylist1` will become a list containing `[1, 2, 3, 4, 5, 6]`.

   Note: if n_1 is the number of items in `mylist1`, and n_2 is the number of items in `mylist2`, then `mylist1 += mylist2` should run in $O(n_1 + n_2)$

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.

   For example, if mylist1 is a list containing [1, 2, 3], mylist1[-1] is 3, mylist1[-2] is 2 and mylist1[-3] is 1.

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

e. Implement the __mul__ method for the MyList class, so that the expression mylist1 * n (where n is a positive integer) is evacuated to a **new** MyList object, which contains n copies of the elements in mylist1.

   For example, if mylist1 is a list containing [1, 2, 3], and we evaluate the expression mylist2 = mylist1 * 2, then mylist2 will be a new MyList object containing [1, 2, 3, 1, 2, 3].

   Note: This function should run in linear time, proportional to the length of the new list being created.

f. Implement the __rmul__ method to also allow the expression n * mylist1. The behavior of n * mylist1 should be equivalent to the behaviour of mylist1 * n.

g. Extra: Implement slicing for the MyList class.