- This lab will cover dynamic arrays (list) and run-time analysis of list methods.
- 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 the 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 do not have to stay for the duration of the lab if you finish early. 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. Ideally, you should not spend more time than suggested for each problem.
- Your TAs are available to answer questions in the lab, during office hours, and on Piazza.

Vitamins (45 minutes)

---

1. For each of the following $f(n)$, write out the summation results, and provide a tight bound $\Theta(f(n))$, using the Θ *notation* (5 minutes).

   Given $log(n)$ numbers, where *n* is a power of *2*:

   $1 + 2 + 4 + 8 + 16 \ldots + n =$ ____________ = $\Theta($ ___________ $)$

   $n + n/2 + n/4 + n/8 \ldots + 1 =$ ____________ = $\Theta($ ___________ $)$

   Provide a tight bound $\Theta(f(n))$, using the Θ *notation*

   $1 + 2 + 3 + 4 + 5 \ldots + \sqrt{n} =$ ____________ = $\Theta($ ___________ $)$

   $1 + 2 + 4 + 8 + 16 \ldots + \sqrt{n} =$ ____________ = $\Theta($ ___________ $)$

2. For each of the following code snippets, find $f(n)$for which the algorithm's time complexity is $\Theta(f(n))$in its **worst case** run and explain why. (10 minutes)

```
a. def func(lst):
        i = 1
        n = len(lst)
        while (i < n):
            print(lst[i])
            i *= 2
```

```
b. def func(lst):
        i = 1
        n = len(lst)
        while (i < n):
            print(lst)
            i *= 2
```

```
c. def func(lst):
        i = 1
        n = len(lst)
        while (i < n):
            print(lst[ : i])
            i *= 2
```

```
d. def func(lst):
        i = 1
        n = len(lst)
        while (i < n ** 2):
            print(lst)
            i *= 2
```

3. Give the **worst case** run-time for each of the following list methods. Write your answer in asymptotic notation in terms of n, the length of the list. Provide an appropriate summation for multiple calls. (25 minutes)

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

   What will be the **worst-case** run-time when calling the following for lst?

Method	1 Call	n Calls `for i in range(n): ...`
`append()`		What will be the total cost if lst = [ ] instead? Will the overall run-time change?
`insert(0, val)`		What will be the total cost if lst = [ ] instead? Will the overall run-time change?

Derive the **amortized cost** of a single call of append.

4. Given the following mystery functions: (5 minutes)
   - **i.** Replace `mystery` with an appropriate name (what does the function do?)
   - **ii.** Determine the function's **worst-case runtime** and **extra space usage** with respect to the input size.

a.
```
def mystery(n):
    lst = []
    for i in range(n):
        lst.insert(i, i)
```

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

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 /Content/Labs on NYU Brightspace

1. Extend the ArrayList class implemented during lecture with the following methods (note: for each of these methods, simulate the same behaviors as those of the built-in python list):

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. (10 minutes)

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

<u>Note</u>: Your implementation should create the string in $\Theta(n)$, where n = len(arr).

b. Implement the `__add__` method for the ArrayList class, so that the expression arr1 + arr2 is evaluated to a **new** ArrayList object representing the concatenation of these two lists. (10 minutes) (*think of this as a shallow concatenation of the lists*)

ex) `arr1` is an `ArrayList` with `[1, 2, 3]`
`arr2` is an `ArrayList` with `[4, 5, 6]`
→ `arr3 = arr1 + arr2`
`arr3` is a new `ArrayList` with `[1, 2, 3, 4, 5, 6]`.

<u>Note</u>: If n_1 is the size of arr1, and n_2 is the size of arr2, then __add__ should run in $\Theta(n_1 + n_2)$

c. Implement the `__iadd__` method for the ArrayList class, so that the expression arr1 += arr2 **mutates** arr1 to contain the concatenation of these two lists.
You may remember that this operation produces the same result as the **extend method**.

Your implementation should return *self*, which is the object being mutated. (10 minutes)

```
ex)   arr1 is an ArrayList with [1, 2, 3]
      arr2 is an ArrayList with [4, 5, 6]
 →    arr1 += arr2
      arr1 is mutated and now has [1, 2, 3, 4, 5, 6].
```

Note: If n_1 is the size of arr1, and n_2 is the size of arr2, 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. (20 minutes)

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

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

e. Implement the `__mul__` method for the ArrayList class, so that the expression arr1 * k (where k is a positive integer) creates a **new** ArrayList object, which contains k copies of the elements in arr1. (15 minutes)

```
ex)    arr1 is an ArrayList with [1, 2, 3]
 →     arr2 = arr1 * 2
       arr2 is a new ArrayList with [1, 2, 3, 1, 2, 3].
```

Note: If *n* is the size of arr1 and k is the int, then __mul__ should run in $\Theta(k * n)$.

f. Implement the `__rmul__` method to also allow the expression n * arr1. The behavior of n * arr1 should be equivalent to the behavior of arr1 * n. (5 minutes)

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

g. Modify the constructor `__init__` to include an option to pass in an iterable collection such as a string and return an ArrayList object containing each element of the collection. Do not account for dictionaries.(10 minutes)

```
ex)    arr = ArrayList("Python")
→      print(arr)  outputs ['P','y','t',h','o','n']

→      arr2 = ArrayList(range(10))
→      print(arr2) outputs [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

h. Implement a remove( ) method that will remove the first instance of val in the ArrayList. **You do not have to account for physically resizing the array for this question.** (20 minutes)

ex) `arr` is an `ArrayList` with `[1, 2, 3, 2, 3, 4, 2, 2]`

→ `arr.remove(2)`
→ `print(arr2)` outputs `[1, 3, 2, 3, 4, 2, 2]`

i. Implement a removeAll( ) method that will remove all instances of val in the ArrayList. The implementation should be in-place and maintain the relative order of the other values. It must also be done in $\Theta(n)$ **run-time**. **You do not have to account for physically resizing the array for this question.**

ex) `arr` is an `ArrayList` with `[1, 2, 3, 2, 3, 4, 2, 2]`

→ `arr.removeAll(2)`
→ `print(arr2)` outputs `[1, 3, 3, 4]`