- This lab will cover asymptotic analysis, and problem solving.
- It is assumed that you have reviewed chapter 3 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!
- If you finish early, you may help other students or leave early after showing your work to the TA. 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 Ed Discussion.

Vitamins (30 minutes)

For **big-O proof**, show that there exists constants c, and n_0 such that f(n) ≤ c*g(n) for every n ≥ n_0, then f(n) = O(g(n)).

For **big-Θproof**, show that there exists constants c1, c2, and n_0 such that c1*g(n) ≤ f(n) ≤ c2*g(n) for every n ≥ n_0, then f(n) = Θ(g(n)).

1. Use the **formal proof of big-O and big-Θ** in order to show the following (10 minutes):

   a) $n^2 + 5n - 2$ is $O(n^3)$ $< n^2 + 5n \leq n^2 + 5n^2 \leq 6n^2$

   b) $\frac{n^2 - 1}{n + 1}$ is $O(n)$ $< \frac{\cancel{(n+1)}(n-1)}{\cancel{n+1}} = n - 1 \leq n -$

   c) $\sqrt{5n^2 - 3n + 2}$ is $\Theta(n)$

$\sqrt{2n} \leq \sqrt{5n^2 - 3n} \leq \sqrt{5n^2 - n} < \sqrt{5n^2 - 3n + 2} \leq \sqrt{5n^2 + 3n + 2} \leq \sqrt{5n^4 + 3n^2}$

$\rightarrow \sqrt{n^2} = n$

2. State **True** or **False** and explain why for the following (10 minutes):

   a) $8n^2(\sqrt{n})$ is $O(n^3)$ True

   b) $8n^2(\sqrt{n})$ is $\Theta(n^3)$ False

3. Given the generator function, write the output: (10 minutes)

```
def sum_to(n): #also known as triangle numbers

    for i in range(1, n+1):
        total = i * (i + 1)//2
        yield total

for i in sum_to(10):
    print(i, end = ', ')
```

____

1, 3, 6, 10, 15, 21, 28, 36, 45, 55,

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code. **You can not use pop() or append() for any problem**.

1.
   a. Given a list of values (`int, float, str, …`), write a function that reverses its order in-place. You are <u>not allowed to create a new list</u>. Your solution must run in $\Theta(n)$, where n is the length of the list (10 minutes).

```
def reverse_list(lst):
      """
      : lst type: list[]
      : return type: None
      """
```

   b. Modify the function to include low and high parameters that represent the positive indices to consider. Your function should reverse the list from index low to index high, inclusively. By default, low and high will be None so these parameters are optional. If they're both None (no parameters passed), set low to 0 and high to len(lst) - 1 just like in the previous function above.

      You are <u>not allowed to create a new list</u>. Your solution must run in $\Theta(n)$, where n is the length of the list (5 minutes).

```
def reverse_list(lst, low = None, high = None):
    """
    : lst type: list[]
    : low, high type: int
    : return type: None
    """
```

Example:

lst = [1, 2, 3, 4, 5, 6], low = 0, high = 5
`reverse_list(lst) #default, no parameters passed`
`print(lst) → [6, 5, 4, 3, 2, 1]`

lst = [1, 2, 3, 4, 5, 6], low = 1, high = 3
`reverse_list(lst, 1, 3)`
`print(lst) → [1, 4, 3, 2, 5, 6]`

2. Define a generator that takes in a number n and returns the first n powers of 2:
   For example: powers_of_two(6) will yield 1, 2, 4, 8, 16, 32
   a. *def powers_of_two(n)*

3. Given a list of positive integers with zeros mixed in, write a function to move all zeros to the end of the list while maintaining the order of the non-zero numbers. For example, given the list [0, 1, 0, 3, 13, 0], the function will modify the list to become [1, 3, 13, 0, 0, 0]. Your solution must be in-place and run in $\Theta(n)$, where n is the length of the list. (25 minutes)

```
def move_zeros(nums):
    """
    : nums type: list[int]
    : return type: None
    """
```

Hint: You should traverse the list with 2 pointers, both starting from the beginning. One pointer will traverse through the entire list but when should the other pointer move?

4. Recall the following question from Homework 1 Question 2:

```
def shift(lst, k):
    """
    : lst type: list
    : k type: int
    : return type: None
    """
```

The function takes in a list and shifts it to the left by k steps.

ex) shift([1, 2, 3, 4, 5, 6], **2**) → [3, 4, 5, 6, **1, 2**]

In the homework, you probably solved it using the list methods, pop() and insert() to shift the list or manually shifted the list each time using an extra loop. Know that your homework solution was not linear if you used either of these methods. Since the run-time of the list methods have not been discussed at this point, **<u>do not</u> use any of the methods for this question**.

This time, you will attempt to solve this with run-time in mind. That is, your solution must run in Θ(n), where n is the length of the list (15 minutes).

The direction will also change so that the **shift function will shift to the right instead.**

ex) shift([1, 2, 3, 4, 5, 6], **2**) → [**5, 6,** 1, 2, 3, 4]

Hint: You should use the function reverse_list(lst, low, high) function from part 1b to solve this problem.

—------------------------------------ *optional* —------------------------------------

1. Sort the following 18 functions in an increasing asymptotic order and write < or <= between each two subsequent functions to indicate if the first is asymptotically less than or asymptotically equivalent to the second function respectively.

For example, if you were to sort: $f_1(n) = n$, $f_2(n) = log(n)$, $f_3(n) = 3n$, $f_4(n) = n^2$,

your answer could be $log(n) < (n <= 3n) < n^2$

Hint: Try grouping the functions like so: linear, quadratic, cubic, exponential … etc

$f_1(n) = n$

$f_2(n) = 500n$

$f_3(n) = \sqrt{n}$

$f_4(n) = log(\sqrt{n})$

$f_5(n) = \sqrt{log(n)}$

$f_6(n) = 1$

$f_7(n) = 3^n$

$f_8(n) = n \cdot log(n)$

$f_9(n) = \frac{n}{log(n)}$

$f_{10}(n) = 700$

$f_{11}(n) = log(n)$

$f_{12}(n) = \sqrt{9n}$

$f_{13}(n) = 2^n$

$f_{14}(n) = n^2$

$f_{15}(n) = n^3$

$f_{16}(n) = \frac{n}{3}$

$f_{17}(n) = \sqrt{n^3}$

$f_{18}(n) = n!$