- This lab will focus on more 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 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 40 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*.

   Given n numbers:

   $1 + 1 + 1 + 1 + 1 \ldots + 1 =$ _____ $= \Theta($ _____ $)$

   $n + n + n + n + n \ldots + n =$ _____ $= \Theta($ _____ $)$

   $1 + 2 + 3 + 4 + 5 \ldots + n =$ _____ $= \Theta($ _____ $)$

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.

a)
```
def func(lst):
    for i in range(len(lst)):
        if (lst[i] % 2 == 0):
            print("Found an even number!")
            return
```

b)
```
def func(lst):
    for i in range(len(lst)):
        if (lst[i] % 2 == 0):
            print("Found an even number!")
        else:
            print("No luck.")
            return
```

c)
```
def func(lst):
    for i in range(0, len(lst), 2):
        for j in range(i):
            print(lst[j], end = " ")
```

d)
```
def func(n):
    for i in range(n):
        j = 1
        while j <= 80:
            print("i = ",i,", j =", j)
            j *= 2
```

e)
```
def func(n):
    for i in range(n):
        j = 1
        while j <= n:
            print("i = ",i,", j =", j)
            j *= 2
```

3. Sort the following 18 functions in an increasing asymptotic order and write `<, <=,` between each two subsequent functions to indicate if the first is asymptotically less than, asymptotically greater 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 $log(n) < (n <= 3n) < n^2$

$f_1(n) = n$
$f_2(n) = 500n$
$f_3(n) = \sqrt{n}$
$f_4(n) = log(\sqrt{n})$
$f_5(n) = \sqrt{log(n)}$
$f_6(n) = 1$
$f_7(n) = 3^n$
$f_8(n) = n \cdot log(n)$
$f_9(n) = \frac{n}{log(n)}$
$f_{10}(n) = 700$
$f_{11}(n) = log(n)$
$f_{12}(n) = \sqrt{9n}$
$f_{13}(n) = 2^n$
$f_{14}(n) = n^2$
$f_{15}(n) = n^3$
$f_{16}(n) = \frac{n}{3}$
$f_{17}(n) = \sqrt{n^3}$
$f_{18}(n) = n!$

Coding

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

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

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

2. Given a string, write a function that returns True, if it is a palindrome, and False, if not. A string is a palindrome if its characters are read the same backwards and forwards.

   For example, `"1racecar1"` is a palindrome but `"1racecar2"` is not. You are not allowed to create a new list or use any str methods. Your solution must run in $\Theta(n)$, where n is the length of the input string.

```
def is_palindrome(s):
    """
    : s type: str
    : return type: bool
    """
```

3. Given a **sorted** 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.

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

4.

a. The function below takes in a **sorted** list with n numbers, all taken from the range 0 to n, with one of the numbers removed. Also, none of the numbers in the list is repeated. The function searches through the list and returns the missing number.

   For instance, lst = [0, 1, 2, 3, 4, 5, 6, 8] is a list of 8 numbers, from the range 0 to 8, with the number 7 missing. Therefore, the function below will return 7.

   Analyze the worst case run-time of the function:

```
def find_missing(lst):

      for num in range(len(lst) + 1):
            if num not in lst:
                  return num
```

b. Rewrite the function so that it finds the missing number with a better run-time: **Hint:** the list is sorted. Also, make sure to consider the edge cases.

```
def find_missing(lst):
      """
      : nums type: list[int] (sorted)
      : return type: int
      """
```

c. Suppose the given list is **not sorted** but still contains all the numbers from 0 to n with one missing.

   For instance, lst = [8, 6, 0, 4, 3, 5, 1, 2] is a list of numbers from the range 0 to 8, with the number 7 missing. Therefore, the function below will return 7.

   How would you solve this problem? Do not use the idea in step a, or sort the list and reuse your solution in step b.

```
def find_missing(lst):
      """
      : nums type: list[int] (unsorted)
      : return type: int
      """
```