- This lab will cover Asymptotic Analysis in more detail.
- 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. 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. 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. Show your work.

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!")
            return
        else:
            print("No luck.")
            return
```

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

d)
```
def func(n):
    for i in range(n):
        j = i
        while j >= 0:
            print("i =",i,", j =",j)
            j -= 2
```

Coding

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

1. Write a function that takes in a string as input and reverse only the vowels of the string. For example, an input of “tandon” would return “tondan”. Your function must run in $\Theta(n)$ and you may assume all strings contain only lowercase characters.

   Hint: you may want to use the .join() string method, which is guaranteed to run in linear time.

   ```
   def reverse_vowels(input_str):
       """
       : input_str type: string
       : return type: string
   ```

2. Write a python function that takes in 2 **sorted** lists of numbers and returns the intersection list, that is a list that contains the elements that show in both input lists. Aim for linear time complexity. That is if the first input list has n elements, and the second has m elements, the runtime should be $\theta(n+m)$

   For example, if lst_A = [1,6,14,15] and lst_B = [2, 6, 14, 19], the returned list should be: [6,14]

3.
   a. Implement a function that calculates an approximation of the square root of a number with two-decimal points accuracy that runs in $\Theta(\sqrt{n})$.

   ```
   def square_root(num):
   """
   : num type: positive int
   : return type: float
   """
   ```

   b. Implement the square root function that runs in $\Theta(\log n)$.

4.

a. Consider the `Polynomial` class that you wrote during Lab 1. Think about how you might optimize the runtime of the following methods. What is the best runtime that you can come up with in terms of Θ for an input of size n?

   the `__init__` method

   the `__repr__` method

   the `eval` method

   the `__add__` method

   the `__mul__` method

b. Using the `Polynomial` class you wrote during Lab 1 and the `PolyTimer` class that you used during Lab 2, time the `eval` method for polynomials with 100, 200, 500, 1000, and 10000 coefficients, all randomly generated. Export the timings you get to a .csv file with each stringified Polynomial and its running time.

c. Extra: Time the other polynomial methods using the polynomials generated in part a. Export the timings in the same manner with both stringified operands and the timing results.