- This lab will cover recursion.
- It is assumed that you have reviewed chapters 4 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 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. Ideally you should not spend more time than suggested for each problem.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (20 minutes)

---

1. For each of the following code snippets:
    a. Given the following inputs, trace the execution of each code snippet. Write down all outputs in order and what the functions return.

    b. Analyze the running time of each. For each snippet:
        i. Draw the recursion tree that represents the execution process of the function, and the cost of each call
        ii. Conclude the total (asymptotic) run-time of the function.

a.
```
def func1(n):          #for tracing: n = 16
    if (n <= 1):
        return 0
    else:
        return 10 + func1(n-2)
```

b.
```
def func2(n):          #for tracing: n = 16
    if (n <= 1):
        return 1
    else:
        return 1 + func2(n//2)
```

```
#for tracing: lst = [1, 2, 3, 4, 5, 6, 7, 8]
```
c.
```
def func3(lst):
    if (len(lst) == 1):
        return lst[0]
    else:
        return lst[0] + func3(lst[1:])
```

Coding

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

1. Write a **recursive** function that returns the **sum** of all numbers from 0 to n. (5 minutes)

```
def sum_to(n):
    """
    : n type: int
    : return type: int
    """
```

2. Write a **recursive** function that returns the product of all the even numbers from 1 to n. (5 minutes)

   ex) if n = 8 → 2 * 4 * 6 * 8 = 384, so the function returns 384.

```
def product_evens(n):
    """
    : n type: int
    : return type: int
    """
```

3. Write a **recursive** function to find the maximum element in a **non-empty**, **non-sorted** list of numbers. ex) if the input list is [13, 9, 16, 3, 4, 2], the function will return 16.

   a. Determine the runtime of the following implementation. (7 minutes)

```
def find_max(lst):
    if len(lst) == 1:
        return lst[0]      #base case
    prev = find_max(lst[1:])
    if prev > lst[0]:
        return prev
    return lst[0]
```

b. Update the function parameters to include low and high. Low and high are int values that are used to determine the range of indices to consider. Implementation run-time must be linear. (10 minutes)

```
def find_max(lst, low, high):
      """
      : lst type: list[int]
      : low, high type: int
      : return type: int
      """
```

4. In lab 3, you wrote an iterative implementation that determines whether a string is a palindrome (characters are read the same backward and forward). This time, you will do this **recursively**. The function parameters, low and high, are int values that are used to determine the range of indices to consider. Implementation run-time must be linear. (10 minutes)

   ex) is_palindrome("racecar", 0, 6) returns True #racecar
   is_palindrome("racecar", 1, 5) returns True #aceca
   is_palindrome("racecar", 1, 3) returns False #ace

```
def is_palindrome(input_str, low, high):
    """
    : input_str type: str
    : low, high type: int
    : return type: bool
    """
```

5. Give a **recursive** implementation for the binary search algorithm. The function is given a **sorted** list, lst, a value, val, to search for, and two indices, low and high, representing the lower and upper bounds to consider. (20 minutes)

   If the value is on the list (between low and high), return the index at which the value is located. If val is not found, the function should return None.

```
def binary_search(lst, low, high, val):
    """
    : lst type: list[int]
    : val type: int
    : low type, high type: int
    : return type: int (found), None
    """
```

6. Write a **recursive** function that takes in a list of integers and **mutates** it so that all the even numbers are at the front and all the odd numbers are in the back. You do not have to maintain the relative order; just focus on separating them. You are also given low and high, the range of indices to consider. Implementation run-time must be linear.
(15 minutes)

ex) lst = [4, -5, 2, 3, -1, -6, 7, 9, 0]
split_parity(lst) → [4, 0, 2, -6, -1, 3, 7, 9, -5]

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

7. Given a string of letters representing a word(s), write a **recursive** function that returns a **tuple** of 2 integers: the number of vowels, and the number of consonants in the word.

   Remember that tuples are not mutable so you'll have to create a new one to return each time when you're updating the counts. Since we are always creating a tuple of 2 integers each time, the cost is constant. Implementation run-time must be linear. (25 minutes)

ex) word = "NYUTandonEngineering"
vc_count(word, 0, len(word)-1) → (8, 12) # 8 vowels, 12 consonants

```
def vc_count(word, low, high):
    """
    : word type: str
    : low, high type: int
    : return type: tuple (int, int)
    """
```

8. **OPTIONAL**

A nested list of integers is a list that stores integers in some hierarchy. The list can contain integers and other nested lists of integers. An example of a nested list of integers is [[1, 2], 3, [4, [5, 6, [7], 8]]]. (30 minutes)
Write a **recursive** function to find the total sum of a nested list of integers.

ex) If lst = [[1, 2], 3, [4, [5, 6, [7], 8]]], the function should return 36.

```
def nested_sum(lst):
    """
    : lst type: list
    : output type: int
    """
```

Note:
To check the type of an object, use the isinstance function. You may use for loops inside your function. No run-time requirement.

ex) lst = [1, 2, 3, 4]
if isinstance(lst, list): #returns True
if isinstance(lst, int): #returns False

9. **OPTIONAL**

Given a list of integers, write a function to print the sum triangle of the array. Each row in the sum triangle is defined as the sum of each number and the one exactly after it.

Note that if there is no number after it, this number is removed. That is, there will always be one fewer number after each row.

Example, if `lst = [1, 2, 3, 4, 5]`, the sum triangle will be printed as followed:

```
[48]            ← #[20+28]
[20, 28]        ← #[8+12, 12+16]
[8, 12, 16]     ← #[3+5, 5+7, 7+9]
[3, 5, 7, 9]    ← #[1+2, 2+3, 3+4, 4+5]
[1, 2, 3, 4, 5] ← starting list
```

```
def list_sum_triangle(lst):
    """
    : lst type: list
    : output type: None
    """
```

You may write additional recursive functions to solve the problem. The sum triangle must still be a recursive function while calling your helper recursive functions.

Do not use any while or for loops. No run-time requirement.