- This lab will review Copying, recursion and Dynamic Arrays.
- When approaching the problems, think before you code. Doing so is good practice and can help you layout 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 the lab, during office hours, and on Piazza.

Submission Instructions

- Submit the completed lab on gradescope by Saturday, 18th July 2020, 11:55pm.
- Submit one pdf file with all your answers to the vitamin section in it. And submit a yourNetID_Lab2.py file with all the solutions to the coding questions in it.

Vitamins

---

1. For each section below, write the correct output shown after the Python code is run. Explain your answer by **drawing the memory image** for the execution of these lines of code. That is, you should draw the variables as they are organized in the call stack, and the data they each point to.

A.
```
import copy
lst = [1, 2, [3, 4]]
lst_copy = copy.copy(lst)
lst_copy[0] = 10
lst_copy[2][0] = 30
print(lst)

______________________________________

print(lst_copy)

______________________________________
```

B.
```
import copy
lst = [1, [2, "abc"], [3, [4]], 7]
lst_deepcopy = copy.deepcopy(lst)
lst[0] = 10
lst[1][1] = "ABC"
lst_deepcopy[2][1][0] = 40
```

```
print(lst)

________________________________________
print(lst_deepcopy)

________________________________________
```

```
C. lst = [1, [2, 3], ["a", "b"] ]
   lst_slice = lst[:]
   lst_assign = lst
   lst.append("c")
   for i in range(1, 3):
   lst_slice[i][0] *= 2

   print(lst)

   _______________________________________

   print(lst_slice)

   ________________________________________

   print(lst_assign)

   ________________________________________
```

2. Analyze the running time of each. For each snippet:
   - Draw the recursion tree that represents the execution process of the function, and the cost of each call
   - 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. This will be good practice for when you write code by hand on the exams.

1. 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. (30 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)
            """
```

   Hint: A string can be converted into a list using the list constructor e.g. list("abc") gives ["a", "b", "c"].

2. 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
```

3. Refer to the Dynamic_Array_Students.py and complete the tasks 6-10.