- This lab will cover Recursion.
- It is assumed that you have reviewed chapter 4 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 45 minutes)

---

1. For each of the following code snippets:
    a. If `lst=[1,2,3,4,5,6,7,8,9,10]` and `n = 4` trace the execution of each code snippet. Write down all outputs in order and what the functions returns.
    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) running time of the function

```
a.
def f(n):
    if (n <= 1):
        return 0
    else:
        return 10 + f(n-2)

b.
def f(n):
    if (n <= 1):
       return 1
    else:
       return 1 + f(n/2)

c.
def f(ls):
    if (len(ls) == 1):
        return ls[0]
    else:
        return ls[0] + f(ls[1:])
```

2. You are given 2 implementations for a recursive algorithm that calculates the sum of all of the elements in a list (of integers):

```
def sum_lst1(lst, low, high):
    print("sum_lst1(", low, ", ", high, ")", sep='')
    if (low == high):
        return lst[high]
    else:
        rest = sum_lst1(lst, low+1, high)
        sum = lst[low] + rest
        return sum


def sum_lst2(lst, low, high):
    print("sum_lst2(", low, ", ", high, ")", sep='')
    if (low == high):
        return lst[high]
    else:
        mid = (low + high)//2
        part1 = sum_lst2(lst, low, mid)
        part2 = sum_lst2(lst, mid+1, high)
        sum = part1 + part2
        return sum
```

Note: The print statements in the code above is for tracing purposes (it is not significant for calculating the sum).

   a. Make sure you understand the recursive idea of each implementation.

   b. If `lst=[2, 4, 6, 8, 10, 12, 14, 16]`, trace the execution of calling each version above to calculate the sum of the elements of lst (the initial call passes `low=0` and `high=7`). Write down all outputs in order and what the functions returns.

   c. Analyze the running time of the implementations above. For each version:
      i. Draw the recursion tree that represents the execution process of the function, and the cost of each call
      ii. Conclude the total (asymptotic) running time of the function

   d. Which version is asymptotically faster?

Coding

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

1. Write a **recursive** function to find the maximum element in a list of numbers.
   For example, if the input list is `[1,2,3,4,5,100,12,2]`, the function should return 100.

```
def find_lst_max(lst):
    """
    : input_str type: lst
    : output type: int
    """
```

2. Write a **recursive** function that takes a list as input and returns the product of even numbers less than or equal to *n* (the number of elements in the list). For example, if the list is `[1,2,3,4,5,100,12,2], n = 8 and 2 * 4 * 2 = 16 so the function would return the value 16.`

```
def product_evens(lst):
    """
    : input_str type: lst
    : output type: int
    """
```

3. Give a **recursive** implementation of a function that checks whether a word is a palindrome. A palindrome is a word that is read the same backward or forward. The function is given a string, `input_str`, and two indices: `low` and `high` (`low` ≤ `high`), which indicate the range of indices that needs to be considered. The function should return a boolean that indicates whether the substring between `low` and `high` positions is a palindrome. If the string is a palindrome, the function will return `True`, if it is not a palindrome, the function will return `False`.

   For example, if the `input_str = "kayak"` the function will return `True`. However, if the `input_str = "python"` the function will return `False`.

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

4. Give a **recursive** implementation for the binary search algorithm. The function is given a sorted list, `srt_lst`, a value `val` to search for, and two indices , `low` and `high`, representing the boundaries (in `srt_lst`) where we need to search.
   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(srt_lst, val, low, high):
      """
      : srt_lst type: list[int]
      : val type: int
      : low type: int
      : high type: int
      """
```

3. Modern operating systems define file-system directories (which are also sometimes called “folders”) in a recursive way. Namely, a file system consists of a top-level directory, and the contents of this directory consists of files and/or other directories, which in turn can contain files and/or other directories, and so on. Given the recursive nature of the file-system representation, it should not come as a surprise that many common behaviors of an operating system, such as copying a directory or deleting a directory, are implemented with recursive algorithms.

   To help us implement common behaviors of an operating system, we rely on Python’s `os module`, which provides robust tools for interacting with the operating system during the execution of a program.

   Some of the functions include:

   - **`os.path.getsize(path)`**
     Return the immediate disk usage (measured in bytes) for the file or directory that is identified by the string path (e.g., /user/rt/courses).

   - **`os.path.isdir(path)`**
     Return `True` if entry designated by string path is a directory; `False` otherwise

   - **`os.listdir(path)`**
     Return a list of strings that are the names of all entries within a directory designated by string path. For example, if the parameter is /user/rt/courses, this can return `[‘cs016’, ‘cs252’]`

   - **`os.path.join(path, filename)`**
     Compose the path string and filename string using an appropriate operating

system separator between the two (e.g., the / character for a Unix/Linux/OSX system, and the \ character for Windows). Return the string that represents the full path to the file.

Give a **recursive** implementation for calculating the disk usage of a folder. The function is given a string, `path`, representing the location of the folder. The cumulative disk space is equal to the immediate disk space used by the entry plus the sum of the cumulative disk space usage of any entries that are stored directly within the entry.

For example, given the diagram below, the cumulative disk space for python is 15K because it uses 1K itself (immediate disk space), 11K cumulatively in labs and 3K cumulatively in hw.

```
def disk_usage(path):
    """
    : path type: str
    : return value type: int
    """
```