Homework Set 1
CSCI 204.01
Prof Meng

Assigned: Monday, 02/10/2020
Due: Monday, 02/17/2020

Please complete the following homework. While discussions are encouraged, everyone must complete their work themselves. Submit your work to Moodle with one single file, either in PDF, or in Word by the deadline.

1. Given two Python lists, one is a collection of city names **city_name[]**, the other is a collection of the population in these cities **city_pop[]**. Create a dictionary **d** that uses the city name as its key and the population as its value.
2. Given the following partial class definitions for **College** and for **Major** of various majors in a college, write a class function to compute the total number of students in the college.

```
class Major:
   def __init__(self, major_name, major_count):
      self.name = major_name    # name of the major, e.g., 'CS'
      self.count = major_count  # number of students in the major
…
class College:
   def __init__(self, majors):
       self.majors = []                 # self.majors is a list
       for m in range(len(majors):
           self.majors.append(majors[m])# each is a major object
```

3. Given the following Python code segments, identify the complexity using the big-Oh notation. Explain briefly your answers.

```
   a. # search for a name in a list
      i = 0
      while (i < len(name_list) and name_list[i] != name):
            i += 1
   b. # selection sort, my_nums is a list of integers
      for k in range(len(my_nums)):
           min = my_nums[k]
           min_index = k
           for m in range(k+1, len(my_nums)):
                if  my_nums[m] < min:
                     min = my_nums[m]
                    min_index = m
        swap(my_nums, k, min_index)    # swap my_nums[k] and
      my_nums[min_index]
   c. # some recursion, try out some small numbers first
      def  fun(n):
         if n <= 0:
              return n
         else:
              return fun(n-2) + 2*fun(n-1)
```

4. We studied various recursive programs. One of the well-known example is the Tower of Hanoi. A sample code for Tower of Hanoi is as follows.

```
def tower_of_hanoi( n, src, dest, helper ):

   if n == 1:
       print( 'moving ', n, ' from ', src, ' to ', dest )
   else:
       tower_of_hanoi( n-1, src, helper, dest )
       tower_of_hanoi( 1, src, dest, helper )
       tower_of_hanoi( n-1, helper, dest, src )
```

   Try out the program with n = 3, 6, and 10. That is

```
tower_of_hanoi(3, 0, 1, 2)
tower_of_hanoi(6, 0, 1, 2)
tower_of_hanoi(10, 0, 1, 2)
```

   and count how many lines of output each of the execution generates. Using this to estimate the complexity of the program, i.e., O(f(n)), what is f(n) in this case?

   The easiest way of estimate the lines of output is to run at the Linux command line

```
% python tower_of_hanoi(3, 0, 1, 2) | wc
% python tower_of_hanoi(3, 0, 1, 2) | wc
% python tower_of_hanoi(3, 0, 1, 2) | wc
```

   Here **wc** is a Linux command that counts the number of lines. You can also do it within **spyder** or **idle**. You can copy and paste the output to a text file and then find out how many lines of output.
5. Write a function to compute the sum from **1** to **n** using recursion.
6. Write a recursive function to return a list consisting of multiples of 4 for a given parameter **n**. For example, **multiple_of_four(3)** returns **[4, 8, 12]**, **multiple_of_four(5)** returns **[4, 8, 12, 16, 20]**.
7. Given the following recursive function, show what is printed and explain briefly why.

```
def fun_six ( counter ):
      if counter ==0:
            return
      else :
            print ('Before '+str( counter ))
            fun_six ( counter - 1)
            print ('After '+str( counter ))
```

8. Given a singly linked list definition, **UserList** and **ListNode**, as discussed in the lectures, write an overloading function **`__add__()`** for the UserList that allows the use of **`list = list + node`**
9. For the same singly linked list definition in Problem 7, write an overloading function **`__len__()`** that returns the length of the list.
10. Write a function to delete a node from a doubly linked list. (The topic of doubly linked list will be discussed on Wednesday 2/12.)