CS 115 Exam 3, Fall 2016, Sections 5-8

Your name: ______________________________________

Rules

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only resource you may consult during this exam.
- Explain/show work if you want to receive partial credit for wrong answers.
- As long as your code is correct, you will get full credit. No points for style.
- When you write code, be sure that the indentation level of each statement is clear.

	Your Score	**Max Score**
Problem 1: Binary search		10
Problem 2: Selection sort		10
Problem 3: Mergesort		10
Problem 4: Recursion		20
Problem 5: Defining classes		20
Problem 6: Using classes		30
Total		**100**

Reference code for Problems 1 and 2

The functions below are just for your reference on Problems 1 and 2. You do not need to read them if you understand the algorithms.

```
def binary_search(search_list, value_to_find):
      first = 0
      last = len(search_list) - 1

      while first <= last:
           middle = (first + last) // 2
           # Problem 1: state the values of first, last,
           # and middle at this point in the code
           if value_to_find == search_list[middle]:
                return middle
           elif value_to_find < search_list[middle]:
                last = middle - 1
           else:
                first = middle + 1
      return None
```

```
def selection_sort(list_to_sort):
  for i in range(len(list_to_sort) - 1):
    min_index = find_min_index(list_to_sort, i)
    list_to_sort[i], list_to_sort[min_index] =
               list_to_sort[min_index], list_to_sort[i]
    # Problem 2: Show list contents at this point


def find_min_index(L, s):
     min_index = s
     for i  in range(s, len(L)):
        if L[i] < L[min_index]:
            min_index = i
    return min_index
```

Reference code for Problem 3

The functions below are just for your reference on Problem 3. You do not need to read them if you understand the algorithms.

```
def merge(L, start_index, sublist_size):
    index_left = start_index
    left_stop_index = start_index + sublist_size
    index_right = start_index + sublist_size
    right_stop_index = min(start_index + 2 * sublist_size,
                           len(L))
    L_tmp = []

    while (index_left < left_stop_index and
           index_right < right_stop_index):
        if L[index_left] < L[index_right]:
           L_tmp.append(L[index_left])
           index_left += 1
        else:
           L_tmp.append(L[index_right])
           index_right += 1

    if index_left < left_stop_index:
           L_tmp.extend(L[index_left : left_stop_index])
    if index_right < right_stop_index:
           L_tmp.extend(L[index_right : right_stop_index])

    L[start_index : right_stop_index] = L_tmp

def merge_sort(L):
    chunksize = 1
    while chunksize < len(L):
        left_start_index = 0  # Start of left chunk in each pair
        while left_start_index + chunksize < len(L):
            merge(L, left_start_index, chunksize)
            left_start_index += 2 * chunksize

        chunksize *= 2
        # Problem 3: Show list contents at this point
```

Problem 1: Binary search (10 points)

Consider the following sorted list:

```
L = [   'append',
        'computer',
        'deck',
        'discard',
        'game',
        'hand',
        'pop',
        'racko' ]
```

and the binary search code on page 2. You may want to label the elements of L with their numeric index values before proceeding.

(a) Fill out the following table tracing the call `v = binary_search(L, 'deck')`, a binary search for `'deck'` in this list, according to the comment in the code. ***You should fill out one row per iteration of the loop***. If there are more rows than iterations, leave the extra rows blank. At the end, write the value `v` returned by the function

Iteration	Value of `first`	Value of `last`	Value of `middle`	Value of `L[middle]`
1				
2				
3				
4				

Return value `v`:

(b) Fill out the following table tracing call to `v = binary_search(L, 'human')`, a binary search for `'human'` in this list. At the end, write the value `v` returned by the function

Iteration	Value of `first`	Value of `last`	Value of `middle`	Value of `L[middle]`
1				
2				
3				
4				

Return value `v`:

Problem 2: Selection sort (10 points)

Consider the following list:

```
L = [   'pop',
        'game',
        'hand',
        'deck',
        'racko',
        'append',
        'discard',
        'computer']
```

In the diagram below, show the contents of the list after each of the first 4 iterations of the for-loop in `selection_sort`.

INDEX	INITIAL ORDER	AFTER i=0 ITERATION	AFTER i=1 ITERATION	AFTER i=2 ITERATION	AFTER i=3 ITERATION
0	pop				
1	game				
2	hand				
3	deck				
4	racko				
5	append				
6	discard				
7	computer				

Problem 3: Mergesort (10 points)

Consider the following list:

```
L = [   'pop',
        'game',
        'deck',
        'hand',
        'racko',
        'append',
        'discard',
        'computer']
```

In the diagram below, show the contents of the list after each of the first 3 iterations of the outer while-loop in `merge_sort`.

INDEX	INITIAL ORDER	AFTER chunksize=1 ITERATION	AFTER chunksize=2 ITERATION	AFTER chunksize=4 ITERATION
0	pop			
1	game			
2	deck			
3	hand			
4	racko			
5	append			
6	discard			
7	computer			

Problem 4a: Recursion (10 points)

Consider the following function definition:

```
def fun(s):
    # parameter s is a string

    if len(s) <= 1:
        return s
    else:
        return fun(s[1:]) + s[0]
```

A. What does the following snippet of code return?

```
     fun('o')
```

B. Show the chain of recursive calls, and state what the return value is for each call, starting with:

```
     fun('racko')
```

C. How would you summarize what this function does in one sentence? Don't explain the code line-by-line. Provide a higher-level description like "adds *x* and *y*" or "computes *x* factorial."

Problem 4b: Recursion (10 points)

Consider the following function definition:

```
def func(L) :
    # parameter L is a list
    print("Current list is", L)

    if len(L) == 1:
        print("Base case! Returning", L[0] % 2)
        return L[0] % 2   #Gives remainder on dividing by 2

    val = func(L[1:])
    print("Adding return value", val, "to", L[0] % 2)
    return val + L[0] % 2
```

Specify the output obtained upon executing the following statement:

```
print ("Final answer is", func([11, 24, 7]))
```

Problem 5: Defining classes (20 points)

In this problem, you will define a class to represent profile of a `Facebook Member`. Your class should be named `FacebookProfile`, and you should define the following methods:

`__init__`: This method initializes an `FacebookProfile` object. Initialize the attributes to store the member's *name* (**eg**. Tim Baker) and *status* (**eg**. idle). There is another attribute to store a *list of friends names* but it is initially an empty list.

`get_name`: This method returns the member's name

`get_status`: This method returns the member's status

`get_num_friends`: This methods returns the number of friends of this member

`__str__`: This method returns a string with the `FacebookProfile` object's attributes, formatted as follows:

```
  Tim Baker (status: idle) has 27 friends
```

The above output is just an example: you should use the actual values in place of values that are underlined.

`__lt__`: This method compares `self` to another `FacebookProfile` object. It returns `True` if the `self` object has less number of friends than another `FacebookProfile` object, and `False` otherwise

`add_friend`: This method adds a new friend's name to the *list of friends names* of the current `FacebookProfile` object (`self`), where the name of the new friend is a parameter to the method

`update_status`: This method updates the status of the current `FacebookProfile` object (`self`), where the new status is a parameter to the method

[Write code in next page]

[WRITE YOUR PROBLEM 5 CODE HERE]

The last page of this exam has extra space for you to write your solution.

Problem 6: Using classes (30 points)

For this problem, you must write a **complete program**. However, you can assume that the `FacebookProfile` class from Problem 5 has already been correctly defined for you.

To earn full credit, you must **use the methods** of the `FacebookProfile` class whenever possible.

Read the instructions carefully before you start coding!

Your program should do the following:

1. A function called `CreateProfile` to do the following:
   - Ask the user to enter the member's name and status on two lines. For example:
     ```
     Tim Baker
     idle
     ```
   - If the member's status is "`sleepy`", **return** None
   - Create a `FacebookProfile` object that uses the information the user entered (name, status).
   - Now prompt the user to enter the number of friends of this member.
   - For each friend, ask the user their name and add them to the friend list of `FacebookProfile` object just created.
   - Return the `FacebookProfile` object.

2. A main function to do the following:
   - Call `CreateProfile` repeatedly until the member's status is "`sleepy`".
   - Use the results of `CreateProfile` to build a list of Facebook member profiles.
   - *After* creating the list, use the methods of the `FacebookProfile` class to find
     - Members who have more than 100 friends and print the status of each such member.
     - The member with the least number of friends. Call it *minFr*
     - The member with the maximum number of friends. Call it *maxFr*
   - Update *minFr*'s status to "`friend me`". Print *minFr's* profile information summarizing their name, status and number of friends.
   - Add *minFr* and *maxFr* to each other's friend's list.
   - Update *maxFr*'s status to "`made a new friend`". Print *maxFr's* profile information summarizing their name, status and number of friends.

The last page of this exam has extra space for you to write your solution.

[WRITE YOUR PROBLEM 6 CODE HERE]

[EXTRA SPACE FOR PROBLEMS 5 AND 6]

[EXTRA SPACE FOR PROBLEMS 5 AND 6]