CS 115 Exam 3, Spring 2015, 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		15
Problem 2: Selection sort		10
Problem 3: Mergesort		10
Problem 4: Recursion		15
Problem 5: Defining classes		30
Problem 6: Using classes		25
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
```

```
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 (15 points)

Consider the following sorted list:

```
L = [   'black',
        'blue',
        'green',
        'orange',
        'purple',
        'red',
        'white',
        'yellow' ]
```

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 a binary search for `'white'` 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.

Iteration	Value of `first`	Value of `last`	Value of `middle`	Value of `L[middle]`
1				
2				
3				
4				
5				

(b) Fill out the following table tracing a binary search for `'brown'` in this list.

Iteration	Value of `first`	Value of `last`	Value of `middle`	Value of `L[middle]`
1				
2				
3				
4				
5				

Problem 1 continued

If we doubled the number of elements of the list L, how would you expect the worst-case number of binary search loop iterations to change (circle one)?

A. No change.

B. It would increase by one.

C. It would increase by two.

D. It would double.

E. It would quadruple.

Problem 2: Selection sort (10 points)

Consider the following list:

```
L = [   'red',
        'orange',
        'yellow',
        'green',
        'blue',
        'purple',
        'white',
        'black'     ]
```

In the diagrams below, show the contents of the list after each of the first 4 iterations of the for-loop in `selection_sort`. If the list does not change from one iteration to the next, you can write "SAME" for the next iteration.

INDEX	INITIAL ORDER	AFTER i=0 ITERATION	AFTER i=1	AFTER i=2	AFTER i=3
0	red				
1	orange				
2	yellow				
3	green				
4	blue				
5	purple				
6	white				
7	black				

Problem 3: Mergesort (10 points)

Consider the following list:

```
L = [   'red',
        'orange',
        'yellow',
        'green',
        'blue',
        'purple',
        'white',
        'black'     ]
```

In the diagrams below, show the contents of the list after each of the first 4 iterations of the outer while-loop in `merge_sort`. If the list does not change from one iteration to the next, you can write "SAME" for the next iteration.

INDEX	INITIAL ORDER	AFTER chunksize=1 ITERATION	AFTER chunksize=2 ITERATION	AFTER chunksize=4 ITERATION
0	red			
1	orange			
2	yellow			
3	green			
4	blue			
5	purple			
6	white			
7	black			

Problem 4: Recursion (15 points)

Consider the following function definition:

```
def rec(L, start):
    # parameter L is a list of numbers
    # parameter start is a number
    if len(L) - start <= 1:
        return True
    if L[start] > L[start + 1]: return False
    return rec(L, start + 1)
```

A. What does the following snippet of code print?

```
L = [2]
print(rec(L, 0))
```

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

```
L = [1, 2, 3, 4, 5]
rec(L, 0)
```

C. How would you summarize what this function does in just a few words, if you always pass a value of 0 for *start* in the initial call to *rec*?

Don't explain the code line-by-line. Provide a higher-level description like "adds *x* and *y*" or "computes *x* factorial."

Problem 5: Defining classes (25 points)

In this problem, you will define a class to represent a rectangle.

If you use the input() or print() functions in your solution to this problem, you're doing it wrong!

Your class should be named `Rectangle`, and you should define the following methods:

`__init__`: This method initializes a `Rectangle` object, given a length and width. You can assume that the length and the width are numbers. However, if the length (or width) parameter is negative, you should initialize the object's length (or width) to 0.

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

```
Length = 5 and Width = 10
```

This example assumes that the length and width are 5 and 10, but you should use the actual values.

`__lt__`: Returns `True` if this `Rectangle` object's area is less than another `Rectangle` object's area, and `False` otherwise

`contains`: Compares this Rectangle object and another Rectangle object. Returns `True` if this `Rectangle` object's length is larger than the length of the other object AND this `Rectangle` object's width is larger than the width of the other object. Otherwise, returns `False`.

`area`: Returns the area of the rectangle.

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

Problem 6: Using classes (25 points)

For this problem, you must write a **complete program**. However, you can assume that the following pieces of code will be cut and pasted into your program:

- A correctly working implementation of the class described in Problem 5
- The definition of a `readfile` function, similar to the ones you used in the labs, that takes a string (filename) as a parameter and returns a list of strings (the lines read from the file)

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

Read the instructions carefully before you start coding!

Your program should do the following.

- Ask the user how many rectangles to read. You can assume they enter a positive integer.
- Open up the file rectangles.txt, and read the lengths and widths of the user-specified number of rectangles. You may assume that the file contains one valid number per line (a length, then a width, etc.) and contains enough numbers.
- Print the area of the largest rectangle, in this format:

```
Largest rectangle:
Length = 10 and Width = 100
Area = 1000
```

- If we define "Rectangle A *dominates* Rectangle B" to mean that Rectangle A's length is larger than Rectangle B's length, AND Rectangle A's width is larger than Rectangle B's width, print the dimensions of all of the rectangles that are NOT dominated by any other rectangles in the list. This could be an example:

```
Undominated rectangles:
Length = 10 and Width = 100
Length = 8 and Width = 200
```

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

[EXTRA SPACE FOR PROBLEMS 5 AND 6]

[EXTRA SPACE FOR PROBLEMS 5 AND 6]