**Name:** ____________________

## Rules and Hints

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- When you write code, be sure that the indentation level of each statement is clear.
- 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.
- As always, the SSU rules on academic integrity are in effect.

| **Problem** | **Max Score** | **Your Score** |
|---|---|---|
| *Problem 1:* Binary Search | 10 | |
| *Problem 2:* Selection Sort | 10 | |
| *Problem 3:* Mergesort | 10 | |
| *Problem 4:* Object-Oriented Analysis | 15 | |
| *Problem 5:* Defining classes | 30 | |
| *Problem 6:* Using classes | 25 | |
| **Total** | 100 | |

## Cheat Sheet Additions

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)
        swap(list_to_sort, i, min_index)
        # Problem 2: Show list contents at this point

def swap(L, i, j):
    x = L[i]
    L[i] = L[j]
    L[j] = x

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
```

## Cheat Sheet Additions

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 = ['biggie',
     'branch',
     'cooper',
     'creek',
     'guy-diamond',
     'mr.dinkles',
     'poppy',
     'smidge']
```

**Problem 1A** Fill out the below table, tracing `v = binary_search(L, 'cooper')`, a binary search for `'cooper'` in this list. Fill out one row per iteration of the loop (per the comment in the code on page 2). If there are more rows than iterations, leave extra rows blank. At the end, write the function's return value `v`.

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

**Return value** `v`: __________

**Problem 1B** Fill out the below table, tracing the call `v = binary_search(L, 'king-gristle')`, a binary search for `'king-gristle'` in this list. Fill out one row per iteration of the loop (per the comment in the code on page 2). If there are more rows than iterations, leave extra rows blank. At the end, write the function's return value `v`.

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

**Return value** `v`: __________

***Problem 2:*** **Selection Sort (10 points)**

Consider the following list:

```
L = ['poppy',
     'guy-diamond',
     'creek',
     'smidge',
     'mr.dinkles',
     'biggie',
     'cooper',
     'branch']
```

In the table below, show the *contents* of the list after each of the first four iterations of the for-loop in `selection_sort` (per the comment in the code on page 2).

You may just draw a horizontal line between cells if a word has *not* changed position.

| Index | Initial Order | After $i = 0$ iteration | After $i = 1$ iteration | After $i = 2$ iteration | After $i = 3$ iteration |
|---|---|---|---|---|---|
| 0 | poppy | | | | |
| 1 | guy-diamond | | | | |
| 2 | creek | | | | |
| 3 | smidge | | | | |
| 4 | mr.dinkles | | | | |
| 5 | biggie | | | | |
| 6 | cooper | | | | |
| 7 | branch | | | | |

***Problem 3:*** **Mergesort (10 points)**

Consider the following list:

```
L = ['smidge',
     'cooper',
     'mr.dinkles',
     'biggie',
     'poppy',
     'creek',
     'guy-diamond',
     'branch']
```

In the diagrams below, show the contents of the list after each of the first three iterations of the outer while-loop in `merge_sort` (per the comment in the code on page 3).

| Index | Initial Order | After `chunksize == 1` | After `chunksize == 2` | After `chunksize == 4` |
|---|---|---|---|---|
| 0 | smidge | | | |
| 1 | cooper | | | |
| 2 | mr.dinkles | | | |
| 3 | biggie | | | |
| 4 | poppy | | | |
| 5 | creek | | | |
| 6 | guy-diamond | | | |
| 7 | branch | | | |

***Problem 4:*** **Object-Oriented Analysis (15 points)**

Answer the questions below, using the following Python code:

```
class Car:
    def __init__(self, c):
        self.color = c
        self.cylinders = 4

    def __str__(self):
        return self.color + ':V'
          + str(self.cylinders)

    def upgrade(self, c):
        self.cylinders += c
```

```
# 4E:
```

```
color = 'red'                           # 4D: Calls method ___________

c = Car(color)                          # 4D: Calls method ___________

c.upgrade(2)                            # 4D: Calls method ___________

print(c)                                # 4D: Calls method ___________

print(color)                            # 4D: Calls method ___________
```

**Problem 4A** What is the data type of the variable `c`? ________________________

**Problem 4B** What is the data type of the variable `color`? ________________________

**Problem 4C** List the methods of class `Car`.

**Problem 4D** In each comment labeled 4D above, fill in the blank with the method(s) of class `Car` that are called in the execution of that line. If a line does not call a method of class `Car`, write N/A.

**Problem 4E** In the box labeled 4E above, write the output of the code.

***Problem 5:*** **Defining classes (30 points)**

In this problem, you will define a class to represent a students preparation for final exams at Sonoma State University. Your class should be named `ExamPrep`, and you should define the methods below. Hint: if you are using the `print` or `input` functions to implement these methods, you are doing it wrong.

`__init__`: Initializes an `ExamPrep` object. Takes three parameters: name of the student (a string), number of exams they are taking (an integer) and a list of relaxing strategies (a list of strings) and saves these in appropriate attributes.

In addition, it defines an attribute to

- store a list containing hours of preparation planned for each exam (size of this list is equal to total number of exams).

Initially, 0 hours of preparation has been planned for each exam.

`get_name`: Returns the name of the student. It takes no parameters.

`get_num_exams`: Returns the number of exams that the student is taking. It takes no parameters.

`set_exam_preptime`: Takes two parameters —i) an integer indicating exam number, and ii) a float indicating number of hours— and updates the list attribute that is storing hours of preparation. You may assume that the first parameter will get valid values (that is, not exceed the number of exams) and the second parameter will be positive.

`get_total_prep`: Returns the total number of hours that the student will spend preparing for all their exams. It takes no parameters.

`is_relaxed`: Returns `True` if the student has adopted at least 2 relaxing strategies or their *average* preparation time per exam is greater than 5 hours, and `False` otherwise.

`__lt__`: Compares `self` to another `ExamPrep` object. It returns `True` if `self` has a smaller total preparation time than the other `ExamPrep` object, and `False` otherwise.

`__str__`: Returns a string summarizing the `ExamPrep` object, following the format below exactly:

```
Jane Doe is spending a total of 18.5 hours and
2 relaxation strategies to prepare for 5 exams.
```

You should use values derived from attributes in place of the underlined values.

*Start your solution on the next page...*
*Toward the end of the exam, there are extra pages if needed.*

**Problem 5, continued . . .**

***Problem 6:*** **Using classes (25 points)**

For this problem, you must write a complete program. This includes logic in `def main()`, a call to `main()`, any necessary library imports, etc. You do *not* need to write any docstrings.

To earn full credit, you must *use the methods* from the `ExamPrep` class whenever appropriate. You may assume that the class, as described in Problem 5, has already been correctly implemented for you. Read the instructions carefully before you start coding!

Your program should do the following:

1. Define a function called `CreateProfile` that does the following:
   - Prompt for the name of the student.

     ```
     Name: John Smith
     ```
   - Prompt for the number of exams. If number is 0, return `None`.

     ```
     Number of exams: 3
     ```
   - Prompt the user for how many relaxing strategies they will adopt.

     ```
     Number of relaxation strategies: 2
     ```
   - Prompt for name of each relaxation strategy in the following format:

     ```
     Strategy 1: Pet dogs
     Strategy 2: Work out
     ```
   - Create an `ExamPrep` object, using all the data that has been entered above.
   - Prompt the user for number of hours spent in preparing for each exam in the following format:

     ```
     Exam 1: 7
     Exam 2: 2.5
     Exam 3: 9
     ```
   - Update the `ExamPrep` object created, and return it.
2. Define a function called `ExamPrepProfiles` that creates and *returns* a list of exam prepration profiles by:
   - Calling `CreateProfile` repeatedly, until it returns `None`.
   - Each time `CreateProfile` returns a `ExamPrep` object, it inserts it into a list.
3. Define a function called `main` that does the following:
   - Call `ExamPrepProfiles` to get a list of `ExamPrep` objects and print information about each of them (as provided by the `__str__` method).
   - Determine and print the total number of relaxed students.
   - Among all the relaxed students, determine the profile with least amount of total preparation time (as provided by the `__lt__` method) and print its information.

*Start your solution on the next page...*
*Toward the end of the exam, there are extra pages if needed.*

**Problem 6, continued . . .**

**Extra Pages . . .**