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: Recursion	20	
Problem 5: Defining classes	20	
Problem 6: Using classes	30	
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 = ['blaster',
     'crown',
     'evilgnome',
     'goggles',
     'lich',
     'pipe',
     'raptor',
     'star',
     'trojan',
     'walrus']
```

Problem 1A Fill out the below table, tracing the call `v = binary_search(L, 'goggles')`, a binary search for `'goggles'` 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 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				
5				

Return value `v`: _____

Problem 1B Fill out the below table, tracing the call `v = binary_search(L, 'septre')`, a binary search for `'septre'` 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 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				
5				

Return value `v`: _____

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

Consider the following list:

```
L = ['psyduck',
     'pikachu',
     'jigglypuff',
     'charizard',
     'brock',
     'turtwig',
     'bulbasaur',
     'magikarp']
```

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	psyduck				
1	pikachu				
2	jigglypuff				
3	charizard				
4	brock				
5	turtwig				
6	bulbasaur				
7	magikarp				

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

Consider the following list:

```
L = ['psyduck',
     'bulbasaur',
     'jigglypuff',
     'pikachu',
     'magikarp',
     'brock',
     'charizard',
     'turtwig']
```

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	psyduck			
1	bulbasaur			
2	jigglypuff			
3	pikachu			
4	magikarp			
5	brock			
6	charizard			
7	turtwig			

Problem 4: **Recursion (20 points)**

Consider the following function definition:

```
def fun(x, y):
    # Parameters x and y are strings
    # Recall: if x == 'a', then x[1:] is ''
    if len(x) == 0 or len(y) == 0:
        return 0
    elif x[0].lower() == y[0].lower():
        return 1 + fun(x[1:], y[1:])
    else:
        return fun(x[1:], y[1:])
```

Problem 4A What are the return values of each code snippet, below?

```
fun('', 'At')     # return value is:

fun('C', 'cAt')   # return value is:
```

Problem 4B Show the chain of recursive calls, and state what the return value is for each call, starting with:

```
fun('NaClC', 'TaCOcAt')
```

Problem 4C Summarize what this function does in one sentence. (Do not explain the code line-by-line; instead, provide a high-level description.)

Problem 4D Consider the following function definition:

```
def func(n):
    # Parameter n is an integer

    print("Current value is", n)

    if n <= 1:
        print("Base case:", n % 2)
        return str(n % 2)
    else:
        val = func(n//2)
        print("Returning", val + str(n % 2))
        return val + str(n % 2)
```

Write the output obtained upon executing the following statement:

```
print("Final answer is", func(6))
```

Extra Page . . .

Problem 5: **Defining classes (20 points)**

In this problem, you will define a class to represent a package. Your class should be named `SpecialDelivery`, 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 a `SpecialDelivery` object. Takes two parameters: a numerical package weight, and a Boolean value indicating if the package is fragile. It saves these in appropriate attributes, and initializes any attributes used by other methods.

`is_fragile`: Returns `True` if the package is fragile, and `False` otherwise.

`get_weight`: Returns the weight of the package.

`set_from`: Takes five parameters—the name, street address, city, state and zip code—each of which is a string related to the sender, and saves these in an attribute.

`set_to`: Takes five parameters—the name, street address, city, state and zip code—each of which is a string related to the recipient, and saves these in an attribute.

`get_from`: If no sender has been added, returns `None`; else, returns a list holding the five strings describing the sender: name, street address, city, state and zip code.

`get_to`: If no recipient has been added, returns `None`; else, returns a list holding the five strings describing the recipient: name, street address, city, state and zip code.

`__lt__`: Compares `self` to another `SpecialDelivery` object. It returns `True` if `self` has a smaller weight than the other object, and `False` otherwise.

`__str__`: Returns a string summarizing the `SpecialDelivery` object, following the format below exactly (two examples follow):

```
12.1 lb package:
From: None
To:   None
```

or this fragile example, with full addresses filled out:

```
1.12 lb package (Fragile):
From: Sally Sonoma
      1801 East Cotati Dr.
      Rohnert Park, CA 93928
To:   Ramona R. Sonoma
      2137 Indiana Avenue
      Honolulu, Hawaii 96814
```

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 (30 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 `SpecialDelivery` 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 `SendPackages` that does the following:
   - Prompt for the sender's name. If this input is empty, return an empty list.

     ```
     From: Sally Sonoma
     ```

   - Prompt for the sender's other data, as in the below interaction.

     ```
     Street: 1801 East Cotati Dr.
     City: Rohnert Park
     State: CA
     Zip: 93928
     ```

   - Prompt the user for how many packages they will send using this sender.
   - For each package:
     - Prompt for recipient and package information, as in the below interaction:

       ```
       To: Ramona R. Sonoma
       Street: 2137 Indiana Avenue
       City: Honolulu
       State: Hawaii
       Zip: 96814
       Weight: 1.12
       Fragile (y/n): y
       ```

     - Create a `SpecialDelivery` object, using all the data that has been entered above: sender (From), recipient (To), weight and fragility.
   - Returns a list of all the `SpecialDelivery` objects created.
2. Define a function called `main` that does the following:
   - Call `SendPackages` repeatedly, until it returns an empty list. Each time `SendPackages` returns a non-empty list, aggregate these into a single list.
   - Search this list for the largest and smallest `SpecialDelivery` objects, and print the summary data of these packages (as provided by the `__str__` method).
   - Sum the weight of all the packages marked fragile, and print the total weight of those packages.

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

Problem 6, continued . . .

Extra Pages . . .

Extra Pages . . .