Lab 11 Problems

Problem 1: Vocabulary. In computer science, in addition to developing programming and problem solving skills, it's also necessary to learn the vocabulary of common terms used in the field. For this problem, answer the questions below to test your understanding of the concepts. You can refer to the slides for help.

What does it mean for a data type to be immutable?

What is a hash function?

Describe how linear probing works.

Give a description of the queue abstract data type

Problem 2: Complexity.

1. The following function takes a list of integers and performs some computations based on that list.

```
def my_function(data):
    pairs = []
    for i in range(len(data)):
        for j in range(i + 1, len(data)):
            if data[i] + data[j] == 0:
                pairs.append((data[i], data[j]))
    return pairs
```

a) What does this function do?

b) What is its worst-case big-O complexity?

2. When searching for an item in a list, what is the name of the algorithm that we covered in lecture that has complexity O(log n)? Also, what must be true of the list for this algorithm to work?

3. In the code snippets below, `alist` is a previously defined list of an arbitrary length. Give the complexity of the two snippets of code below:

a)

```
n = len(alist) - 1
while n > 0:
    print(alist[n])
    n = n // 2
```

b)

```
n = len(alist) - 1
n = n // 2
while n > 0:
    print(alist[n])
    n = n - 1
```

Problem 3: Testing.

1. Black Box testing. You are given the following program specification:
   "This program reads a text file containing names and ages of people (one name and age per line, separated by a space) and prints out the average age of all the people in the file. If the file is empty, no output is given. All lines of data should contain a valid name-age pair."

   Give descriptions of the following black box test cases:

   a) Two error cases

   b) Two edge cases

   c) One regular (normal) case

2. White Box testing. Write four unit tests for the function below. Each test should provide specific values for the arguments `lst` and `target`.

```
# Returns the number of times a target element
# appears in the list.
def count_occurrences(lst, target):
    count = 0
    for item in lst:
        if item == target:
            count += 1
    return count
```

   Unit test 1:

   Unit test 2:

   Unit test 3:

   Unit test 4:

Problem 4 Hashing with separate chaining.

Step 1. Use the hash function `hash(key) = key % 7` and **separate chaining** to insert the keys below into the hash table. On a collision, add to the *front* of the linked list. Make sure that when you add a key to a slot, that the slot points to the first node in the linked list.

```
8, 24, 19, 17, 26
```

0	1	2	3	4	5	6

Step 2. Download the starter code `lab11_starter.py` from the class website.

Step 3. Implement the `put()` method for the Hashtable class. For this implementation, you are only putting in the keys, with no corresponding value (as is shown in Step 1). You can assume that all keys will be unique.

Write your code below or in your IDE using the starter code.

Step 4. The starter code contains a very limited implementation of the `LinkedList` class. What method would need to be implemented in the `LinkedList` class in order to write a `__contains__(self, key)` method for the `Hashtable` class?