These problems were generated by TAs and instructors in previous semesters. They may or may not match the actual difficulty of problems on Quiz3.

Lists

1. Write the function `onlyNegative(L)` which takes a list of numbers, `L`, and returns a **new list** that contains only the negative numbers in `L`, in the order they originally appeared. Your function should **not** destructively modify the original list `L`.

For example, `onlyNegative([-2, 1, 2, -1, 0, -3, 3])` would return the list `[-2, -1, -3]`.

2. Write a function commonElement that takes in two lists and returns True if they share a common element (ie. there is an element that occurs in both lists).

3. Given a word search puzzle (format is a 2D list of strings), check to see if a given word is in the board. Words can be left->right or up->down, but not right->left, down->up, or diagonal. For example,

```
puzzle = [ [ 'a', 'q', 'r', 't' ],
           [ 'd', 'o', 'g', 'a' ],
           [ 'w', 'm', 'z', 'c' ] ]

assert(wordSearch(puzzle, "dog") == True)

assert(wordSearch(puzzle, "cat") == False)
```

References and Memory

1. Are lists considered mutable or immutable objects? If mutable, what are some built in methods you can use to directly change a list?

2. What does the following code print?

```
def ct1(P, M):
      A = [ [ 1, 5, 7 ], [ 1 ] ]
      P[1][0] = P[1][0] * 3
      M[1] = A[0]
      A = P
      print("A =", A)
      print("P =", P)
      print("M =", M)

ct1([ [ 2, 4, 6 ], [ 2 ] ] , [ [ 3, 6, 9 ], [ 3 ] ])
```

3. Given that three lists are originally set up with the following code:

```
L = [ "Y" ]
A = [ 1, L, 2 ]
B = A
C = [ 1, L, 2 ]
```

Fill out the following table so that it shows the values in each variable after the line of code in the left column has run. Any changes made should be cumulative (if A is changed in row 1, the change should carry over to row 2).

Code	List A	List B	List C
Initial List	`[ 1, ["Y"], 2 ]`	`[ 1, ["Y"], 2 ]`	`[ 1, ["Y"], 2 ]`
`A.remove(2)`			
`B[0] = 9`			
`C.append(5)`			
`L.append("Z")`			
`A = [3, 4]`			

Recursion

1. Define what a base case is in terms of a recursive problem.

2. In this problem, you will translate the algorithm below into a recursive Python function `b2d(s)` that will compute the decimal value of a binary string of 1's and 0's. For example, `b2d("101") = 5` and `b2d("1011") = 11`.

    A. Base case: If the string is empty, return 0
    B. Recursive case:
        a. Compute the decimal value of the all but the first digit recursively by setting a variable `value` equal to the recursive call on the string from index 1 to end.
        b. Convert the 0 index character in the string to an integer and set a variable `bit` equal to it.
        c. Compute the bit's power of 2 by setting a variable `power` as 2 to the power of the string length minus 1.
        d. Return the value of the string so far as `bit*power + value`

Example: `b2d("1011")` recursively computes $1*2^3 + (0*2^2 + (1*2^1 + (1*2^0 + (0))))$ = 11

3. Write the recursive function addDeck(deck) that takes in a 2D List representing a deck of cards and **recursively** returns the sum of the numbers on the cards. Each list inside the deck is a card, with the first element representing the number on the card and the second element representing the suit. Assume all cards will be number cards. For example:

```
d = [[1, "Hearts"], [4, "Spades"], [3, "Clubs"], [8, "Diamonds"]]
assert(addDeck(d) == 16)
```

4. Write a recursive function areNearlyEqual(L, M) which takes two lists of integers and returns True if they are "nearly equal" or False otherwise. Two lists are "nearly equal" if they are the same length and the corresponding values in the lists are all within 1 (inclusive) of each other. Your solution must use recursion.

Search Algorithms

1. Answer the following questions about binary and linear search.
    a. Explain one major difference between binary and linear search
    b. For linear search, at what index would the target element be found in the best case scenario runtime? For binary search? Explain.
    c. For linear search, at what index would the target element be found in the worst case scenario runtime? For binary search? Explain.

2. Binary search algorithm trace: Assuming binary search is called on the following list, write the list argument used in the function call after each split when using binary search to search for 19.

Then: How many function calls are required to find the value? And how many calls would be required if linear search were used?

Original list: [3, 12, 26, 28, 32, 39, 44]

Dictionaries

1. The fruits in a bag of groceries are provided as a list (eg. `["apple", "oranges", "banana", "kiwis"]`). For any elements in the list that are plural, like “kiwis”, we say there are 3 of that fruit in the bag. (There are no fruits in the bag where the singular form of the word ends in an “s”.) Write a function `groceryCount` that outputs a dictionary with each fruit name and its frequency as a key-value pair.

For example:

```
groceryCount(["apple", "oranges", "banana", "kiwis", "kiwi"]) ==
{ "apple" : 1, "orange" : 3, "banana" : 1, "kiwi" : 4 }
```

2.

a. What does the following code print?

```
def chainedKeys(dict, startkey):
    key = startkey
    while key in dict.keys():
        key = dict[key]
        print(key)
    return key
result = chainedKeys({3:6, 4:9, 5:7, 6:5, 7:4, 8:9, 9:2}, 3)
print(str(result) + " is missing")
```

b. What is one key whose **value** could be changed to make the function loop infinitely? What should the value be changed to?

Hashed Search

1. What properties does a hash function need to achieve O(1) lookup in a hashtable?

2. What would happen if two objects happen to have the same hash values? Is this allowed?

3. What are some data structures that you can apply hash functions to, and what are the ones you cannot?

4. Each student in 15-110 is asked to create a list of their favorite ice cream flavors, which they can update as their tastes change.

Students A and B say ["chocolate", "vanilla", "strawberry"],
Student C says ["mint chocolate chip", "chocolate", "strawberry"], and
Student D says ["vanilla", "mint chocolate chip", "chocolate"].
Later, Student C finds the best strawberry ice cream ever and destructively shuffles their favorite list.

Your professors want to maintain a dictionary of ice cream preferences (keys) to student counts (values) in order to keep track of which combination is the favorite at the time the preferences were first collected. How should they store the keys, and why that approach?