15-110 Hw3 - Written Portion

Name:

AndrewID:

#1 - Recursion Tracing - 5pts

Trace the following code, then fill out the table below to indicate all the **recursive function calls** that are made, and which **value** is returned by each function call. You may not need all of the rows.

```
def gcd(x, y):
    if y == 0:
        return x
    return gcd(y, x % y)

print(gcd(20, 12))
```

Note: in the second column, make sure to indicate the actual returned **value**, not a set of arguments, a function call, or an expression.

Function Call	Returned Value
`gcd(20, 12)`	

#2 - Linear Search Debugging - 10pts

The following three functions all attempt to implement the algorithm linear search, but with a twist: instead of identifying whether or not the target occurs in the list, each function returns the first index where the item occurs, or -1 if it never shows up. However, only one of the three is correct. Identify which of the three functions is correct, then explain what is wrong with the other two and how they can be fixed.

```
def linearSearchA(lst, target):
    for i in range(len(lst)):
        if lst[i] == target:
            return i
        return -1

def linearSearchB(lst, target):
    i = 0
    while i < len(lst):
        if lst[i] == target:
            return i
        i = i + 1
    return -1

# Initially called with index = 0
def linearSearchC(lst, target, index):
    if lst[0] == target:
        return index
    elif len(lst) == 0:
        return -1
    else:
        return linearSearchC(lst[1:], target, index+1)
```

Which implementation is correct?

- ☐ linearSearchA
- ☐ linearSearchB
- ☐ linearSearchC

Why are the other two incorrect, and how can they be fixed?

#3 - Best Case and Worst Case - 10pts

For each of the following functions, describe an input that would result in **best-case efficiency,** then describe an input that would result in **worst-case efficiency**. This generic input must work at **any possible size**; don't answer 1 for `isPrime`, for example.

```
def getEmail(words):
    # words is a list of strings
    for i in range(len(words)):
        if "@" in words[i]:
            return words[i]
    return "No email found"
```

```
def isPrime(num):
    for factor in range(2, num):
        if num % factor == 0:
            return False
    return True
```

What is a **best case input** for `getEmail`?

What is a **worst case input** for `getEmail`?

What is a **best case input** for `isPrime`?

What is a **worst case input** for `isPrime`?

#4 - Calculating Big-O Families - 15pts

For each of the following functions, check the **Big-O function family** that function belongs to. You should determine the function family by considering how the number of steps the algorithm takes grows as the size of the input grows.

```
def countEven(L): # n = len(L)
  result = 0
  for i in range(len(L)):
    if L[i] % 2 == 0:
      result = result + 1
  return result
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L)
def sumFirstTwo(L):
  if len(L) < 2:
    return 0
  return L[0] + L[1]
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L1) = len(L2)
def linearSearchAll(L1, L2):
  count = 0
  for item in L1:
    # Hint: what's the complexity of
    # linear search?
    if linearSearch(L2, item) == True:
      count = count + 1
  return count
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L1) = len(L2)
def binarySearchAll(L1, L2):
  count = 0
  for item in L1:
    # Hint: what's the complexity of
    # binary search?
    if binarySearch(L2, item) == True:
      count = count + 1
  return count
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L); original call has i = 0
def recursiveSum(L, i):
  if i == len(L):
    return 0
  else:
    return L[i] + recursiveSum(L, i+1)
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

#5 - Dictionary Keys and Values - 5pts

Given the following set of code:

```
d = { "snow" : 7, "rain" : 4, "sun" : 10 }
d["fog"] = d["rain"] - 1
for k in d:
    d[k] = d[k] * 2
d.pop("sun")
```

What are the **keys** of the dictionary `d` after this code has run?

For each of the keys you listed above, what is its **value** associated with that key after the code has run?

What line of code would you write to add the **key-value pair** `("hail", 1)` to the code? Assume the line will be added at the *end* of the current code.

#6 - Good Use of Hashing? - 5pts

Recall our discussion of what hash functions are and what they are used for. Below we've listed four different scenarios. Each scenario contains a data set, a hash function, and which values will need to be looked up in the hashtable. Select **all** the scenarios where you will generally be able to look up the given values in **constant time**.

- ☐ Given a set of integer phone numbers, hash a phone number based on the phone number itself. Use the hashtable to look up an individual phone number.

- ☐ Given a set of all the college essays sent to CMU (as strings), hash an essay based on the first character of the essay ("I want to go to CMU because.." hashes based on "I"). Use the hashtable to look up an individual essay.

- ☐ Given a set of string full names (like "Farnam Jahanian"), hash a name by adding together the numeric ASCII values of all the letters in the name. Use the hashtable to look up an individual name.

- ☐ Given a set of lists of high scores (so each list contains integers), hash a list based on the sum of its scores. Lists can be updated after hashing when new high scores are added. Use the hashtable to look up an individual high-score list.

- ☐ None of the situations described above can be searched in constant time.

#7 - Tracing Sorting Algorithms - 15pts

In the table below, each row represents a 'pass' - a single iteration of the outer loop in the function. Fill in the number of comparisons and swaps that happen in each pass, and the state of the list at the **end** of that pass, for **selection sort** as implemented in class. We've filled in the first row for you.

Pass #	Comparisons	Swaps	List State
Start	-	-	**[** 5, 4, 2, 3, 6, 7, 1 **]**
1	**6**	**1**	**[** 1, 4, 2, 3, 6, 7, 5 **]**
2			
3			
4			
5			
6			

In the next table, each row will represent either a **split-pass** (S) or a **merge-pass** (M). Fill in the number of comparisons and copies that happened in each pass, and the state of each of the list(s) at the **end** of that pass, for **merge sort** as discussed in class. We've filled in the first row for you.

Pass #	Comparisons	Copies	List(s) State
Start	-	-	**[** 5, 4, 2, 3, 6, 7, 1 **]**
S1	**0**	**7**	**[5, 4, 2] [3, 6, 7, 1]**
S2			
S3			
M1			
M2			
M3			

15-110 Hw3 - Programming Portion

Each of these problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Hw3 - Programming for autograding.

All programming problems may also be checked by running the starter file, which calls the function `testAll()` to run test cases on all programs.

#1 - `addToAll(lst, x)` - 5pts

Write the function `addToAll(lst, x)` which takes a list of numbers and a number and **destructively** modifies the list so that every element has `x` added to it, then returns `None`. For example, if `lst = [1, 2, 3]`, calling the function `addToAll(lst, 2)` will evaluate to `None`, but will also change `lst` to hold `[3, 4, 5]`.

#2 - `recursiveMax(lst)` - 5pts

Write a function `recursiveMax(lst)` that takes a list as input and returns the maximum value in the list. You may assume the list contains at least one element. This function must use **recursion** in a meaningful way; a solution that uses a loop or built-in max functions will receive no points.

For example, `recursiveMax([1, 2, 3])` returns `3`, and `recursiveMax([2, 4, 6, 9, 10, 2, 6])` returns `10`.

Hint: consider what properties the recursive result has if the function works as expected.

Another hint: consider what the **base case** for this algorithm should be. It isn't the usual list base case...

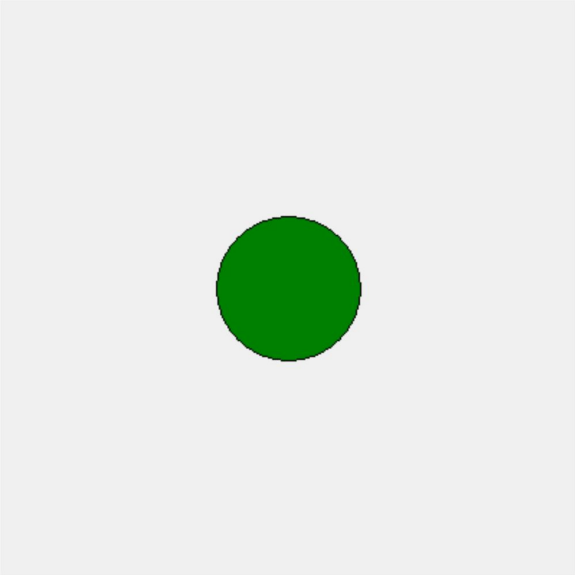
#3 - `generateBubbles(canvas, bubbleList)` - 5pts

Write the tkinter function `generateBubbles(canvas, bubbleList)` which takes a tkinter canvas and a **list of dictionaries**, `bubbleList`, and draws bubbles as described in `bubbleList`.

Each dictionary in the bubble list contains exactly four keys: `"left"`, `"top"`, `"size"`, and `"color"`. The first three all map to integers (the left coordinate, top coordinate, and diameter size of the bubble), and the fourth maps to a string (its color). Use this information to draw the bubble (with `canvas.create_oval`) in the appropriate location, with the correct size and color.

For example, if we make run the function with the bubble list from the first test:
`bubbleList1 = [ {"left":150, "top":150, "size":100, "color":"green"} ]`

We'll get:

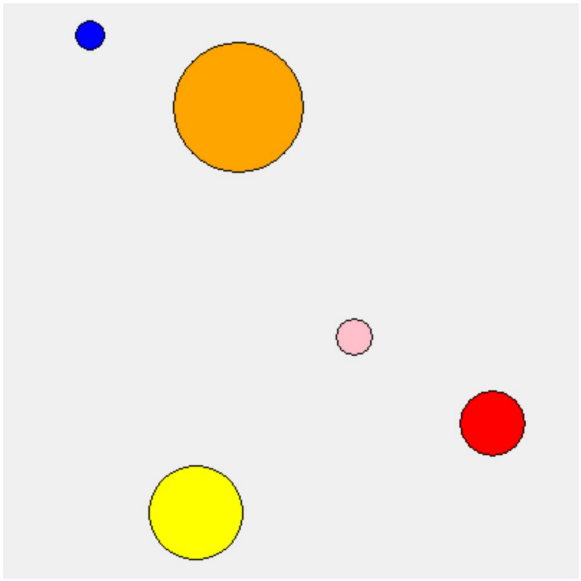


[continued on next page]

And the second test, which has:

```
bubbleList2 = [
      {'left': 317, 'top': 269, 'size': 45, 'color': 'red' },
      {'left': 118, 'top': 27, 'size': 90, 'color': 'orange'},
      {'left': 101, 'top': 321, 'size': 65, 'color': 'yellow'},
      {'left': 231, 'top': 219, 'size': 25, 'color': 'pink' },
      {'left': 50, 'top': 12, 'size': 20, 'color': 'blue' } ]
```

Should produce this:



The third test randomly generates 10 bubbles using the provided `makeNBubbles(n)` function. Try changing the size of `n` to generate more or less bubbles, and see how it looks! Your bubbles will be different every time.

Hint: a list of dictionaries might sound intimidating at first, but it's not so bad! Just loop over the list, access the dictionary using the loop control variable, then index into the dictionary to get the needed values.

#4 - `getBookByAuthor(bookInfo, author)` - 10pts

Dictionaries are very good at searching for keys, but not so good at searching for values. Write the function `getBookByAuthor(bookInfo, author)` which takes a dictionary mapping book titles (strings) to author names (also strings) and an author name (a string) and returns the book associated with that author, or `None` if the author does not appear in the dataset. You are guaranteed that no author will appear more than once in the dictionary.

For example, calling the function on `{ "The Hobbit" : "JRR Tolkein", "Harry Potter and the Sorcerer's Stone" : "JK Rowling", "A Game of Thrones" : "George RR Martin" }` and `"JK Rowling"` would return `"Harry Potter and the Sorcerer's Stone"`.

Hint: you basically want to implement **linear search** over a dictionary instead of a list. Make sure you use the right kind of loop!

#5 - `makeFirstPhonebook(nameList, numberList)` - 10pts

Write the function `makeFirstPhonebook(nameList, numberList)` that takes two lists, a list of names and a list of phone numbers (both strings), and returns a dictionary mapping names to phone numbers. You may assume that the two lists match up, i.e., each person is at the same index as their phone number.

You should use a **loop** to construct the dictionary. You'll need to loop over both the `nameList` and the `numberList` *at the same time* to access the key and value together. To do this, use the same loop control variable on both lists in each iteration.

If a person occurs in `nameList` multiple times (in other words, if they have multiple phone numbers), you should map their name to the **first** phone number they were paired with. For example, given the list of names `["Kelly", "Dave", "Kelly"]` and the list of numbers `["0000", "1234", "9876"]`, the function would return the dictionary `{ "Kelly" : "0000", "Dave" : "1234" }`.