

Name: _____

andrewID: _____

Instructions:

- Do not open this exam until the instructor tells you to.
- This exam tests material from weeks 5-8 of the course. You have **50 minutes**.
- If you have a clarification question, raise your hand and a proctor will come help you.
- You must complete the exam **individually**. You may refer to paper notes during the exam, but do not communicate with anyone else.
- Show your work to get full credit.
- Make sure to display values **unambiguously**. Strings should be in quotes (unless they're being printed), floats should have decimals, Booleans should be capitalized.

#1 - Code Writing - Lists [15pts]	2
#2 - Code Reading - Aliasing [15pts]	3
#3 - Code Reading - Recursion [15pts]	4
#4 - Free Response - Big-O [15pts]	5
#5 - Code Writing - Graphs [15pts]	6
#6 - Short Answer - Search Algorithms [4pts]	7
#7 - Short Answer - Dictionaries [6pts]	7
#8 - Short Answer - Trees [5pts]	8
#9 - Short Answer - Hashtables [6pts]	9
#10 - Short Answer - Tractability [4pts]	9

#1 - Code Writing - Lists [15pts]

Write the function `mostLetters(lst, letter)` which takes a list of strings (`lst`) and a one-character string (`letter`) and returns the string in the list that contains the most instances of the given letter. You are guaranteed that at least one string in the list contains the letter. If there's a tie, you can return any of the tied answers. For example:

```
mostLetters(["abc", "alpha", "aaa", "xyz"], "a") -> "aaa"
mostLetters(["lots of text", "mumble", "m"], "m") -> "mumble"
mostLetters(["www", "aaaaaaaaaaaa"], "w") -> "www"
mostLetters(["cat", "bat"], "a") -> "cat" # "bat" would also be fine
mostLetters(["rest"], "e") -> "rest"
```

Hint: you may want to use the string method `str.count(c)`. This takes a single character and returns the number of times that character occurs in the string.

```
def mostLetters(lst, letter):
    mostWord = ""
    mostCount = 0
    for word in lst:
        count = word.count(letter)
        if count > mostCount:
            mostWord = word
            mostCount = count
    return mostWord
```

#2 - Code Reading - Aliasing [15pts]

Consider the following code:

```
1 def fun(lst1, lst2):
2     result = []
3     for i in range(len(lst1)):
4         lst1[i] = lst1[i] + 2
5         result.append(lst2[i] + 5)
6     return result
7
8 w = [ 2, 3, 5 ]
9 x = w
10 y = [ 2, 3, 5 ]
11 z = y
12
13 val = fun(w, y)
14
15 x.append(42)
16 z = z + [ 110 ]
```

What will lst1 and lst2 hold **right before result is returned**?

lst1	[4, 5, 7]
lst2	[2, 3, 5]

What will these variables hold **at the end of the code**?

w	[4, 5, 7, 42]
x	[4, 5, 7, 42]
y	[2, 3, 5]
z	[2, 3, 5, 110]
val	[7, 8, 10]

Which of the following pairs of variables are aliased **at some point** during code run?
Select all that apply. (Some aliased pairs might not be included in the options.)

☒ y and z

☒ w and x

☐ lst1 and lst2

☒ y and lst2

☐ w and y

☐ lst2 and result

Which of the following pairs of variables are aliased **at the end** of the code run?
Select all that apply. (Some aliased pairs might not be included in the options.)

☐ y and z

☒ w and x

☐ lst1 and lst2

☐ y and lst2

☐ w and y

☐ lst2 and result

#3 - Code Reading - Recursion [15pts]

Consider the following code:

```
1 def rec(s):
2     if len(s) <= 1:
3         print("Base: " + s)
4         return s
5     else:
6         print("Split: " + s)
7         result = rec(s[1:]) + s[0]
8         print("Build: " + result)
9         return result
```

What would be printed if you called
rec("pet")?

Split: pet
Split: et
Base: t
Build: te
Built: tep

How many times would the function rec be called if you ran rec("future")?
Include **all** function calls that occur when the code runs (including the initial call).

Six - five recursive cases, one base case.

#4 - Free Response - Big-O [15pts]

For the following programs, identify the Big-O runtime of each of the required lines on the right and the total Big-O runtime of the function. Make sure to represent your Big-O runtimes in simplified terms and using Big-O notation.

<pre># lst is a list # assume len(lst) = n and n > 1000 # Note: max(..) runs in O(n) 1. def mystery1(lst): 2. x = 0 3. for i in range(750, len(lst)): 4. for j in range(1000): 5. if j % 2 == 0: 6. x = x + max(lst) 7. return x</pre>	Line 2 runtime: ____O(1)____ Line 3 runtime: ____O(n)____ Line 4 runtime: ____O(1)____ Line 5 runtime: ____O(1)____ Line 6 runtime: ____O(n)____ Total runtime: ____O(n^2)____
<pre># num is an integer # n = num 1. def mystery2(num): 2. i = num 3. count = 0 4. while i > 0: 5. if i % 11 == 0: 6. count = count + 1 7. i = i // 2 8. return count</pre>	Line 2 runtime: ____O(1)____ Line 3 runtime: ____O(1)____ Line 4 runtime: ____O(logn)____ Line 5 runtime: ____O(1)____ Line 6 runtime: ____O(1)____ Line 7 runtime: ____O(1)____ Total runtime: ____O(logn)____

#5 - Code Writing - Graphs [15pts]

As part of an initiative to help the financial health of your friends, you are keeping track of all your friends' debts. The graph below shows an example of your friends and who they owe money to and how much. For example, **Leslie** owes **Amanda** **3** dollars.

Write a function `collect(g, name)` that takes this graph as a parameter and the name of one of your friends, and returns how much money your friend will collect if everyone who owes them money pays them. For example, calling `collect(debts, "Dennis")` returns 13 because Rich owes Dennis \$5, Niki owes Dennis \$7, and Amanda owes Dennis \$1, which means Dennis would collect \$13 once everyone pays him.

```
debts = { "Rich":  [["Dennis", 5], ["Leslie", 5], ["Amanda", 10]],
          "Dennis": [["Rich", 2], ["Niki", 2], ["Amanda", 3]],
          "Niki":   [["Leslie", 1], ["Dennis", 7]],
          "Leslie": [["Amanda", 3]],
          "Amanda": [["Leslie", 1], ["Dennis", 1], ["Rich", 2]] }
```

```
def collect(g, name):
    total = 0
    for person in g:
        for pair in g[person]:
            if pair[0] == name:
                total = total + pair[1]
    return total
```

#6 - Short Answer - Search Algorithms [4pts]

You're given the list [5, 7, 8, 10, 12, 21, 80]. Which elements would be compared to the target if you ran...

... **linear search** with the target value 2?

5, 7, 8, 10, 12, 21, 80

... **binary search** with the target value 2?

10, 7, 5

#7 - Short Answer - Dictionaries [6pts]

Consider the following code:

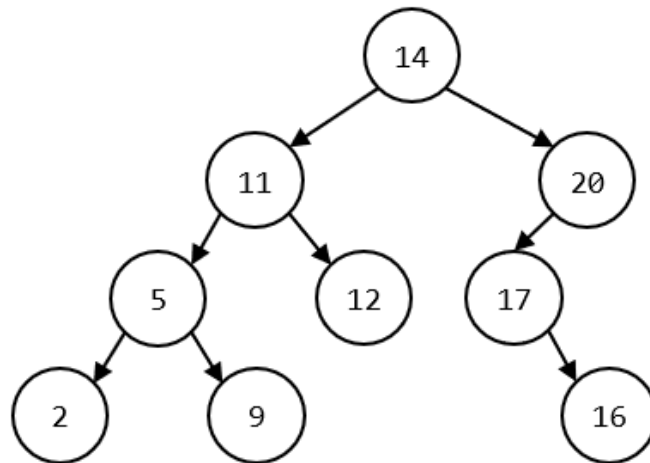
```
d = { "y" : 2, "z" : 5 }  
d["x"] = 5  
d["z"] = 11  
d["y"] = d["y"] + 4  
if 5 in d:  
    d.pop(5)
```

Please list all the **key-value** pairs in d after the code above runs. You may not need all the lines.

```
d = { ____"y"____ : ____6____, ____"z"____ : ____11____,  
      ____"x"____ : ____5____, _____ : _____,  
      _____ : _____, _____ : _____, }
```

#8 - Short Answer - Trees [5pts]

Given the tree below, identify the following:



Which node(s) are the root ?	14
Which node(s) are the leaf nodes?	2, 9, 12, 16
Which node(s) are the children of node 20 ?	17
Is the above tree a Binary Search Tree ? Explain why or why not in one sentence.	No, because 16 is to the right of node 17.

#9 - Short Answer - Hashtables [6pts]

Each scenario below 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** and **the hash is valid**.

- ☐ Given a set of shopping carts at Target (with total prices between \$25-\$200), hash a cart based on its current total price rounded to nearest ten dollars. Use the hashtable to look up an individual shopping cart.
- ☐ Given a set of lists of temperatures (each list contains floats), hash a list based on the average temperature. Lists can be updated after hashing when temperatures are added. Use the hashtable to look up a list of temperatures.
- ☒ Given a set of car licence plates (which consist of a mix of numbers and letters), hash a licence plate by adding together the numeric ASCII values of all the characters in the plate, where each value is multiplied by 10 to the power of its index in the string. Use the hashtable to look up an individual licence plate.
- ☐ None of the above scenarios will let you look up values in constant time.

#10 - Short Answer - Tractability [4pts]

For each of the following statements, select whether it is True, False, or Unknown.

Finding the max element in an unsorted list is an example of a tractable problem. ☒ True ☐ False ☐ Unknown	If a problem is in the complexity class P, it is not in the class NP. ☐ True ☒ False ☐ Unknown
The Subset Sum problem can be solved using a brute-force approach in $O(n^3)$ time. ☐ True ☒ False ☐ Unknown	Using heuristics, we can solve problems like Exam Scheduling and Travelling Salesperson in much less time without sacrificing accuracy or optimality. ☐ True ☒ False ☐ Unknown