fullName:_____andrewID:_____ recitationLetter:_____

15-112 S23

Quiz8 version A (25 min)

You **MUST** stop writing and hand in this **entire** quiz when instructed in lecture.

- You may not unstaple any pages.
- Failure to hand in an intact quiz will be considered cheating. Discussing the quiz with anyone in any way, even briefly, is cheating. (You may discuss it only once the quiz has been posted to the course website.)
- You may not use your own scrap paper. If you must use additional scrap paper, raise your hand and we will provide some. You must hand any scrap paper in with your paper quiz, and we will not grade it.
- Please try to limit questions so as to not distract your peers. **We will answer two questions at most per person.** If you are unsure how to interpret a problem, take your best guess.
- Unless otherwise stated, you may not use any concepts (including builtin functions) which we have not covered in the notes in weeks 1-8. You may not use recursion.
- Assume almostEqual(x, y) and rounded(n) are both supplied for you. You must write all other helper functions you wish to use, unless we specify otherwise.

Multiple Choice [3pts ea, 12 total]

MC1. Out of the provided options, which of the following is the fastest (ie. most efficient) function family for large N?
Select the best answer (fill in one circle).

- ○ O(N)
- ○ O(logN)
- ○ O(NlogN)
- ○ O(2**N)
- ○ O(N**2)

MC2. What is the efficiency of linear search?
Select the best answer (fill in one circle).

- ○ O(N)
- ○ O(logN)
- ○ O(NlogN)
- ○ O(2**N)
- ○ O(N**2)

MC3. What is the efficiency of merge sort?
Select the best answer (fill in one circle).

- ○ O(N)
- ○ O(logN)
- ○ O(NlogN)
- ○ O(2**N)
- ○ O(N**2)

MC4. What is the best efficiency for sorting (assuming we don't have additional information about the list to be sorted)?
Select the best answer (fill in one circle).

- ○ O(N)
- ○ O(logN)
- ○ O(NlogN)
- ○ O(2**N)
- ○ O(N**2)

True/False [2pts ea, 8 total]

Mark each option as either True or False. (Fill in one circle for each question.)

TF1. Values in sets are unordered and must be unique and immutable

◯ True

◯ False

TF2. Dictionaries contain key-value pairs. Keys must be immutable, but values may be mutable or immutable

◯ True

◯ False

TF3. In 15-112, we don't care about constant coefficients, so O(100N**2) simplifies to O(N**2)

◯ True

◯ False

TF4. In 15-112, we don't care about higher-order terms, so O(10000N + N**2) simplifies to O(N)

◯ True

◯ False

CT1: Code Tracing [10pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. If the result contains elements that are unordered in Python, you may write them in any order.

```
#Hint: Sets are mutable, so
#   what does u = t create?
def ct1(L):
    s = set()
    t = set()
    for i in range(len(L)):
        if L[i]==i:
            u = t
        else:
            u = s
        u.add(L[i])
    return (u, s, t)

for elem in ct1([4, 1, 6, 3, 6]):
    print(elem)
```

Free Response 1: stringMatches(L) [35 points]

Write the function stringMatches(L) which takes a list L of non-empty strings and returns a set of all strings in L which can be formed by combining two other strings in L. Carefully consider the test cases below, and note that each string in the result must be formed from two strings in L, though those strings may be equivalent. Your function must have an **efficiency of O(N**2) or better,** where N is the length of the list. Less efficient solutions will be eligible for no more than half credit.

```
def testStringMatches():
  assert(stringMatches(['a', 'b', 'c', 'bc', 'ca']) == {'ca', 'bc'})

  assert(stringMatches(['car', 'carp', 'speed', 'race', 'y', 'speedy', 'p', 'racecar'])
                      == {'carp', 'speedy', 'racecar'})

  assert(stringMatches(['marf', 'no', 'matches', 'here']) == set())

  assert(stringMatches(['xo', 'xoxo']) == set()) # Can't use 'xo' twice for 'xoxo'!

  assert(stringMatches(['xoxo', 'xo', 'xo']) == {'xoxo'})

  assert(stringMatches(['a', 'b', 'ab', 'ba']) == {'ab', 'ba'})
```

Begin your FR1 answer here or on the next page

You may continue your FR1 answer here

Free Response 2: bigramFrequency(text) [35 points]

Write the function bigramFrequency(text) which takes a string, text, and returns a dictionary mapping each bigram in the string to its frequency. Within a text, a bigram is any pair of consecutive words. Frequencies should be floats between 0 and 1, such that all frequencies add up to 1.
Consider the following string:
"Beth is a student. Jimothy is a student. 42 is a number."
This contains the following bigrams: "beth is", "is a", "a student", "student jimothy", "jimothy is", "student is", "a number"

Notice that "a student" shows up twice and "is a" shows up three times in the string. Notice also that **bigrams are lowercase, and they ignore punctuation and digits.**

To get the bigram frequency, we recommend the following steps:

1. First, create a string containing only letters (converted to lowercase) and spaces. So, for the example above, we want:
   ```
   s = "beth is a student jimothy is a student  is a number"
   ```

2. Now split that string into a list:
   ```
   L = ["beth", "is", "a", "student", "jimothy", "is", "a", "student", "is", "a", "number"]
   ```

3. Use this list to create a dictionary mapping each bigram to the number of times it appears:
   ```
   d = {"beth is": 1,
        "is a": 3,
        "a student": 2,
        "student jimothy": 1,
        "jimothy is": 1,
        "student is": 1,
        "a number": 1}
   ```

4. Find the total number of bigrams (in this example we have 10) and use it to return a dictionary of bigram frequencies such that all frequencies add up to 1:
   ```
   result == {"beth is": 0.1,
              "is a": 0.3,
              "a student": 0.2,
              "student jimothy": 0.1,
              "jimothy is": 0.1,
              "student is": 0.1,
              "a number": 0.1}
   ```

Here is an additional test case:

```
text = "A cat is a cat is a cat!  Cat5 is a cable... cable is not a cat. "
assert(bigramFrequency(text) == {'a cat': 0.25,        # 4/16
                                 'cat is': 0.1875,     # 3/16
                                 'is a': 0.1875,       # 3/16
                                 'cat cat': .0625,     # 1/16
                                 'a cable': .0625,     # 1/16
                                 'cable cable': .0625, # 1/16
                                 'cable is': .0625,    # 1/16
                                 'is not': .0625,      # 1/16
                                 'not a': .0625})      # 1/16
```

Make reasonable assumptions for any edge cases not illustrated here.

Begin your FR2 answer here

You may continue your FR2 answer here

bonusCT: Code Tracing [2pts bonus]

This question is optional. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def bonusCt1(L, s):
    L = [chr(ord('a')+L[i]+i) for i in range(len(L))]
    s = sorted(set(s) - set(L)) * len(set(s))**len(L)
    return ''.join(s).count('rb')
print(bonusCt1([2, -1], 'abracadabra'))
```