Name:_____ andrewID:_____ Recitation:___

15-112 F24
Quiz8 version A (30 min)

Read these instructions carefully before starting:

1. Quiz versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. Stop writing and submit the entire quiz when instructed by the proctor.
    - Do not unstaple any pages.
    - You must submit the entire quiz with all pages intact.
3. Do not discuss the quiz with anyone else until after 4pm.
    - This applies to everyone, including students in either lecture.
4. Do not use your own scrap paper.
    - You should not need scrap paper, there is plenty of room for you on the quiz.
    - However, if you absolutely must use scrap paper, raise your hand and we will provide some. Then, you must write your andrew id clearly on the scrap paper, and hand in the scrap paper with your paper quiz. We will not grade anything on your scrap paper.
5. You may not ask questions during the quiz.
    - The one exception is for English-language clarifications.
    - If you are unsure how to interpret a problem, just take your best guess.
6. Do not use any concepts (including built-in functions) not covered in the notes through week 8.
    - Do not use OOP or recursion.
7. Do not hardcode your solutions.
    - We may test your code using additional test cases.
    - Hardcoding will receive zero points.
8. 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.
9. Good luck!

Code Tracing (CT) [10 pts, 5 pts each]
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

CT1:

```
def ct1(L):
    s = set(L)
    t = set([2*v for v in L])
    return s-t, t-s, s.intersection(t)
print(ct1([1,2,3,4]))
```

CT2:

```
def ct2(s):
    d = dict()
    for v in s:
        k = v%2
        d[k] = d.get(k, 0) + v
    return d
print(ct2(set([3,3,3,1,2,2])))
```

True/False [20 pts, 2 pts each]
For each of the following, bubble in True or False (do not bubble in both!).
Assume L and M are both lists with N integers.
Assume S is a set with N integers.

O True O False 1. We represented each Tetris shape with a 2d list of booleans.

O True O False 2. We represented the Tetris board as a 2d list of instances of SimpleNamespace.

O True O False 3. Sets can contain sets.

O True O False 4. Dict keys can be dicts but dict values cannot be dicts.

O True O False 5. For a sorted list, binary search is typically much faster than linear search.

O True O False 6. If len(set(L)) < N, then L contains duplicate values.

O True O False 7. Testing if (v in S) is O(1) in the worst case.

O True O False 8. If L == M, then set(L) == set(M).

O True O False 9. If len(set(L) - set(M)) == 0, then L and M do not share any values.

O True O False 10. Testing if (v in L) is O(N) in the best case (note: NOT the worst case).

Fill-in-the-blanks: Red and Blue Lines [20 pts, 2.5 pts each]
For this problem, you are given a working solution, but with some lines (or parts of lines) replaced with blanks. Fill in the blanks so the code works properly.

This app should work as such:

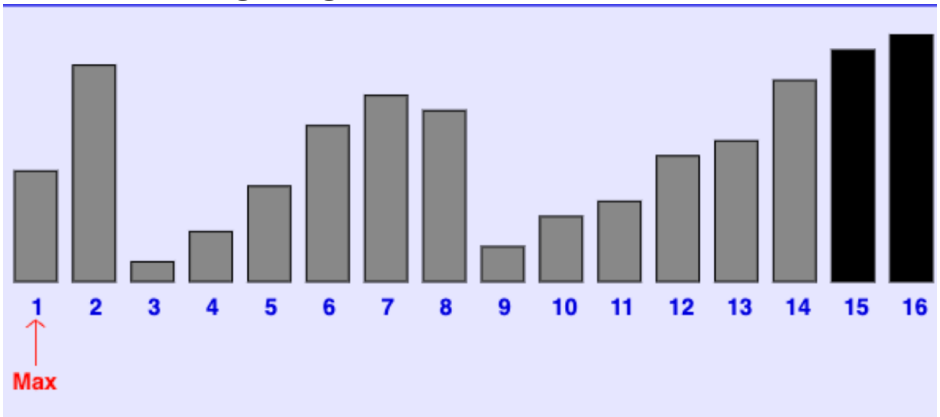
- When started, the canvas is empty.
- When the user presses 'r', a random red line with lineWidth of 1 is drawn.
- When the user presses 'b', a random blue line with lineWidth of 1 is drawn.
- Once every 2 seconds, all the red lines have their lineWidth increase by 1.
- Once every 10 seconds, all the lines are removed, so the canvas is empty.

```
from cmu_graphics import *
from types import SimpleNamespace
import random

def onAppStart(app):
    app.counter = 0
    app.stepsPerSecond = 10
    app.lines = [ ]

def onKeyPress(app, key):
    if key in 'rb':

        color = ____________________________________________ #1

        addRandomLine(app, color)

def addRandomLine(app, color):
    x1, y1, x2, y2 = [random.randrange(50, 350)
                      for _ in range(4)]
    line = makeLine(x1, y1, x2, y2, color)

    ____________________________________________________________ #2
```

```
def makeLine(x1, y1, x2, y2, color):

    line = ____________________________________________________#3
    line.x1 = x1
    line.y1 = y1
    line.x2 = x2
    line.y2 = y2
    line.color = color

    ____________________________________________________ #4

    return ____________________________________________ #5


def onStep(app):
    app.counter += 1

    if ____________________________________________ : #6

        makeRedLinesThicker(app)

    if ____________________________________________ : #7

        app.lines = [ ]


def makeRedLinesThicker(app):
    for line in app.lines:
        if line.color == 'red':

            ____________________________________________ #8


def redrawAll(app):
    for line in app.lines:
        drawLine(line.x1, line.y1, line.x2, line.y2,
                 fill=line.color, lineWidth=line.lineWidth)

runApp()
```

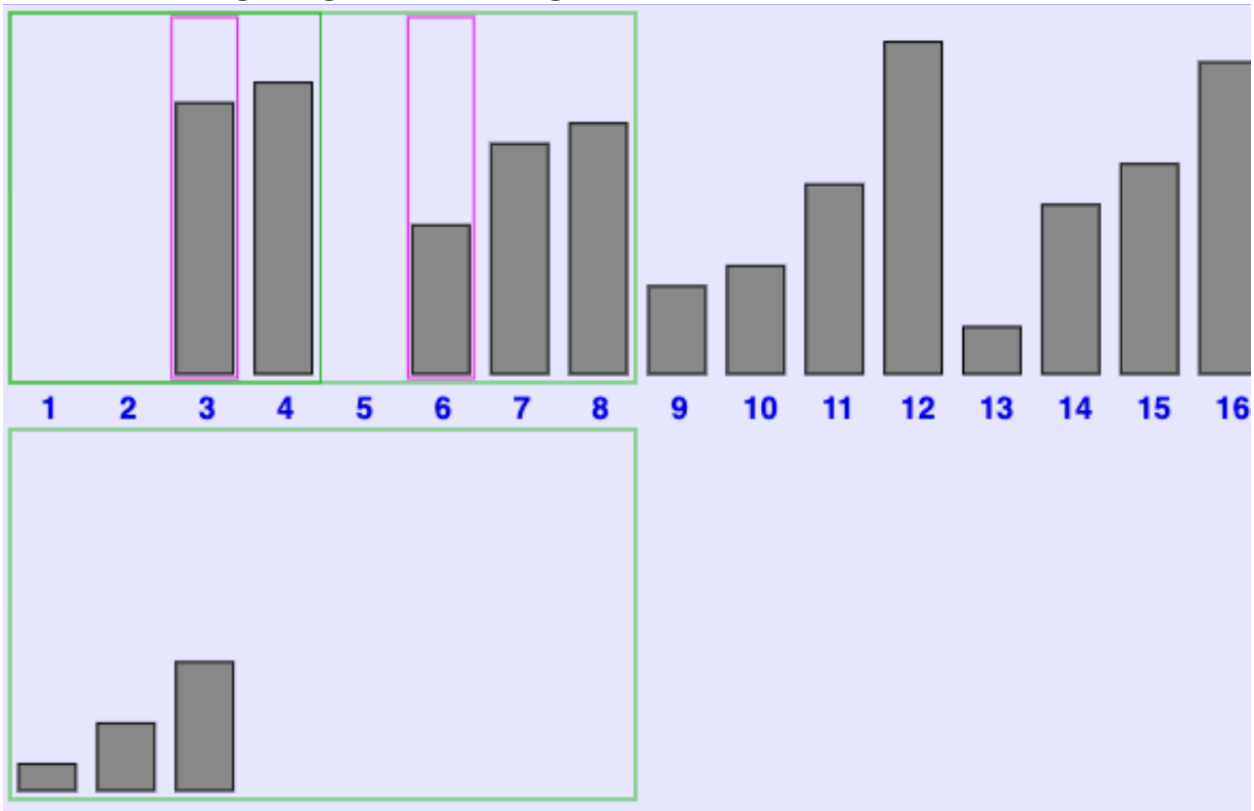
Searching, Sorting, and Hashing [20 pts, 2 pts each]

1. The following image is from selection sort in xSortLab:



What are the indexes of the next two values to be swapped?

2. The following image is from merge sort in xSortLab:



What is the index of the next value to be copied to the temporary list?

3. When running selection sort over a list of N integers, how many steps are taken on the kth pass (where a step is either a compare or a swap, and where k=0 on the first pass)?

4. When running merge sort over a list of N integers, where you may assume that N is a power of 2...

A) How many steps per pass are required (where a step is either a compare or a copy)?

B) How many total passes are required?

5. If selection sort takes 1 minute to sort 2,000 values, how long (to the nearest minute) would we expect selection sort to take to sort 8,000 values on the same computer?

6. If linear search takes 0.01 seconds to search 2,000 values, how long (to the nearest hundredth of a second) would we expect linear search to take to search 8,000 values on the same computer?

7. When we represent a dict using a hash table, each element in each bucket is not a single value but actually a tuple of length 2. In just a few words, what do those two values represent?

Note: for the next 2 questions:

- you may assume that 2**10 == 1,000, and
- we will accept answers within 2 of the correct answer.

8. For a list with 1 million values, how many comparisons would be required in the worst case for linear search?

9. For a sorted list with 1 million values, about how many comparisons would be required in the worst case for binary search?

10. For this question, assume:
    1) hash tables are implemented as described in lecture,
    2) each bucket in a hash table has a max bucket size of 2,
    3) hash tables start with 5 buckets, and
    4) hash(x) == x for any integer x.

Say we have this code:

```
s = set()
s.add(88)
s.add(24)
s.add(24)
s.add(13)
```

With the assumptions above, write the values in each bucket for the hash table that represents this set after running those lines of code:

0:

1:

2:

3:

4:

Free Response / FR: fasterFoo in O(N) [30 pts]

For this exercise, you are given a function foo(L) that takes a list L with N integers. Study the function to understand what it is doing.

Then write the function fasterFoo(L) which does the same thing in general as foo(L) but runs in O(N).

Answers that do not run in O(N) will not receive any credit.

```
def foo(L):
    M = [ ]
    for v in L:
        if L.count(v) == v:
            M.append(v)
    return set(M)

def testFoo():
    print('Testing foo()...', end='')
    assert(foo([1,2,3,2,3,4,3,1]) == {2,3})
    print('Passed!')

testFoo()
```

Begin your FR answer on the next page.

Begin your answer to the FR here:

Bonus Code Tracing (BonusCT)) [Optional, 4 pts, 2 pts each]
Bonus problems are not required.
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

BonusCT1:

```
def bonusCt1(s):
    return eval(''.join([v[:s.count('1-')]
                         for v in s.split()]) + ')')
s = 'see  t(42)  [1-dog]  2,3-cats  13-mice  ,1-1,  2][2'
print(bonusCt1(s))
```

BonusCT2:

```
def bonusCt2(m):
    def f(k): return set(str(list(range(k))))
    def g(m, n): return int(''.join(sorted(f(m) - f(n))))
    return g(m, 2) - 2
print(bonusCt2(6))
```