fullName:_____andrewID:_____
recitationLetter:_____ Lecture (1, 2, or 3)_____

Quiz8 version A

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

- You may not unstaple any pages.
- 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 this in with your paper quiz, and we will not grade it.
- 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 not use any concepts we have not covered in the notes this semester. We may test your code using additional test cases. Do not hardcode. Assume almostEqual(x, y) and roundHalfUp(n) are both supplied for you.You must write all other helper functions you wish to use.

Multiple Choice [2pt ea, 12pts total]

Clearly place an X in the blank next to the letter of each correct answer for the following 6 questions.

MC1: Which of the following is NOT a property of a set s in Python?
____ A) (v in s) is O(1)
____ B) s.add(v) is O(1)
____ C) set(list(s)) == s
____ D) All values in s must be immutable
____ E) The values in s are not ordered
____ F) None of these (that is, these ALL are properties of a set s)

MC2: Which of the following is NOT a property of a dictionary d in Python?
____ A) d.get(key) is O(1)
____ B) d[key] = value is O(1)
____ C) d.get(key, 42) returns 42 if and only if (key in d) returns False
____ D) All keys in d must be immutable
____ E) None of these (that is, these ALL are properties of a dictionary d)

MC3: Which of the following is the worst-case big-oh runtime of linear search?

____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

MC4: Which of the following is the worst-case big-oh runtime of binary search?

____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

MC5: Which of the following is the worst-case big-oh runtime of selection sort?

____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

MC6: Which of the following is the worst-case big-oh runtime of merge sort?

____ A) O(1)
____ B) O(logN)
____ C) O(N)
____ D) O(NlogN)
____ E) O(N**2)
____ F) None of these

Short Answer [3pts ea, 24pts total]

Clearly write your answer in the blank for the following 8 questions. For simplicity, assume that 2**10 == 1000.

SA1: What is the maximum number of values linear search must check to find a value in a list with 2 billion values?

_____

SA2: What is the maximum number of values binary search must check to find a value in a list with 2 billion values?

_____

SA3: How many passes will selection sort require to sort a list of 4000 values?

_____

SA4: How many passes will merge sort require to sort a list of 4000 values?

_____

SA5:If on a given computer, selection sort takes 4 seconds to a sort a list with N values, **roughly** how long will it take on the same computer to sort a list with 7N values?

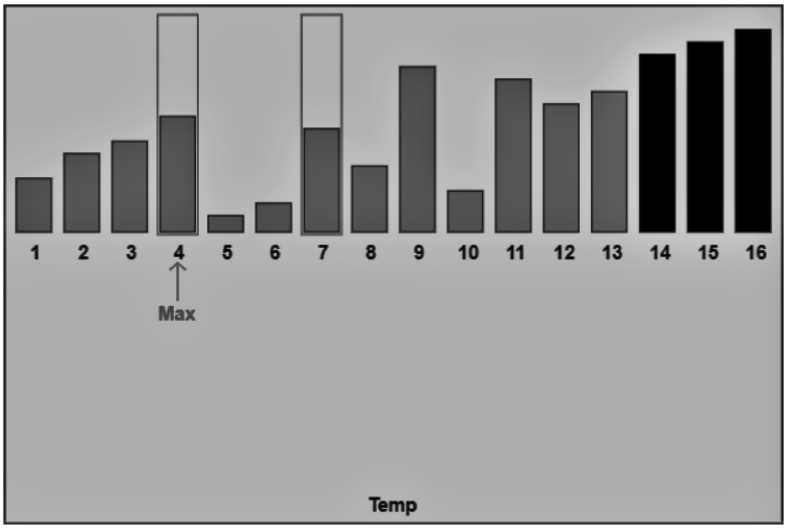
_____

SA6: If on a given computer, merge sort takes 4 seconds to a sort a list with N values, **roughly** how long will it take on the same computer to sort a list with 7N values?

_____

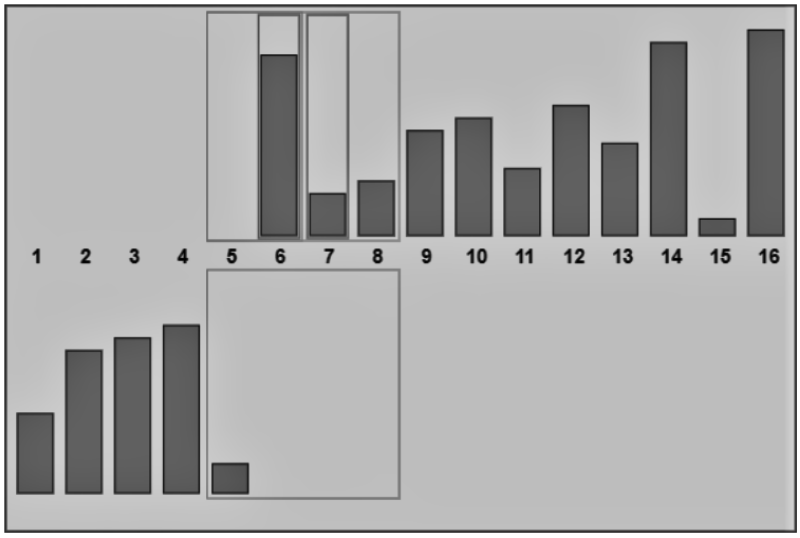
SA7: In the picture below of xSortLab running selection sort, what are the indexes of the next two values to be swapped? (Use xSortLab's indices, which begin at 1 rather than 0)

_____



SA8:In this picture below of xSortLab running merge sort, what is the index of the next value to be copied? (Use xSortLab's indices, which begin at 1 rather than 0)

_____



CT1: Code Tracing [8pts]

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

```
import copy
class A:
    def __init__(self):
        self.L = [[0]]
def ct1():
    a1 = A()
    a2 = A()
    a3 = a1
    a4 = copy.copy(a1)
    a1.L = [[1]]
    a2.L[0].append(2)
    a3.L[0].append(3)
    print(a1.L, a2.L, a3.L, a4.L)
ct1()
```

Free Response 1: mostCommonValues(d) [24pts]

Write the function mostCommonValues(d) that takes a dictionary d and returns a set of all the values (not the keys) that occur the most times in d. For full credit, your function must run in O(N) time (assuming d has N keys). Slower functions will receive a -10 deduction. Functions using lists or tuples will receive 0 credit.

```
def testMostCommonValues():
    print('Testing mostCommonValues()...', end='')
    d = { 'a':1, 'b':2, 'c':1, 'd':3, 'e':3 }
    assert(mostCommonValues(d) == {1, 3})
    assert(mostCommonValues({}) == set())
    print('Passed!')  #Note that we will use additional test cases

testMostCommonValues()
```

Free Response 2: Marble class [32pts]

Write the Marble class to pass the following test cases (or any very similar ones). Read through all the test cases first!

```
# A Marble takes a string (not a list) of comma-separated color names
m1 = Marble('Pink,cyan,CYAN')
assert(m1.colorCount() == 2) # pink and cyan
# The .getColors method should return a set of unique colors present in the marble
assert(m1.getColors() == {'cyan', 'pink'})
# Note that these colors are lowercase.  Ignore case and duplicates
m2 = Marble('Red,Orange,yellow,GREEN,green')
assert(m2.getColors() == {'yellow','red','orange','green'})
assert(m2.colorCount() == 4) #Still just four colors

m3 = Marble('cyan')
# You can add one new color at a time:
assert(m3.addColor('cyan') == False) # Return False if the color is already present
assert(m3.addColor('PINK') == True) # Return True if we added a new color
assert(m3.colorCount() == 2)
assert(m3.getColors() == {'cyan', 'pink'})
assert((m3.getColors() == m1.getColors()) and (m3.getColors != m2.getColors()))
# Once more, but with a new color:
assert(m3.addColor('light green') == True) # True means the color was added!
# and so these all change:
assert(m3.colorCount() == 3)
assert(m3.getColors() == {'cyan', 'pink', 'light green'})

# Lastly, all marbles start out perfect
assert(m1.status == 'perfect')
# ...unless you drop them
m1.drop()
assert(m1.status == 'cracked')
# and they break if you drop them twice
m1.drop()
assert(m1.status == 'broken')
m1.drop()
assert(m1.status == 'broken') # still broken
assert(m2.status == 'perfect') # m2 is still perfect though
```

bonusCT: Code Tracing [2pts]

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

```
def bonusCt(L):
    M = [ (L, 0) ]
    N = [ ]
    while M != [ ]:
        v,c = M.pop(0)
        if isinstance(v, list):
            for u in v: M.append((u, c+1))
            if (c > 2): N.append((u, c+1))
    return N
print(bonusCt([1, [2, [3, [4], [5, [6]]]]]))
```