fullName:_____

andrewID:_____

recitationLetter:_____

15-112 F22

Quiz8 version B

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.
- You may not ask questions during the quiz, except for English-language clarifications. If you are unsure how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) we have not covered in the notes this semester.
- You may not use recursion.
- 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.
- **Write your answers entirely inside the boxes!**

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**2)
- ○ O(2**n)
- ○ O(nlogn)
- ○ O(logn)
- ○ O(n)

MC2. 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**2)
- ○ O(2**n)
- ○ O(nlogn)
- ○ O(logn)
- ○ O(n)

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

- ○ O(n**2)
- ○ O(2**n)
- ○ O(nlogn)
- ○ O(logn)
- ○ O(n)

MC4. What is the efficiency of binary search?
Select the best answer (fill in one circle).

- ○ O(n**2)
- ○ O(2**n)
- ○ O(nlogn)
- ○ O(logn)
- ○ O(n)

True/False [2pts ea, 8 total]

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

TF1. In 15-112, we don't care about lesser-order terms, so O(N**2 - 10000N + 100) simplifies to O(N**2)

◯ True

◯ False

TF2. In 15-112, we don't care about constant coefficients, so O(100N**2) simplifies to O(N**2)

◯ True

◯ False

TF3. Dictionaries contain key-value pairs. Keys must be immutable, but values may be mutable or immutable.

◯ True

◯ False

TF4. Values in sets are unordered and must be unique and immutable

◯ 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.

```
def ct1(L):
    a = set()
    b = set()
    c = dict()
    d = c
    for item in L:
        if item in a:
            b.add(item)
        a.add(item)

    c[L[0]] = a
    c[L[1]] = b
    c[L[2]] = len(a)
    c[L[3]] = len(b)

    x = (d == c)
    c[L[4]] = x
    return c

print(ct1(['n', 5, True, True, (3,5), 'n']))
```

Free Response 1: findChanges(original, new) [30 points]

Background: The profs just downloaded the newest list of students enrolled in 112, and need to know who (if anyone) has been added to the class and who (if anyone) has dropped it. They also have access to the original list of students enrolled at the beginning of the semester. Each list contains student names as strings, like so:

```
original = ["Jimothy Stego", "Kimchee Wiggles", "Keanu Reeves", "Daisy Ridley"]
new = ["Luz Noceda", "Jimothy Stego", "Kimchee Wiggles",
       "Raine Whispers", "Daisy Ridley"]
```

Write the nondestructive function findChanges(original, new) that takes two unsorted lists of names and returns a tuple containing two sets: The first contains the names of anyone who added the course, and the last contains the names of those who dropped the course. For the example above:
`assert(findChanges(original, new) == ({"Luz Noceda", "Raine Whispers"} , {"Keanu Reeves"})`

...because Luz and Raine joined the class, and Keanu dropped the class.
See the test cases for examples!

Efficiency: Your solution must have an efficiency of O(n) or better, where n is the length of the new list. Assume that the original list and the new list are approximately the same size.

```
def testFindChanges():
    original = ["Jimothy Stego", "Kimchee Wiggles", "Keanu Reeves", "Daisy Ridley"]
    new = ["Luz Noceda", "Jimothy Stego", "Kimchee Wiggles",
           "Raine Whispers", "Daisy Ridley"]
    (added, dropped) = findChanges(original, new)
    assert(added == {"Luz Noceda", "Raine Whispers"})
    assert(dropped == {"Keanu Reeves"})

    original = ["A", "B", "C", "D"]
    new = ["A", "B", "D", "E"]
    assert(findChanges(original, new) == ({"E"}, {"C"}))

    original = ["A", "B", "C", "D"]
    new = ["X", "A", "D", "Y", "C", "B", "Z"]
    assert(findChanges(original, new) == ({"X", "Y", "Z"}, set()))

    original = ["A", "B", "C", "D"]
    new = ["D", "C", "B", "A"]
    assert(findChanges(original, new) == (set(), set()))
```

Begin your FR1 answer here

Free Response 2: Shop class [40 points]

Write the Shop class so that it passes the following test cases. Your class should also pass similar test cases (so do not hardcode or assume that shops can only buy/sell lemons and potatoes). Make reasonable assumptions.

```
def testShopClass():
    #Create a grocery with $100 and no items in stock
    grocery = Shop(100)
    assert(grocery.getMoney() == 100)
    assert(grocery.getItems() == dict())

    #The grocery buys 5 lemons for $2 each
    assert(grocery.buy("lemon", 5, 2) == "Bought 5")
    #The grocery buys 20 potatoes for $1 each
    assert(grocery.buy("potato", 20, 1) == "Bought 20")

    #The grocery now has 5 lemons and 20 potatoes
    assert(grocery.getItems() == {"lemon" : 5,
                                  "potato" : 20})
    #The grocery has $70 left
    assert(grocery.getMoney() == 70)

    #Not enough money to buy golden eggs...
    assert(grocery.buy("goldenEgg", 3, 50) == "Can't afford!")
    #Items and money are unchanged
    assert(grocery.getItems() == {"lemon" : 5, "potato" : 20})
    assert(grocery.getMoney() == 70)

    #Sell 15 potatoes for $5 each
    assert(grocery.sell("potato", 15, 5) == "Sold 15")
    assert(grocery.getItems() == {"lemon" : 5, "potato" : 5})
    assert(grocery.getMoney() == 145)

    #Sell 5 lemons for $3
    assert(grocery.sell("lemon", 5, 3) == "Sold 5")
    assert(grocery.getItems() == {"lemon" : 0, "potato" : 5})
    assert(grocery.getMoney() == 160)

    #Can't sell something you don't have...
    assert(grocery.sell("waffle", 5, 2) == "Not enough in stock!")

    #Can't sell 10 potatoes
    assert(grocery.sell("potato", 10, 3) == "Not enough in stock!")
    assert(grocery.getItems() == {"lemon" : 0, "potato" : 5})
    assert(grocery.getMoney() == 160)
```

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 f(s):
    s, d, prevc = s.replace(' ',''), dict(), 'x'
    for c in s:
        d[c], prevc = prevc, c
    return ''.join([d.get(str(n), '') for n in range(10)])

def bonusCT1(s):
    while (len(f(s)) > 1):
        s = f(s)
    return s
print(bonusCT1('Maine has 14 counties, and 432 towns.'))
```