fullName:_____andrewID:_____ recitationLetter:_____

15-112 N23

Quiz4 version A (40 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.
- You may not ask questions during the quiz, except for English-language clarifications. Read problem statements carefully and if you are unsure of how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) which we have not covered in the notes in week 4.
- You may not use recursion.
- We may test your code using additional test cases. Do not hardcode.
- 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 [10pts total]

MC1. [2pt] What is the efficiency of the following code? Select the best answer (fill in one circle).

```
def bigOh1(L):
    if len(L) > 0:
        return L.pop()
    else:
        return None
```

- ○ O(1)
- ○ O(N)
- ○ O(N^2)
- ○ O(logN)
- ○ O(NlogN)

MC2. [2pt] What is the efficiency of the following code? Select the best answer (fill in one circle).

```
def bigOh2(L):
    d = dict()
    s = set(L)
    for value in s:
        if value in d:
            d[value] = d[value] + 1
        else:
            d[value] = 0
    return d
```

- ○ O(1)
- ○ O(N)
- ○ O(N^2)
- ○ O(logN)
- ○ O(NlogN)

MC3. [2pt] What is the efficiency of the following code? Select the best answer (fill in one circle).

```
def bigOh3(L):
    n = len(L)
    x = y = 0
    while (x < n**2):
        while (y < n**2):
            print("y")
            y += 3
        print("x")
        x += 4
```

- ○ O(1)
- ○ O(N)
- ○ $O(N^2)$
- ○ $O(N^3)$
- ○ $O(N^4)$

MC4. [2pt] What does the following code print? Select the best answer (fill in one circle).

```
class A:
    d = {1:2, 1:3}
    def __init__(self, x, y):
        self.x = x
        self.y = y
        A.d[x] = A.d.get(x,y) + 1

# What does this print?
a = A(1,1)
print(A.d)
```

- ○ `{1:2, 1:3}`
- ○ `{1:3, 1:4}`
- ○ `{1:2}`
- ○ `{1:3}`
- ○ `{1:4}`
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

MC5. [1pt each] Given a set `S = {1,2,3,4}`, mark the following as mutating, non-mutating, a syntax error, or a runtime error. Select the best answer (fill in one circle):

```
S.add((1,2))
```

- ○ mutating
- ○ non-mutating
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

```
S & {5,6}
```

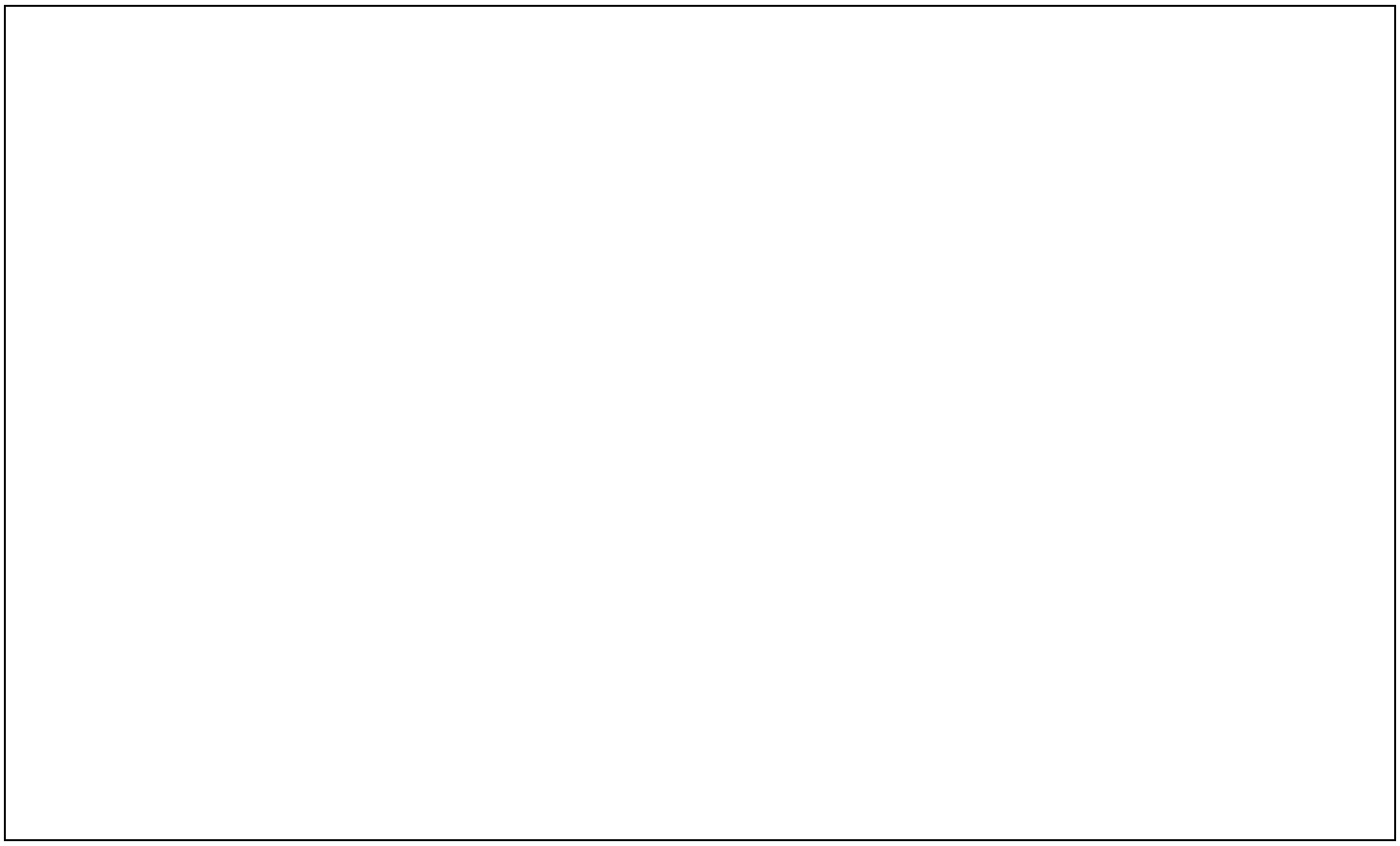
- ○ mutating
- ○ non-mutating
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

CT1: Code Tracing [12pts]

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

```
def ct1(S):
    x = set()
    y = set()
    for elem in S:
        if elem in x:
            y.add(elem)
        x.add(elem)
    print(f'x:{x}')
    print(f'y:{y}')

S = 'bananas'
ct1(S)
```



CT2: Code Tracing [12pts]

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

```
def ct2(L):
    d = dict()
    for i in range(len(L)):
        for k in L[i]:
            if k in d:
                d[k].add(i)
            else:
                d[k] = {i}
    return d

d0 = { 3:'z', 1:'x' }
d1 = { 4:'y' }
d2 = { 2:'z', 1:'z', 4:'w' }
print(ct2([d0, d1, d2]))
```

Free Response 1: findNeighbors(L) [34 points]

Write the function findNeighbors(L) that takes a rectangular 2D list of integers `L` and returns a dictionary mapping each integer value `v` in `L` to a set of all of its neighbors. If a number `v` appears more than once in `L`, the set of all of its neighbors should include all adjacent integers to any occurrence of `v` in `L`. For example:

```
L = [[ 3, 2, 5],
     [ 4, 3, 4],
     [ 1, 3, 3]]

findNeighbors(L) == {
    1 : { 3, 4 },
    2 : { 3, 4, 5 },
    3 : { 1, 2, 3, 4, 5 },
    4 : { 1, 2, 3, 5 },
    5 : { 2, 3, 4 }
}
```

Important notes:

- Each cell has up to 8 adjacent cells (including diagonals, just as in wordSearch)
- If L is empty, you should return the empty set.
- **The function should be non-mutating and you cannot use copy.copy, copy.deepcopy, or L.copy.**

Carefully read the test cases on the following page:

```
def testFindNeighbors():
   L =  [[ 3, 2, 5],
         [ 4, 3, 4],
         [ 1, 3, 3]]

   assert(findNeighbors(L) == {
         1 : { 3, 4 },
         2 : { 3, 4, 5 },
         3 : { 1, 2, 3, 4, 5 },
         4 : { 1, 2, 3, 5 },
         5 : { 2, 3, 4 }
       })

   L = [[ 1, 2, 3, 4],
        [ 5, 6, 7, 8]]

   assert(findNeighbors(L) == {
         1 : { 2, 5, 6 },
         2 : { 1, 3, 5, 6, 7 },
         3 : { 2, 4, 6, 7, 8 },
         4 : { 3, 7, 8 },
         5 : { 1, 2, 6 },
         6 : { 1, 2, 3, 5, 7 },
         7 : { 2, 3, 4, 6, 8 },
         8 : { 3, 4, 7 }
       })
```

Begin your FR1 answer on the following page.

Begin your FR1 answer here

You may continue your FR1 answer here

Free Response 2: Polynomial Class [32 points]

Write the class Polynomial along with the required methods so that the following test cases pass. Do not hardcode any test cases. Your implementation should pass similar test cases. You must use OOP properly. Do not add unnecessary methods to the class.

Carefully read the test cases:

```
def testPolynomialClass():

    f = Polynomial([2,3,1]) # 2x**2 + 3x + 1
    assert(f.evalAt(4) == 2*4**2 + 3*4 + 1) # returns f(4), which is 45
    assert(f.evalAt(5) == 2*5**2 + 3*5 + 1) # returns f(5), which is 66
    assert(f.getCoefficient(0) == 1) # get the x**0 coefficient
    assert(f.getCoefficient(1) == 3) # get the x**1 coefficient
    assert(f.getCoefficient(2) == 2) # get the x**2 coefficient
    assert(f.getCoefficient(33) == 0) # assume leading 0's...

    g = f.times(10) # g is a new polynomial, which is 10*f
                    # just multiply each coefficient in f by this value
                    # so g = 20x**2 + 30*x + 10
    assert(g.getCoefficient(0) == 10) # get the x**0 coefficient
    assert(g.getCoefficient(1) == 30) # get the x**1 coefficient
    assert(g.getCoefficient(2) == 20) # get the x**2 coefficient
    assert(g.getCoefficient(33) == 0) # assume leading 0's...
    assert(g.evalAt(4) == 20*4**2 + 30*4 + 10) # returns g(4), which is 450

    h = Polynomial([20,30,10])
    assert(h == g)          # polynomials are equal if their coefficients are equal
    assert(h != 'a string') # should not crash!
```

Begin your FR2 answer on the following page.

Begin your FR2 answer here

You may continue your FR2 answer here

bonusCT: Code Tracing [2pt]

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

```
def bonusCT(s):
    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)])
    while (len(f(s)) > 1): s = f(s)
    return s

s = 'Maine has 14 counties, and 432 towns.'
print(bonusCT(s))
```