fullName:_____________________________andrewID:_____________________________ recitationLetter:______

15-112 S23

Midterm2 version A (80 min)

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

- You may not unstaple any pages.
- Failure to hand in an intact exam will be considered cheating. Discussing the exam with anyone in any way, even briefly, is cheating. (You may discuss it only once the exam 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 exam, 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 1-11.
- Assume almostEqual(x, y), rounded(n), and distance(x1, y1, x2, y2) are both supplied for you. You must write all other helper functions you wish to use, unless we specify otherwise.

Sign your initials below to assert that you will not discuss or share information about the exam with anyone in any way, even briefly, until the exam has been posted on the course website.

x______________

Multiple Choice [10pts total]

MC1. What is the worst-case big-O runtime of linear search, where N is the length of the list? Select the best answer (fill in one circle).

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

MC2. What is the worst-case big-O runtime of selection sort, where N is the length of the list? Select the best answer (fill in one circle).

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

MC3. For a sorted list, what is the worst-case big-O runtime of binary search, where N is the length of the list?
Select the best answer (fill in one circle).

○ O(1)

○ O(logN)

○ O(N**2)

○ O(NlogN)

○ O(N)

MC4. What is the worst-case big-O runtime of merge sort, where N is the length of the list?
Select the best answer (fill in one circle).

○ O(1)

○ O(logN)

○ O(N**2)

○ O(NlogN)

○ O(N)

MC5. Which one of the following is True?
Select the best answer (fill in one circle).

○ Sorting a list and performing a binary search is usually much faster than performing one linear search

○ Checking if a key is in a dictionary is O(1)

○ Converting a list to a set and then checking for membership is usually much faster than performing one linear search

○ Tuples cannot be hashed

○ The best sorting algorithms run in O(logN)

CT1: Code Tracing [6pts]

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

```
def ct1(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 = { 1:'d', 3:'b' }
d1 = { 4:'a' }
d2 = { 1:'b', 2:'b', 4:'c' }
print(ct1([d0, d1, d2]))
```

CT2: Code Tracing [6pts]

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

```
def ct2(n):
    if n == 0:
        return 0
    else:
        return 10 * ct2(n//100) + n//10%10

print(ct2(123456))
```

CT3: Code Tracing [6pts]

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

```
def ct3(L):
    d = dict()
    for i in range(len(L)):
        k = A(L[i], i)
        d[k] = i  # Hint: The keys will be strings,
                  # as returned by k.__repr__()
    a1 = A(1, 4)
    print(d)
    print(d.get('A(3)', True))
    print(d.get(a1, True))


class A:
    def __init__(self, i, j):
        self.x = i + j

    def __repr__(self):
        return f'A({self.x})'

    def __eq__(self, other):
        return str(self) == str(other)

    def __hash__(self):
        return hash(str(self))

L = [1,2,3]
ct3(L)
```

Free Response 1: findChanges [16pts]

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', 'Kimchee', 'Keanu', 'Daisy']
new = ['Luz', 'Jimothy', 'Kimchee', 'Raine', 'Daisy']
```

Write the non-mutating 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', 'Raine'} , {'Keanu'})
```

...because Luz and Raine joined the class, and Keanu dropped the class.
Efficiency: In order to receive full credit, your solution must have an efficiency of O(N) or better, where N is the length of the longer input list (original or new). Assume that the original list and the new list are approximately the same size.

```
def testFindChanges():
    original = ['Jimothy', 'Kimchee', 'Keanu', 'Daisy']
    new = ['Luz', 'Jimothy', 'Kimchee', 'Raine', 'Daisy']
    (added, dropped) = findChanges(original, new)
    assert(added == {'Luz', 'Raine'})
    assert(dropped == {'Keanu'})

    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()))

    assert(findChanges([],[]) == (set(), set()))
```

Begin your FR1 answer here

You may continue your FR1 answer here

Free Response 2: getStringCounts [16pts]

Write the function getStringCounts(L) where L is a list that may contain sublists (which themselves may further contain sublists). Non-list elements in L are guaranteed to be strings. getStringCounts should return a dictionary mapping each string to the number of times it appears anywhere inside L.
For example:
getStringCounts(['a', ['b', 'c'], 'a', [['a'], 'c']])
should return {'a' : 3, 'b' : 1, 'c' : 2}

Your solution must use recursion. If you use any loops, comprehensions, or iterative functions or methods (including .count()), you will receive no points on this problem.

Hint: You may wish to first flatten the list recursively, then use a separate recursive function to get the dictionary.

Note: If in doubt, you can earn half credit by writing a function that works for a 1D list.

```
def testGetStringCounts():
    assert(getStringCounts(['a', ['b', 'c'], 'a', [['a'], 'c']]) == {
        'a': 3,
        'b': 1,
        'c': 2,
      })

    assert(getStringCounts([[['a', 'A']], 'a']) == {
        'a': 2,
        'A': 1,
      })

    assert(getStringCounts(['marf', 'Jimothy', '[1string]', 'taters']) == {
        'marf': 1,
        'Jimothy': 1,
        '[1string]': 1,
        'taters': 1
      })

    assert(getStringCounts([[[[]]], [[], 'a'], 'b', []]) == {
        'a': 1,
        'b': 1,
      })

    assert(getStringCounts([]) == dict())
```

Begin your FR2 answer here

You may continue your FR2 answer here

You may continue your FR2 answer here

Free Response 3: Matrix class [20pts]

Write the class Matrix so that the following test code passes. Hardcoding will not receive any credit. Your code must work for valid arguments in general, not just for the specific values in the test cases. Assume we will only check for valid arguments unless the tests specify otherwise. Note: You will lose points if you create unnecessary/unused methods.

```python
def testMatrixClass():
    print('Testing Matrix class...', end='')
    m1 = Matrix([[1,2,3],   # We're guaranteed the input list will be rectangular
                 [4,5,6]])
    assert(m1.getDims() == (2, 3))  # 2 rows, 3 columns
    assert(m1.getRow(0) == [1,2,3])
    assert(m1.getCol(0) == [1,4])
    assert(m1.getRow(5) == m1.getCol(42) == None) # handle out-of-bounds indexes

    assert(str(m1) == '2x3 Matrix: [[1, 2, 3], [4, 5, 6]]')
    m2 = Matrix([[10,20,30],[40,50,60]]) # make another matrix
    assert(str(m2) == '2x3 Matrix: [[10, 20, 30], [40, 50, 60]]')
    # This should work for lists too
    assert(str([m2]) == '[2x3 Matrix: [[10, 20, 30], [40, 50, 60]]]')

    # addMatrix adds each value in m1 to the corresponding value in m2
    m3 = m1.addMatrix(m2)
    assert(str(m3) == '2x3 Matrix: [[11, 22, 33], [44, 55, 66]]')
    # Be sure the previous operation was non-mutating:
    assert(str(m1) == '2x3 Matrix: [[1, 2, 3], [4, 5, 6]]')
    m4 = Matrix([[1]])
    assert(str(m4) == '1x1 Matrix: [[1]]')
    assert(m1.addMatrix(m4) == None) # handle mismatched dimensions when adding

    assert(m3 == Matrix([[11, 22, 33], [44, 55, 66]]))
    assert(m3 != m4)
    assert(m3 != 42) # don't crash here
    print('Passed!')
```

Begin your FR3 answer on the following page

Begin your FR3 answer here

You may continue your FR3 answer here

You may continue your FR3 answer here

Free Response 4: canBeNumberLadder [20pts]

Note: this is a backtracking problem. To receive credit, you must use backtracking properly. In particular, you must use recursion, and you must check for legality as you make each "move," instead of expanding every possible choice and only checking for correctness at the end.
Also note that so long as you use backtracking properly, you may use 'for' or 'while' loops here.

Background: we will say that a list of integers forms a "number ladder" (a coined term) if each value (except the first two values) is the sum of the previous two values.
For example, `L = [8, -3, 5, 2]` is a number ladder because:

- `8 + -3 == 5`
- `-3 + 5 == 2`

Similarly, `L = [8, -3, 2, 5]` is NOT a number ladder because `8 + -3 != 2`

With this in mind, and using backtracking, write the function canBeNumberLadder(L) that takes a list of integers and returns True if the list can be reordered in some way that it becomes a number ladder, and False otherwise. **Note that any list shorter than length 3 cannot be a number ladder.**

For example, `canBeNumberLadder([-3, 2, 5, 8])` returns True, because this list can be reordered into `[8, -3, 5, 2]`, which we already saw is a number ladder.

Also, `canBeNumberLadder([9, 4, 6])` returns False because there are 6 ways to order the list `[9, 4, 6]` and none of those orderings is a number ladder.

Note that you may want to make `canBeNumberLadder(L)` a wrapper function around a recursive helper function.

```
def testCanBeNumberLadder():
    assert(canBeNumberLadder([-3, 2, 5, 8]) == True)
    assert(canBeNumberLadder([9, 4, 6]) == False)
    assert(canBeNumberLadder([9, 4, 5]) == True) # [4, 5, 9]
    assert(canBeNumberLadder([-5, -3, -1, 2, 7]) == True) # [7, -5, 2, -3, -1]
    assert(canBeNumberLadder([9, 4, 6]) == False)
    assert(canBeNumberLadder([8, 3]) == False)
    assert(canBeNumberLadder([]) == False)
```

Begin your FR4 answer on the following page

Begin your FR4 answer here:

Continue your FR4 answer here

Continue your FR4 answer here

bonusCT1: Code Tracing [1pt]

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 bonusCT1(d, L):
    for v in L[:]:
        L *= v
    for v in L[:]:
        d[v] = 1 + d.get(v,42)
    return sum([d[v] for v in d])

print(bonusCT1({1:2, 2:3, 3:1}, list(range(1,5))))
```

bonusCT2: Code Tracing [1pt]

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 g(x):
    if x > 10:
        return g(2)
    if x < 5:
       print(x, end = "")
    if x < 1:
       return 1
    return x + g(x - 2) if (x%2 == 0) else x + g(x + 2)

def bonusCT2(x):
    print(x, end = "")
    return g(x + 1)

print(f" --> {bonusCT2(2)}")
```