15-112 Spring 2023 Lecture 3/4
Quiz 2
20 minutes

Name: _____

Andrew ID: _____@andrew.cmu.edu

Section: _____

- **You may not use any books, notes, or electronic devices during this quiz.**
- **You may not ask questions about the quiz except for language clarifications.**
- **Show your work on the quiz (not scratch paper) to receive credit.**
- **If you use scratch paper, you must submit it with your andrew id on it, and we will ignore it.**
- **All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.**
- **Do not use these topics: sets/dictionaries, recursion.**
- **You may use almostEqual() and rounded() without writing them. You must write everything else.**

1. CT [10 pts]

Indicate what these print. Place your answers (and nothing else) in the box next to each block of code.

```
def ct1(n, L):
    M = [0] * n
    for j,k,d in L:
        for i in range(j, k+1):
            M[i%n] += d
    return M
print(ct1(3, [(1,2,1), (0,0,2), (-2,-1,4), (2,3,8)]))
```

Note: because this CT used (x%y) where y is negative, and that is outside the scope of 112, this problem was dropped (so everyone got full credit for it).

```
def ct2(L):
    M = L
    M += [2]
    N = M + [3]
    L.append(N.pop(0))
    L = sorted(L)
    M = M + [max(N)]
    M.sort()
    return (M, N)
L = [1]
M, N = ct2(L)
print(L)
print(M)
print(N)
```

2. Free Response: moveNegativesToEnd(L) [40pts]

Write the function moveNegativesToEnd(L) that takes a possibly-empty list L of integers, and mutatingly moves all the negative values in L to the end of the list, while keeping them in the same order. Your function should return the number of negative values in the list (which is unusual, because most mutating functions return None).

For example:

```
L1 = [1, -2, 3, 4, -5]
assert(moveNegativestoEnd(L1) == 2) # there were 2 negatives
assert(L1 == [1, 3, 4, -2, -5])     # and now they are at the end, but
                                    # the order is otherwise unchanged

L2 = [1, 2, 3, 4]
assert(moveNegativestoEnd(L2) == 0)
assert(L2 == [1, 2, 3, 4])

L3 = [-4]
assert(moveNegativestoEnd(L3) == 1)
assert(L3 == [ -4 ])

L4 = [ ]
assert(moveNegativestoEnd(L4) == 0)
assert(L3 == [ ])
```

This page is blank (for your moveNegativesToEnd solution).

3. Free Response: comfyNumbers(L) [40pts]

Background: given a list of non-negative integers `L`, we will say that a value `x` in `L` is "comfy" (a coined term) if there is another value `y` in `L` such that `x` and `y` are not equal but are within 2 of each other.

For example, in the list `[2, 5, 8, 4, 8, 11]`:

- 2 is comfy (within 2 of 4).
- 5 is comfy (within 2 of 4).
- 4 is comfy (within 2 of both 2 and 5).
- 8 is not comfy, even though there are two 8's, because 8 is more than 2 away from the nearest other values (5 and 11).
- 11 is not comfy, because it is more than 2 away from the nearest other value (8).

Thus, the comfy numbers in `L` are: `[2, 5, 4]`.

With this in mind, write the function `comfyNumbers(L)` that takes a list `L` of non-negative integers and returns a new list of all the comfy numbers in `L`, appearing in the same order that they appear in `L`.

Note that two different `x` values can be comfy with the same `y` value. Thus, in the list `[2, 3, 2]`, the comfy values are the entire list `[2, 3, 2]`.

Note: this function must be non-mutating.

```
def comfyNumbers(L):
    pass

############################################################
# Test Function
############################################################
def testComfyNumbers():
    print('Testing comfyNumbers()...', end='')
    assert(comfyNumbers([2, 5, 8, 4, 8, 11]) == [2, 5, 4])
    assert(comfyNumbers([2, 3, 2 ]) == [2, 3, 2])
    assert(comfyNumbers([2, 2, 2, 2, 2, 2 ]) == [ ])

    # Be sure to be non-mutating
    L = [2, 5, 8, 4, 8, 11]
    assert(comfyNumbers([2, 5, 8, 4, 8, 11]) == [2, 5, 4])
    assert(L == [2, 5, 8, 4, 8, 11])

    print('Passed!')

testComfyNumbers()
```

This page is blank (for your comfyNumbers solution).

4. Bonus [5 pts]

Indicate what these print. Place your answers (and nothing else) in the box next to each block of code.

```
def bonusCt1(L):

    while sum(L) < L[0]**L[1]:
        L += L * (sum(L)//4)
    return L
print(len(bonusCt1([2,4])))
```

```
def bonusCt2(L, n):

    while L:
        L, n = L[:-1], 10*n+L[-1]%10
    L = list(str(n))
    return [int('4' + ''.join(L[:i]))
            for i in range(0, len(L), 2)]
print(bonusCt2([1,23,456,78],9))
```