Name: _____ Andrew Id: _____

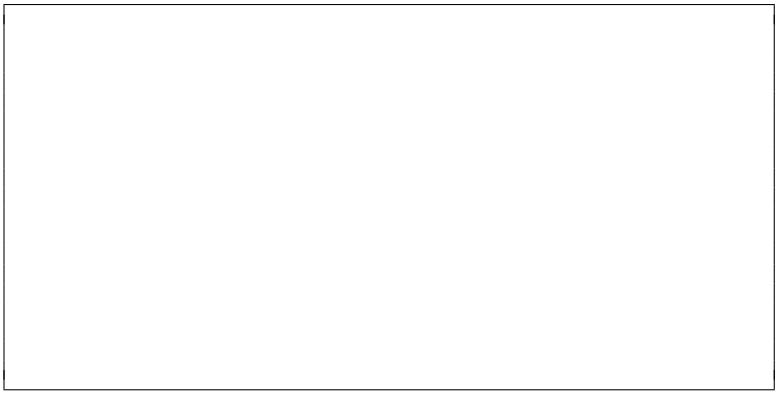
15-112 Spring 2026 Quiz 6

Up to 30 minutes. No calculators, no notes, no books, no computers. Show your work!
Do not use try/except or recursion on this quiz.

1. (5 points) **Code Tracing**: Indicate what the following program prints. Place your answers (and nothing else) in the box next to the code.

```
def ctA(L):
    d1 = {}
    d2 = d1
    for i in range(len(L)):
        key = L[i] % 3
        d1[key] = d1.get(key, set())
        d1[key].add(L[i])
    for n in d2:
        d2[n] = sorted(d2[n])
    return d2

print(ctA([2, 5, 8, 3, 6, 4]))
```



2. **Big O**: In this problem, you will be calculating the efficiency of a provided piece of code. Consider the following example:

```
def bigOSample(n): # N is n
    print("Simple")     # O(1)
    for i in range(n):  # loop runs N times
        print(i)        # O(1)

# BigO Time Efficiency: O(N)
```

(a) (3 points) For the function shown below, write next to each line of the function either the Big-O runtime of the line or the number of times the line loops. Then write the total Big-O runtime of the function in terms of N in part (b). **All answers must be simplified—do not include lower-order terms! For full credit, you must include line-by-line Big-O.**

```
def bigO(L):  # L is an NxN list

    result = set()                      # (1)_____________________________

    for i in range(len(L)):             # (2)_____________________________

        row = L[i]                      # (3)_____________________________

        if row[i] not in result:        # (4)_____________________________

            result.add(row[i])          # (5)_____________________________

            L.append([i]*len(L[0]))     # (6)_____________________________

        else:                           # (7)_____________________________

            temp = list(result)         # (8)_____________________________

            print("repeat:", row[i])    # (9)_____________________________

    return len(sorted(result))          # (10)____________________________
```

(b) (1 point) Write the simplified BigO complexity (i.e. not including lower order terms or coefficients).

BigO Time Efficiency = _____

3. **Free Response**: Improving Efficiency

The following function takes a list `L` and returns the first element that appears exactly once in the list. If no such element exists, it returns `None`.

```
def firstUnique(L):
    for x in L:
        if L.count(x) == 1:
            return x
    return None

# Example Test Cases
assert(firstUnique([2, 3, 2, 4, 3, 5]) == 4)
assert(firstUnique([1, 1, 2, 2]) == None)
assert(firstUnique([7]) == 7)
assert(firstUnique([]) == None)
```

(a) (1 point) What is the Big O time complexity of the function above? Briefly justify your answer.

(b) (4 points) Rewrite the function so that it runs in O(n) time.

4. (6 points) **Free Response**: Missing Numbers

   Write the function `missingNumbers(L, n)` that takes a list of integers `L` and an integer `n` and returns a set of all integers from `1` to `n` (inclusive) that do not appear in `L`. **Your solution must run in O(N) time.**

   Consider the following example:

```
assert(missingNumbers([1, 2, 4, 6], 6) == {3, 5})
assert(missingNumbers([2, 2, 2], 3) == {1, 3})
assert(missingNumbers([], 4) == {1, 2, 3, 4})
assert(missingNumbers([1,2,3,4], 4) == set())
```