Name: _____ Andrew Id: _____

15-112 Fall 2025 Quiz 6

Up to 25 minutes. No phones, no calculators, no notes, no books, no computers. Show your work!
Do not try/except or recursion on this quiz.

1. **Code Tracing**: Indicate what the following programs print. Place your answer (and nothing else) in the box next to the code.

   (a) (3 points) CT1

```
def ct1(L, d):
    for i in range(len(L)):
        for j in range(i, len(L)):
            d[L[i]+L[j]] = {i,j}
    return d
print(ct1([2,1,1], {0: [0,0]}))
```

   (b) (3 points) CT2

```
def ct2(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(ct2([3, 'm', True, True, (5,3), 'm']))
```

2. (6 points) **Big-O**

For each function shown below, write the total Big-O runtime of the function in terms of N, the length of the function argument, in the box to the right of the code. All answers must be simplified – do not include lower-order terms! Only the total big-O will be graded.

```
def f1(S): # S is a set, len(S) = N
    P = dict()
    for i in S:
        avg = sum(S) / len(S)
        if i < avg:
            P[i] = avg
    return P
```

```
def f2(L): # L is list, len(L) = N
    N = len(L)
    for i in range(0, N, 2):
        for j in range(i+1, N):
            L[i] += L[j]//2
```

```
def f3(L): # L is list, len(L) = N
    i = len(L)-1
    t = 0
    while i > 1:
        t += sum(L[i:i+3])
        print(i)
        i //= 3
    return t
```

3. (8 points) **Free Response: Six-Seven Twins**

Two **distinct** positive integers `a` and `b` are *Six-Seven-Twins* (coined term) if they contain the same number of 6s and the same number of 7s in their decimal representation, and each number contains at least one 6 and one 7. Other digits, if any, do not matter.

For example:

- 1667 and 7606 are Six-Seven-Twins (each has two 6s and one 7)
- 1667 and 7617 are **not** Six-Seven-Twins (different counts of 6s and 7s)
- 177 and 771 are **not** Six-Seven-Twins (no 6s)

With this in mind, write a function `containsSixSevenTwins(L)` that takes a list `L` of positive integers and returns `True` if there exist two **distinct** numbers in the list that are Six-Seven-Twins, and `False` otherwise.

Your solution must run in $O(N)$ time, where $N =$ `len(L)`.

Notes: When analyzing efficiency, assume that scanning the digits of a single number takes constant time. What matters for efficiency is the length of the list, not the number of digits in each number. Your goal is to design an $O(N)$ solution with respect to the number of elements in the list.

Here are some test cases:

```
assert containsSixSevenTwins([67, 76]) == True
assert containsSixSevenTwins([7, 77, 777]) == False
assert containsSixSevenTwins([166, 606, 700]) == False
assert containsSixSevenTwins([1, 2, 3]) == False
assert containsSixSevenTwins([671, 761, 42]) == True
assert containsSixSevenTwins([123, 890, 45, 67]) == False
assert containsSixSevenTwins([676, 676]) == False   # same value twice: not distinct
assert containsSixSevenTwins([676]) == False        # only one element
assert containsSixSevenTwins([666777, 67677, 776766]) == True
```

Complexity of Python Built-ins

General	
Function/Method	**Complexity**
Print	O(N)
Range in Iteration	Number of iterations = (end - start)/step

Strings: s is a string with N characters	
Function/Method	**Complexity**
Len	O(1)
Membership	O(N)
Get single character	O(1)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
Chr() and Ord()	O(1)
Concatentation	O(len(s1) + len(s2))
Character Type Methods	O(N)
String Edit Methods	O(N)
Substring Search Methods	O(N)

Lists: L is a list with N elements	
Function/Method	**Complexity**
Len	O(1)
Append	O(1)
Extend	O(K)
Concatentation with +=	O(K), where K is the size of the list to concatenate
Concatentation with +	O(N + K), where K is the size of the list to concatenate
Membership Check	O(N)
Pop Last Value	O(1)
Pop Intermediate Value	O(N)
Count values in list	O(N)
Insert	O(N)
Get value	O(1)
Set value	O(1)
Remove	O(N)
Get slice	O(end - start)
Get slice with step	O((end - start)/step)
Sort	O(N log (N))
Multiply	O(N*D)
Minimum	O(N)
Maximum	O(N)
Copy	O(N)
Deep Copy	O(N*M)

Sets: s is a set with N elements	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Adding an Element	O(1)
Removing an Element	O(1)
Union	O(len(s) + len(t))
Intersection	O(min(len(s), len(t)))
Difference	O(len(s))
Clear	O(len(s))
Copy	O(len(s))

Dictionaries: d is a dictionary with N key-value pairs	
Function/Method	**Complexity**
Len	O(1)
Membership	O(1)
Get Item	O(1)
Set Item	O(1)
Delete Item	O(1)
Clear	O(N)
Copy	O(N)