15-112 Fall 2022 Lecture 3
Quiz 7A
45 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: recursion.**
- **You may use almostEqual() and rounded() without writing them. You must write everything else.**

Do not write below here

Question	**Points**	**Score**
1. CT	12	
2. FR: getSingletons	25	
3. FR: averageMap	35	
4. Big O	18	
5. Sorting	10	
3. Bonus	5 (bonus)	
TOTAL	100	

1. CT [12 pts, 6 pts each]

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

```
def ct1(d):
    e = dict()
    for k in sorted(d):
        n = (d[k]**2)%10
        e[n] = k
    return e
print(ct1({5:4, 7:8, 1:2, 3:6}))
```

```
def ct2(L):
    s = set()
    for v in L:
        if isinstance(v, dict):
            for k in v:
                s.add(v[k])
        else:
            s = s.union(set(v))
    return s
print(ct2(['AB', {2:'AC', 'BC':3}, [4,4,4]]))
```

2. Free Response: getSingletons(L) [25 pts]
Write the function getSingletons(L) that takes a list L of sets, and returns a single set which contains the values that occur in exactly one of the sets in L (we're calling these "singletons").

For example:

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

We see that 1, 3, and 4 are each in more than one set in the list L, but both 2 and 5 occur in only one set in L. Thus, for this list:

```
assert(getSingletons(L) == {2,5})
```

Here are two more test cases:

```
M = [ {1}, {2}, {3}, {1,2}, {3,4} ]
assert(getSingletons(M) == {4})

N = [ {1}, {2}, {3}, {1,2}, {3,4}, {1,4} ]
assert(getSingletons(N) == set())
```

Important note: assume that L is of length N, and that each set in L contains no more than 10 values. Your solution must run in O(N).

3. Free Response: averageMap(L) [35 pts]

Background: This problem works with a list L of dicts that each map an integer to a possibly-empty list of integers. For example, here is one such list:

```
L = [ {1:[2,3], 7:[5], 8:[9], },
      {1:[4],   7:[1,1,1], 6:[]} ]
```

With that, write the function averageMap(L) that takes such a list, and returns a dict mapping each integer key K in any of the dicts in L to the integer average value (using //) calculated from all lists d[K], where d is each dictionary in L. Ignore keys that only map to empty lists.

For example, consider each key in any dict in L from above:

- For the key 8, there is only one dict with 8 as a key, and the average of that one list is 9. So the result maps 8 to 9.
- For the key 1, there are two dicts with 1 as a key, and they map to the lists [2,3] and [4]. The integer average of all these values is (2+3+4)//3 which is 3. So the result maps 1 to 3.
- For the key 7, again, there are two dicts with 7 as a key, and they map to the lists [5] and [1,1,1], which average to (5+1+1+1)//4 which is 2. So the result maps 7 to 2.
- For the key 6, there is only one dict with 6 as a key, and it maps to the empty list [], so we ignore this key.

Thus, for the list L above:

```
assert(averageMap(L) == {1:3, 7:2, 8:9})
```

Here is another test case for you:

```
M = [ {5:[2,3], 4:[9,2,3,4], 3:[]},
      {5:[1],   4:[1],       3:[]} ]
assert(averageMap(M) == {5:2, 4:3})
```

This page intentionally blank for your answer to averageMap(L).

4. Big O [18 pts, 3 pts each]

For each of the following, indicate which Big O family the code runs in (in the worst case). Each function takes a list L, and N is len(L). Circle your answers.

1)

```
def f(L):
    N = len(L)
    M = sorted(L + L)
    return sum(M) // len(M)
```

A) N**2 B) NlogN C) N D) N**0.5 E) logN F) 1

2)

```
def f(L):
    N = len(L)
    for i in range(N):
        for j in range(i+1, N):
            L[i] += L[j]
```

A) N**2 B) NlogN C) N D) N**0.5 E) logN F) 1

3)

```
def f(L):
    N = len(L)
    M = [ ]
    s = set(L)
    for v in s:
        M.append(L.count(v))
    return M
```

A) N**2 B) NlogN C) N D) N**0.5 E) logN F) 1

4)
```
def f(L):
    N = len(L)
    M = [v**2 for v in L]
    return set(L) == set(M)
```

A) N**2   B) NlogN   C) N   D) N**0.5   E) logN   F) 1

5)
```
def f(L):
    N = len(L)
    i = N-1
    while i > 0:
        L[i] += i
        i //= 2
```

A) N**2   B) NlogN   C) N   D) N**0.5   E) logN   F) 1

6)
```
def f(L):
    # assume len(L) >= 10
    N = len(L)
    for i in range(10):
        L[i] *= i
```

A) N**2   B) NlogN   C) N   D) N**0.5   E) logN   F) 1

5. Sorting [10 pts]
State and briefly prove the worst-case Big O for merge sort. Your proof should just be the picture that was drawn in the video in the course notes, along with a short note explaining the number of passes and the steps per pass in terms of N (the length of the list).

6. Bonus [5 pts]

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

```
def bonusCt1(K,V):
    d = { k:v for (k,v) in zip(K,V)}
    r, c = '', 'a'
    while c not in r:
        r, c = r+c, d[c]
    return r
print(bonusCt1('abcbad', 'abadbc'))
```

```
def bonusCt2(L):
    M = [ ]
    for v in L:
        M.extend(list(range(0, 100, v)))
    N = [set() for _ in range(3)]
    for v in M:
        for s in N:
            if v not in s:
                s.add(v)
                break # exit inner loop
    return sorted(N[-1])
print(bonusCt2([2,3,5]))
```