Name:___________________________________ Section:___ Andrew Id: _____________________

15-112 Spring 2017 Quiz9a

*** Up to 30 minutes. No calculators, no notes, no books, no computers. * Show your work! * No recursion**

1. **Big-Oh** [10 pts]:
   What is the worst-case big-oh runtime of each of the following, in terms of N? Assume L is a list of N integers, and x is a positive integer where N = math.log(x, 2). Place your answer (and nothing else) in the box next to the code.

```
def bigOh1(L):
    N = len(L)
    M1 = [L[i]**2 for i in range(1, len(L), 3)]
    M2 = [L[i]**3 for i in range(1, 3)]
    M3 = sorted([x*y for x in L for y in L])
    return sum(sorted(M1 + M2 + M3))
```

```
def bigOh2(L):
    N = len(L)
    R = [ ]
    for k in L:
        M, s, d = copy.copy(L), set(), dict()
        while (M != [ ]):
            s.add(M[0]**2)
            try: d[M.pop()] = M.pop(0) + k
            except: print("Uh oh!")
            M = M[::-1]
        R += [(k + v**3) for v in s]
    return (min(R), max(R))
```

2. **Short Answer** [10 pts]
   Be very brief.

   a. Given the list [4,7,2,3,5], what will the list be after exactly 2 swaps are made in selectionSort, as it works in xSortLab?

   b. State and prove the worst-case big-oh of mergesort. Note that a well-labeled picture can be sufficient proof.

3. **True or False** [10 pts] Circle your answers.

a. True or False — While mergesort is generally faster than bubblesort, if L is already sorted in increasing order, then bubblesort(L) is faster than mergesort(L).

b. True or False — Given some function H, and some value N > 10**5, if:

```
(almostEqual(math.log(N, 2), len(set([H(x) for x in range(N)])))
```

Then H is probably a good hash function.

c. True or False — If (x == hash(x)), then x must be an integer.

d. True or False — A hashtable implementation may require lists even though the hashtable is not allowed to include lists as keys.

e. True or False — In 'ternary search', as opposed to binary search, we divide the list into 3 parts on each pass, and so we can eliminate 2/3rds of the list on each pass. While this will generally reduce the number of passes required to find an element in a sorted list, it does not improve the worst-case big-oh runtime compared to binary search.

4. **Fill in the Blank** [10 pts]

This implementation of mergesort and merge is from the course notes. Fill in the blanks with the missing code.

```
def merge(a, start1, start2, end):
    index1 = start1
    index2 = start2
    length = end - start1
    aux = ____________________________________________
    for i in range(length):

        if ((____________________________________________) or
            ((index2 != end) and (a[index1] > a[index2]))):
            aux[i] = a[index2]
            index2 += 1
        else:
            aux[i] = a[index1]
            index1 += 1
    for i in range(start1, end):
        a[i] = ____________________________________________

def mergeSort(a):
    n = len(a)
    step = 1
    while (step < n):
        for start1 in range(0, n, 2*step):

            start2 = min(________________________________________, n)
            end = min(start1 + 2*step, n)
            merge(a, start1, start2, end)

        ____________________________________________ *= 2
```

Name:___________________________________ Section:___ Andrew Id: _____________________

5. **Free Response: antisocialScore(d, person)** [40 pts]

Recall that friendsOfFriends(d) takes a dictionary d like this:

```
d = dict()
d["fred"]  = set(["betty", "barney"])
d["wilma"] = set(["fred", "betty"])
d["betty"] = set(["barney"])
d["barney"] = set([ ])
```

We will consider a person A's antisocial score to be the number of people B such that A is a friend of B, but B is not a friend of A. The higher the score, the more antisocial the person is. In the example above:

Person	Antisocial Score	Reason
wilma	0	nobody likes wilma, so wilma cannot dislike anyone who likes her
fred	1	wilma likes fred, but fred does not like wilma
betty	2	fred and wilma like betty, but betty does not like them
barney	2	fred and betty like barney, but barney does not like them

With this in mind, write the function antisocialScore(d, person) that takes a dictionary of that form, and a person who you may assume is in the dictionary, and returns the antisocial score of that person. Thus, in the example above:

```
assert(antisocialScore(d, "wilma")  == 0)
assert(antisocialScore(d, "fred")   == 1)
assert(antisocialScore(d, "betty")  == 2)
assert(antisocialScore(d, "barney") == 2)
```

6. **Code Tracing** [10 pts]:Indicate what this prints. Place your answer (and nothing else) in the box below the code.

```
def ct1(n):
    d = dict()
    while (n > 0):
        (i, j, n) = (n%10, n//10%10, n//100)
        d[j] = d.get(j, set())
        d[j].add(i)
    return [(key,sorted(d[key])) for key in sorted(d.keys())]
print(ct1(32421))
```

7. **Reasoning Over Code** [10 pts]:
Find an argument for the following function that makes it return True. Place your answer (and nothing else) in the box below the code:

```
def rc1(k):
    n = 100
    s = set(range(n))
    for i in range(2,n):
        if (i in s):
            for j in range(2*i,n, i):
                if (j in s): s.remove(j)
    return ((k >= 10) and (k in s) and (2*k-1 in s))
```

k =

8. **Bonus/Optional: Code Tracing** [5 pts; 2.5 pts each] What will these print? Clearly circle your answers.

```
def bonusCt1(s = 'Maine has 14 counties, and 432 towns.'):
    def f(s):
        s, d, prevc = s.replace(' ',''), dict(), 'x'
        for c in s: d[c], prevc = prevc, c
        return ''.join([d.get(str(n), '') for n in range(10)])
    while (len(f(s)) > 1): s = f(s)
    return s
print(bonusCt1())

def bonusCt2():
    def f(s, S=set()):
        try: ugh = eval(s)
        except Exception as e:
            S.add(str(e)[len('unhashable type: '):].replace("'",''))
        return ''.join(sorted(S))
    f("{1:{2},3:{4:5},{6}:{7,8}}")
    return f("{1:{2},3:{4:5},{6:7}:{8}}")
print(bonusCt2())
```