fullName: _____ andrewID: _____ section: ____

15-112 S26 Quiz8 Version B
Time: 30 Minutes

You must write your name on this paper and hand this back in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. Do not unstaple any pages. All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your quiz. We will not grade anything on scrap paper.

You may not ask questions during the quiz, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the quiz, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use material we have not yet covered. Thus, do not use OOP or recursion.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than the specified time noted above on this assessment.

FR: missingVisitorsMap(L) [25 pts]

Write the function missingVisitorsMap(L) that takes a list L of tuples, where each tuple is a person's name and a city that person visited. For example:

```
L = [('Ann', 'Erie'),
     ('Ben', 'Reno'),
     ('Ann', 'Troy'),
     ('Cam', 'Troy')]
```

The function should return a dict mapping each city name to a set of the names of the people who did NOT visit that city.

For the list L above, we see:

- Erie was not visited by Ben or Cam
- Reno was not visited by Ann or Cam
- Troy was not visited by Ben

And so:

```
d = { 'Erie' : { 'Ben', 'Cam' },
      'Reno' : { 'Ann', 'Cam' },
      'Troy' : { 'Ben' }
    }

assert(missingVisitorsMap(L) == d)
```

And since this is non-mutating:

```
assert(L == [('Ann', 'Erie'),
             ('Ben', 'Reno'),
             ('Ann', 'Troy'),
             ('Cam', 'Troy')])
```

Write your solution here:

True/False [5 pts, 1 pt each]

Indicate True or False for each of the following. Indicate your answer by filling in the dot.
Assume L is a list of ints.
Assume S and T are sets of ints.
Assume D and E are dicts.

A) If `(S == set(L))` is True, then `(len(S) <= len(L))` must be True.

◯ True

◯ False

B) E can be a value in D, but E cannot be a key in D.

◯ True

◯ False

C) If `((n in S) and (len(S) == 1))` is True, then `(S.pop() == n)` must be True.

◯ True

◯ False

D) If `((S - T) == set())` is True, then `(len(S) <= len(T))` must be True.

◯ True

◯ False

E) If `((S|T) == S)` is True, then `(S == T)` must be True.

◯ True

◯ False

Big O [20 pts, 4 pts each]

For each of the following functions, state the worst-case big-oh runtime in terms of N, where N = len(L). Assume the functions never crash.

All the answers will be one of these:
O(1), O(logN), O(N), O(NlogN), $O(N^2)$, $O(N^2 \log N)$, $O(N^3)$, $O(2^N)$

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

```
def f1(L):
    M = sorted(L)
    P = [ ]
    for i in range(len(L)):
        avg = sum(L) / len(L)
        if M[i] < avg:
            P.append(i)
    return P
```

```
# note: this is the same as f1(L), only here we compute avg just once:
def f2(L):
    M = sorted(L)
    P = [ ]
    avg = sum(L) / len(L)
    for i in range(len(L)):
        if M[i] < avg:
            P.append(i)
    return P
```

```
def f3(L):

    return L.pop(0) + L.pop()
```

```
def f4(L):
    i = len(L)-1
    t = 0
    while i > 0:
        t += L[i]
        i //= 2
    return t
```

```
def f5(L):
    L = copy.copy(L)
    M = [ ]
    N = len(L)
    for x in L:
        for y in L:
            P = [x*y] * N
            M.extend(P)
    while M != [ ]:
        L.append(M.pop())
    return L
```

More Big O + Efficiency [20 pts, 4 pts each]
Answer the following. Be sure to show your work as appropriate.

1. If a function f(L) runs in O(N**2) time, and f takes 2 seconds on a list of length 1000, how long (in seconds) will f take on a list of length 3000?

2. If a function g(L) takes 4 seconds on a list of length 300, and g takes 20 seconds on a list of length 1500, what is the most likely Big O of g?

3. Assume L is a list with N values.

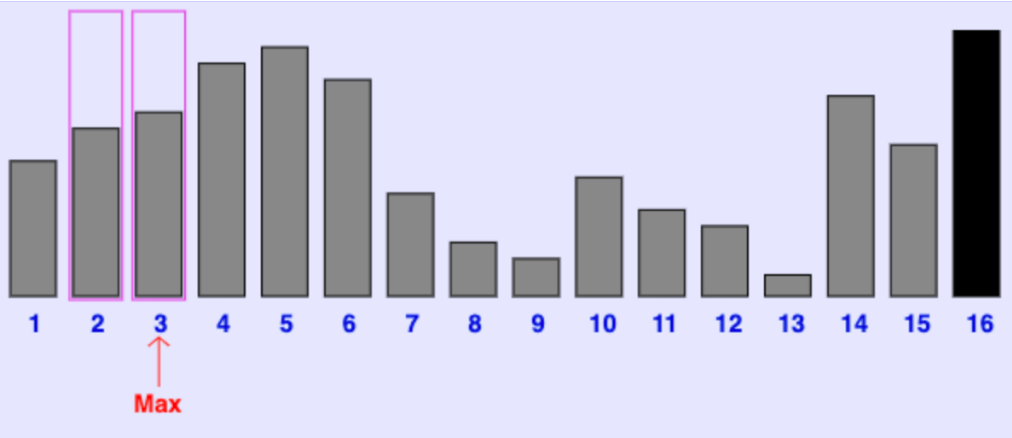
3A) What is the Big O of the number of passes for selectionSort(L)?

3B) What is the Big O number of passes for mergeSort(L)?

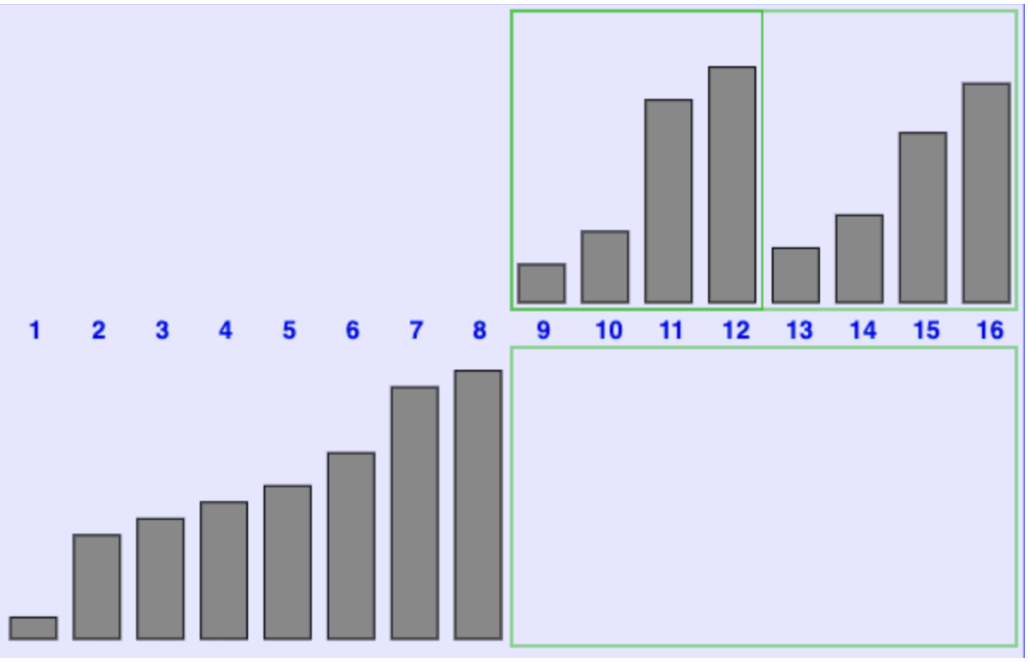
3C) What is the worst-case Big O for selectionSort(L)?

3D) What is the worst-case Big O for mergeSort(L)?

4. The following is a snapshot of xSortLab running selectionSort. What are the indexes of the next two values to be **swapped**?



5. The following is a snapshot of xSortLab running mergeSort. What are the indexes of the next two values to be **compared**?



Fill in the Blank [10 pts, 5 pts each]

FitB1: Fill in the blank in g(n) so that f(n) == g(n) for all integer values of n:

```
def f(n):
    L = list(range(n))
    M = [k**2 for k in L]
    return sorted(set(L) - set(M))


def g(n):
    s = set()
    N = list()
    for v in range(2, n):
        if v not in s:
            N.append(v)
        s.add(_____________________________________________________)
    return N
```

FitB2: Fill in the blank in g(n) so that f(n) == g(n) for all integer values of n:

```
def f(n):
    if n < 1: return None
    s = set()
    for k in range(1, n):
        t = 0
        for f in range(1, k):
            if k % f == 0:
                t += f
        if t == k:
            s.add(k)
    return s


def g(n):
    if n < 1: return None
    d = dict()
    for k in range(1, n):
        for f in range(1, k):
            if k % f == 0:
                d[k] = d.get(k, list()) + [f]
    s = set()
    for k in d:
        if ____________________________________________________________:
            s.add(k)
    return s
```

Code Tracing [20 pts, 5 pts each]

For each of the following, indicate what the code prints. Place your answer (and nothing else) in the box beside or below each block of code. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

CT1:

```
def ct1(s):
    t = set()
    u = set()
    for v in s:
        for c in v:
            if c in t:
                u.add(c)
            else:
                t.add(c)
    return u

print(ct1({'BCD', 'BE', 'C', 'TTTTT'}))
```

CT2:

```
def ct2(s, t):
    print(s | t)
    print(s & t)
    print(s - t)
    print(t - s)

ct2({1,3}, {1,4})
```

CT3:

```
def ct3(L, M):
    d = dict()
    for v in L:
        d[v] = 1
    for v in M:
        d[v] = d.get(v, 0) + 1
    print(d) # <-- DON'T MISS THIS
    for v in sorted(d):
        d[v] += d.get(d[v], 10)
    return d

print(ct3([1, 9], [6, 9]))
```

CT4:

```
def ct4(s):
    d = dict()
    for i in range(len(s)):
        c = s[i]
        d[c] = d.get(c, set()) | {i}
        d[i%2] = sum(d[c])
    return d

print(ct4('BABAB'))
```

BonusCT [2 pts, 1 pt each]

These CTs are optional, and intended to be very challenging. They are worth very few points. Indicate what the following code prints. Place your answers (and nothing else) in the boxes to the right of each code block. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1: [1pt]

```
import copy
def f(a, b, c):
    L = list(range(a, b, c))
    M = copy.copy(L)
    while L:
        i = L.index(max(L))
        M *= L.pop(i)
    return sum(M) + b

print(f(1, 10, 3))
```

bonusCT2: [1pt]

```
# note: bin(n) is the string representation of n in base 2,
# preceded with '0b'. So bin(6) is '0b110'
def g(x):
    def h(n):
        count = 0
        b = bin(n)
        for i in range(1, len(b)):
            count += int(b[i] == b[i-1] == '1')
        return count
    n = 0
    while h(n) != 3:
        n += 1
    while h(n) != 1:
        n += 1
    return n + x

print(g(-1))
```