fullName: _____ andrewID: _____ section: ____

15-112 F26 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. Also: do not use eval() or exec().

For all problems, assume that the required imports (such as 'import math') are present.

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.

Advanced Topic #1: Lisp:

True or False [7 pts, 1 pt each]

Indicate True or False for each of the following. Indicate your answer by filling in the dot.

1. Parsing starts with the 1d list of tokens, and returns a parse tree, which is a hierarchical data structure that captures the structure of the program.

○ True ○ False

2. Evaluating takes a parse tree and performs the computations that produce the output of the program.

○ True ○ False

3. In Lisp, (+ 1 (* 2 3)) evaluates to 6.

○ True ○ False

4. Our interpreter followed these steps: Tokenize, Parse, Predict, Evaluate.

○ True ○ False

5. Tokenizing is where we remove whitespace and comments.

○ True ○ False

6. We used recursion because expressions can be arbitrarily nested.

○ True ○ False

7. We used a 2d list to map variable names to their values.

○ True ○ False

Advanced Topic #2: Robotics
True or False [3 pts, 1 pt each]
Indicate True or False for each of the following. Indicate your answer by filling in the dot.

1. Early walking robots were attached to larger computing hardware and power supplies using a long cable.

○ True ○ False

2. The Sense Think Act paradigm is often used to describe the operation of robotic/autonomous systems.

○ True ○ False

3. Series-Elastic Actuators typically have very good position control, but poor force control.

○ True ○ False

Code Tracing [10 pts, 5 pts each]

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

 # CT1:

```
def ct1(L):
    M = [ (v+1) for v in L ]
    s = set(L)
    t = set(M)
    return [ sorted(s), sorted(s-t) ]

print(ct1([3, 5, 3, 6, 5]))
```

 # CT2:

```
def ct2(L):
    d = dict()
    for i in range(1, len(L), 2):
        d[L[i]] = L[i-1] + d.get(L[i], 5)
    return d

print(ct2([4, 3, 2, 3, 1, 5]))
```

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

Fill in the blank so f(L) == g(L) for all lists L where the values in L are all integers.

Important note: To receive any credit, the code you provide in the blank must run in O(1) time.

 # FitB1:

```
def f(L):
    s = set()
    for v in L:
        if L.count(v) > 1:
            s.add(v)
    return sorted(s)

def g(L):
    s = set()
    t = set()
    for v in L:
        if ____________________________________________________________:
            s.add(v)
        else:
            t.add(v)
    return sorted(t)
```

 # FitB2:

```
def f(L):
    s = set()
    for v in L:
        if v == L.count(v):
            s.add(v)
    return s

def g(L):
    d = dict()
    s = set()
    for v in L:
        d[v] = 1 + d.get(v, 0)
    for k in d:
        if ____________________________________________________________:
            s.add(k)
    return s
```

Big O [20 pts, 5 pt each]

If L is a list of N integers, what is the worst-case Big O runtime of each of the following functions. Indicate your answer by filling in the dot.

 # BigO1:

```
def bigO1(L):
    i = len(L) - 1
    r = 0
    while i > 0:
        r += L[i]
        i //= 2
    return r
```

○ O(1) ○ O(logN) ○ O(N) ○ O(NlogN) ○ $O(N^2)$ ○ $O(2^N)$

 # BigO2:

```
def bigO2(L):
    M = [ ]
    for v in L:
        M.insert(0, 2**v)
    return list(reversed(M)) + sorted(M)
```

○ O(1) ○ O(logN) ○ O(N) ○ O(NlogN) ○ $O(N^2)$ ○ $O(2^N)$

 # BigO3:

```
def bigO3(L):
    M = sorted(L)
    s = set([v**2 for v in L])
    for v in M:
        s.add(v)
    return s
```

○ O(1) ○ O(logN) ○ O(N) ○ O(NlogN) ○ $O(N^2)$ ○ $O(2^N)$

 # BigO4:

```
def bigO4(L):
    count = 0
    N = len(L)
    M = L[:10]
    for i in range(0, N, N//4):
        x = L[i] ** 3
        if x in M:
            count += 1
    return count
```

○ O(1) ○ O(logN) ○ O(N) ○ O(NlogN) ○ $O(N^2)$ ○ $O(2^N)$

FR: getSumMap [50 pts]

Background: For this problem, assume that L is a non-empty list containing N positive integers. For any value v in L, we will define its "valueSum" (a coined term) as the sum of all the occurrences of that value in L. For example, if L = [1,5,5,1,5,7], the valueSum of 1 is 2 (1+1), the valueSum of 5 is 15 (5+5+5), and the valueSum of 7 is just 7.

To continue, we will define the "sumMap" of L (a coined term) as a dictionary that maps each possible valueSum in L to a set of the values that have that valueSum.

For example, if L = [1,2,1]:

- The valueSum of 1 is 2
- The valueSum of 2 is 2
- Thus, the sumMap of L is:
  { 2 : { 1, 2 } } # both 1 and 2 have a valueSum of 2

As another example, if L = [1,2,1,1,3]:

- The valueSum of 1 is 3
- The valueSum of 2 is 2
- The valueSum of 3 is 3
- Thus, the sumMap of L is:
  {2: {2}, 3: {1, 3}} # 2 has a valueSum of 2, 1 and 3 have valueSums of 3

And finally, if L = [1,2,1,1,3,4,2]:

- The valueSum of 1 is 3
- The valueSum of 2 is 4
- The valueSum of 3 is 3
- The valueSum of 4 is 4
- Thus, the sumMap of L is:
  {3: {1, 3}, 4: {2, 4}} # 1 and 3 have valueSums of 3, 2 and 4 have valueSums of 4

With that, write the function getSumMap(L) that takes a list L and returns its sumMap as just described. Here are the same examples as they appear in the test function:

```
assert(getSumMap([1,2,1]) == {2: {1,2}})
assert(getSumMap([1,2,1,1,3]) == {2: {2}, 3: {1, 3}})
assert(getSumMap([1,2,1,1,3,4,2]) == {3: {1, 3}, 4: {2, 4}})
```

Important Note: For full credit, your function must run in O(N) time. You can receive up to half-credit if your function requires more than this time.

Write your solution on the next page.

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 below or to the right of each function. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1: [1pt]

```
def bonusCt1(L):
    # hint: ord('<') < ord('>')
    s = set(str(L))
    t = set(str(set(L)))
    u = sorted(s-t)
    for v in reversed(sorted(t-s)): u.insert(1, v)
    u.insert(2, str(sum(L)))
    return eval(''.join(u))
print(bonusCt1([1,2,3,3]))
```

bonusCT2: [1pt]

```
def bonusCt2(n):
    s = set(range(2))
    t = set()
    for v in range(n+1):
        if v not in s:
            t.add(v)
            s |= set(range(2*v, n+1, v))
    r = [ ]
    for v in sorted(t):
        e = 0
        while n%v == 0:
            e += 1
            n //= v
        if e: r.append(f'{v}^{e}')
    return ' * '.join(r)

print(bonusCt2(60))
```