fullName: _____ andrewID: _____ section: ____

15-112 F26 Quiz10 Version A
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 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.

Code Tracing [30 pts, 7.5 pts each]

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

 # CT1:

```
def ct1(L):
    if L == [ ]:
        return [ ]
    else:
        i = len(L) // 2
        M = L[:i]
        return ct1(M) + [M]

print(ct1([2, 4, 1, 5, 6, 3]))
```

 # CT2:

```
def ct2(L, k):
    if L == [ ]:
        return [ ]
    else:
        M = L[:k]
        N = L[k:]
        return [sum(M)] + ct2(N, k)

print(ct2([2, 5, 3, 1, 4, 6], 2))
```

 # CT3:

```
def ct3(lo, hi):
    if lo > hi:
        return 'done'
    else:
        return f'{lo},{hi} {ct3(2*lo, hi//2)}'

print(ct3(2, 32))
```



 # CT4:

```
from cmu_cpcs_utils import Tree

def ct4(t):
  if t.isLeaf():
      return set()
  else:
      x = [child.value for child in t.children]
      result = { sum(x) }
      for child in t.children:
          result = result.union(ct4(child))
      return result

t1 = Tree(5,
          Tree(4,
               Tree(3),
               Tree(4)),
          Tree(2),
          Tree(2,
               Tree(1),
               Tree(1,
                    Tree(1),
                    Tree(6))))

print(ct4(t1))
```



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

FitB1:

Fill in the blank so f(x) == g(x) for all positive integers x, where g must be recursive and must not use for loops, while loops, comprehensions, or any loop-like / iterative built-ins.

Also, your function g() cannot call f().

And do not worry about stack overflow for large values of x.

```
def f(x):
    L = [ ]
    while x > 0:
        L.append(x % 10)
        x //= 10
    return L

def g(x):
    if x == 0:
        return [ ]
    else:
        # recall that g MUST be recursive here:

        return ____________________________________________________________
```

FitB2:

Fill in the blank so f(L) == g(L) for all non-empty lists L of integers.

Also, your function g() must NOT be recursive and cannot call f().

And do not worry about stack overflow for large lists.

```
def f(L):
    if len(L) == 1:
        return L[0]
    else:
        first = L[0]
        rest = L[1:]
        result = f(rest)
        if result > first:
            result = first
        return result

def g(L):
    # recall that g MUST NOT be recursive here:

    return _____
```

FR1: getCloseNeighbors(L) [30 pts]

Note: your answer to this problem must be recursive and must not use for loops, while loops, comprehensions, or any loop-like / iterative built-ins.

Background: given a list L of integers, we will say that two values are "neighbors" if they occur at L[i] and L[i+1] for some index i. Further, we will say that two values are "close neighbors" (a coined term) if they are neighbors and their absolute difference is no larger than 2.

For example, if L = [2, 5, 3, 4, 8, 6]:

- 2 and 5 are NOT close neighbors (absolute difference is 3)
- 5 and 3 are close neighbors (absolute difference is 2)
- 3 and 4 are close neighbors (absolute difference is 1)
- 4 and 8 are NOT close neighbors (absolute difference is 4)
- 8 and 6 are close neighbors (absolute difference is 2)

Also, note:

- 2 and 3 are NOT close neighbors (they are not consecutive elements)

With that, write the recursive function getCloseNeighbors(L) that takes a list L of integers, and (without using loops) returns a list of the tuples of values that are close neighbors in L, in the order they appear in L.

For example, for the list L above:

```
assert(getCloseNeighbors([2, 5, 3, 4, 8, 6]) ==
                         [(5, 3), (3, 4), (8, 6)])
```

Write your solution on the next page.

FR2: getCharIndexesMap(s) [30 pts]

Note: your answer to this problem must be recursive and must not use for loops, while loops, comprehensions, or any loop-like / iterative built-ins.

With that, write the recursive function getCharIndexesMap(s) that takes a string s and (without using loops) returns a dict mapping each char c in s to a set of all the indexes i such that s[i] == c.

For example, if s = 'ababa!':

- 'a' occurs at indexes 0, 2, 4
- 'b' occurs at indexes 1, 3
- '!' occurs at index 5

Thus:

```
assert(getCharIndexesMap('ababa!') == {'a': {0, 2, 4},
                                       'b': {1, 3},
                                       '!': {5}
                                      })
```

Hint: while you would usually process the first character in your recursive case, we found it easier here to process the LAST character (because we know the index of that character).

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(n):
    def f(n): return (1 + f(n//2)) if n else 0
    def g(k): return k if f(k) == n else g(k+1)
    return g(n)
print(bonusCt1(7))
```

bonusCT2: [1pt]

```
def bonusCt2(L):
    def x(L):
        if not L: return L
        return x(L[:-1]) + [L[-1] - 1]
    def y(L):
        if not L: return L
        x = [L[0]] if L[0] else [ ]
        return x + y(L[1:])
    def z(L):
        if len(L) <= 1: return L
        return z(y(x(L)))
    return z(L)
print(bonusCt2([22, 17, 31, 19, 3]))
```