4. (8 points) Composition

(a) (4 pt) Implement `compose`, which takes a positive integer `n`. It returns a function that, when called repeatedly on n one-argument functions $f_1, f_2, \ldots, f_n$, returns a one-argument function of `x` that returns $f_1(f_2(\ldots f_n(x)\ldots))$. You may **not** call the `compose1` function from the Midterm 1 Study Guide.

```
def compose(n):
    """Return a function that, when called n times repeatedly on unary
    functions f1, f2, ..., fn, returns a function g(x) equivalent to
    f1(f2( ... fn(x) ... )).

    >>> add1 = lambda y: y + 1
    >>> compose(3)(abs)(add1)(add1)(-4)  # abs(add1(add1(-4)))
    2
    >>> compose(3)(add1)(add1)(abs)(-4)  # add1(add1(abs(-4)))
    6
    >>> compose(1)(abs)(-4)              # abs(-4)
    4
    """
    assert n > 0


    if n == 1:


        return lambda f: f


    def call(f):


        def on(g):


            return compose(n-1)(lambda x: f(g(x)))


        return on


    return call
```

1.5 Write a procedure `merge(n1, n2)` which takes numbers with digits in decreasing order and returns a single number with all of the digits of the two, in decreasing order. Any number merged with 0 will be that number (treat 0 as having no digits). Use recursion.

Hint: If you can figure out which number has the smallest digit out of both, then we know that the resulting number will have that smallest digit, followed by the merge of the two numbers with the smallest digit removed.

```
def merge(n1, n2):
    """ Merges two numbers
    >>> merge(31, 42)
    4321
    >>> merge(21, 0)
    21
    >>> merge (21, 31)
    3211
    """


    if n1 == 0:
        return n2
    elif n2 == 0:
        return n1
    elif n1 % 10 < n2 % 10:
        return merge(n1 // 10, n2) * 10 + n1 % 10
    else:
        return merge(n1, n2 // 10) * 10 + n2 % 10
```

disc 05

1. **So Many Options...**

(a) Implement the following function `partition_options` which outputs all the ways to partition a number `total` using numbers no larger than `biggest`.

```
def partition_options(total, biggest):
    """
    >>> partition_options(2, 2)
    [[2], [1, 1]]
    >>> partition_options(3, 3)
    [[3], [2, 1], [1, 1, 1]]
    >>> partition_options(4, 3)
    [[3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]
    """
    if total == 0:
        return [[]]
    elif total < 0 or biggest_num == 0:
        return []
    else:
        with_biggest = partition_options(total-biggest_num, biggest_num)
        without_biggest = partition_options(total, biggest_num-1)
        with_biggest = [[biggest_num] + elem for elem in with_biggest]
        return with_biggest + without_biggest
```

(b) Return the minimum number of elements from the list that need to be summed in order to add up to `T`. The same element can be used multiple times in the sum. For example, for `T = 11` and $lst = [5, 4, 1]$ we should return 3 because at minimum we need to add 3 numbers together (5, 5, and 1). You can assume that there always exists a linear combination of the elements in `lst` that equals `T`.

```
def min_elements(T, lst):
    """
    >>> min_elements(10, [4, 2, 1]) # 4 + 4 + 2
    3
    >>> min_elements(12, [9, 4, 1]) # 4 + 4 + 4
    3
    >>> min_elements(0, [1, 2, 3])
    0
    """

    if T == 0:

        return 0

    return min([1 + min_elements(T - i, lst) for i in lst if T - i >= 0])
```

(c) Reminder: don't forget to check your quiz solutions @ cs61a.org Quiz solutions can be found on the last page of the discussion solutions, which are posted at the end of each week. If you do not know where to find discussion solutions @ cs61a.org, see links.cs61a.org/quiz-sols-location