Chapter 5: Solutions

Recursion is really really hard, but please give the problems your best shot before checking these solutions!

1) md

```python
def md(number):
    if number < 10:
        return number

    rightmost_digit = number % 10
    rest_of_number = number // 10

    return rightmost_digit * md(rest_of_number)
```

This was identical to sum digits, the only difference being we multiplied instead of adding. Good review though!

2) rec_power

```python
def rec_power(base, exponent):
    if exponent == 0:
        return 1
    exponent = exponent - 1
    return base * rec_power(base, exponent)
```

The key realization for this problem was that for 2 ^ 3 = 2 x (2 ^ 2)

In the general case, base ^ exponent = base x (base ^ (exponent - 1))

Then, the recursive call is straightforwards, and the base case is because for all n, n ^ 0 = 1.

(Note that you could simplify this code by returning base * rec_power(base, exponent - 1). This saves you from having to declare exponent = exponent - 1 on the previous line.)

3) count8

```python
def count8(number):
    if number == 8:
        return 1
    elif number < 10:
        return 0

    rightmost_digit = number % 10
    rest_of_number = number // 10

    if rightmost_digit == 8:
        return 1 + count8(rest_of_number)
    else:
        return count8(rest_of_number)
```

We make the problem smaller each time in the same way as we did in sum digits and the product of digits, but the hard part about this problem was realizing that we change the recursive call based on some condition.

We split the problem of counting 8's into:

1. An easily solvable problem: is the last digit 8?
2. A problem that can be split further: the rest of the number.

The counting in this problem was conditional, which might have been tricky.

Recap

Hopefully, you either got the problems mostly correct, or were able to fully understand the solutions. If you're still having trouble with these concepts, it may be worthwhile for you to review the previous examples and attempt the problems again without looking at the solutions.