Recursion Problems

These problems involve the use of recursion; if you know of a way of doing it with higher order functions, pretend to forget it temporarily for the sake of practicing recursion.

Don't be surprised if you find yourself using stuff from previous problems. After you answer a Question, you can use your results to solve subsequent Questions. (In fact, you'll need to; otherwise the problems are too large to handle!) They're arranged in order of difficulty, so we recommend that you do them in sequence. Finally, if you feel the need to write some helper functions on any Question, it's perfectly ok.

Question 1 (**Easy**) It's often useful to write a function that, given a sentence or word, *counts* how many elements are in it. This exercise asks you to implement your own version of the `count` function.

- Write a `count-word` function that, given a word, returns its length. For example,

  `(count-word 'gangrene)`

  should return the value `8`.

- Write a `count-sent` function that, given a sentence, returns its length. For example,

  `(count-sent '(this song is just six words long))`

  should return the value `7`. Yes, that particular sentence is contradictory. Blame Weird Al.

- Finally, write a `count` function that takes either a sentence or word, and returns the appropriate answer.

Question 2 (**Easy**) In the long distant, barbaric past, secretaries used an archaic device called a *typewriter* to enter in documents. As a secretary typed a line, if the line extended past a specified line width, the typewriter would `<ding>` a bell. This `<ding>` would tell the secretary to go down to the beginning of the next line.

For example, given a line width of 10 characters, after seeing a line begin like this

`Strong␣roots`

a typewriter would `<ding>` because "Strong roots" is 12 characters long, including the space.

Write a predicate function called `ding?` which will take in two arguments—a sentence that represents a line, and the width of a line. It should return `#f` if the sentence's width is shorter than or equal to the line width, and should otherwise return `#t`. Here are a few sample calls:

```
> (ding? '(strong roots) 10)
#t
> (ding? '(strong roots) 12)
#f
> (ding? '(so the spear danes in days gone by) 34)
#f
> (ding? '(so the spear danes in days gone by) 33)
#t
> (ding? '(could not think of any lyrics) 80)
#f
```

Question 3 (**Medium**) It's also convenient to have a function that, given a sentence, selects a small portion of a sentence for us. For example, if we had the sentence:

`(russians declare war rington vodka to be excellent)`

we could imagine using a hypothetical `subsentence` function that would let us pull out the first few words of that sentence, if we tell it where to start and stop the selection:

```
> (subsentence '(russians declare war rington
                  vodka to be excellent) 1 3)
(russians declare war)
> (subsentence '(no shirt no shoes no service) 4 4)
(shoes)
```

Write the function `subsentence`, which takes in three arguments: a `sent`ence, the `start`ing endpoint, and the `stop`ping endpoint. It should return back a sentence that includes the words between the `start` and `stop` endpoints. Assume that the user is nice, and won't give weird input. In Scheme notation, we mean that we can assume `(<= 1 start stop (count sent))` is always true.

Question 4 (**Hard**) Write a function called `insert-dings` that simulates the `<ding>`'ing of a typewriter; given a sentence and line width, it should insert `<ding>` marks wherever it's appropriate. Use and write any helper functions that you think will make this problem manageable.

Here are a few sample calls, indented to more clearly show `insert-dings` behavior:

```
> (insert-dings '(strong roots are not hurt by the frost) 7)
(strong roots <ding>
 are not hurt <ding>
 by the frost <ding>)

> (insert-dings '(dont write back theyll probly sikowanalize your letter) 12)
(dont write back <ding>
 theyll probly <ding>
 sikowanalize your <ding>
 letter)
```