The following exercises are designed to test your understanding of recursion. The functions are defined using a variant of LISP known as meta-LISP. In order to aid your understanding, the function defined in problem 1 is identical to the one below:

\[ \text{drop}(x) = \begin{cases} \text{nil} & \text{if } x \text{ is null} \\ \text{car}(x) \text{ cons drop(cdr}(x)) & \text{otherwise} \end{cases} \]

The idea is that:
- \( a \cdot x = \text{car}(x) \)
- \( d \cdot x = \text{cdr}(x) \)
- \( n \cdot x = \text{null}(x) \)
- \( a \cdot b = \text{conc}(a, b) \) (i.e. append list \( b \) to list \( a \))
- \( \text{reverse}(x) = \text{reverses the top level list } x \). For example \( \text{reverse}([A B C]) = [C B A] \). But \( \text{reverse}(((A B C)(D E))) = ((D E)(A B C)) \).

1. Consider the function \( \text{drop} \) defined by
   \[
   \text{drop}(x) \leftarrow \begin{cases} \text{nil} & \text{if } x \text{ is null} \\ \text{a \cdot x \cdot \text{drop}(d \cdot x)} & \text{otherwise} \end{cases}
   \]
   Compute (by hand) \( \text{drop}([A B C]) \). What does \( \text{drop} \) do to lists in general?

2. What does the function \( \text{r2}(x) \) do to lists of lists? How about
   \[
   \text{r3}(x) \leftarrow \begin{cases} \text{nil} & \text{if } x \text{ is null} \\ \text{reverse}([a \cdot x \cdot \text{r2}(d \cdot x)] & \text{otherwise} \end{cases}
   \]

3. Compare the following function with the function \( \text{r3} \) of the preceding example:
   \[
   \text{r3'}(x) \leftarrow \begin{cases} \text{nil} & \text{if } x \text{ is null} \\ \text{a \cdot x \cdot \text{r3'}(d \cdot x)} & \text{otherwise} \end{cases}
   \]

4. Consider \( \text{r5} \) defined by
   \[
   \text{r5}(x) \leftarrow \begin{cases} \text{nil} & \text{if } x \text{ or } d \cdot x \text{ is null} \\ \text{a \cdot r5}(d \cdot x) \cdot \text{r5}(x) \cdot \text{r5}(d \cdot r5(d \cdot x))] & \text{otherwise} \end{cases}
   \]
   Compute \( \text{r5}([A B C D]) \). What does \( \text{r5} \) do in general. Needless to say, this is not a good way of computing this function even though it involves no auxiliary functions.