Recursive Programming Extravaganza II

Really think about and discuss your edge cases and time complexities in all of these!

- **Length of string**
    - Implement a recursive function that calculates the length of a string
    - Takes a string as a parameter
    - Returns an integer (+1 for each character)

  Note that you can use built in syntax for a string exactly as if it were a list where each item is a character. “abcd”[2] == ‘c’ ... “abcd”[:-1] == “abc” ... “abcd”[1:] == “bcd”

- **Linear search**
    - Implement a recursive function that searches for a value in a list
    - Takes a list and a single value as parameters
    - Returns a ***boolean*** value
        - ***True*** if the value is in the list, otherwise ***False***

- **Count instances**
    - Implement a recursive function that counts a specific value in a list
    - Takes a list and a single value as parameters
    - Returns an ***integer*** value
        - *How many times does that value appear in the list?*

- **Are there duplicates in a list?**
    - Implement a recursive function that checks for duplicate values in a list
    - Takes a list as a parameter
    - Returns a ***boolean*** value
        - ***True*** if any value in the list appears more than once, otherwise ***False***

- **Remove duplicates**
    - Implement a recursive function that removes duplicate values in a list
    - Takes a ***list*** as a parameter
    - Returns another ***list***
        - *List with all the same values, but only one instance of each value*

- **Binary search**
    - Implement binary search in an ordered list using recursive programming

- **Substring**
    - Implement the function ***is_substring(substring, a_str)*** that answers the following:
        - Is the string ***substring*** actually a substring in the list ***a_str***?
            - Examples:
                - is_substring(“a”, “gagnaskipan”) -> True
                - is_substring(“gnask”, “gagnaskipan”) -> True
                - is_substring(“iganpsk”, “gagnaskipan”) -> False
                - is_substring(“gnAsk”, “gagnaskipan”) -> False
                - is_substring(“gnesk”, “gagnaskipan”) -> False
        - **Use recursion**
        - Hint: Try to first implement the function ***prefix(prefix, a_str)***
            - Only checks whether ***prefix*** is an exact duplicate of the beginning of ***a_str***
        - *There are two recursive/iterative “loops”.*
            - Try to implement both loops with recursion
                - *This solution can look elegant and clean!*

- **Elfish / X-ish**
    - Implement the function ***x_ish(a_str, x)*** that answers the following:
        - Does the string ***a_str*** include all letters in the string ***x***?
            - Examples:
                - x_ish(“gagnaskipan”, “a”) -> True
                - x_ish(“gagnaskipan”, “gnask”) -> True
                - x_ish(“gagnaskipan”, “iganpsk”) -> True
                - x_ish(“gagnaskipan”, “gnAsk”) -> False
                - x_ish(“gagnaskipan”, “gnesk”) -> False
        - **Use recursion**
        - Hint: Try to first implement the function ***elf_ish(a_str)***
            - Only checks whether ***a_str*** includes all letters in the substring ***“elf”***
        - *There are two recursive/iterative “loops” and it affects the efficiency of the solution which one is the “outer loop” and which one the “inner loop”.*
            - Try to implement both loops with recursion
                - *This solution can look elegant and clean!*

- **Palindrome**
    - Implement the function ***palindrome*** that takes a string as a parameter:
        - Returns ***True*** if the string reads exactly the same forwards and backwards.
        - Hint: You can make a helper function that takes different parameters
            - One possibility (of many) is a helper that takes two strings
                - *But send the same string into both parameters?*