Recursion, Dynamic Programming, and Binary Search

Instructions

- The references fir this problem set are Chapters 9, 10, and 11 of the textbook and Levitin's book.
- The textbook is available at `https://github.com/rcpsilva/PCC104_DesignAndAnalysisOfAlgorithms/blob/main/2025-1/Course%20Material/A%20First%20Course%20on%20Data%20Structures%20in%20Python.pdf`
- Presente the complexity analysis of your solutions.

Recursion (Chapter 9)

1. **Recursive Maximum Finder**

   Write a recursive function `find_max(L)` that returns the maximum element in a list `L`.

   Example:

   ```
   find_max([1, 5, 3, 9, 2])  # Output: 9
   ```

2. **Recursive String Reversal**

   Implement a recursive function `reverse(s)` that returns the reversed version of the string `s`.

   Example:

   ```
   reverse("python")  # Output: "nohtyp"
   ```

3. **Sum of Digits**

   Write a recursive function `sum_digits(n)` that returns the sum of the digits of an integer `n`.

   Example:

   ```
   sum_digits(1234)  # Output: 10
   ```

4. **Palindrome Check**

   Write a recursive function `is_palindrome(s)` that returns `True` if the string `s` is a palindrome and `False` otherwise.

   Example:

   ```
   is_palindrome("racecar")  # Output: True
   is_palindrome("hello")    # Output: False
   ```

Dynamic Programming (Chapter 10)

5. **Fibonacci with Memoization**

   Implement a memoized version of the Fibonacci sequence. The function `fib(n)` should return the nth Fibonacci number.

   Example:

   ```
   fib(10)  # Output: 55
   ```

6. **Minimum Coin Change**

   Given coins of certain denominations and a total amount, write a function `min_coins(coins, amount)` that computes the minimum number of coins needed to make the amount.

   Example:

   ```
   min_coins([1, 3, 4], 6)  # Output: 2 (3 + 3)
   ```

7. **Longest Common Subsequence (LCS)**

   Write a function `lcs(X, Y)` that returns the length of the longest common subsequence of two strings `X` and `Y`.

   Example:

   ```
   lcs("abcde", "ace")  # Output: 3
   ```

8. **0/1 Knapsack Problem**

   Given weights and values of n items, write a function `knapsack(W, weights, values)` to determine the maximum value that can be put in a knapsack of capacity W.

   Example:

   ```
   knapsack(50, [10, 20, 30], [60, 100, 120])  # Output: 220
   ```

Binary Search (Chapter 11)

9. **Binary Search Implementation**

   Implement a recursive function `binary_search(L, x)` that returns `True` if `x` is in the sorted list `L`, and `False` otherwise.

   Example:

   ```
   binary_search([1, 3, 5, 7, 9], 3)  # Output: True
   binary_search([1, 3, 5, 7, 9], 4)  # Output: False
   ```

10. **First Occurrence in Sorted Array**

    Modify your binary search to find the index of the first occurrence of a number in a sorted list with duplicates.

    Example:

    ```
    first_occurrence([1, 2, 2, 2, 3, 4], 2)  # Output: 1
    ```

11. **Square Root Using Binary Search**

    Implement a function `sqrt_binary(n)` that returns the integer square root of a non-negative integer n using binary search.

    Example:

    ```
    sqrt_binary(10)  # Output: 3
    sqrt_binary(16)  # Output: 4
    ```

12. **Peak Element Finder**

    Write a function `find_peak(L)` that returns an index of a peak element using a binary search-like approach. An element is a peak if it is greater than or equal to its neighbors.

    Example:

    ```
    find_peak([1, 3, 20, 4, 1, 0])  # Output: 2 (index of 20)
    ```