RECURSION EXERCISES

In this mini-workbook, we'll get you up and running with recursion. Recursion is a fundamental concept in computer science and a precursor to more involved algorithms such as dynamic programming so it is important to deeply understand the inner workings of recursion.

In the guide, we will quickly highlight the key concepts that you need to know for your interview. If you're unfamiliar with any of these, I highly recommend taking some time to review them.

Then, I've provided you with a series of exercises to help you get up and running quickly. I recommend setting aside 2-3 days to work through these exercises. Simply start from the top and work through them as much as you have time.

Make sure to track the time you spend here in your Interview Prep Tracker. You can categorize this as "Study" time.

For more information on recursive patterns, see this post.

TABLE OF CONTENTS

KEY RECURSION TERMINOLOGY

- **Basic Recursion Terms**
    - Base case
    - Recursive step
    - Tail recursion
    - Backtracking

KEY RECURSION PATTERNS

- **Recursion Fundamentals**
  The foundations of recursion.
- **Iteration**
  iterate over a variety of data structures using recursion, both in one and multiple dimensions.
- **Subproblems**
  We'll go over the most fundamental pattern in all of recursion: Subproblems.
- **Selection**
  The selection pattern is one of the most commonly occurring patterns that you will see in your interview and is fundamental to understanding dynamic programming.
- **Ordering**
  Learn how to use permutations to solve many common recursive problems.
- **Divide & Conquer**
  These are two common subpatterns that will be critical to your interview success.

RECURSION EXERCISES

1. Iteration pattern: Iterate over an array or list using recursion. This is rarely useful except in the circumstances of simplifying code.

- Insert an item at the bottom of a stack (Full Problem Definition)
- Print a linked list in reverse order (Full Problem Definition)
- Find all substrings of a string (Full Problem Definition)

2. Breaking into Subproblems pattern: This includes all of the "classic" recursive problems. Technically all recursive-problems, but many are not obvious.

- Towers of Hanoi (Full Problem Definition)
- Is a string a palindrome? (Full Problem Definition)
- Stair steps (Full Problem Definition)

3. Selection (Combinations) Pattern: Fundamentally, problems that can be solved by finding all valid combinations. Optimize by validating as we go/backtracking

- Find all combinations of a set of inputs (Full Problem Definition)
- 0-1 Knapsack (Full Problem Definition)
- Find all ways to interleave 2 strings (Full Problem Definition)

4. Ordering (Permutations) Pattern: The Ordering Pattern is similar to the Selection Pattern except now, order matters.

- Find all permutations of a set of inputs (Full Problem Definition)
- Find all permutations when the input includes duplicates (Full Problem Definition)
- Find all N-digit numbers whose digits sum up to a target (Full Problem Definition)

5. Divide and Conquer Pattern: This pattern is common with searching, sorting, and tree data structures. In this pattern, we ask whether we can solve the problem for each half of the input and easily combine the results.

- Implement binary search (Full Problem Definition)
- Implement mergesort (Full Problem Definition)
- Find all unique binary search trees (Full Problem Definition)