That’s a great approach! Giving yourself **two weeks per concept** will let you dive deep into each topic and gain a solid understanding. This will definitely help with more challenging problems, especially during interviews where a thorough understanding is key. Here’s a more refined breakdown for your two-week focus on each topic:

### ****Weeks 1–2: Arrays and Strings****

**Days 1–3**:

1. **Review Basics**: Start with sliding window, two-pointer techniques, and sorting.
2. **Solve**: Easy problems like "Maximum Sum Subarray of Size K" and "Longest Substring Without Repeating Characters."

**Days 4–7**:

1. **Focus**: Subarrays, subsequences, and combinations.
2. **Solve**: Medium problems like "Longest Substring with K Distinct Characters" and "Minimum Window Substring."

**Days 8–11**:

1. **Advanced Practice**: Work on problems involving more complex conditions, like dynamic windows.
2. **Solve**: Hard problems like "Sliding Window Maximum" and "Subarrays with K Different Integers."

**Days 12–14**:

1. **Mock Interviews**: Solve problems in a timed setting or use platforms like LeetCode's mock interview feature. Focus on covering edge cases and improving speed.

### ****Weeks 3–4: Linked Lists****

**Days 1–3**:

1. **Understand Basics**: Linked list traversal, reversing a linked list, and merging techniques.
2. **Solve**: Easy problems like "Reverse Linked List" and "Merge Two Sorted Lists."

**Days 4–7**:

1. **Intermediate Problems**: Practice cycle detection, adding two numbers, and middle of the linked list problems.
2. **Solve**: Problems like "Linked List Cycle" and "Add Two Numbers."

**Days 8–11**:

1. **Focus**: More challenging linked list problems like "Flattening a Multilevel Doubly Linked List" and "Copy List with Random Pointer."

**Days 12–14**:

1. **Concept Combination**: Work on problems where you combine different linked list techniques, such as merging multiple linked lists or rearranging nodes.
2. **Solve**: "Merge K Sorted Lists."

### ****Weeks 5–6: Stacks and Queues****

**Days 1–3**:

1. **Learn Basics**: Operations on stacks and queues, monotonic stacks/queues.
2. **Solve**: Easy problems like "Valid Parentheses" and "Implement Queue using Stacks."

**Days 4–7**:

1. **Intermediate Practice**: Solve medium-level problems such as "Min Stack" and "Next Greater Element."
2. **Focus**: Sliding window problems using stacks/queues like "Sliding Window Maximum."

**Days 8–11**:

1. **Advanced Problems**: Tackle harder problems like "Largest Rectangle in Histogram."
2. **Solve**: More tricky stack/queue problems that involve dynamic ranges or partitioning.

**Days 12–14**:

1. **Review and Refine**: Focus on edge cases in stack/queue problems and do mock interviews.

### ****Weeks 7–8: Recursion and Backtracking****

**Days 1–3**:

1. **Master Recursion**: Review the basics of recursion and simple recursion problems like Fibonacci, factorial, and basic tree traversal.
2. **Solve**: Easy problems like "Climbing Stairs" and "Power of Two."

**Days 4–7**:

1. **Backtracking**: Learn the backtracking approach with problems like "N-Queens" and "Subsets."
2. **Solve**: Medium problems involving decision trees, permutations, and combinations.

**Days 8–11**:

1. **Hard Backtracking**: Tackle problems that require deep recursion and complex state management.
2. **Solve**: Hard problems like "Word Search" and "Sudoku Solver."

**Days 12–14**:

1. **Deep Dive**: Focus on problems that combine recursion with dynamic programming or optimization techniques.

### ****Weeks 9–10: Trees and Binary Search Trees****

**Days 1–2**:

1. **Tree Basics**: Revise tree traversals (inorder, preorder, postorder) and the structure of binary trees.
2. **Solve**: Easy problems like "Max Depth of Binary Tree" and "Same Tree."

**Days 3–5**:

1. **BST**: Understand the properties of Binary Search Trees and solve problems like validating a BST and searching within a BST.
2. **Solve**: Medium problems like "Lowest Common Ancestor of a BST."

**Days 6–9**:

1. **Advanced Tree Problems**: Focus on DFS and BFS approaches to tree problems.
2. **Solve**: Hard problems like "Serialize and Deserialize Binary Tree."

**Days 10–14**:

1. **Combining Concepts**: Solve problems involving combinations of tree traversal, DFS/BFS, and dynamic programming in trees.

### ****Weeks 11–12: Binary Search****

**Days 1–3**:

1. **Binary Search Basics**: Revise the concept of binary search, mid-point technique, and edge cases.
2. **Solve**: Easy problems like "Binary Search" and "Guess Number Higher or Lower."

**Days 4–7**:

1. **Advanced Binary Search**: Work on problems like "Search in Rotated Sorted Array" and "Find Peak Element."
2. **Solve**: Medium problems involving binary search on unsorted arrays or finding the insertion point.

**Days 8–11**:

1. **Binary Search on Answers**: Solve more advanced problems like "Median of Two Sorted Arrays."
2. **Solve**: Hard problems that involve variations of binary search like finding minimum in rotated arrays.

**Days 12–14**:

1. **Practice and Review**: Revisit and solve harder binary search problems. Mock interviews are key here.

### ****Weeks 13–14: Dynamic Programming****

**Days 1–3**:

1. **DP Basics**: Start with simple problems like Fibonacci and basic memoization techniques.
2. **Solve**: Easy DP problems like "Climbing Stairs" and "House Robber."

**Days 4–7**:

1. **Tabulation**: Move on to solving problems like "0/1 Knapsack" and "Coin Change" using tabulation.
2. **Solve**: Medium problems involving overlapping subproblems and recursive solutions.

**Days 8–11**:

1. **Advanced DP**: Tackle harder problems like "Longest Increasing Subsequence" and "Palindrome Partitioning."
2. **Solve**: Multi-step DP problems that involve optimization.

**Days 12–14**:

1. **Review and Deep Practice**: Focus on combining DP with other techniques like recursion and graphs.

### ****Weeks 15–16: Graphs****

**Days 1–3**:

1. **Graph Basics**: Learn graph representations, DFS, and BFS.
2. **Solve**: Easy problems like "Number of Islands" and "Graph Traversal."

**Days 4–7**:

1. **Intermediate Graph Problems**: Solve problems like cycle detection, connected components, and shortest paths.
2. **Solve**: Medium problems involving directed graphs and undirected graphs.

**Days 8–11**:

1. **Advanced Graph Problems**: Focus on more complex graph algorithms like Dijkstra's or Floyd-Warshall.
2. **Solve**: Hard problems like "Topological Sort" and "Network Delay Time."

**Days 12–14**:

1. **Review**: Combine graph concepts with recursion or dynamic programming.

### ****Weeks 17–18: Heaps and Priority Queues****

**Days 1–3**:

1. **Heap Operations**: Review heap basics, min-heap vs. max-heap, and basic heap operations.
2. **Solve**: Easy problems like "Kth Largest Element in an Array."

**Days 4–7**:

1. **Intermediate Heap Problems**: Solve problems like "Top K Frequent Elements" and "Find Median from Data Stream."
2. **Solve**: Practice medium heap-based problems like merging sorted lists.

**Days 8–11**:

1. **Advanced Heap Problems**: Work on harder problems like "Sliding Window Maximum."
2. **Solve**: Complex heap problems that combine other data structures.

**Days 12–14**:

1. **Review and Refine**: Work on edge cases and do mock interviews with heap-based problems.

### Summary:

By giving **two weeks per topic**, you’re making sure that you fully understand each concept, solve a variety of problems, and improve steadily over time. The slow, in-depth learning will help you build confidence, which is essential for interviews!