

# Video Summaries

## 1. 70 Leetcode problems in 5+ hours (every data structure) (full tutorial)

Video ID: lvO88XxNAzs

### Original Summary

English

This comprehensive guide to LeetCode problem-solving, presented as a YouTube video, provides a structured approach to tackling coding interview questions. The video emphasizes the importance of a solid foundation in both literature and science, and a dedicated mindset for success.

#### **\*\*I. Foundations for Success:\*\***

**\* \*\*Mindset and Focus:\*\*** The video stresses the critical role of focus and uninterrupted work sessions. It highlights the detrimental effects of constant distractions and the importance of cultivating a focused mindset, drawing parallels to the fast-paced nature of modern content consumption and its impact on attention spans. **\* \*\*Programming Fundamentals:\*\*** The video recommends Python as a beginner-friendly language for LeetCode problem-solving. It emphasizes the importance of object-oriented programming concepts like classes, inheritance, and abstraction, and the ability to build small, independent projects to solidify language proficiency. **\* \*\*Data Structures and Algorithms:\*\*** The video prioritizes data structures as more crucial than algorithms for interview preparation, emphasizing their daily use in software development. **\* \*\*Foundation as a Software Engineer:\*\*** The video highlights the importance of a 20-page document covering everything from bits to the cloud, acting as a personal cheat sheet. **\* \*\*Active Learning:\*\*** The video strongly advocates for active learning, discouraging passive consumption of content and encouraging hands-on practice through note-taking, code writing, solving problems repeatedly, and participating in mock interviews.

#### **\*\*II. Problem-Solving Framework:\*\***

**\* \*\*Read and Understand:\*\*** The first step is to read the problem statement carefully, ideally twice, and to ask clarifying questions to ensure a complete understanding. **\* \*\*Brainstorm and Verbalize:\*\*** The next step is to brainstorm different approaches to solving the problem, starting with a brute-force solution to establish a baseline. **\* \*\*Deep Dive and Visualize:\*\*** Explore the best solutions in depth, considering the end-to-end process from input to output. Drawing out solutions with pseudocode is recommended for improved visualization and pattern recognition. **\* \*\*Code and Iterate:\*\*** Write the code based on the chosen solution, and then iterate to improve it. **\* \*\*Analyze and Learn:\*\*** Review other solutions, paying attention to readability and time complexity.

#### **\*\*III. LeetCode Problem Walkthrough (Array Section):\*\***

The video then dives into a detailed walkthrough of several LeetCode array problems, providing solutions, explanations, and insights into the thought process:

\* \*\*Contains Duplicate:\*\* Discusses using a set in Python for an  $O(n)$  solution. \*  
\* \*\*Missing Number:\*\* Explores sorting and a mathematical approach (sum of range minus the sum of the array) for an  $O(n)$  solution. \* \*\*Find All Numbers Disappeared in an Array:\*\* Highlights the use of sets and range iteration. \* \*\*Two Sum:\*\* Presents a brute-force  $O(n^2)$  solution and a more efficient hashmap-based  $O(n)$  solution. \* \*\*How Many Numbers Are Smaller Than the Current Number:\*\* Demonstrates sorting and the use of a dictionary to store counts. \* \*\*Minimum Time Visiting All Points:\*\* Explains a unique solution based on the maximum difference between coordinates for an  $O(n)$  time complexity. \* \*\*Spiral Matrix:\*\* Provides a detailed walkthrough of a medium-level problem, emphasizing the importance of drawing out solutions and identifying patterns. \*  
\* \*\*Number of Islands:\*\* Introduces breadth-first search (BFS) as a traversal algorithm, explaining its use in solving the problem. \* \*\*Two Pointers:\*\* Explains the two-pointer technique and applies it to the "Best Time to Buy and Sell Stock" problem, resulting in an  $O(n)$  solution. \* \*\*Squares of a Sorted Array:\*\* Explores three solutions: a simple sorting approach ( $O(n \log n)$ ), a split-and-merge approach ( $O(n)$ ), and an absolute-and-merge approach ( $O(n)$ ). \* \*\*Three Sum:\*\* Builds upon the Two Sum problem and presents a solution using sorting and two pointers for an  $O(n^2)$  time complexity. \* \*\*Longest Mountain in Array:\*\* Explains the concept of a mountain array and provides a solution using two pointers for an  $O(n)$  time complexity.

#### \*\*IV. LeetCode Problem Walkthrough (Sliding Window Section):\*\*

\* \*\*Contains Duplicate II:\*\* Applies the sliding window technique using a set for an  $O(n)$  solution. \* \*\*Minimum Absolute Difference:\*\* Demonstrates sorting and a sliding window of size two to find pairs with the minimum difference. \* \*\*Minimum Size Subarray Sum:\*\* Explains the sliding window technique with two pointers for an  $O(n)$  solution.

#### \*\*V. LeetCode Problem Walkthrough (Bit Manipulation Section):\*\*

\* \*\*Single Number:\*\* Introduces the exclusive OR (XOR) bitwise operation and explains how it can be used to efficiently find the single non-duplicate element in an array, achieving a linear runtime and constant space complexity.

#### \*\*VI. LeetCode Problem Walkthrough (Dynamic Programming Section):\*\*

\* \*\*Coin Change:\*\* Presents a dynamic programming approach to solve the coin change problem, explaining the bottom-up approach and the use of an array to store intermediate results. \* \*\*Climbing Stairs:\*\* Demonstrates the dynamic programming approach to the climbing stairs problem, highlighting the use of the previous two subproblems to calculate the current solution. \* \*\*Maximum Subarray:\*\* Explains the dynamic programming approach to find the maximum subarray sum, showing how to build the solution iteratively. \* \*\*Counting Bits:\*\* Generalizes the solution for counting bits using a dynamic programming approach, leveraging the pattern of repeating bit patterns. \* \*\*Range Sum Query - Immutable:\*\* Presents a solution using prefix sums to efficiently calculate the sum of a range of numbers.

#### \*\*VII. LeetCode Problem Walkthrough (Backtracking Section):\*\*

\* \*\*Letter Case Permutation:\*\* Explains an iterative approach to generate all possible letter case permutations of a string. \* \*\*Subsets:\*\* Demonstrates a recursive backtracking approach to generate all subsets of an array. \* \*\*Combinations:\*\* Explains a recursive backtracking approach to generate all combinations of  $K$  numbers chosen from a range. \*  
\* \*\*Permutations:\*\* Presents a recursive backtracking approach to generate all permutations

of an array.

#### **\*\*VIII. LeetCode Problem Walkthrough (Linked List Section):\*\***

\* **Middle of the Linked List:** Uses the slow and fast pointer approach to find the middle node in a linked list. \* **Linked List Cycle:** Uses the slow and fast pointer approach to detect a cycle in a linked list. \* **Reverse Linked List:** Provides an iterative solution to reverse a linked list using three pointers. \* **Remove Linked List Elements:** Shows how to remove nodes with a specific value from a linked list using a dummy head. \* **Reverse Linked List II:** Explains how to reverse a specific portion of a linked list using a dummy head and three pointers. \* **Palindrome Linked List:** Combines finding the middle node, reversing the second half, and comparing the two halves to determine if a linked list is a palindrome. \* **Merge Two Sorted Lists:** Demonstrates merging two sorted linked lists into a single sorted list using a dummy head and a while loop.

#### **\*\*IX. LeetCode Problem Walkthrough (Stacks Section):\*\***

\* **Implement Queue using Stacks:** Implements a queue using two stacks, demonstrating the fundamental differences between the two data structures. \* **Valid Parentheses:** Uses a stack to check if a string of parentheses is valid, highlighting how language parsers work. \* **Evaluate Reverse Polish Notation:** Uses a stack to evaluate an arithmetic expression in reverse Polish notation. \* **Sort Stack:** Provides a pseudo-code solution for sorting a stack of integers using a temporary stack.

#### **\*\*X. LeetCode Problem Walkthrough (Queues Section):\*\***

\* **Implement Stack using Queues:** Implements a stack using a single queue, demonstrating the differences between the two data structures. \* **Time Needed to Buy Tickets:** Presents a linear solution to the problem of calculating the time needed for a person to buy tickets in a queue, emphasizing the importance of logical reasoning. \* **Reverse First K Elements of Queue:** Explains how to reverse the first K elements of a queue using a stack.

#### **\*\*XI. LeetCode Problem Walkthrough (Binary Trees Section):\*\***

\* **Average of Levels in Binary Tree:** Uses breadth-first search (BFS) to calculate the average value of nodes at each level. \* **Minimum Depth of Binary Tree:** Uses breadth-first search (BFS) to find the minimum depth of a binary tree. \* **Maximum Depth of Binary Tree:** Presents both iterative and recursive solutions to find the maximum depth of a binary tree. \* **Maximum and Minimum Node in Binary Tree:** Uses breadth-first search (BFS) to find the minimum and maximum value nodes in a binary tree. \* **Level Order Traversal:** Performs a level order traversal (BFS) and returns the nodes at each level in separate arrays. \* **Same Tree:** Implements a depth-first search (DFS) to check if two binary trees are identical. \* **Path Sum:** Uses a depth-first search (DFS) to determine if a binary tree has a root-to-leaf path that sums to a given target. \* **Diameter of Binary Tree:** Presents both iterative and recursive solutions to find the diameter of a binary tree. \* **Invert Binary Tree:** Shows how to invert a binary tree (swap left and right children) using a depth-first search (DFS). \* **Lowest Common Ancestor of a Binary Tree:** Provides both iterative (BFS) and recursive (DFS) solutions to find the lowest common ancestor of two nodes in a binary tree.

#### **\*\*XII. LeetCode Problem Walkthrough (Binary Search Trees Section):\*\***

\* **Search in a Binary Search Tree:** Demonstrates searching for a node with a specific value in a binary search tree. \* **Insert into a Binary Search Tree:** Provides an iterative solution to insert a new node into a binary search tree. \* **Convert Sorted Array to Binary Search Tree:** Presents a solution to convert a sorted array into a balanced binary search

tree, utilizing concepts from the array section. \* \*\*Two Sum IV - Input is a BST:\*\* Uses breadth-first search (BFS) and a set to determine if two nodes in a binary search tree sum up to a given target. \* \*\*Lowest Common Ancestor of a Binary Search Tree:\*\* Provides an iterative solution to find the lowest common ancestor in a binary search tree. \* \*\*Minimum Absolute Difference in BST:\*\* Uses an in-order traversal to find the minimum absolute difference between any two nodes in a binary search tree. \* \*\*Balance a Binary Search Tree:\*\* Combines in-order traversal and the "Convert Sorted Array to BST" problem to balance a binary search tree. \* \*\*Delete Node in a BST:\*\* Presents an iterative solution to delete a node from a binary search tree, covering all three cases (leaf, one child, two children). \* \*\*Kth Smallest Element in a BST:\*\* Uses an in-order traversal to find the kth smallest element in a binary search tree.

#### **\*\*XIII. LeetCode Problem Walkthrough (Heaps Section):\*\***

\* \*\*Kth Largest Element in an Array:\*\* Explores three solutions: using the `nlargest` function, creating a heap manually, and a hybrid approach. \* \*\*K Closest Points to Origin:\*\* Uses a Max Heap to find the K closest points to the origin. \* \*\*Top K Frequent Elements:\*\* Uses a Max Heap to find the K most frequent elements in an array. \* \*\*Task Scheduler:\*\* Uses a Max Heap to determine the minimum time required to schedule tasks with a cooling time constraint.

#### **\*\*XIV. LeetCode Problem Walkthrough (Graphs Section):\*\***

\* \*\*Clone Graph:\*\* Implements a breadth-first search (BFS) to create a deep copy of a graph. \* \*\*Core Graph Operations:\*\* Discusses finding the largest node, detecting cycles, and counting edges in a graph. \* \*\*Cheapest Flights Within K Stops:\*\* Uses the Bellman-Ford algorithm to find the cheapest flights with a maximum number of stops. \* \*\*Course Schedule:\*\* Explains how to use a graph to represent course prerequisites and uses an iterative depth-first search (DFS) to determine if it's possible to complete all courses.

#### **\*\*XV. Conclusion:\*\***

The video concludes with a strong encouragement to continue practicing and developing coding skills, emphasizing that success in LeetCode and interviews is achievable through consistent effort and dedication.