Here are the notes for all 75 Blind 75 LeetCode problems:

1. **Two Sum: Use a hash table to store each element and its index. For each element, check if its complement exists in the hash table.**
2. **Best Time to Buy and Sell Stock: Traverse the array and keep track of the minimum price so far and the maximum profit that can be made by selling at each price.**
3. **Maximum Subarray: Use Kadane's algorithm to find the maximum subarray sum.**
4. **Merge Intervals: Sort the intervals by start time, and then merge overlapping intervals.**
5. **Group Anagrams: Use a hash table to group anagrams by their sorted forms.**
6. **Merge Two Sorted Lists: Use a dummy node to build a new list by comparing the nodes of the two input lists.**
7. **Valid Parentheses: Use a stack to keep track of open parentheses. For each closing parenthesis, check if its corresponding open parenthesis is at the top of the stack.**
8. **Merge k Sorted Lists: Use a priority queue to merge the lists in a sorted order.**
9. **Climbing Stairs: Use dynamic programming to calculate the number of ways to climb the stairs.**
10. **Unique Paths: Use dynamic programming to calculate the number of unique paths from the top left to the bottom right corner of a grid.**
11. **Min Stack: Use two stacks, one to store the values and another to store the minimum values seen so far.**
12. **Best Time to Buy and Sell Stock II: Traverse the array and buy at each local minimum and sell at each local maximum.**
13. **Word Search: Use backtracking to check if a word can be formed by traversing the grid.**
14. **Maximum Product Subarray: Use dynamic programming to find the maximum product subarray.**
15. **Word Break: Use dynamic programming to check if a string can be segmented into a space-separated sequence of dictionary words.**
16. **Course Schedule: Use a topological sort to check if a directed graph is acyclic.**
17. **Merge Intervals II: Use a priority queue to merge overlapping intervals.**
18. **Intersection of Two Linked Lists: Traverse the two lists to find their length and align them at the same starting point. Then traverse them together until the nodes match.**
19. **Subarray Sum Equals K: Use a hash table to store the cumulative sum up to each index, and check if the complement of the target sum exists in the hash table.**
20. **Kth Largest Element in an Array: Use a priority queue to keep track of the k largest elements seen so far.**

21. **Lowest Common Ancestor of a Binary Tree: Use recursion to find the lowest common ancestor of two nodes in a binary tree.**
22. **Product of Array Except Self: Use dynamic programming to calculate the product of all elements except the current one.**
23. **Serialize and Deserialize Binary Tree: Use recursion to serialize and deserialize a binary tree.**
24. **Maximum Depth of Binary Tree: Use recursion to find the maximum depth of a binary tree.**
25. **Valid Sudoku: Use three hash tables to keep track of the validity of each row, column, and box.**
26. **Binary Tree Maximum Path Sum: Use recursion to find the maximum path sum in a binary tree.**
27. **Longest Consecutive Sequence: Use a hash table to keep track of the longest consecutive sequence seen so far.**
28. **Merge k Sorted Arrays: Use a priority queue to merge the arrays in a sorted order.**
29. **Longest Increasing Subsequence: Use dynamic programming to find the length of the longest increasing subsequence.**
30. **Course Schedule II: Use a topological sort to check if a directed graph is acyclic and return the order of courses.**
31. **Palindrome Linked List: Use two pointers to find the middle of the list, reverse the list.**
32. **Reorder List: Use two pointers to find the middle of the list, reverse the second half of the list, and then merge the two halves.**
33. **Binary Tree Level Order Traversal: Use a queue to traverse a binary tree level by level.**
34. **LRU Cache: Use a doubly linked list and a hash table to implement a least recently used cache.**
35. **Sort Colors: Use three pointers to keep track of the boundaries of each color.**
36. **Combination Sum: Use backtracking to find all combinations of numbers that add up to a target sum.**
37. **Minimum Window Substring: Use two pointers to maintain a sliding window that contains all the characters of a target substring.**
38. **Find the Duplicate Number: Use Floyd's cycle detection algorithm to find a duplicate number in an array.**
39. **Game of Life: Use two bits to store the current and next states of each cell in a matrix.**
40. **Reverse Nodes in k-Group: Use recursion to reverse groups of k nodes in a linked list.**
41. **First Missing Positive: Use an in-place algorithm to partition an array into positive and negative numbers, and then use a hash table to find the first missing positive.**

42. **Trapping Rain Water: Use two pointers to maintain a sliding window that can trap rain water.**
43. **Multiply Strings: Use long multiplication to multiply two strings.**
44. **Wildcard Matching: Use dynamic programming to match a string against a pattern that contains wildcard characters.**
45. **Jump Game II: Use dynamic programming to find the minimum number of jumps to reach the end of an array.**
46. **Permutations: Use backtracking to find all permutations of a set of numbers.**
47. **Rotate Image: Use matrix transposition and reflection to rotate an image in place.**
48. **Group Anagrams II: Use a hash table to group anagrams by their character count.**
49. **Pow(x, n): Use recursion to calculate the power of a number.**
50. **N-Queens: Use backtracking to find all solutions to the N-Queens problem.**
51. **N-Queens II: Use backtracking to find the number of solutions to the N-Queens problem.**
52. **Maximum Subarray II: Use dynamic programming to find the maximum subarray sum that can be obtained by removing one element from the array.**
53. **Find Minimum in Rotated Sorted Array: Use binary search to find the minimum element in a rotated sorted array.**
54. **Find Peak Element: Use binary search to find a peak element in an array.**
55. **Word Search II: Use a trie to efficiently find all words that can be formed by traversing a grid.**
56. **Merge Intervals III: Use a binary search tree to merge overlapping intervals.**
57. **Insert Interval: Use two pointers to maintain a sliding window that can merge an interval into a list of intervals.**
58. **Length of Last Word: Use string manipulation to find the length of the last word in a sentence.**
59. **Spiral Matrix II: Use simulation to generate a spiral matrix of integers.**
60. **Unique Paths II: Use dynamic programming to calculate the number of unique paths from the top left to the bottom right corner of a grid with obstacles.**
61. **Minimum Path Sum: Use dynamic programming to find the minimum sum of a path from the top left to the bottom right corner of a grid.**
62. **Valid Number: Use regular expressions to validate a string as a number.**
63. **Maximum Gap: Use bucket sort to find the maximum gap between adjacent numbers in an array.**
64. **Text Justification: Use greedy algorithm to justify a list of words into lines of equal width.**
65. **Sudoku Solver: Use backtracking to solve a Sudoku puzzle.**
66. **Palindrome Partitioning: Use backtracking to find all palindromic partitions of a string.**

67. **Binary Tree Maximum Path Sum: Use recursion to find the maximum sum path in a binary tree.**
68. **Valid Parentheses: Use a stack to validate a string containing only parentheses.**
69. **Merge Sorted Array: Use two pointers to merge two sorted arrays in place.**
70. **Subsets: Use backtracking to find all subsets of a set of numbers.**
71. **Simplify Path: Use a stack to simplify a Unix-style file path.**
72. **Edit Distance: Use dynamic programming to find the minimum number of operations to convert one string into another.**
73. **Largest Rectangle in Histogram: Use a stack to maintain a sliding window of heights and find the largest rectangle that can be formed by the heights.**
74. **Maximal Rectangle: Use dynamic programming to find the maximum area rectangle that can be formed by a matrix of ones and zeros.**
75. **Unique Binary Search Trees: Use dynamic programming to find the number of unique binary search trees that can be formed from a set of numbers.**
66.