# DSA Cheat Sheet

This cheat sheet summarizes the 24 LeetCode problems (14 Easy, 10 Medium) with brief descriptions, key approaches, time complexities, and space complexities. Approaches are based on optimal solutions in Golang (or general DSA principles). For full code and details, refer to the original document.

## Easy Problems

### 1. Two Sum (#1)

- **Description:** Find indices of two numbers in an array that add up to a target.
- **Approach:** Use a hash map to store numbers and indices; check for complement while iterating.
- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

### 2. Valid Parentheses (#20)

- **Description:** Check if a string of parentheses is valid (properly opened and closed).
- **Approach:** Use a stack to push opening brackets and pop/match closing ones.
- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

### 3. Merge Two Sorted Lists (#21)

- **Description:** Merge two sorted linked lists into one sorted list.
- **Approach:** Use a dummy node and two pointers to compare and link smaller nodes.
- **Time Complexity:** O(m + n) where m and n are list lengths
- **Space Complexity:** O(1) (excluding output)

### 4. Best Time to Buy and Sell Stock (#121)

- **Description:** Find maximum profit from buying and selling a stock once.
- **Approach:** Track minimum price and maximum profit while iterating.
- **Time Complexity:** O(n)

- **Space Complexity:** O(1)

### 5. Valid Palindrome (#125)

- **Description:** Check if a string is a palindrome ignoring non-alphanumeric and case.
- **Approach:** Two pointers from ends, skip non-alphanumeric, compare lowercase.
- **Time Complexity:** O(n)
- **Space Complexity:** O(1)

### 6. Linked List Cycle (#141)

- **Description:** Detect if a linked list has a cycle.
- **Approach:** Floyd's cycle detection with slow and fast pointers.
- **Time Complexity:** O(n)
- **Space Complexity:** O(1)

### 7. Maximum Depth of Binary Tree (#104)

- **Description:** Find the maximum depth of a binary tree.
- **Approach:** Recursive DFS to compute max of left/right depths +1.
- **Time Complexity:** O(n)
- **Space Complexity:** O(h) where h is tree height (worst O(n))

### 8. Binary Search (#704)

- **Description:** Search for a target in a sorted array, return index or -1.
- **Approach:** Two pointers (low, high), compute mid, adjust based on comparison.
- **Time Complexity:** O(log n)
- **Space Complexity:** O(1)

### 9. Flood Fill (#733)

- **Description:** Change color of connected pixels in an image starting from a point.
- **Approach:** DFS or BFS to visit 4-directional connected same-color pixels.
- **Time Complexity:** O(m * n) where m,n are grid dimensions
- **Space Complexity:** O(m * n) (recursion stack or queue)

## 10. Lowest Common Ancestor of a BST (#235)

- **Description:** Find LCA of two nodes in a binary search tree.
- **Approach:** Traverse based on values: go left/right if both smaller/larger, else current is LCA.
- **Time Complexity:** O(h) where h is height (worst O(n))
- **Space Complexity:** O(1)

## 11. Balanced Binary Tree (#110)

- **Description:** Check if a binary tree is height-balanced (subtree depths differ <=1).
- **Approach:** Recursive height calculation; return -1 if unbalanced.
- **Time Complexity:** O(n)
- **Space Complexity:** O(h) (worst O(n))

## 12. Implement Queue using Stacks (#232)

- **Description:** Implement FIFO queue using two stacks.
- **Approach:** One stack for input (push), transfer to output stack for pop/peek (amortized).
- **Time Complexity:** Push O(1), Pop/Peek amortized O(1)
- **Space Complexity:** O(n)

## 13. Min Stack (#155)

- **Description:** Stack with push/pop/top and O(1) getMin.
- **Approach:** Two stacks: one for values, one for current mins.
- **Time Complexity:** O(1) for all operations
- **Space Complexity:** O(n)

## 14. Invert Binary Tree (#226)

- **Description:** Invert (swap left/right) a binary tree.
- **Approach:** Recursive DFS: swap children and recurse.
- **Time Complexity:** O(n)
- **Space Complexity:** O(h) (worst O(n))

# Medium Problems

## 15. Product of Array Except Self (#238)

- **Description:** Compute array where each element is product of all others (no division).
- **Approach:** Two passes: left products, then multiply by right products.
- **Time Complexity:** O(n)
- **Space Complexity:** O(n) (can optimize to O(1) with output array)

## 16. Maximum Subarray (Kadane's) (#53)

- **Description:** Find contiguous subarray with maximum sum.
- **Approach:** Kadane's: track current sum (reset if negative), update max.
- **Time Complexity:** O(n)
- **Space Complexity:** O(1)

## 17. 3Sum (#15)

- **Description:** Find unique triplets that sum to 0.
- **Approach:** Sort array, fix i, two pointers for j/k; skip duplicates.
- **Time Complexity:** O(n^2)
- **Space Complexity:** O(1) (excluding output)

## 18. Container With Most Water (#11)

- **Description:** Find two lines forming max area container with heights.
- **Approach:** Two pointers from ends, move smaller height; compute area.
- **Time Complexity:** O(n)
- **Space Complexity:** O(1)

## 19. Longest Substring Without Repeating Characters (#3)

- **Description:** Find length of longest substring without repeats.
- **Approach:** Sliding window with set; move left on duplicate.
- **Time Complexity:** O(n)
- **Space Complexity:** O(min(n, charset)) ~ O(1) for ASCII

## 20. Top K Frequent Elements (#347)

- **Description:** Find k most frequent elements in array.
- **Approach:** Frequency map + bucket sort (buckets by freq).
- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

## 21. Number of Islands (#200)

- **Description:** Count islands ('1's connected 4-directionally) in grid.
- **Approach:** DFS/BFS on each '1' to mark connected as '0'; count starts.
- **Time Complexity:** O(m * n)
- **Space Complexity:** O(m * n) (worst recursion/queue)

## 22. Clone Graph (#133)

- **Description:** Deep copy an undirected graph.
- **Approach:** DFS/BFS with map for visited/cloned nodes.
- **Time Complexity:** O(n + e) where e is edges
- **Space Complexity:** O(n)

## 23. House Robber (#198)

- **Description:** Max money robbing non-adjacent houses.
- **Approach:** DP: max = max(prev, prev_prev + current); use variables.
- **Time Complexity:** O(n)
- **Space Complexity:** O(1)

## 24. Course Schedule (#207)

- **Description:** Check if courses can be completed without cycle in prerequisites.
- **Approach:** Graph cycle detection via DFS (visiting/visited states).
- **Time Complexity:** O(n + e)
- **Space Complexity:** O(n + e)