Complete LeetCode DSA Patterns Cheatsheet

Table of Contents

1. Two Pointers
2. Sliding Window
3. Fast & Slow Pointers
4. Merge Intervals
5. Cyclic Sort
6. In-place Reversal of LinkedList
7. Tree Breadth First Search
8. Tree Depth First Search
9. Two Heaps
10. Subsets
11. Modified Binary Search
12. Bitwise XOR
13. Top K Elements
14. K-way Merge
15. 0/1 Knapsack
16. Unbounded Knapsack
17. Fibonacci Numbers
18. Palindromic Subsequence
19. Longest Common Substring
20. Topological Sort
21. Trie
22. Union Find
23. Monotonic Stack
24. Backtracking
25. Best Practices & Next Steps

---

1. Two Pointers

When to use: Problems involving sorted arrays, pairs, triplets, or subarrays.

Time Complexity: O(n) or O(n^2) **Space Complexity:** O(1)

Pattern Recognition:

- Target sum problems
- Removing duplicates
- Comparing elements from both ends
- Palindrome verification

Template:

```python
def two_pointers(arr):
    left, right = 0, len(arr) - 1

    while left < right:
        # Process current pair
        current_sum = arr[left] + arr[right]

        if current_sum == target:
            return [left, right]
        elif current_sum < target:
            left += 1
        else:
            right -= 1

    return []
```

Example: Two Sum II (Sorted Array)

```python
def two_sum(numbers, target):
    left, right = 0, len(numbers) - 1

    while left < right:
        current_sum = numbers[left] + numbers[right]
        if current_sum == target:
            return [left + 1, right + 1]  # 1-indexed
        elif current_sum < target:
            left += 1
        else:
            right -= 1

    return []
```

Three Sum Pattern:

```python
def three_sum(nums):
    nums.sort()
    result = []

    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i-1]:  # Skip duplicates
            continue

        left, right = i + 1, len(nums) - 1

        while left < right:
            current_sum = nums[i] + nums[left] + nums[right]

            if current_sum == 0:
```

```python
                result.append([nums[i], nums[left], nums[right]])

                # Skip duplicates
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1

                left += 1
                right -= 1
            elif current_sum < 0:
                left += 1
            else:
                right -= 1

    return result
```

Practice Problems:

1. Two Sum II - Input array is sorted (LeetCode 167)
2. 3Sum (LeetCode 15)
3. 3Sum Closest (LeetCode 16)
4. Remove Duplicates from Sorted Array (LeetCode 26)
5. Container With Most Water (LeetCode 11)

---

2. Sliding Window

When to use: Problems involving contiguous subarrays/substrings with specific conditions.

Time Complexity: O(n) **Space Complexity:** O(1) to O(k)

Pattern Recognition:

- Maximum/minimum subarray of size K
- Substrings with K distinct characters
- String permutation problems
- Longest substring with condition

Fixed Window Template:

```python
def sliding_window_fixed(arr, k):
    window_sum = sum(arr[:k])
    max_sum = window_sum

    for i in range(k, len(arr)):
        # Slide window: remove first element, add new element
        window_sum = window_sum - arr[i - k] + arr[i]
        max_sum = max(max_sum, window_sum)
```

```
    return max_sum
```

Variable Window Template:

```python
def sliding_window_variable(s, condition):
    left = 0
    result = 0
    window_state = {}

    for right in range(len(s)):
        # Expand window
        char = s[right]
        window_state[char] = window_state.get(char, 0) + 1

        # Contract window if condition violated
        while not is_valid(window_state, condition):
            left_char = s[left]
            window_state[left_char] -= 1
            if window_state[left_char] == 0:
                del window_state[left_char]
            left += 1

        # Update result
        result = max(result, right - left + 1)

    return result
```

Example: Longest Substring Without Repeating Characters

```python
def length_of_longest_substring(s):
    char_index = {}
    left = 0
    max_length = 0

    for right in range(len(s)):
        if s[right] in char_index and char_index[s[right]] >= left:
            left = char_index[s[right]] + 1

        char_index[s[right]] = right
        max_length = max(max_length, right - left + 1)

    return max_length
```

Example: Minimum Window Substring

```python
def min_window(s, t):
    if not s or not t:
        return ""

    # Count characters in t
    dict_t = {}
    for char in t:
        dict_t[char] = dict_t.get(char, 0) + 1

    required = len(dict_t)
    left, right = 0, 0
    formed = 0
    window_counts = {}

    ans = float("inf"), None, None

    while right < len(s):
        char = s[right]
        window_counts[char] = window_counts.get(char, 0) + 1

        if char in dict_t and window_counts[char] == dict_t[char]:
            formed += 1

        while left <= right and formed == required:
            char = s[left]

            if right - left + 1 < ans[0]:
                ans = (right - left + 1, left, right)

            window_counts[char] -= 1
            if char in dict_t and window_counts[char] < dict_t[char]:
                formed -= 1

            left += 1

        right += 1

    return "" if ans[0] == float("inf") else s[ans[1]:ans[2] + 1]
```

Practice Problems:

1. Maximum Average Subarray I (LeetCode 643)
2. Longest Substring Without Repeating Characters (LeetCode 3)
3. Minimum Window Substring (LeetCode 76)
4. Longest Substring with At Most K Distinct Characters (LeetCode 340)
5. Permutation in String (LeetCode 567)

---

3. Fast & Slow Pointers

When to use: Linked list cycle detection, finding middle element, palindrome checking.

Time Complexity: O(n) **Space Complexity:** O(1)

Pattern Recognition:

- Cycle detection in linked lists
- Finding middle of linked list
- Palindrome linked list
- Happy number problem

Template:

```python
def has_cycle(head):
    if not head or not head.next:
        return False

    slow = fast = head

    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            return True

    return False
```

Finding Cycle Start:

```python
def detect_cycle(head):
    if not head or not head.next:
        return None

    # Phase 1: Detect if cycle exists
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            break
    else:
        return None  # No cycle

    # Phase 2: Find cycle start
    slow = head
    while slow != fast:
        slow = slow.next
        fast = fast.next

    return slow
```

Finding Middle Node:

```python
def find_middle(head):
    slow = fast = head

    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    return slow
```

Palindrome Linked List:

```python
def is_palindrome(head):
    if not head or not head.next:
        return True

    # Find middle
    slow = fast = head
    while fast.next and fast.next.next:
        slow = slow.next
        fast = fast.next.next

    # Reverse second half
    second_half = reverse_list(slow.next)

    # Compare
    first_half = head
    while second_half:
        if first_half.val != second_half.val:
            return False
        first_half = first_half.next
        second_half = second_half.next

    return True

def reverse_list(head):
    prev = None
    current = head

    while current:
        next_temp = current.next
        current.next = prev
        prev = current
        current = next_temp

    return prev
```

Happy Number:

```python
def is_happy(n):
    def get_sum_of_squares(num):
        total_sum = 0
        while num > 0:
            digit = num % 10
            total_sum += digit * digit
            num //= 10
        return total_sum

    slow = fast = n

    while True:
        slow = get_sum_of_squares(slow)
        fast = get_sum_of_squares(get_sum_of_squares(fast))

        if fast == 1:
            return True
        if slow == fast:
            return False
```

Practice Problems:

1. Linked List Cycle (LeetCode 141)
2. Linked List Cycle II (LeetCode 142)
3. Middle of the Linked List (LeetCode 876)
4. Palindrome Linked List (LeetCode 234)
5. Happy Number (LeetCode 202)

---

4. Merge Intervals

When to use: Problems involving overlapping intervals, scheduling, range merging.

Time Complexity: O(n log n) **Space Complexity:** O(n)

Pattern Recognition:

- Overlapping intervals
- Meeting room problems
- Insert intervals
- Interval intersection

Template:

```python
def merge_intervals(intervals):
    if not intervals:
        return []
```

```python
    # Sort by start time
    intervals.sort(key=lambda x: x[0])
    merged = [intervals[0]]

    for current in intervals[1:]:
        last_merged = merged[-1]

        if current[0] <= last_merged[1]:  # Overlapping
            merged[-1] = [last_merged[0], max(last_merged[1], current[1])]
        else:  # Non-overlapping
            merged.append(current)

    return merged
```

Example: Insert Interval

```python
def insert(intervals, newInterval):
    result = []
    i = 0
    n = len(intervals)

    # Add all intervals that end before newInterval starts
    while i < n and intervals[i][1] < newInterval[0]:
        result.append(intervals[i])
        i += 1

    # Merge overlapping intervals
    while i < n and intervals[i][0] <= newInterval[1]:
        newInterval[0] = min(newInterval[0], intervals[i][0])
        newInterval[1] = max(newInterval[1], intervals[i][1])
        i += 1

    result.append(newInterval)

    # Add remaining intervals
    while i < n:
        result.append(intervals[i])
        i += 1

    return result
```

Example: Meeting Rooms II

```python
def min_meeting_rooms(intervals):
    if not intervals:
        return 0

    starts = sorted([interval[0] for interval in intervals])
```

```python
    ends = sorted([interval[1] for interval in intervals])

    start_pointer = end_pointer = 0
    used_rooms = 0

    while start_pointer < len(intervals):
        if starts[start_pointer] >= ends[end_pointer]:
            used_rooms -= 1
            end_pointer += 1

        used_rooms += 1
        start_pointer += 1

    return used_rooms
```

Example: Interval Intersection

```python
def interval_intersection(A, B):
    result = []
    i = j = 0

    while i < len(A) and j < len(B):
        # Find intersection
        start = max(A[i][0], B[j][0])
        end = min(A[i][1], B[j][1])

        if start <= end:
            result.append([start, end])

        # Move pointer with smaller end time
        if A[i][1] < B[j][1]:
            i += 1
        else:
            j += 1

    return result
```

Practice Problems:

1. Merge Intervals (LeetCode 56)
2. Insert Interval (LeetCode 57)
3. Meeting Rooms (LeetCode 252)
4. Meeting Rooms II (LeetCode 253)
5. Interval List Intersections (LeetCode 986)

---

5. Cyclic Sort

When to use: Problems with arrays containing numbers in a given range, missing numbers.

Time Complexity: O(n) **Space Complexity:** O(1)

Pattern Recognition:

- Array contains numbers from 1 to n
- Finding missing/duplicate numbers
- First missing positive

Template:

```python
def cyclic_sort(nums):
    i = 0
    while i < len(nums):
        correct_index = nums[i] - 1

        if nums[i] != nums[correct_index]:
            nums[i], nums[correct_index] = nums[correct_index], nums[i]
        else:
            i += 1

    return nums
```

Example: Find Missing Number

```python
def find_missing_number(nums):
    i = 0
    n = len(nums)

    # Cyclic sort
    while i < n:
        if nums[i] < n and nums[i] != nums[nums[i]]:
            nums[nums[i]], nums[i] = nums[i], nums[nums[i]]
        else:
            i += 1

    # Find missing number
    for i in range(n):
        if nums[i] != i:
            return i

    return n
```

Example: Find All Duplicates

```python
def find_duplicates(nums):
    i = 0
```

```
    # Cyclic sort
    while i < len(nums):
        correct_index = nums[i] - 1
        if nums[i] != nums[correct_index]:
            nums[i], nums[correct_index] = nums[correct_index], nums[i]
        else:
            i += 1

    # Find duplicates
    duplicates = []
    for i in range(len(nums)):
        if nums[i] != i + 1:
            duplicates.append(nums[i])

    return duplicates
```

Example: First Missing Positive

```
def first_missing_positive(nums):
    n = len(nums)

    # Place each positive number at its correct position
    for i in range(n):
        while 1 <= nums[i] <= n and nums[nums[i] - 1] != nums[i]:
            nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]

    # Find first missing positive
    for i in range(n):
        if nums[i] != i + 1:
            return i + 1

    return n + 1
```

Practice Problems:

1. Missing Number (LeetCode 268)
2. Find All Numbers Disappeared in an Array (LeetCode 448)
3. Find All Duplicates in an Array (LeetCode 442)
4. First Missing Positive (LeetCode 41)
5. Find the Duplicate Number (LeetCode 287)

---

6. In-place Reversal of LinkedList

When to use: Reversing linked lists or parts of linked lists without extra space.

Time Complexity: O(n) **Space Complexity:** O(1)

Pattern Recognition:

- Reverse entire linked list
- Reverse sublist
- Reverse in groups

Basic Reversal Template:

```
def reverse_list(head):
    prev = None
    current = head

    while current:
        next_temp = current.next
        current.next = prev
        prev = current
        current = next_temp

    return prev
```

Example: Reverse Sublist

```
def reverse_between(head, m, n):
    if m == n:
        return head

    # Find the node before position m
    dummy = ListNode(0)
    dummy.next = head
    prev = dummy

    for _ in range(m - 1):
        prev = prev.next

    # Reverse sublist from m to n
    current = prev.next
    for _ in range(n - m):
        next_node = current.next
        current.next = next_node.next
        next_node.next = prev.next
        prev.next = next_node

    return dummy.next
```

Example: Reverse Nodes in k-Group

```
def reverse_k_group(head, k):
    # Check if we have k nodes left
    count = 0
```

```
    current = head
    while current and count < k:
        current = current.next
        count += 1

    if count == k:  # We have k nodes
        current = reverse_k_group(current, k)  # Recursively process rest

        # Reverse current k nodes
        while count > 0:
            next_temp = head.next
            head.next = current
            current = head
            head = next_temp
            count -= 1

        head = current

    return head
```

Example: Reverse Alternate k-Group

```
def reverse_alternate_k_group(head, k):
    if k <= 1 or not head:
        return head

    current = head
    prev = None

    while current:
        last_node_prev_part = prev
        last_node_sub_list = current

        # Skip k nodes
        for _ in range(k):
            if not current:
                break
            next_temp = current.next
            current.next = prev
            prev = current
            current = next_temp

        if last_node_prev_part:
            last_node_prev_part.next = prev
        else:
            head = prev

        last_node_sub_list.next = current

        # Skip k nodes
        for _ in range(k):
```

```python
            if not current:
                break
            prev = current
            current = current.next

    return head
```

Practice Problems:

1. Reverse Linked List (LeetCode 206)
2. Reverse Linked List II (LeetCode 92)
3. Reverse Nodes in k-Group (LeetCode 25)
4. Swap Nodes in Pairs (LeetCode 24)
5. Rotate List (LeetCode 61)

---

7. Tree Breadth First Search

When to use: Level-order traversal, finding shortest path in unweighted trees.

Time Complexity: O(n) **Space Complexity:** O(w) where w is maximum width of tree

Pattern Recognition:

- Level-order traversal
- Finding minimum depth
- Connecting nodes at same level
- Zigzag traversal

Template:

```python
from collections import deque

def level_order(root):
    if not root:
        return []

    result = []
    queue = deque([root])

    while queue:
        level_size = len(queue)
        current_level = []

        for _ in range(level_size):
            node = queue.popleft()
            current_level.append(node.val)

            if node.left:
                queue.append(node.left)
```

```
            if node.right:
                queue.append(node.right)

        result.append(current_level)

    return result
```

Example: Binary Tree Right Side View

```
def right_side_view(root):
    if not root:
        return []

    result = []
    queue = deque([root])

    while queue:
        level_size = len(queue)

        for i in range(level_size):
            node = queue.popleft()

            # Add rightmost node of each level
            if i == level_size - 1:
                result.append(node.val)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

    return result
```

Example: Zigzag Level Order

```
def zigzag_level_order(root):
    if not root:
        return []

    result = []
    queue = deque([root])
    left_to_right = True

    while queue:
        level_size = len(queue)
        current_level = deque()

        for _ in range(level_size):
            node = queue.popleft()
```

```
            if left_to_right:
                current_level.append(node.val)
            else:
                current_level.appendleft(node.val)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        result.append(list(current_level))
        left_to_right = not left_to_right

    return result
```

Example: Minimum Depth

```
def min_depth(root):
    if not root:
        return 0

    queue = deque([(root, 1)])

    while queue:
        node, depth = queue.popleft()

        if not node.left and not node.right:
            return depth

        if node.left:
            queue.append((node.left, depth + 1))
        if node.right:
            queue.append((node.right, depth + 1))

    return 0
```

Practice Problems:

1. Binary Tree Level Order Traversal (LeetCode 102)
2. Binary Tree Right Side View (LeetCode 199)
3. Binary Tree Zigzag Level Order Traversal (LeetCode 103)
4. Minimum Depth of Binary Tree (LeetCode 111)
5. Average of Levels in Binary Tree (LeetCode 637)

---

8. Tree Depth First Search

When to use: Path problems, tree validation, finding all paths.

Time Complexity: O(n) **Space Complexity:** O(h) where h is height of tree

Pattern Recognition:

- All root-to-leaf paths
- Path sum problems
- Tree validation
- Finding diameter/height

Template (Recursive):

```python
def dfs(root):
    if not root:
        return

    # Process current node
    print(root.val)

    # Recursively process children
    dfs(root.left)
    dfs(root.right)
```

Template (Iterative):

```python
def dfs_iterative(root):
    if not root:
        return

    stack = [root]

    while stack:
        node = stack.pop()

        # Process current node
        print(node.val)

        # Add children to stack (right first for left-to-right processing)
        if node.right:
            stack.append(node.right)
        if node.left:
            stack.append(node.left)
```

Example: Path Sum

```python
def has_path_sum(root, sum_target):
    if not root:
        return False
```

```python
    # Leaf node
    if not root.left and not root.right:
        return root.val == sum_target

    # Recursively check left and right subtrees
    remaining_sum = sum_target - root.val
    return (has_path_sum(root.left, remaining_sum) or
            has_path_sum(root.right, remaining_sum))
```

Example: All Root-to-Leaf Paths

```python
def binary_tree_paths(root):
    result = []

    def dfs(node, path):
        if not node:
            return

        path.append(str(node.val))

        # Leaf node
        if not node.left and not node.right:
            result.append("->".join(path))
        else:
            dfs(node.left, path)
            dfs(node.right, path)

        path.pop()  # Backtrack

    dfs(root, [])
    return result
```

Example: Maximum Path Sum

```python
def max_path_sum(root):
    max_sum = float('-inf')

    def max_gain(node):
        nonlocal max_sum

        if not node:
            return 0

        # Maximum gain from left and right subtrees
        left_gain = max(max_gain(node.left), 0)
        right_gain = max(max_gain(node.right), 0)

        # Maximum path sum through current node
```

```
        current_max = node.val + left_gain + right_gain
        max_sum = max(max_sum, current_max)

        # Return maximum gain if we continue path through current node
        return node.val + max(left_gain, right_gain)

    max_gain(root)
    return max_sum
```

Example: Diameter of Binary Tree

```
def diameter_of_binary_tree(root):
    diameter = 0

    def depth(node):
        nonlocal diameter

        if not node:
            return 0

        left_depth = depth(node.left)
        right_depth = depth(node.right)

        # Update diameter
        diameter = max(diameter, left_depth + right_depth)

        return max(left_depth, right_depth) + 1

    depth(root)
    return diameter
```

Practice Problems:

1. Path Sum (LeetCode 112)
2. Path Sum II (LeetCode 113)
3. Binary Tree Paths (LeetCode 257)
4. Maximum Path Sum (LeetCode 124)
5. Diameter of Binary Tree (LeetCode 543)

---

9. Two Heaps

When to use: Finding median in data stream, sliding window median.

Time Complexity: O(log n) for insertion, O(1) for median **Space Complexity:** O(n)

Pattern Recognition:

- Finding median

- Sliding window median
- Scheduling problems with priorities

Template:

```python
import heapq

class MedianFinder:
    def __init__(self):
        self.small_heap = []  # Max heap (negate values)
        self.large_heap = []  # Min heap

    def add_num(self, num):
        # Add to appropriate heap
        if not self.small_heap or num <= -self.small_heap[0]:
            heapq.heappush(self.small_heap, -num)
        else:
            heapq.heappush(self.large_heap, num)

        # Balance heaps
        if len(self.small_heap) > len(self.large_heap) + 1:
            val = -heapq.heappop(self.small_heap)
            heapq.heappush(self.large_heap, val)
        elif len(self.large_heap) > len(self.small_heap) + 1:
            val = heapq.heappop(self.large_heap)
            heapq.heappush(self.small_heap, -val)

    def find_median(self):
        if len(self.small_heap) == len(self.large_heap):
            return (-self.small_heap[0] + self.large_heap[0]) / 2.0
        elif len(self.small_heap) > len(self.large_heap):
            return -self.small_heap[0]
        else:
            return self.large_heap[0]
```

Example: Sliding Window Median

```python
def median_sliding_window(nums, k):
    result = []

    for i in range(len(nums) - k + 1):
        window = sorted(nums[i:i+k])

        if k % 2 == 1:
            result.append(float(window[k // 2]))
        else:
            result.append((window[k // 2 - 1] + window[k // 2]) / 2.0)

    return result
```

```python
# Optimized version using two heaps
def median_sliding_window_optimized(nums, k):
    from collections import defaultdict
    import heapq

    def get_median(max_heap, min_heap, k):
        if k % 2 == 1:
            return -max_heap[0]
        else:
            return (-max_heap[0] + min_heap[0]) / 2.0

    max_heap = []  # Left half (negated for max heap)
    min_heap = []  # Right half
    hash_table = defaultdict(int)
    result = []

    # Initialize first window
    for i in range(k):
        heapq.heappush(max_heap, -nums[i])

    # Move half to min_heap
    for _ in range(k // 2):
        val = -heapq.heappop(max_heap)
        heapq.heappush(min_heap, val)

    result.append(get_median(max_heap, min_heap, k))

    # Slide window
    for i in range(k, len(nums)):
        # Remove outgoing element
        out_num = nums[i - k]
        hash_table[out_num] += 1

        # Add incoming element
        in_num = nums[i]
        if max_heap and in_num <= -max_heap[0]:
            heapq.heappush(max_heap, -in_num)
        else:
            heapq.heappush(min_heap, in_num)

        # Balance heaps and clean invalid elements
        balance_heaps(max_heap, min_heap, hash_table)
        result.append(get_median(max_heap, min_heap, k))

    return result

def balance_heaps(max_heap, min_heap, hash_table):
    # Remove invalid elements from tops
    while max_heap and hash_table[-max_heap[0]] > 0:
        hash_table[-max_heap[0]] -= 1
        heapq.heappop(max_heap)

    while min_heap and hash_table[min_heap[0]] > 0:
        hash_table[min_heap[0]] -= 1
```

```python
            heapq.heappop(min_heap)

    # Balance heap sizes
    if len(max_heap) > len(min_heap) + 1:
        val = -heapq.heappop(max_heap)
        heapq.heappush(min_heap, val)
    elif len(min_heap) > len(max_heap) + 1:
        val = heapq.heappop(min_heap)
        heapq.heappush(max_heap, -val)
```

Example: IPO Problem

```python
def find_maximized_capital(k, w, profits, capital):
    import heapq

    # Min heap for projects by capital requirement
    min_capital_heap = []
    # Max heap for profits of affordable projects
    max_profit_heap = []

    # Add all projects to capital heap
    for i in range(len(profits)):
        heapq.heappush(min_capital_heap, (capital[i], profits[i]))

    # Select up to k projects
    for _ in range(k):
        # Move all affordable projects to profit heap
        while min_capital_heap and min_capital_heap[0][0] <= w:
            cap, profit = heapq.heappop(min_capital_heap)
            heapq.heappush(max_profit_heap, -profit)

        # If no affordable projects, break
        if not max_profit_heap:
            break

        # Select most profitable project
        w += -heapq.heappop(max_profit_heap)

    return w
```

Practice Problems:

1. Find Median from Data Stream (LeetCode 295)
2. Sliding Window Median (LeetCode 480)
3. IPO (LeetCode 502)
4. Find Right Interval (LeetCode 436)
5. Maximum Capital (Custom problem)

---

10. Subsets

When to use: Generating all combinations, permutations, or subsets.

Time Complexity: O(2^n) for subsets, O(n!) for permutations **Space Complexity:** O(n) for recursion depth

Pattern Recognition:

- Generate all subsets/combinations
- Generate all permutations
- Parentheses generation
- Letter combinations

Subsets Template:

```python
def subsets(nums):
    result = []

    def backtrack(start, path):
        # Add current subset
        result.append(path[:])

        # Generate subsets starting from 'start'
        for i in range(start, len(nums)):
            path.append(nums[i])
            backtrack(i + 1, path)
            path.pop()

    backtrack(0, [])
    return result
```

Iterative Subsets:

```python
def subsets_iterative(nums):
    result = [[]]

    for num in nums:
        new_subsets = []
        for subset in result:
            new_subsets.append(subset + [num])
        result.extend(new_subsets)

    return result
```

Example: Subsets with Duplicates

```python
def subsets_with_dup(nums):
    nums.sort()  # Sort to handle duplicates
    result = []

    def backtrack(start, path):
        result.append(path[:])

        for i in range(start, len(nums)):
            # Skip duplicates
            if i > start and nums[i] == nums[i-1]:
                continue

            path.append(nums[i])
            backtrack(i + 1, path)
            path.pop()

    backtrack(0, [])
    return result
```

Example: Permutations

```python
def permute(nums):
    result = []

    def backtrack(path):
        if len(path) == len(nums):
            result.append(path[:])
            return

        for num in nums:
            if num in path:
                continue

            path.append(num)
            backtrack(path)
            path.pop()

    backtrack([])
    return result
```

Example: Generate Parentheses

```python
def generate_parenthesis(n):
    result = []

    def backtrack(path, open_count, close_count):
        if len(path) == 2 * n:
            result.append(path)
```

```
            return

        # Add opening parenthesis
        if open_count < n:
            backtrack(path + '(', open_count + 1, close_count)

        # Add closing parenthesis
        if close_count < open_count:
            backtrack(path + ')', open_count, close_count + 1)

    backtrack('', 0, 0)
    return result
```

Example: Letter Combinations of Phone Number

```
def letter_combinations(digits):
    if not digits:
        return []

    phone_map = {
        '2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
        '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'
    }

    result = []

    def backtrack(index, path):
        if index == len(digits):
            result.append(path)
            return

        letters = phone_map[digits[index]]
        for letter in letters:
            backtrack(index + 1, path + letter)

    backtrack(0, '')
    return result
```

Practice Problems:

1. Subsets (LeetCode 78)
2. Subsets II (LeetCode 90)
3. Permutations (LeetCode 46)
4. Generate Parentheses (LeetCode 22)
5. Letter Combinations of a Phone Number (LeetCode 17)

---

11. Modified Binary Search

When to use: Searching in rotated/modified sorted arrays, finding peak elements.

Time Complexity: O(log n) **Space Complexity:** O(1)

Pattern Recognition:

- Search in rotated sorted array
- Find peak element
- Search in infinite array
- Find minimum in rotated sorted array

Template:

```python
def binary_search_modified(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = left + (right - left) // 2

        if arr[mid] == target:
            return mid

        # Determine which side is sorted
        if arr[left] <= arr[mid]:  # Left side is sorted
            if arr[left] <= target < arr[mid]:
                right = mid - 1
            else:
                left = mid + 1
        else:  # Right side is sorted
            if arr[mid] < target <= arr[right]:
                left = mid + 1
            else:
                right = mid - 1

    return -1
```

Example: Search in Rotated Sorted Array

```python
def search(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2

        if nums[mid] == target:
            return mid

        # Left half is sorted
        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums[mid]:
```

```
                right = mid - 1
            else:
                left = mid + 1
        # Right half is sorted
        else:
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1

    return -1
```

Example: Find Minimum in Rotated Sorted Array

```
def find_min(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = (left + right) // 2

        # Right half has minimum
        if nums[mid] > nums[right]:
            left = mid + 1
        # Left half has minimum
        else:
            right = mid

    return nums[left]
```

Example: Find Peak Element

```
def find_peak_element(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = (left + right) // 2

        # Peak is on the right
        if nums[mid] < nums[mid + 1]:
            left = mid + 1
        # Peak is on the left
        else:
            right = mid

    return left
```

Example: Search in Infinite Array

```python
def search_in_infinite_array(reader, target):
    # Find bounds
    left, right = 0, 1
    while reader.get(right) < target:
        left = right
        right *= 2

    # Binary search in bounds
    while left <= right:
        mid = (left + right) // 2
        val = reader.get(mid)

        if val == target:
            return mid
        elif val < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
```

Example: Find First and Last Position

```python
def search_range(nums, target):
    def find_first():
        left, right = 0, len(nums) - 1
        result = -1

        while left <= right:
            mid = (left + right) // 2

            if nums[mid] == target:
                result = mid
                right = mid - 1  # Continue searching left
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1

        return result

    def find_last():
        left, right = 0, len(nums) - 1
        result = -1

        while left <= right:
            mid = (left + right) // 2

            if nums[mid] == target:
                result = mid
```

```
                left = mid + 1  # Continue searching right
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1

        return result

    return [find_first(), find_last()]
```

Practice Problems:

1. Search in Rotated Sorted Array (LeetCode 33)
2. Find Minimum in Rotated Sorted Array (LeetCode 153)
3. Find Peak Element (LeetCode 162)
4. Search for a Range (LeetCode 34)
5. Search in a Sorted Array of Unknown Size (LeetCode 702)

---

12. Bitwise XOR

When to use: Finding single numbers, missing numbers in arrays.

Time Complexity: O(n) **Space Complexity:** O(1)

Key Properties:

- $a \oplus a = 0$
- $a \oplus 0 = a$
- XOR is commutative and associative

Pattern Recognition:

- Single number problems
- Missing number in array
- Two single numbers

Template:

```python
def single_number(nums):
    result = 0
    for num in nums:
        result ^= num
    return result
```

Example: Single Number II

```python
def single_number_ii(nums):
    ones = twos = 0

    for num in nums:
        # Update twos with bits that appeared twice
        twos |= ones & num
        # Update ones with current number
        ones ^= num
        # Remove bits that appeared three times
        threes = ones & twos
        ones &= ~threes
        twos &= ~threes

    return ones
```

Example: Two Single Numbers

```python
def single_number_iii(nums):
    # XOR all numbers
    xor_all = 0
    for num in nums:
        xor_all ^= num

    # Find rightmost set bit
    rightmost_bit = xor_all & (-xor_all)

    # Divide numbers into two groups
    num1 = num2 = 0
    for num in nums:
        if num & rightmost_bit:
            num1 ^= num
        else:
            num2 ^= num

    return [num1, num2]
```

Example: Missing Number

```python
def missing_number(nums):
    n = len(nums)
    result = n  # Start with n

    for i in range(n):
        result ^= i ^ nums[i]

    return result
```

Example: Complement of Base 10 Integer

```
def find_complement(num):
    # Find bit length
    bit_length = num.bit_length()

    # Create mask with all 1s
    mask = (1 << bit_length) - 1

    # XOR with mask to flip bits
    return num ^ mask
```

Example: Flip and Invert Image

```
def flip_and_invert_image(A):
    for row in A:
        # Flip (reverse) and invert (XOR with 1)
        for i in range((len(row) + 1) // 2):
            j = len(row) - 1 - i
            row[i], row[j] = row[j] ^ 1, row[i] ^ 1

    return A
```

Practice Problems:

1. Single Number (LeetCode 136)
2. Single Number II (LeetCode 137)
3. Single Number III (LeetCode 260)
4. Missing Number (LeetCode 268)
5. Complement of Base 10 Integer (LeetCode 476)

---

13. Top K Elements

When to use: Finding K largest/smallest elements, K closest elements.

Time Complexity: O(n log k) with heap, O(n) with quickselect **Space Complexity:** O(k)

Pattern Recognition:

- K largest/smallest elements
- K most frequent elements
- K closest points

Template with Min Heap (for K largest):

```python
import heapq

def find_k_largest(nums, k):
    heap = []

    for num in nums:
        heapq.heappush(heap, num)
        if len(heap) > k:
            heapq.heappop(heap)

    return list(heap)
```

Template with Max Heap (for K smallest):

```python
def find_k_smallest(nums, k):
    heap = []

    for num in nums:
        heapq.heappush(heap, -num)  # Negate for max heap
        if len(heap) > k:
            heapq.heappop(heap)

    return [-x for x in heap]
```

Example: K Most Frequent Elements

```python
def top_k_frequent(nums, k):
    from collections import Counter
    import heapq

    count = Counter(nums)

    # Use min heap to keep top k frequent elements
    heap = []
    for num, freq in count.items():
        heapq.heappush(heap, (freq, num))
        if len(heap) > k:
            heapq.heappop(heap)

    return [num for freq, num in heap]
```

Example: K Closest Points to Origin

```python
def k_closest(points, k):
    import heapq
```

```
    heap = []

    for point in points:
        x, y = point
        dist = x*x + y*y

        heapq.heappush(heap, (-dist, point))  # Max heap
        if len(heap) > k:
            heapq.heappop(heap)

    return [point for dist, point in heap]
```

Example: Kth Largest Element in Array (QuickSelect)

```
def find_kth_largest(nums, k):
    def quickselect(left, right, k_smallest):
        if left == right:
            return nums[left]

        # Partition around random pivot
        pivot_index = partition(left, right)

        if k_smallest == pivot_index:
            return nums[k_smallest]
        elif k_smallest < pivot_index:
            return quickselect(left, pivot_index - 1, k_smallest)
        else:
            return quickselect(pivot_index + 1, right, k_smallest)

    def partition(left, right):
        pivot = nums[right]
        i = left

        for j in range(left, right):
            if nums[j] < pivot:
                nums[i], nums[j] = nums[j], nums[i]
                i += 1

        nums[i], nums[right] = nums[right], nums[i]
        return i

    # Kth largest is (n-k)th smallest
    return quickselect(0, len(nums) - 1, len(nums) - k)
```

Example: Find K Pairs with Smallest Sums

```
def k_smallest_pairs(nums1, nums2, k):
    import heapq
```

```
    if not nums1 or not nums2:
        return []

    heap = []
    result = []

    # Initialize heap with first row
    for j in range(min(k, len(nums2))):
        heapq.heappush(heap, (nums1[0] + nums2[j], 0, j))

    while heap and len(result) < k:
        sum_val, i, j = heapq.heappop(heap)
        result.append([nums1[i], nums2[j]])

        # Add next element from same column
        if i + 1 < len(nums1):
            heapq.heappush(heap, (nums1[i + 1] + nums2[j], i + 1, j))

    return result
```

Practice Problems:

1. Kth Largest Element in an Array (LeetCode 215)
2. Top K Frequent Elements (LeetCode 347)
3. K Closest Points to Origin (LeetCode 973)
4. Find K Pairs with Smallest Sums (LeetCode 373)
5. Kth Smallest Element in a Sorted Matrix (LeetCode 378)

---

14. K-way Merge

When to use: Merging K sorted arrays/lists, finding smallest range.

Time Complexity: O(n log k) where n is total elements **Space Complexity:** O(k)

Pattern Recognition:

- Merge K sorted lists
- Smallest range covering elements from K lists
- Kth smallest in sorted matrix

Template:

```
import heapq

def merge_k_sorted_arrays(arrays):
    heap = []
    result = []

    # Initialize heap with first element from each array
```

```python
    for i, array in enumerate(arrays):
        if array:
            heapq.heappush(heap, (array[0], i, 0))

    while heap:
        val, array_idx, element_idx = heapq.heappop(heap)
        result.append(val)

        # Add next element from same array
        if element_idx + 1 < len(arrays[array_idx]):
            next_val = arrays[array_idx][element_idx + 1]
            heapq.heappush(heap, (next_val, array_idx, element_idx + 1))

    return result
```

Example: Merge K Sorted Lists

```python
def merge_k_lists(lists):
    import heapq

    heap = []

    # Initialize heap
    for i, head in enumerate(lists):
        if head:
            heapq.heappush(heap, (head.val, i, head))

    dummy = ListNode(0)
    current = dummy

    while heap:
        val, i, node = heapq.heappop(heap)

        current.next = node
        current = current.next

        if node.next:
            heapq.heappush(heap, (node.next.val, i, node.next))

    return dummy.next
```

Example: Smallest Range Covering Elements from K Lists

```python
def smallest_range(nums):
    import heapq

    heap = []
    max_val = float('-inf')
```

```python
    # Initialize heap with first element from each list
    for i, lst in enumerate(nums):
        if lst:
            heapq.heappush(heap, (lst[0], i, 0))
            max_val = max(max_val, lst[0])

    range_start, range_end = 0, float('inf')

    while len(heap) == len(nums):
        min_val, list_idx, element_idx = heapq.heappop(heap)

        # Update range if current is smaller
        if max_val - min_val < range_end - range_start:
            range_start, range_end = min_val, max_val

        # Add next element from same list
        if element_idx + 1 < len(nums[list_idx]):
            next_val = nums[list_idx][element_idx + 1]
            heapq.heappush(heap, (next_val, list_idx, element_idx + 1))
            max_val = max(max_val, next_val)

    return [range_start, range_end]
```

Example: Kth Smallest Element in Sorted Matrix

```python
def kth_smallest(matrix, k):
    import heapq

    n = len(matrix)
    heap = []

    # Initialize heap with first column
    for i in range(n):
        heapq.heappush(heap, (matrix[i][0], i, 0))

    for _ in range(k):
        val, row, col = heapq.heappop(heap)

        if col + 1 < n:
            heapq.heappush(heap, (matrix[row][col + 1], row, col + 1))

    return val
```

Practice Problems:

1. Merge k Sorted Lists (LeetCode 23)
2. Kth Smallest Element in a Sorted Matrix (LeetCode 378)
3. Smallest Range Covering Elements from K Lists (LeetCode 632)
4. Find K Pairs with Smallest Sums (LeetCode 373)

5. Merge k Sorted Arrays (Custom)

---

15. 0/1 Knapsack

When to use: Optimization problems with binary choices (take or don't take).

Time Complexity: O(n × W) where W is knapsack capacity **Space Complexity:** O(n × W) or O(W) with optimization

Pattern Recognition:

- Subset sum problems
- Partition problems
- Target sum problems

Basic Template:

```python
def knapsack_01(weights, values, capacity):
    n = len(weights)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(capacity + 1):
            # Don't take item i-1
            dp[i][w] = dp[i-1][w]

            # Take item i-1 if possible
            if weights[i-1] <= w:
                dp[i][w] = max(dp[i][w],
                              dp[i-1][w - weights[i-1]] + values[i-1])

    return dp[n][capacity]
```

Space Optimized:

```python
def knapsack_01_optimized(weights, values, capacity):
    dp = [0] * (capacity + 1)

    for i in range(len(weights)):
        # Traverse backwards to avoid using updated values
        for w in range(capacity, weights[i] - 1, -1):
            dp[w] = max(dp[w], dp[w - weights[i]] + values[i])

    return dp[capacity]
```

Example: Subset Sum

```python
def can_partition(nums):
    total_sum = sum(nums)

    if total_sum % 2 != 0:
        return False

    target = total_sum // 2
    dp = [False] * (target + 1)
    dp[0] = True

    for num in nums:
        for j in range(target, num - 1, -1):
            dp[j] = dp[j] or dp[j - num]

    return dp[target]
```

Example: Target Sum

```python
def find_target_sum_ways(nums, S):
    total = sum(nums)

    if S > total or S < -total or (S + total) % 2 == 1:
        return 0

    target = (S + total) // 2
    dp = [0] * (target + 1)
    dp[0] = 1

    for num in nums:
        for j in range(target, num - 1, -1):
            dp[j] += dp[j - num]

    return dp[target]
```

Example: Ones and Zeroes

```python
def find_max_form(strs, m, n):
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for s in strs:
        zeros = s.count('0')
        ones = s.count('1')

        # Traverse backwards
        for i in range(m, zeros - 1, -1):
            for j in range(n, ones - 1, -1):
                dp[i][j] = max(dp[i][j], dp[i - zeros][j - ones] + 1)
```

```
    return dp[m][n]
```

Practice Problems:

1. Partition Equal Subset Sum (LeetCode 416)
2. Target Sum (LeetCode 494)
3. Ones and Zeroes (LeetCode 474)
4. Last Stone Weight II (LeetCode 1049)
5. Partition to K Equal Sum Subsets (LeetCode 698)

---

16. Unbounded Knapsack

When to use: Optimization problems where items can be used unlimited times.

Time Complexity: O(n × W) **Space Complexity:** O(W)

Pattern Recognition:

- Coin change problems
- Rod cutting problems
- Perfect squares

Template:

```python
def unbounded_knapsack(weights, values, capacity):
    dp = [0] * (capacity + 1)

    for i in range(1, capacity + 1):
        for j in range(len(weights)):
            if weights[j] <= i:
                dp[i] = max(dp[i], dp[i - weights[j]] + values[j])

    return dp[capacity]
```

Example: Coin Change

```python
def coin_change(coins, amount):
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0

    for coin in coins:
        for i in range(coin, amount + 1):
            dp[i] = min(dp[i], dp[i - coin] + 1)

    return dp[amount] if dp[amount] != float('inf') else -1
```

Example: Coin Change II (Number of Ways)

```
def change(amount, coins):
    dp = [0] * (amount + 1)
    dp[0] = 1

    for coin in coins:
        for i in range(coin, amount + 1):
            dp[i] += dp[i - coin]

    return dp[amount]
```

Example: Perfect Squares

```
def num_squares(n):
    dp = [float('inf')] * (n + 1)
    dp[0] = 0

    for i in range(1, n + 1):
        j = 1
        while j * j <= i:
            dp[i] = min(dp[i], dp[i - j * j] + 1)
            j += 1

    return dp[n]
```

Example: Combination Sum IV

```
def combination_sum4(nums, target):
    dp = [0] * (target + 1)
    dp[0] = 1

    for i in range(1, target + 1):
        for num in nums:
            if num <= i:
                dp[i] += dp[i - num]

    return dp[target]
```

Practice Problems:

1. Coin Change (LeetCode 322)
2. Coin Change 2 (LeetCode 518)
3. Perfect Squares (LeetCode 279)
4. Combination Sum IV (LeetCode 377)
5. Minimum Cost For Tickets (LeetCode 983)

---

17. Fibonacci Numbers

When to use: Problems with recurrence relations, optimization with overlapping subproblems. **Time Complexity:** O(n) **Space Complexity:** O(1) with optimization

Pattern Recognition:

- Climbing stairs problems
- House robber problems
- Decode ways
- Min cost climbing stairs

Template:

```python
def fibonacci(n):
    if n <= 1:
        return n

    prev2, prev1 = 0, 1

    for i in range(2, n + 1):
        current = prev1 + prev2
        prev2, prev1 = prev1, current

    return prev1
```

Example: Climbing Stairs

```python
def climb_stairs(n):
    if n <= 2:
        return n

    prev2, prev1 = 1, 2

    for i in range(3, n + 1):
        current = prev1 + prev2
        prev2, prev1 = prev1, current

    return prev1
```

Example: House Robber

```python
def rob(nums):
    if not nums:
        return 0
    if len(nums) == 1:
```

```
        return nums[0]

    prev2, prev1 = nums[0], max(nums[0], nums[1])

    for i in range(2, len(nums)):
        current = max(prev1, prev2 + nums[i])
        prev2, prev1 = prev1, current

    return prev1
```

Example: House Robber II (Circular)

```python
def rob_circular(nums):
    if len(nums) == 1:
        return nums[0]

    def rob_linear(houses):
        prev2, prev1 = 0, 0
        for num in houses:
            temp = max(prev1, prev2 + num)
            prev2, prev1 = prev1, temp
        return prev1

    return max(rob_linear(nums[:-1]), rob_linear(nums[1:]))
```

Example: Decode Ways

```python
def num_decodings(s):
    if not s or s[0] == '0':
        return 0

    prev2, prev1 = 1, 1

    for i in range(1, len(s)):
        current = 0
        if s[i] != '0':
            current += prev1
        if 10 <= int(s[i-1:i+1]) <= 26:
            current += prev2
        prev2, prev1 = prev1, current

    return prev1
```

Practice Problems:

1. Climbing Stairs (LeetCode 70)
2. House Robber (LeetCode 198)

3. House Robber II (LeetCode 213)
4. Decode Ways (LeetCode 91)
5. Min Cost Climbing Stairs (LeetCode 746)

18. Palindromic Subsequence

When to use: Problems involving palindromes and subsequences. **Time Complexity:** $O(n^2)$ **Space Complexity:** $O(n^2)$ or $O(n)$ optimized

Pattern Recognition:

- Longest palindromic subsequence
- Minimum deletions to make palindrome
- Count palindromic subsequences

Template:

```
def longest_palindromic_subsequence(s):
    n = len(s)
    dp = [[0] * n for _ in range(n)]

    # Every single character is a palindrome
    for i in range(n):
        dp[i][i] = 1

    # Fill for substrings of length 2 to n
    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j]:
                dp[i][j] = dp[i+1][j-1] + 2
            else:
                dp[i][j] = max(dp[i+1][j], dp[i][j-1])

    return dp[0][n-1]
```

Example: Longest Palindromic Subsequence

```
def longest_palindromic_subsequence(s):
    n = len(s)
    dp = [[0] * n for _ in range(n)]

    for i in range(n):
        dp[i][i] = 1

    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j]:
```

```
                dp[i][j] = dp[i+1][j-1] + 2
            else:
                dp[i][j] = max(dp[i+1][j], dp[i][j-1])

    return dp[0][n-1]
```

Example: Minimum Deletions to Make Palindrome

```
def min_deletions_palindrome(s):
    n = len(s)
    lps = longest_palindromic_subsequence(s)
    return n - lps
```

Example: Count Palindromic Subsequences

```
def count_palindromic_subsequences(s):
    n = len(s)
    dp = [[0] * n for _ in range(n)]

    for i in range(n):
        dp[i][i] = 1

    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j]:
                dp[i][j] = dp[i+1][j] + dp[i][j-1] + 1
            else:
                dp[i][j] = dp[i+1][j] + dp[i][j-1] - dp[i+1][j-1]

    return dp[0][n-1]
```

Practice Problems:

1. Longest Palindromic Subsequence (LeetCode 516)
2. Palindromic Substrings (LeetCode 647)
3. Minimum Insertion Steps to Make String Palindrome (LeetCode 1312)
4. Count Different Palindromic Subsequences (LeetCode 730)

19. Longest Common Substring

When to use: String comparison problems, finding similarities. **Time Complexity:** O(m × n) **Space Complexity:** O(m × n) or O(n) optimized

Pattern Recognition:

- Longest common subsequence

- Edit distance problems
- String similarity

Template:

```
def longest_common_subsequence(text1, text2):
    m, n = len(text1), len(text2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if text1[i-1] == text2[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])

    return dp[m][n]
```

Example: Longest Common Subsequence

```
def longest_common_subsequence(text1, text2):
    m, n = len(text1), len(text2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if text1[i-1] == text2[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])

    return dp[m][n]
```

Example: Edit Distance

```
def min_distance(word1, word2):
    m, n = len(word1), len(word2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Initialize base cases
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    for i in range(1, m + 1):
        for j in range(1, n + 1):
```

```
            if word1[i-1] == word2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])

    return dp[m][n]
```

Example: Longest Common Substring

```
def longest_common_substring(str1, str2):
    m, n = len(str1), len(str2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    max_length = 0

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if str1[i-1] == str2[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
                max_length = max(max_length, dp[i][j])
            else:
                dp[i][j] = 0

    return max_length
```

Practice Problems:

1. Longest Common Subsequence (LeetCode 1143)
2. Edit Distance (LeetCode 72)
3. Delete Operation for Two Strings (LeetCode 583)
4. Minimum ASCII Delete Sum (LeetCode 712)
5. Distinct Subsequences (LeetCode 115)

20. Topological Sort

When to use: Dependency resolution, ordering problems with directed acyclic graphs. **Time Complexity:** O(V + E) **Space Complexity:** O(V)

Pattern Recognition:

- Course scheduling
- Task ordering
- Build dependencies

Template (Kahn's Algorithm):

```
def topological_sort(graph):
    in_degree = {node: 0 for node in graph}
    for node in graph:
```

```python
        for neighbor in graph[node]:
            in_degree[neighbor] += 1

    queue = deque([node for node in in_degree if in_degree[node] == 0])
    result = []

    while queue:
        node = queue.popleft()
        result.append(node)

        for neighbor in graph[node]:
            in_degree[neighbor] -= 1
            if in_degree[neighbor] == 0:
                queue.append(neighbor)

    return result if len(result) == len(graph) else []
```

Example: Course Schedule

```python
def can_finish(num_courses, prerequisites):
    graph = [[] for _ in range(num_courses)]
    in_degree = [0] * num_courses

    for course, prereq in prerequisites:
        graph[prereq].append(course)
        in_degree[course] += 1

    queue = deque([i for i in range(num_courses) if in_degree[i] == 0])
    completed = 0

    while queue:
        course = queue.popleft()
        completed += 1

        for next_course in graph[course]:
            in_degree[next_course] -= 1
            if in_degree[next_course] == 0:
                queue.append(next_course)

    return completed == num_courses
```

Example: Course Schedule II

```python
def find_order(num_courses, prerequisites):
    graph = [[] for _ in range(num_courses)]
    in_degree = [0] * num_courses

    for course, prereq in prerequisites:
        graph[prereq].append(course)
```

```python
            in_degree[course] += 1

    queue = deque([i for i in range(num_courses) if in_degree[i] == 0])
    result = []

    while queue:
        course = queue.popleft()
        result.append(course)

        for next_course in graph[course]:
            in_degree[next_course] -= 1
            if in_degree[next_course] == 0:
                queue.append(next_course)

    return result if len(result) == num_courses else []
```

Example: Alien Dictionary

```python
def alien_order(words):
    graph = {}
    in_degree = {}

    # Initialize graph
    for word in words:
        for char in word:
            if char not in graph:
                graph[char] = []
                in_degree[char] = 0

    # Build graph
    for i in range(len(words) - 1):
        word1, word2 = words[i], words[i + 1]
        min_len = min(len(word1), len(word2))

        if len(word1) > len(word2) and word1[:min_len] == word2[:min_len]:
            return ""

        for j in range(min_len):
            if word1[j] != word2[j]:
                graph[word1[j]].append(word2[j])
                in_degree[word2[j]] += 1
                break

    # Topological sort
    queue = deque([char for char in in_degree if in_degree[char] == 0])
    result = []

    while queue:
        char = queue.popleft()
        result.append(char)
```

```
        for neighbor in graph[char]:
            in_degree[neighbor] -= 1
            if in_degree[neighbor] == 0:
                queue.append(neighbor)

    return ''.join(result) if len(result) == len(in_degree) else ""
```

Practice Problems:

1. Course Schedule (LeetCode 207)
2. Course Schedule II (LeetCode 210)
3. Alien Dictionary (LeetCode 269)
4. Minimum Height Trees (LeetCode 310)
5. Sequence Reconstruction (LeetCode 444)

21. Trie

When to use: Prefix-based operations, word searches, autocomplete. **Time Complexity:** O(m) for insert/search **Space Complexity:** O(ALPHABET_SIZE × N × M)

Pattern Recognition:

- Word search problems
- Prefix matching
- Autocomplete systems

Template:

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end = True

    def search(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                return False
```

```
            node = node.children[char]
        return node.is_end

    def starts_with(self, prefix):
        node = self.root
        for char in prefix:
            if char not in node.children:
                return False
            node = node.children[char]
        return True
```

Example: Word Search II

```
def find_words(board, words):
    class TrieNode:
        def __init__(self):
            self.children = {}
            self.word = None

    root = TrieNode()

    # Build trie
    for word in words:
        node = root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.word = word

    def dfs(i, j, node):
        char = board[i][j]
        if char not in node.children:
            return

        node = node.children[char]
        if node.word:
            result.add(node.word)

        board[i][j] = '#'  # Mark as visited
        for di, dj in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
            ni, nj = i + di, j + dj
            if 0 <= ni < len(board) and 0 <= nj < len(board[0]) and board[ni][nj]
!= '#':
                dfs(ni, nj, node)
        board[i][j] = char  # Restore

    result = set()
    for i in range(len(board)):
        for j in range(len(board[0])):
            dfs(i, j, root)
```

```
    return list(result)
```

Example: Design Add and Search Words

```
class WordDictionary:
    def __init__(self):
        self.root = TrieNode()

    def add_word(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end = True

    def search(self, word):
        def dfs(node, i):
            if i == len(word):
                return node.is_end

            char = word[i]
            if char == '.':
                for child in node.children.values():
                    if dfs(child, i + 1):
                        return True
                return False
            else:
                if char not in node.children:
                    return False
                return dfs(node.children[char], i + 1)

        return dfs(self.root, 0)
```

Practice Problems:

1. Implement Trie (LeetCode 208)
2. Word Search II (LeetCode 212)
3. Design Add and Search Words Data Structure (LeetCode 211)
4. Replace Words (LeetCode 648)
5. Map Sum Pairs (LeetCode 677)

22. Union Find

When to use: Connectivity problems, grouping elements, cycle detection. **Time Complexity:** O(α(n)) amortized **Space Complexity:** O(n)

Pattern Recognition:

- Connected components
- Cycle detection in undirected graphs
- Minimum spanning tree

Template:

```
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n
        self.components = n

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])  # Path compression
        return self.parent[x]

    def union(self, x, y):
        px, py = self.find(x), self.find(y)
        if px == py:
            return False

        # Union by rank
        if self.rank[px] < self.rank[py]:
            px, py = py, px

        self.parent[py] = px
        if self.rank[px] == self.rank[py]:
            self.rank[px] += 1

        self.components -= 1
        return True

    def connected(self, x, y):
        return self.find(x) == self.find(y)
```

Example: Number of Connected Components

```
def count_components(n, edges):
    uf = UnionFind(n)
    for a, b in edges:
        uf.union(a, b)
    return uf.components
```

Example: Redundant Connection

```python
def find_redundant_connection(edges):
    uf = UnionFind(len(edges) + 1)
    for a, b in edges:
        if not uf.union(a, b):
            return [a, b]
    return []
```

Example: Accounts Merge

```python
def accounts_merge(accounts):
    uf = UnionFind(len(accounts))
    email_to_id = {}

    for i, account in enumerate(accounts):
        for email in account[1:]:
            if email in email_to_id:
                uf.union(i, email_to_id[email])
            else:
                email_to_id[email] = i

    groups = defaultdict(list)
    for email, id in email_to_id.items():
        groups[uf.find(id)].append(email)

    result = []
    for id, emails in groups.items():
        name = accounts[id][0]
        result.append([name] + sorted(emails))

    return result
```

Practice Problems:

1. Number of Connected Components in an Undirected Graph (LeetCode 323)
2. Redundant Connection (LeetCode 684)
3. Accounts Merge (LeetCode 721)
4. Most Stones Removed with Same Row or Column (LeetCode 947)
5. Satisfiability of Equality Equations (LeetCode 990)

23. Monotonic Stack

When to use: Finding next/previous greater/smaller elements. **Time Complexity:** O(n) **Space Complexity:** O(n)

Pattern Recognition:

- Next greater element
- Largest rectangle problems

- Temperature problems

Template:

```python
def next_greater_element(nums):
    stack = []
    result = [-1] * len(nums)

    for i in range(len(nums)):
        while stack and nums[stack[-1]] < nums[i]:
            result[stack.pop()] = nums[i]
        stack.append(i)

    return result
```

Example: Daily Temperatures

```python
def daily_temperatures(temperatures):
    stack = []
    result = [0] * len(temperatures)

    for i, temp in enumerate(temperatures):
        while stack and temperatures[stack[-1]] < temp:
            prev_day = stack.pop()
            result[prev_day] = i - prev_day
        stack.append(i)

    return result
```

Example: Largest Rectangle in Histogram

```python
def largest_rectangle_area(heights):
    stack = []
    max_area = 0

    for i, h in enumerate(heights):
        while stack and heights[stack[-1]] > h:
            height = heights[stack.pop()]
            width = i if not stack else i - stack[-1] - 1
            max_area = max(max_area, height * width)
        stack.append(i)

    while stack:
        height = heights[stack.pop()]
        width = len(heights) if not stack else len(heights) - stack[-1] - 1
        max_area = max(max_area, height * width)

    return max_area
```

Example: Next Greater Element II

```python
def next_greater_elements(nums):
    n = len(nums)
    result = [-1] * n
    stack = []

    for i in range(2 * n):
        while stack and nums[stack[-1]] < nums[i % n]:
            result[stack.pop()] = nums[i % n]
        if i < n:
            stack.append(i)

    return result
```

Practice Problems:

1. Daily Temperatures (LeetCode 739)
2. Next Greater Element I (LeetCode 496)
3. Next Greater Element II (LeetCode 503)
4. Largest Rectangle in Histogram (LeetCode 84)
5. Trapping Rain Water (LeetCode 42)

24. Backtracking

When to use: Exploring all possible solutions, constraint satisfaction. **Time Complexity:** O(b^d) where b is branching factor **Space Complexity:** O(d)

Pattern Recognition:

- Generating all combinations/permutations
- Sudoku solving
- N-Queens problem

Template:

```python
def backtrack(result, current, remaining):
    # Base case
    if is_valid_solution(current):
        result.append(current[:])  # Make a copy
        return

    # Try all possible choices
    for choice in get_choices(remaining):
        # Make choice
        current.append(choice)
```

```
        # Recurse
        backtrack(result, current, update_remaining(remaining, choice))

        # Backtrack
        current.pop()
```

Example: Generate Parentheses

```
def generate_parenthesis(n):
    result = []

    def backtrack(current, open_count, close_count):
        if len(current) == 2 * n:
            result.append(current)
            return

        if open_count < n:
            backtrack(current + '(', open_count + 1, close_count)

        if close_count < open_count:
            backtrack(current + ')', open_count, close_count + 1)

    backtrack('', 0, 0)
    return result
```

Example: Combination Sum

```
def combination_sum(candidates, target):
    result = []

    def backtrack(start, current, remaining):
        if remaining == 0:
            result.append(current[:])
            return

        for i in range(start, len(candidates)):
            if candidates[i] <= remaining:
                current.append(candidates[i])
                backtrack(i, current, remaining - candidates[i])
                current.pop()

    backtrack(0, [], target)
    return result
```

Example: N-Queens

```python
def solve_n_queens(n):
    result = []
    board = [['.' for _ in range(n)] for _ in range(n)]

    def is_safe(row, col):
        # Check column
        for i in range(row):
            if board[i][col] == 'Q':
                return False

        # Check diagonals
        for i, j in zip(range(row-1, -1, -1), range(col-1, -1, -1)):
            if board[i][j] == 'Q':
                return False

        for i, j in zip(range(row-1, -1, -1), range(col+1, n)):
            if board[i][j] == 'Q':
                return False

        return True

    def backtrack(row):
        if row == n:
            result.append([''.join(row) for row in board])
            return

        for col in range(n):
            if is_safe(row, col):
                board[row][col] = 'Q'
                backtrack(row + 1)
                board[row][col] = '.'

    backtrack(0)
    return result
```

Practice Problems:

1. Generate Parentheses (LeetCode 22)
2. Combination Sum (LeetCode 39)
3. Permutations (LeetCode 46)
4. N-Queens (LeetCode 51)
5. Word Search (LeetCode 79)

25. Best Practices & Next Steps

Problem-Solving Strategy:

1. **Understand the Problem**: Read carefully, identify constraints
2. **Pattern Recognition**: Match problem to known patterns
3. **Start Simple**: Begin with brute force, then optimize

4. **Test Edge Cases**: Empty inputs, single elements, duplicates
5. **Optimize**: Consider time/space trade-offs

Time Complexity Quick Reference:

- **O(1)**: Hash table operations, array access
- **O(log n)**: Binary search, heap operations
- **O(n)**: Single pass through array
- **O(n log n)**: Efficient sorting, divide and conquer
- **O(n^2)**: Nested loops, some DP problems
- **O(2^n)**: Exponential algorithms, some backtracking

Space Complexity Optimization:

- Use in-place algorithms when possible
- Consider iterative vs recursive approaches
- Optimize DP space usage (rolling arrays)
- Use bit manipulation for compact storage

Common Pitfalls:

- **Off-by-one errors**: Carefully check array bounds
- **Integer overflow**: Consider using long for large numbers
- **Edge cases**: Empty inputs, single elements
- **Modular arithmetic**: Remember to take mod when required

Next Steps:

1. **Practice Regularly**: Solve 2-3 problems daily
2. **Time Yourself**: Simulate interview conditions
3. **Review Solutions**: Study multiple approaches
4. **Mock Interviews**: Practice explaining your thought process
5. **System Design**: Learn large-scale system concepts

Advanced Topics to Explore:

- **Segment Trees**: Range queries and updates
- **Fenwick Trees**: Efficient prefix sum operations
- **Heavy-Light Decomposition**: Tree path queries
- **Suffix Arrays/Trees**: Advanced string algorithms
- **Flow Networks**: Max flow problems
- **Linear Programming**: Optimization problems

Resources for Continued Learning:

- **Books**: "Cracking the Coding Interview", "Algorithm Design Manual"
- **Online Platforms**: LeetCode, HackerRank, CodeForces
- **Courses**: Algorithms Specialization (Coursera), CS algorithms courses
- **Communities**: Reddit r/leetcode, Discord study groups

Remember: **Consistency beats intensity**. Regular practice with these patterns will build your algorithmic thinking and problem-solving skills over time.