# DSA Patterns Master Guide (18 Patterns)

*Simple. Clear. Developer-Friendly. 5+ LeetCode Qs per pattern.*

---

## 1. Sliding Window

**Idea**: Slide a window over array/string to track valid subarray in O(n).

**Example**: Max sum of k-size subarray → expand right, shrink left when size > k. Like scanning a log file for errors in a time window.

**Pseudocode**:

text Collapse Wrap Copy

```
left = 0, max_sum = 0, curr = 0
for right = 0 to n-1:
    curr += arr[right]
    if right - left + 1 > k:
        curr -= arr[left]
        left += 1
    max_sum = max(max_sum, curr)
```

**LeetCode**:

1. Longest Substring Without Repeating
2. Minimum Window Substring
3. Longest Repeating Replacement
4. Max Average Subarray I
5. Fruit Into Baskets
6. Permutation in String

---

## 2. Two Pointers

**Idea**: Two pointers on sorted array for pairs/sums in O(n).

**Example**: Find pair summing to target → left at start, right at end; move based on sum too big/small. Like finding matching brackets.

**Pseudocode**:

text Collapse Wrap Copy

```
sort(arr)
left = 0, right = n-1
while left < right:
    sum = arr[left] + arr[right]
    if sum == target: return [left, right]
    if sum > target: right--
    else: left++
```

**LeetCode**:

1. Two Sum II
2. 3Sum
3. Container With Most Water
4. Trapping Rain Water
5. Remove Duplicates Sorted
6. Sort Colors

---

## 3. Fast & Slow Pointers

**Idea**: Slow (1 step), fast (2 steps) for cycles/middle in O(n).

**Example**: Detect cycle in linked list → if they meet, loop exists. Like checking a loop in a conveyor belt.

**Pseudocode**:

```
text
```

```
slow = fast = head
while fast and fast.next:
    slow = slow.next
    fast = fast.next.next
    if slow == fast: return True
return False
```

**LeetCode**:

1. Linked List Cycle
2. Find Duplicate Number
3. Happy Number
4. Linked List Cycle II
5. Middle of Linked List
6. Palindrome Linked List

---

## 4. Merge Intervals

**Idea**: Sort by start, merge if overlap in O(n log n).

**Example**: Merge [[1,3],[2,6]] → [1,6]. Like combining overlapping meeting times.

**Pseudocode**:

```
text
```

```
sort(intervals, key=start)
merged = [intervals[0]]
for interval in intervals[1:]:
    if interval.start <= merged[-1].end:
        merged[-1].end = max(merged[-1].end, interval.end)
    else:
        merged.append(interval)
```

**LeetCode**:

1. Merge Intervals
2. Insert Interval
3. Meeting Rooms II
4. Non-overlapping Intervals
5. Interval List Intersections
6. Employee Free Time

---

## 5. Cyclic Sort

**Idea**: For 1..n, swap arr[i] to arr[arr[i]-1] in O(n).

**Example**: [3,1,2] → sorted [1,2,3], spot missing/duplicates. Like sorting misplaced books by page number.

**Pseudocode**:

text Collapse Wrap Copy

```
for i in 0..n-1:
    while arr[i] != i+1 and arr[i] != -1:
        swap arr[i], arr[arr[i]-1]
```

**LeetCode**:

1. Find Duplicate
2. Missing Number
3. Find All Duplicates
4. Numbers Disappeared
5. First Missing Positive
6. Duplicate Zeros

---

## 6. In-place Linked List Reversal

**Idea**: Reverse pointers iteratively, no extra space.

**Example**: 1→2→3 → 3→2→1. Like flipping a chain link by link.

**Pseudocode**:

text Collapse Wrap Copy

```
prev = None, curr = head
while curr:
    next = curr.next
    curr.next = prev
    prev = curr
    curr = next
return prev
```

**LeetCode**:

1. Reverse Linked List
2. Reverse Nodes k-Group
3. Palindrome Linked List
4. Reorder List
5. Remove Nth from End
6. Swap Nodes in Pairs

---

## 7. Tree BFS (Level Order)

**Idea**: Queue for level-by-level traversal.

**Example**: Tree levels → [root], [left,right], etc. Like processing floors in a building queue.

**Pseudocode**:

text Collapse Wrap Copy

```
from collections import deque
q = deque([root])
while q:
    level = []
    for _ in range(len(q)):
        node = q.popleft()
        level.append(node.val)
        if node.left: q.append(node.left)
        if node.right: q.append(node.right)
    result.append(level)
```

**LeetCode**:

1. Level Order Traversal
2. Zigzag Level Order
3. Min Depth
4. Right Side View
5. Next Right Pointers
6. Complete Binary Tree Inserter

---

## 8. Tree DFS (Recursion)

**Idea**: Recur pre/in/post for paths/sums.

**Example**: Max depth → 1 + max(left, right). Like exploring a maze branch by branch.

**Pseudocode**:

```
text
```

```
def dfs(node):
    if not node: return 0
    left = dfs(node.left)
    right = dfs(node.right)
    return 1 + max(left, right)
```

**LeetCode**:

1. Max Depth
2. Binary Tree Paths
3. Path Sum
4. LCA Binary Tree
5. Validate BST
6. Kth Smallest BST

---

## 9. Subsets / Backtracking

**Idea**: Recur include/exclude for combinations.

**Example**: Subsets [1,2] → [], [1], [2], [1,2]. Like choosing toppings for pizza options.

**Pseudocode**:

```
text
```

```
def backtrack(start, path):
    if start == n: result.append(path[:])
    else:
        backtrack(start+1, path)  # exclude
        path.append(arr[start])
        backtrack(start+1, path)  # include
        path.pop()
```

**LeetCode**:

1. Subsets
2. Subsets II
3. Permutations
4. Combination Sum
5. Letter Combinations Phone
6. Word Search

---

## 10. Modified Binary Search

**Idea**: Adapt for rotated/peaks in O(log n).

**Example**: Search in rotated array [4,5,1,2,3] for 1 → check sorted halves. Like binary search on a twisted number line.

**Pseudocode**:

text Collapse Wrap Copy

```
left, right = 0, n-1
while left <= right:
    mid = (left + right) // 2
    if arr[mid] == target: return mid
    if arr[left] <= arr[mid]:  # left sorted
        if target > arr[mid]: left = mid + 1
        else: right = mid - 1
    else:  # right sorted
        if target < arr[mid]: right = mid - 1
        else: left = mid + 1
return -1
```

**LeetCode**:

1. Search Rotated Array
2. Min in Rotated Array
3. Search 2D Matrix
4. Peak Index Mountain
5. First Bad Version
6. Guess Number Higher Lower

---

## 11. Top 'K' Elements

**Idea**: Heap or quickselect for k largest/smallest in O(n log k).

**Example**: Kth largest → min-heap of size k, pop smallest. Like priority queue for top tasks.

**Pseudocode**:

text Collapse Wrap Copy

```
import heapq
heap = []
for num in arr:
    heapq.heappush(heap, num)
    if len(heap) > k: heapq.heappop(heap)
return heap[0]  # kth largest
```

**LeetCode**:

1. Kth Largest Element
2. Top K Frequent
3. K Closest Points
4. Find K Pairs Smallest Sums
5. Merge K Sorted Lists
6. Task Scheduler

---

## 12. K-way Merge

**Idea**: Min-heap merge k sorted lists in O(n log k).

**Example**: Merge k sorted arrays → heap tracks smallest heads. Like merging multiple log streams.

**Pseudocode**:

text Collapse Wrap Copy

```
heap = []  # (val, list_idx, elem_idx)
for i, lst in enumerate(lists):
    if lst: heapq.heappush(heap, (lst[0], i, 0))
result = []
while heap:
    val, i, j = heapq.heappop(heap)
    result.append(val)
    if j+1 < len(lists[i]):
        next_val = lists[i][j+1]
        heapq.heappush(heap, (next_val, i, j+1))
```

**LeetCode**:

1. Merge K Sorted Lists
2. Smallest Range Covering Elements
3. Find K Pairs Smallest Sums
4. Ugly Number II
5. Super Ugly Number
6. Kth Smallest in Sorted Matrix

---

## 13. 0/1 Knapsack (DP)

**Idea**: DP[i][w] = max value using first i items, weight <=w.

**Example**: Items with weights/values, max value <=W. Like packing a backpack without reuse.

**Pseudocode**:

text Collapse Wrap Copy

```
dp = [[0]*(W+1) for _ in range(n+1)]
for i in 1..n:
    for w in 1..W:
        if weights[i-1] <= w:
            dp[i][w] = max(dp[i-1][w], dp[i-1][w-weights[i-1]] + values[i-1])
        else:
            dp[i][w] = dp[i-1][w]
```

**LeetCode**:

1. Partition Equal Subset Sum
2. Target Sum
3. Last Stone Weight II
4. Partition to K Equal Sum Subsets
5. Minimum Cost for Tickets
6. Coin Change II

---

## 14. Unbounded Knapsack

**Idea**: Allow reuse, DP[i][w] similar but from previous.

**Example**: Coin change min coins → reuse coins. Like unlimited supply in shopping.

**Pseudocode**:

```
text
```



```
dp = [inf]*(amount+1)
dp[0] = 0
for coin in coins:
    for x in coin..amount:
        dp[x] = min(dp[x], dp[x-coin] + 1)
```

**LeetCode**:

1. Coin Change
2. Coin Change II
3. Perfect Squares
4. Minimum Cost to Hire K Workers # Variant
5. Non-negative Integers without Consecutive Ones # Variant
6. Decode Ways # Variant

---

## 15. Longest Common Substring/Subsequence (DP)

**Idea**: 2D DP[i][j] = LCS length.

**Example**: LCS "abcde", "ace" → "ace" len 3. Like finding common code snippets.

**Pseudocode**:

```
text
```



```
dp = [[0]*(m+1) for _ in range(n+1)]
for i in 1..n:
    for j in 1..m:
        if s1[i-1] == s2[j-1]:
            dp[i][j] = dp[i-1][j-1] + 1
        else:
            dp[i][j] = max(dp[i-1][j], dp[i][j-1])
```

**LeetCode**:

1. Longest Common Subsequence
2. Edit Distance
3. Minimum ASCII Delete Sum
4. Delete Operation for Two Strings
5. Longest Increasing Subsequence
6. Shortest Common Supersequence

---

## 16. Palindrome DP

**Idea**: Expand around centers or DP table for substrings.

**Example**: Longest palindrome in "babad" → "bab". Like checking symmetric code.

**Pseudocode** (Expand):

text Collapse Wrap Copy

```
def expand(s, left, right):
    while left >=0 and right < n and s[left] == s[right]:
        left -=1
        right +=1
    return right - left -1
max_len = 0
for i in 0..n-1:
    len1 = expand(s, i, i)
    len2 = expand(s, i, i+1)
    max_len = max(max_len, len1, len2)
```

**LeetCode**:

1. Longest Palindromic Substring
2. Palindrome Partitioning
3. Palindrome Pairs
4. Shortest Palindrome
5. Valid Palindrome II
6. Longest Palindrome

---

## 17. Monotonic Stack

**Idea**: Stack increasing/decreasing for next greater/smaller.

**Example**: Next greater element → pop smaller till larger. Like tracking stock peaks.

**Pseudocode**:

text Collapse Wrap Copy

```
stack = []
result = [-1]*n
for i in range(n-1, -1, -1):
    while stack and stack[-1] <= arr[i]:
        stack.pop()
    if stack: result[i] = stack[-1]
    stack.append(arr[i])
```

**LeetCode**:

1. Next Greater Element I
2. Daily Temperatures
3. Trapping Rain Water
4. Largest Rectangle in Histogram
5. Sum of Subarray Minimums
6. Online Stock Span

---

# 18. Topological Sort

**Idea**: Kahn's (indegrees) or DFS for DAG order.

**Example**: Course schedule → process zero prereqs first. Like build order in software deps.

**Pseudocode** (Kahn):

text Collapse Wrap Copy

```
from collections import deque, defaultdict
indegree = [0]*n
graph = defaultdict(list)
for u, v in edges:
    graph[u].append(v)
    indegree[v] +=1
q = deque([i for i in range(n) if indegree[i]==0])
order = []
while q:
    u = q.popleft()
    order.append(u)
    for v in graph[u]:
        indegree[v] -=1
        if indegree[v]==0: q.append(v)
return order if len(order)==n else []  # cycle
```

**LeetCode**:

1. Course Schedule
2. Course Schedule II
3. Alien Dictionary
4. Sequence Reconstruction
5. Minimum Height Trees
6. Reconstruct Itinerary

---

*Practice 1 pattern/day. Master DSA! To make PDF: Copy this into Markdown to PDF, export.*

Explain Sliding Window deeply

System Design Patterns

Add difficulty ratings