150 Python Coding Patterns

> Your compact playbook of 150 Python coding patterns.

A. Arrays & Two Pointers (1–10)

Catalog

1. Opposite Ends (Pair Sum on Sorted)
2. Move Zeros In-Place
3. Remove Duplicates from Sorted Array
4. Partition by Predicate (Dutch Flag Lite)
5. Dedup K Times
6. Square Sorted Array (Two Pointers Merge)
7. Triplet Sum Close to Target
8. Merge Two Sorted Arrays In-Place
9. Interval Overlap via Pointers
10. Rotate Array via Reversal

Templates

Template A1 – Two Pointers (Opposite Ends)

```
def two_pointers(nums, target):
    i, j = 0, len(nums) - 1
    while i < j:
        s = nums[i] + nums[j]
        if s == target:
            return (i, j)
        elif s < target:
            i += 1
```

```
        else:
            j -= 1
    return None
```

Template A2 – Fast/Slow Pointers

```
def remove_duplicates(nums):
    if not nums: return 0
    slow = 0
    for fast in range(1, len(nums)):
        if nums[fast] != nums[slow]:
            slow += 1
            nums[slow] = nums[fast]
    return slow + 1
```

Worked Examples

Example A1 – Move Zeros In-Place

```
def move_zeros(nums):
    slow = 0
    for fast in range(len(nums)):
        if nums[fast] != 0:
            nums[slow], nums[fast] = nums[fast], nums[slow]
            slow += 1
    return nums
```

- **Idea**: Partition array into non-zeros and zeros.
- **Complexity**: O(n), O(1).
- **Pitfall**: Forgetting in-place swap.

Example A2 – Rotate Array via Reversal

```
def rotate(nums, k):
    n = len(nums)
    k %= n
```

```
    def rev(i, j):
        while i < j:
            nums[i], nums[j] = nums[j], nums[i]
            i += 1; j -= 1

    rev(0, n-1)
    rev(0, k-1)
    rev(k, n-1)
    return nums
```

- **Idea**: Reverse whole → reverse first k → reverse rest.
- **Complexity**: O(n).

Notes

- **Use cases**: Array manipulation, sorted arrays, partitioning.
- **Pitfalls**: Watch indices in in-place ops.
- **Variations**: 3Sum, remove element by value, cyclic shifts.

B. Sliding Window (11–25)

Catalog

1. Fixed Window Max Sum
2. Fixed Window Average
3. Smallest Subarray ≥ Target
4. Longest Substring without Repeating
5. Longest Substring with K Distinct
6. Anagram Occurrences (Frequency Window)
7. Minimum Window Substring
8. Max Consecutive Ones with K Flips
9. Fruit Into Baskets (2 distinct)
10. Longest Repeating Character Replacement

11. Binary Subarray Sum
12. Distinct Count per Window
13. Substrings with At Most K Distinct
14. Subarray Product < K
15. Longest Vowel-Balanced Substring

Templates

Template B1 – Fixed Size Window

```
def max_sum_k(nums, k):
    cur = sum(nums[:k])
    best = cur
    for i in range(k, len(nums)):
        cur += nums[i] - nums[i-k]
        best = max(best, cur)
    return best
```

Template B2 – Variable Size Window

```
def smallest_subarray(nums, target):
    total = l = 0
    best = float('inf')
    for r in range(len(nums)):
        total += nums[r]
        while total >= target:
            best = min(best, r-l+1)
            total -= nums[l]
            l += 1
    return 0 if best == float('inf') else best
```

Worked Examples

Example B1 – Longest Substring without Repeating

```
def longest_unique_substring(s):
    last, l, best = {}, 0, 0
```

```
    for r, ch in enumerate(s):
        if ch in last and last[ch] >= l:
            l = last[ch] + 1
        last[ch] = r
        best = max(best, r-l+1)
    return best
```

Example B2 – Minimum Window Substring

```
from collections import Counter

def min_window(s, t):
    need = Counter(t)
    missing = len(t)
    l = best = (0, float('inf'))
    i = 0
    for j, ch in enumerate(s, 1):
        if need[ch] > 0:
            missing -= 1
        need[ch] -= 1
        while missing == 0:
            if j - i < best[1] - best[0]:
                best = (i, j)
            need[s[i]] += 1
            if need[s[i]] > 0:
                missing += 1
            i += 1
    return "" if best[1] == float('inf') else s[best[0]:best[1]]
```

Notes

- **Use cases**: Strings, subarrays, frequency constraints.
- **Pitfalls**: Shrinking only when condition is satisfied.
- **Variations**: Distinct substrings, sliding product, prefix sums inside window.

C. Prefix / Suffix & Hashing (26–40)

Catalog

1. Subarray Sum Equals K (prefix + hashmap)
2. Longest Subarray with Sum 0
3. Count Subarrays with Sum Divisible by M
4. Max Length Equal 0/1 (binary → ±1)
5. Longest Balanced Parentheses (stack+prefix)
6. Running Sum / Difference Arrays
7. Range Add via Difference Array
8. Equilibrium Index
9. Split Array Equal Sum
10. Longest Arithmetic Subarray
11. Count Nice Pairs (hash of property)
12. Contiguous Array of Evens/Odds
13. First Unique Character (freq)
14. Group Anagrams (signature)
15. Isomorphic Strings Map

Templates

Template C1 – Prefix Sum with Hash Count

```
from collections import defaultdict

def count_subarrays_sum_k(nums, k):
    pref = 0
    seen = defaultdict(int)
    seen[0] = 1
    ans = 0
    for x in nums:
        pref += x
        ans += seen[pref - k]
```

```
        seen[pref] += 1
    return ans
```

Template C2 – Signature Hashing (e.g., anagrams)

```
from collections import defaultdict

def group_by_signature(words):
    groups = defaultdict(list)
    for w in words:
        sig = tuple(sorted(w))  # or 26-count tuple for speed
        groups[sig].append(w)
    return list(groups.values())
```

Worked Examples

Example C1 – Longest Subarray with Sum 0

```
def longest_zero_sum(nums):
    first_idx = {0: -1}
    pref = ans = 0
    for i, x in enumerate(nums):
        pref += x
        if pref in first_idx:
            ans = max(ans, i - first_idx[pref])
        else:
            first_idx[pref] = i
    return ans
```

Example C2 – Contiguous Array (max length equal 0/1)

```
def max_len_equal_zeros_ones(nums):
    pref = 0
    first_idx = {0: -1}
    best = 0
    for i, x in enumerate(nums):
        pref += 1 if x == 1 else -1
        if pref in first_idx:
```

```
            best = max(best, i - first_idx[pref])
        else:
            first_idx[pref] = i
    return best
```

Notes

- Prefix-hash patterns convert range problems to **point lookups**.
- Use **difference arrays** (Template C1 variant) for range updates then prefix once.
- For string hashing, prefer **count-signature** over sorting for performance.

D. Binary Search Patterns (41–55)

Catalog

1. Classic Binary Search (existence)
2. Lower Bound / Upper Bound
3. Binary Search on Answer (min feasible)
4. Peak in Mountain Array
5. Search Rotated Sorted Array
6. Find Rotation Count
7. Kth in Two Sorted Arrays
8. Minimize Maximum Load
9. Allocate Pages / Ship Packages
10. Aggressive Cows (maximize min distance)
11. Sqrt / Nth Root via BS
12. Median of Row-Sorted Matrix
13. First Bad Version (predicate BS)
14. Ceiling/Floor in Sorted
15. Count Occurrences in Sorted

Templates

Template D1 – Lower/Upper Bound

```
def lower_bound(a, x):
    lo, hi = 0, len(a)
    while lo < hi:
        mid = (lo + hi) // 2
        if a[mid] < x:
            lo = mid + 1
        else:
            hi = mid
    return lo

def upper_bound(a, x):
    lo, hi = 0, len(a)
    while lo < hi:
        mid = (lo + hi) // 2
        if a[mid] <= x:
            lo = mid + 1
        else:
            hi = mid
    return lo
```

Template D2 – Binary Search on Answer

```
def minimize_max(loads, days):
    lo, hi = max(loads), sum(loads)
    def feasible(cap):
        d, cur = 1, 0
        for x in loads:
            if cur + x > cap:
                d += 1
                cur = 0
            cur += x
        return d <= days
    while lo < hi:
        mid = (lo + hi) // 2
        if feasible(mid):
```

```
            hi = mid
        else:
            lo = mid + 1
    return lo
```

Worked Examples

Example D1 – Search Rotated Sorted Array

```
def search_rotated(nums, target):
    lo, hi = 0, len(nums)-1
    while lo <= hi:
        mid = (lo + hi) // 2
        if nums[mid] == target:
            return mid
        if nums[lo] <= nums[mid]:  # left sorted
            if nums[lo] <= target < nums[mid]:
                hi = mid - 1
            else:
                lo = mid + 1
        else:  # right sorted
            if nums[mid] < target <= nums[hi]:
                lo = mid + 1
            else:
                hi = mid - 1
    return -1
```

Example D2 – Peak in Mountain Array

```
def peak_index_mountain(arr):
    lo, hi = 0, len(arr)-1
    while lo < hi:
        mid = (lo + hi) // 2
        if arr[mid] < arr[mid+1]:
            lo = mid + 1
        else:
            hi = mid
    return lo
```

Notes

- Separate **index-space search** vs **answer-space search**.
- For counts/ranges on sorted arrays, use lower/upper bound.

E. Stack & Monotonic Structures (56–70)

Catalog

1. Valid Parentheses
2. Min Stack
3. Next Greater Element
4. Next Greater Circular
5. Next Smaller Element
6. Largest Rectangle in Histogram
7. Max Rectangle in Binary Matrix
8. Daily Temperatures
9. Asteroid Collision
10. Remove K Digits
11. Simplify Path
12. Decode String k[pattern]
13. Basic Calculator (stack ops)
14. RPN Evaluation
15. Exclusive Time of Functions

Templates

Template E1 – Monotonic Decreasing (Next Greater Element)

```python
def next_greater(nums):
    st, ans = [], [-1]*len(nums)
    for i, v in enumerate(nums):
        while st and nums[st[-1]] < v:
            ans[st.pop()] = v
```

```
        st.append(i)
    return ans
```

Template E2 – Valid Parentheses / Stack

```
def valid_parentheses(s):
    mp = {')':'(', ']':'[', '}':'{'}
    st = []
    for ch in s:
        if ch in '([{':
            st.append(ch)
        elif ch in ')]}':
            if not st or st.pop() != mp[ch]:
                return False
    return not st
```

Worked Examples

Example E1 – Largest Rectangle in Histogram

```
def largest_rectangle_area(h):
    st, best = [], 0
    h.append(0)
    for i, x in enumerate(h):
        start = i
        while st and st[-1][1] > x:
            idx, height = st.pop()
            best = max(best, height * (i - idx))
            start = idx
        st.append((start, x))
    h.pop()
    return best
```

Example E2 – Decode String

```
def decode_string(s):
    count_st, str_st = [], []
    cur = ""
```

```
    k = 0
    for ch in s:
        if ch.isdigit():
            k = k*10 + int(ch)
        elif ch == '[':
            count_st.append(k)
            str_st.append(cur)
            k, cur = 0, ""
        elif ch == ']':
            cur = str_st.pop() + count_st.pop()*cur
        else:
            cur += ch
    return cur
```

Notes

- Monotonic stacks capture **next greater/smaller** in O(n).
- For calculators/decoders, **stack frames** mirror nested structure.

F. Heaps & Greedy (71–85)

Catalog

1. Kth Largest / Smallest
2. Top-K Frequent Elements
3. Sort Characters by Frequency
4. Reorganize String
5. Merge K Sorted Lists
6. Meeting Rooms II
7. Task Scheduler
8. Huffman Coding Skeleton
9. Connect Ropes (min cost)
10. Fractional Knapsack
11. Activity Selection

12. Gas Station Circuit
13. Jump Game
14. Candy Distribution
15. IPO / Maximize Capital

Templates

Template F1 – Top-K with Heap

```
import heapq
from collections import Counter

def top_k_frequent(nums, k):
    cnt = Counter(nums)
    heap = [(-c, x) for x, c in cnt.items()]
    heapq.heapify(heap)
    return [heapq.heappop(heap)[1] for _ in range(min(k, len(heap)))]
```

Template F2 – Greedy Scheduling (Sort + Sweep)

```
def min_meeting_rooms(intervals):
    if not intervals: return 0
    starts = sorted(s for s, _ in intervals)
    ends   = sorted(e for _, e in intervals)
    i = j = rooms = best = 0
    while i < len(starts):
        if starts[i] < ends[j]:
            rooms += 1; best = max(best, rooms); i += 1
        else:
            rooms -= 1; j += 1
    return best
```

Worked Examples

Example F1 – Connect Ropes (min cost)

```
import heapq
```

```python
def min_cost_connect_ropes(ropes):
    heapq.heapify(ropes)
    cost = 0
    while len(ropes) > 1:
        a = heapq.heappop(ropes)
        b = heapq.heappop(ropes)
        cost += a + b
        heapq.heappush(ropes, a+b)
    return cost
```

Example F2 – Jump Game (greedy reach)

```python
def can_reach_end(nums):
    reach = 0
    for i, x in enumerate(nums):
        if i > reach: return False
        reach = max(reach, i + x)
    return True
```

Notes

- Heaps handle **top-k** and **merge k lists** efficiently.
- Many “choose best now” problems are **sort or heap** + simple invariants.

G. Intervals (86–95)

Catalog

1. Merge Intervals
2. Insert Interval
3. Non-overlapping Intervals (erase min)
4. Interval Intersection
5. Employee Free Time
6. Min Arrows to Burst Balloons
7. Car Pooling (line sweep)

8. My Calendar I/II (conflict check)
9. Summary Ranges
10. Minimum Platforms

Templates

Template G1 – Merge Intervals

```
def merge_intervals(intervals):
    intervals.sort()
    out = []
    for s, e in intervals:
        if not out or s > out[-1][1]:
            out.append([s, e])
        else:
            out[-1][1] = max(out[-1][1], e)
    return out
```

Template G2 – Line Sweep with Balance

```
def max_overlaps(intervals):
    events = []
    for s, e in intervals:
        events.append((s, +1))
        events.append((e, -1))
    events.sort()
    cur = best = 0
    for _, d in events:
        cur += d
        best = max(best, cur)
    return best
```

Worked Examples

Example G1 – Insert Interval

```
def insert_interval(intervals, newI):
    res = []
```

```
    s, e = newI
    placed = False
    for a, b in intervals:
        if b < s:
            res.append([a, b])
        elif e < a:
            if not placed:
                res.append([s, e]); placed = True
            res.append([a, b])
        else:
            s = min(s, a)
            e = max(e, b)
    if not placed:
        res.append([s, e])
    return res
```

Example G2 – Minimum Arrows to Burst Balloons

```
def find_min_arrow_shots(points):
    points.sort(key=lambda p: p[1])
    arrows = 0
    end = float('-inf')
    for s, e in points:
        if s > end:
            arrows += 1
            end = e
    return arrows
```

Notes

- Sort by **start** for merging; by **end** for greedy picks.
- Line sweep converts intervals to **events** with cumulative balance.

H. Trees – Traversals & DFS/BFS (96–110)

Catalog

1. Iterative Inorder/Preorder/Postorder

2. Level Order / Zigzag
3. Height / Balanced Check
4. Path Sum Variants
5. Lowest Common Ancestor
6. Diameter of Binary Tree
7. Serialize/Deserialize (BFS)
8. Kth Smallest in BST
9. Validate BST
10. Build Tree from Traversals
11. Vertical Order
12. Boundary Traversal
13. Binary Tree to DLL
14. Morris Traversal
15. Invert / Mirror Tree

Templates

Template H1 – DFS Traversals (Iterative)

```
def inorder(root):
    st, cur, out = [], root, []
    while cur or st:
        while cur:
            st.append(cur); cur = cur.left
        cur = st.pop()
        out.append(cur.val)
        cur = cur.right
    return out
```

Template H2 – BFS Level Order

```
from collections import deque

def level_order(root):
```

```
    if not root: return []
    q = deque([root])
    out = []
    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)
        out.append(level)
    return out
```

Worked Examples

Example H1 – Lowest Common Ancestor (Binary Tree)

```
def lca(root, p, q):
    if not root or root == p or root == q:
        return root
    L = lca(root.left, p, q)
    R = lca(root.right, p, q)
    if L and R: return root
    return L or R
```

Example H2 – Validate BST

```
def is_valid_bst(root):
    def dfs(node, lo, hi):
        if not node: return True
        if not (lo < node.val < hi): return False
        return dfs(node.left, lo, node.val) and dfs(node.right, node.val, hi)
    return dfs(root, float('-inf'), float('inf'))
```

Notes

- Iterative traversals avoid recursion limits.
- For BST problems, think **inorder ordering** or **range constraints**.

I. Graphs & Shortest Paths (111–120)

Catalog

1. BFS Shortest Path
2. DFS Connected Components
3. Cycle Detection (U/D)
4. Topological Sort (Kahn/DFS)
5. Course Schedule Feasibility
6. Dijkstra
7. Bellman-Ford
8. Floyd-Warshall
9. Union-Find (Kruskal MST)
10. Prim's MST

Templates

Template I1 – BFS Shortest Path (Unweighted)

```
from collections import deque

def bfs_shortest_path(n, adj, src):
    dist = [-1]*n
    dist[src] = 0
    q = deque([src])
    while q:
        u = q.popleft()
        for v in adj[u]:
            if dist[v] == -1:
                dist[v] = dist[u] + 1
                q.append(v)
    return dist
```

Template I2 – Union-Find (DSU)

```python
def dsu(n):
    parent = list(range(n))
    size = [1]*n
    def find(x):
        while x != parent[x]:
            parent[x] = parent[parent[x]]
            x = parent[x]
        return x
    def union(a, b):
        ra, rb = find(a), find(b)
        if ra == rb: return False
        if size[ra] < size[rb]:
            ra, rb = rb, ra
        parent[rb] = ra
        size[ra] += size[rb]
        return True
    return find, union
```

Worked Examples

Example I1 – Dijkstra (Min Heap)

```python
import heapq

def dijkstra(n, adj, src):
    INF = 10**18
    dist = [INF]*n
    dist[src] = 0
    pq = [(0, src)]
    while pq:
        d, u = heapq.heappop(pq)
        if d != dist[u]:
            continue
        for v, w in adj[u]:
            nd = d + w
            if nd < dist[v]:
                dist[v] = nd
```

```
                heapq.heappush(pq, (nd, v))
    return dist
```

Example I2 – Topological Sort (Kahn)

```
from collections import deque, defaultdict

def topo_sort(n, edges):
    g = defaultdict(list); indeg = [0]*n
    for u, v in edges:
        g[u].append(v); indeg[v] += 1
    q = deque([i for i in range(n) if indeg[i]==0])
    order = []
    while q:
        u = q.popleft()
        order.append(u)
        for v in g[u]:
            indeg[v] -= 1
            if indeg[v] == 0:
                q.append(v)
    return order if len(order) == n else []
```

Notes

- Pick **BFS** for unweighted; **Dijkstra** for positive weights.
- DSU is ideal for **connectivity & MST (Kruskal)**.

J. Dynamic Programming – 1D & 2D (121–140)

Catalog

1. Fibonacci / Climbing Stairs
2. House Robber I/II
3. Coin Change (min coins)
4. Coin Change II (count ways)
5. 0/1 Knapsack

6. Unbounded Knapsack
7. Longest Increasing Subsequence
8. Longest Common Subsequence
9. Edit Distance
10. Longest Palindromic Substring
11. Palindrome Partitioning (min cuts)
12. Matrix Chain Multiplication
13. Unique Paths with Obstacles
14. Maximal Square
15. Burst Balloons
16. Partition Equal Subset Sum
17. Rod Cutting
18. Buy & Sell Stock Variants
19. Wildcard/Regex Matching
20. Word Break I/II

Templates

Template J1 – 1D DP (Coin Change – min coins)

```
def coin_change(coins, amount):
    INF = amount + 1
    dp = [0] + [INF]*amount
    for a in range(1, amount+1):
        for c in coins:
            if a - c >= 0:
                dp[a] = min(dp[a], dp[a-c] + 1)
    return -1 if dp[amount] == INF else dp[amount]
```

Template J2 – 2D DP (Edit Distance)

```
def edit_distance(a, b):
    n, m = len(a), len(b)
    dp = [[0]*(m+1) for _ in range(n+1)]
```

```
    for i in range(n+1): dp[i][0] = i
    for j in range(m+1): dp[0][j] = j
    for i in range(1, n+1):
        for j in range(1, m+1):
            if a[i-1] == b[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[n][m]
```

Worked Examples

Example J1 – Longest Increasing Subsequence (Patience)

```
import bisect

def lis(nums):
    tails = []
    for x in nums:
        i = bisect.bisect_left(tails, x)
        if i == len(tails): tails.append(x)
        else: tails[i] = x
    return len(tails)
```

Example J2 – Word Break (I)

```
def word_break(s, word_set):
    n = len(s)
    dp = [False]*(n+1)
    dp[0] = True
    for i in range(1, n+1):
        for j in range(i):
            if dp[j] and s[j:i] in word_set:
                dp[i] = True
                break
    return dp[n]
```

Notes

- Identify **state** and **transition** clearly.
- Prefer 1D optimization when only **previous row** is needed.

K. Backtracking & Search (141–147)

Catalog

1. Subsets / Subsets with Duplicates
2. Permutations / with Duplicates
3. Combination Sum Variants
4. N-Queens
5. Sudoku Solver
6. Word Search
7. Restore IP Addresses

Templates

Template K1 – Generic Backtracking

```
def backtrack_template(choices):
    path, res = [], []
    def dfs(start=0):
        # check goal
        res.append(path[:])
        for i in range(start, len(choices)):
            path.append(choices[i])
            dfs(i+1)
            path.pop()
    dfs()
    return res
```

Template K2 – N-Queens Skeleton

```
def solve_n_queens(n):
    cols, d1, d2 = set(), set(), set()
    board = [['.']*n for _ in range(n)]
```

```
    res = []

    def dfs(r):
        if r == n:
            res.append(["".join(row) for row in board])
            return
        for c in range(n):
            if c in cols or (r-c) in d1 or (r+c) in d2:
                continue
            cols.add(c); d1.add(r-c); d2.add(r+c)
            board[r][c] = 'Q'
            dfs(r+1)
            board[r][c] = '.'
            cols.remove(c); d1.remove(r-c); d2.remove(r+c)

    dfs(0)
    return res
```

Worked Examples

Example K1 – Subsets (no duplicates)

```
def subsets(nums):
    res = [[]]
    for x in nums:
        res += [cur + [x] for cur in res]
    return res
```

Example K2 – Word Search

```
def exist(board, word):
    R, C = len(board), len(board[0])
    def dfs(r, c, i):
        if i == len(word): return True
        if r<0 or c<0 or r>=R or c>=C or board[r][c] != word[i]:
            return False
        ch = board[r][c]
        board[r][c] = '#'
```

```
        ok = dfs(r+1,c,i+1) or dfs(r-1,c,i+1) or dfs(r,c+1,i+1) or dfs(r,c-1,i+1)
        board[r][c] = ch
        return ok
    for r in range(R):
        for c in range(C):
            if dfs(r, c, 0): return True
    return False
```

Notes

- Control **visited state** carefully (modify/restore or use a set).
- Prune early (bounds, sums, conflicts) to keep search small.

L. Math, Bit, and Misc (148–150)

Catalog

1. Bit Tricks (single number, two singles)
2. Fast Power (binary exponentiation)
3. Reservoir Sampling / Fisher–Yates Shuffle

Templates

Template L1 – Fast Power (binary exp)

```
def fast_pow(x, n):
    if n < 0:
        x, n = 1/x, -n
    res = 1.0
    while n:
        if n & 1:
            res *= x
        x *= x
        n >>= 1
    return res
```

Template L2 – Fisher–Yates Shuffle

```python
import random

def shuffle_inplace(a):
    n = len(a)
    for i in range(n-1, 0, -1):
        j = random.randint(0, i)
        a[i], a[j] = a[j], a[i]
    return a
```

Worked Examples

Example L1 – Single Number (XOR)

```python
def single_number(nums):
    x = 0
    for v in nums:
        x ^= v
    return x
```

Example L2 – Two Unique Numbers (XOR partition)

```python
def two_single_numbers(nums):
    x = 0
    for v in nums: x ^= v
    low_bit = x & -x
    a = b = 0
    for v in nums:
        if v & low_bit: a ^= v
        else: b ^= v
    return a, b
```

Notes

- Prefer integer math to avoid float drift; for pow use **binary exponentiation**.
- For randomization, Fisher–Yates ensures **uniform** permutations.

Cheatsheets & Reference

Complexity Cheats (quick rules)

- **Two Pointers / Sliding Window**: O(n)
- **Monotonic Stack**: O(n)
- **Heaps (top-k/merge)**: O(n log k) / O(n log k + m log k)
- **Binary Search on Answer**: O(log R · C)
- **DFS/BFS**: O(V + E)
- **DP**: (#states × transition)

Pitfalls & Best Practices

- Prefer defaultdict(int) / Counter for counting.
- Shrink windows with a while loop **only when** invariant holds.
- For intervals, **sort by start** (merge) or **by end** (greedy pick).
- For graphs, choose **BFS vs Dijkstra** based on edge weights.
- For recursion, memoize; switch to iterative if depth is large.
- Avoid floats for equality; use integers or tolerances.

Pattern → Template Map (quick cues)

- Need smallest/longest substring with property → **Sliding Window (B2)**
- Count subarrays with sum/condition → **Prefix+Hash (C1)**
- Scheduling/rooms/top-k → **Heap (F1)** or **Sweep (G2)**
- Range updates many, apply once → **Difference Array → Prefix**
- Optimize a numeric answer under feasibility → **Binary Search on Answer (D2)**
