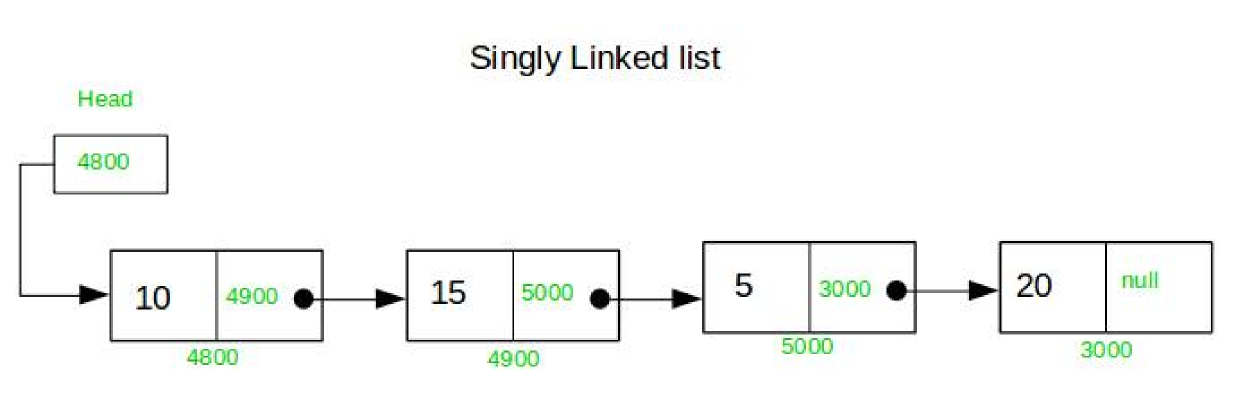
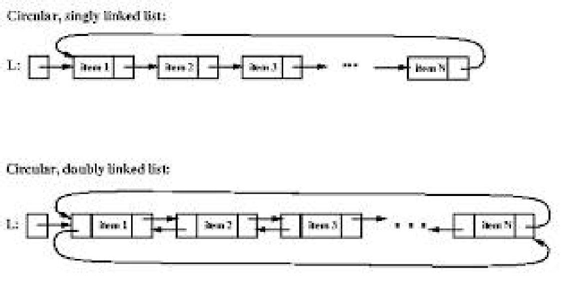
Linked List DSA Guide

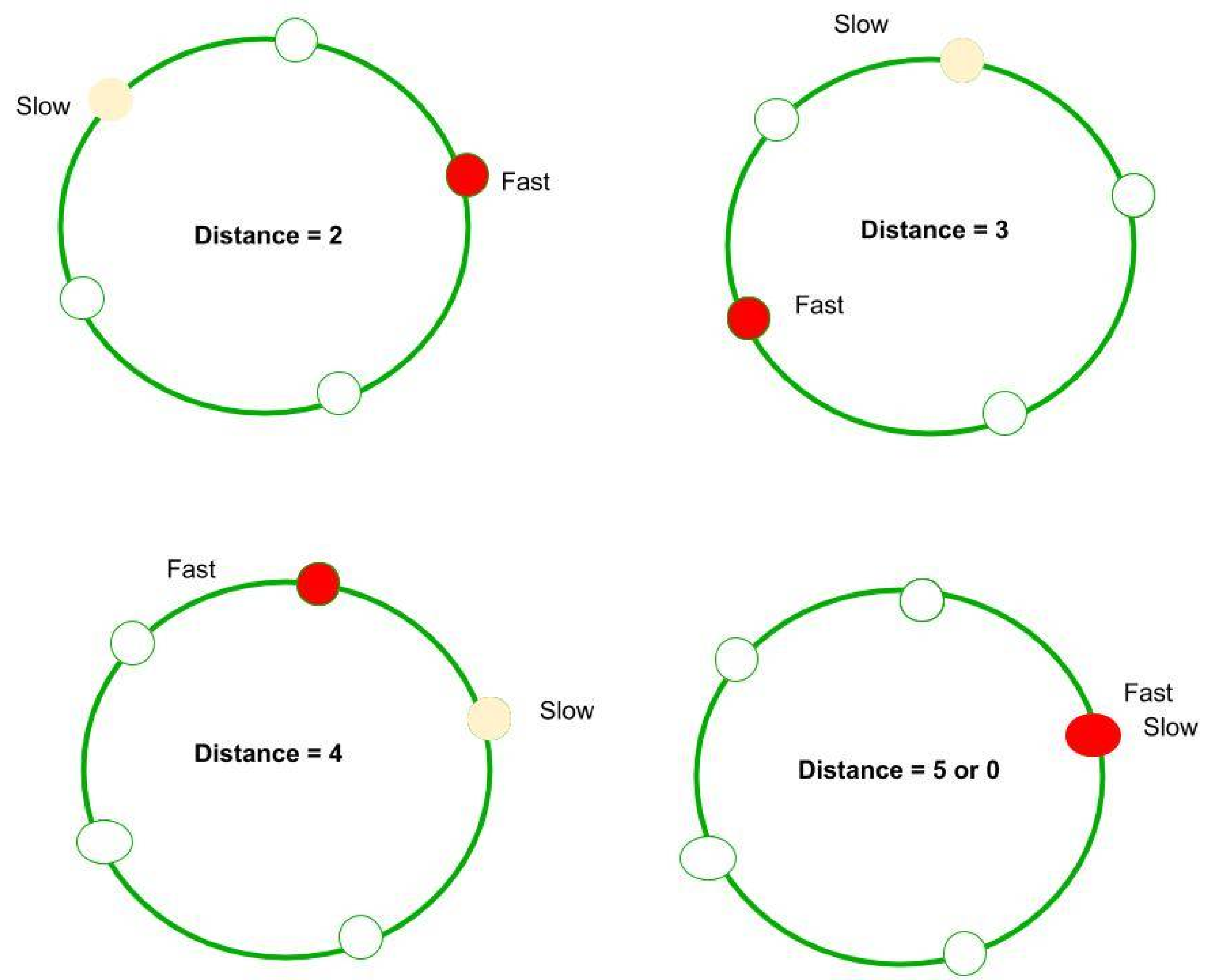
Linked List DSA Mastery

Below is **Linked List DSA Mastery** — a complete, structured, interview-oriented and competitive-programming–oriented guide covering **patterns, intuition, templates, tricks, and must-solve problems**.
Designed for **Python + DSA Mastery workflow**.

🚀 LINKED LIST DSA MASTERY







📌 1. LINKED LIST BASICS — Understanding the Structure

Why Linked Lists?

- Dynamic size
- O(1) insert/delete at head
- No contiguous memory needed
- Great for stack/queue, LRU cache, hash buckets etc.

Types

- **Singly Linked List**
- **Doubly Linked List**
- **Circular Linked List**
- **Skip Lists**
- **Multilevel Linked Lists**

Node Structure (Python)

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
```

📌 2. CORE LINKED LIST PATTERNS (The Only Things You Need)

✅ Pattern 1 — Two-Pointer (Fast & Slow)

Used for:

- Find middle
- Detect cycle
- Find cycle length
- Check palindrome
- Reorder list

Template

```python
slow = fast = head
while fast and fast.next:
    slow = slow.next
    fast = fast.next.next
```

✅ Pattern 2 — Reversal of Linked List (Partial & Full)

Full Reverse Template

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

Reverse first k nodes

Used in **Reverse in K-Groups**

✅ Pattern 3 — Dummy Node Technique

Avoids special-case handling at head.

```python
dummy = ListNode(0, head)
curr = dummy
```

Used in:

- Remove Nth node
- Merge lists
- Add numbers

✅ Pattern 4 — Linked List Manipulation Using References

Useful for:

- Merge two lists
- Sort list
- Partition list

✅ Pattern 5 — Recursion on Linked Lists

Used for:

- Reverse recursively
- Swap nodes in pairs
- Merge sort in LL

📌 3. TOP LINKED LIST OPERATIONS — TEMPLATES

🔹 Insert at Head

```python
node.next = head
head = node
```

🔹 Insert at End

```python
curr = head
while curr.next:
    curr = curr.next
curr.next = node
```

🔹 Delete a Node

(You have the value or node)

📌 4. LINKED LIST GOLDEN TRICKS (Interview Premium)

⭐ Trick 1: To detect cycle entry point

```
pgsql

fast = slow
after meet → slow=head
move both 1 step → where they meet is entry point
```

⭐ Trick 2: To find 2nd half of list

```
scss

slow moves 1x
fast moves 2x
slow stops at middle
```

⭐ Trick 3: Palindrome Linked List

1. Find middle
2. Reverse 2nd half
3. Compare halves

⭐ Trick 4: Delete Nth Node From End

```
pgsql

fast moves n steps, then move both until fast hits end
```

⭐ Trick 5: Odd-Even Linked List

Maintain **odd_head, even_head**, link separately.

⭐ Trick 6: Merge K Sorted Lists = Use MinHeap or Divide & Conquer

⭐ Trick 7: Sorting a Linked List = Merge Sort Only

Because array quicksort methods don't work efficiently here.

📌 5. LINKED LIST INTERVIEW TEMPLATES (Python)

⭐ Remove N-th Node from End

python

```python
def removeNthFromEnd(head, n):
    dummy = ListNode(0, head)
    slow = fast = dummy

    for _ in range(n):
        fast = fast.next

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

    slow.next = slow.next.next
    return dummy.next
```

⭐ Merge Two Sorted Lists

python

```python
def merge(l1, l2):
    dummy = ListNode()
    tail = dummy

    while l1 and l2:
        if l1.val < l2.val:
            tail.next, l1 = l1, l1.next
        else:
            tail.next, l2 = l2, l2.next
        tail = tail.next

    tail.next = l1 or l2
    return dummy.next
```

⭐ Detect Cycle

python

```python
def hasCycle(head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow is fast:
            return True
    return False
```

⭐ Reverse Nodes in K-Group

(Advanced but essential)

📌 6. LINKED LIST PATTERNS — PROBLEM CLASSIFICATION

🧠 Easy

- Reverse Linked List
- Middle of Linked List
- Delete Node
- Merge Two Sorted Lists
- Remove Duplicates

🧠 Medium

- Add Two Numbers
- Odd-Even Linked List
- Remove Nth Node From End
- Partition List
- Sort List (Merge Sort)
- Rotate List
- Intersection of Two Lists
- Linked List Cycle II
- Reorder List

🧠 Hard

- Reverse Nodes in K Group
- Merge K Sorted Lists
- Copy List with Random Pointer
- Flatten Multilevel Linked List
- LFU/LRU cache (LL + HashMap)
- Reverse alternating K groups

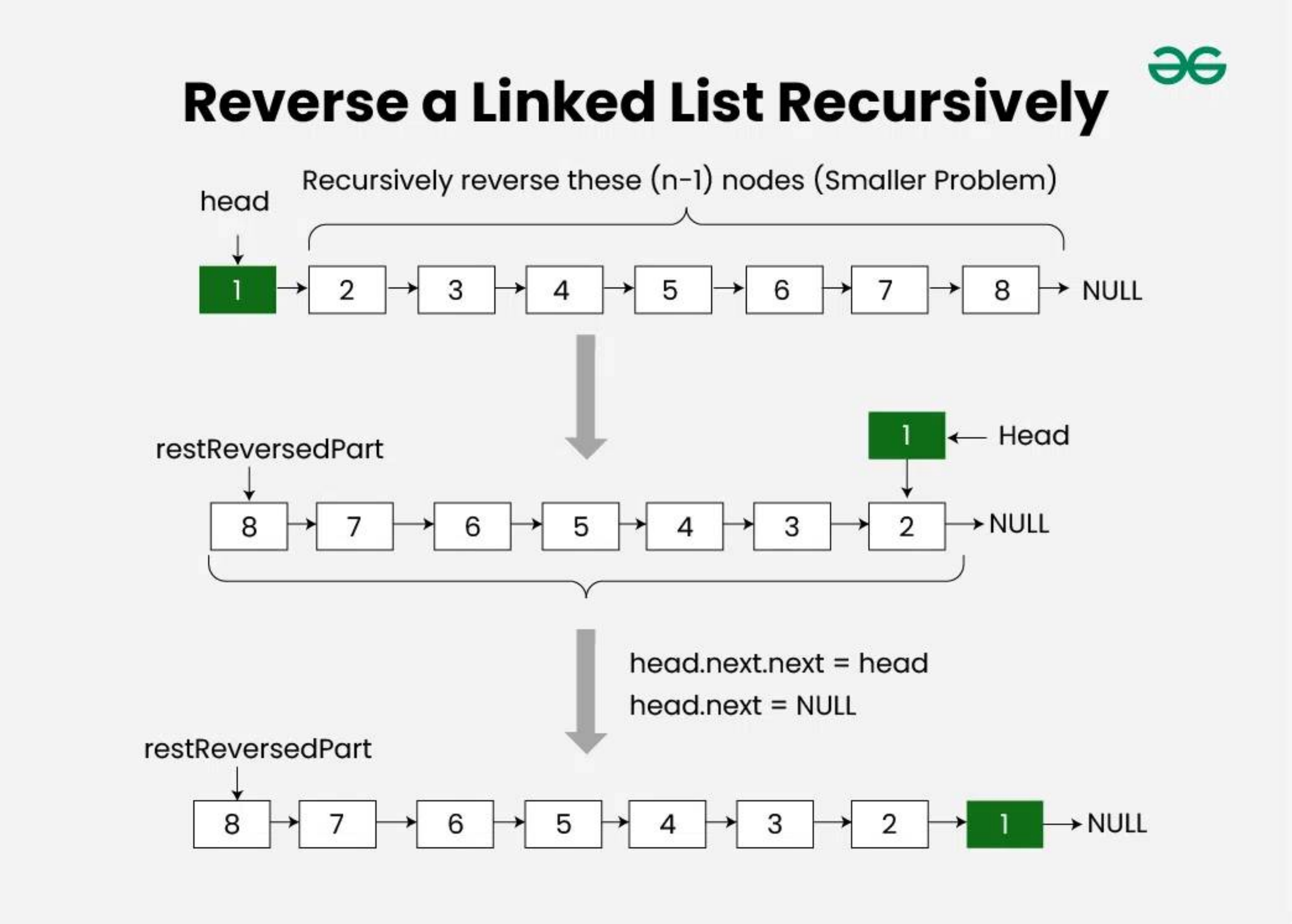
📌 7. LINKED LIST MASTER CHECKLIST (LC + CP)

Must Solve 25

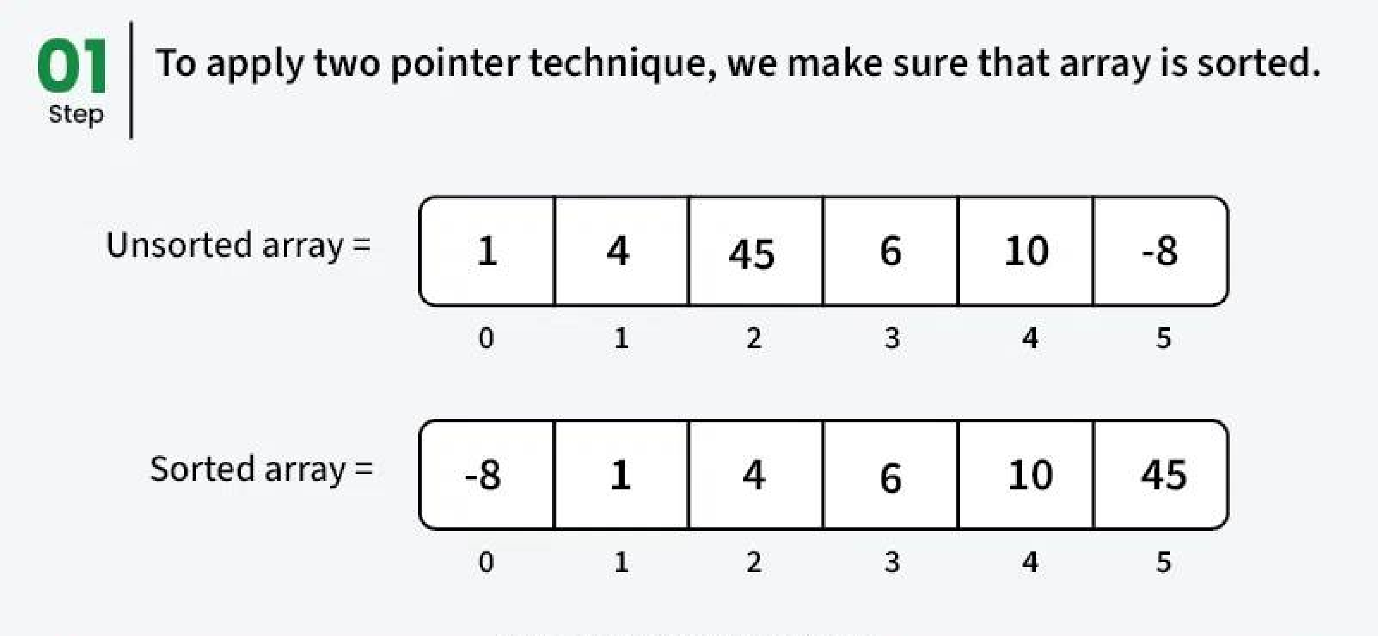
1. Reverse Linked List
2. Middle of Linked List
3. Linked List Cycle
4. Linked List Cycle II

5. Remove Nth Node from End
6. Merge Two Sorted Lists
7. Merge K Lists
8. Add Two Numbers
9. Palindrome Linked List
10. Sort List
11. Reorder List
12. Swap Nodes in Pairs
13. Odd-Even Linked List
14. Partition List
15. Rotate List
16. Delete Node
17. Intersection of Two Lists
18. Remove Duplicates
19. Reverse in K-Group
20. Flatten Multilevel Linked List
21. Copy Random Pointer List
22. Find Cycle Length
23. Split Linked List in Parts
24. Insertion Sort List
25. LRU Cache (LL + dict)

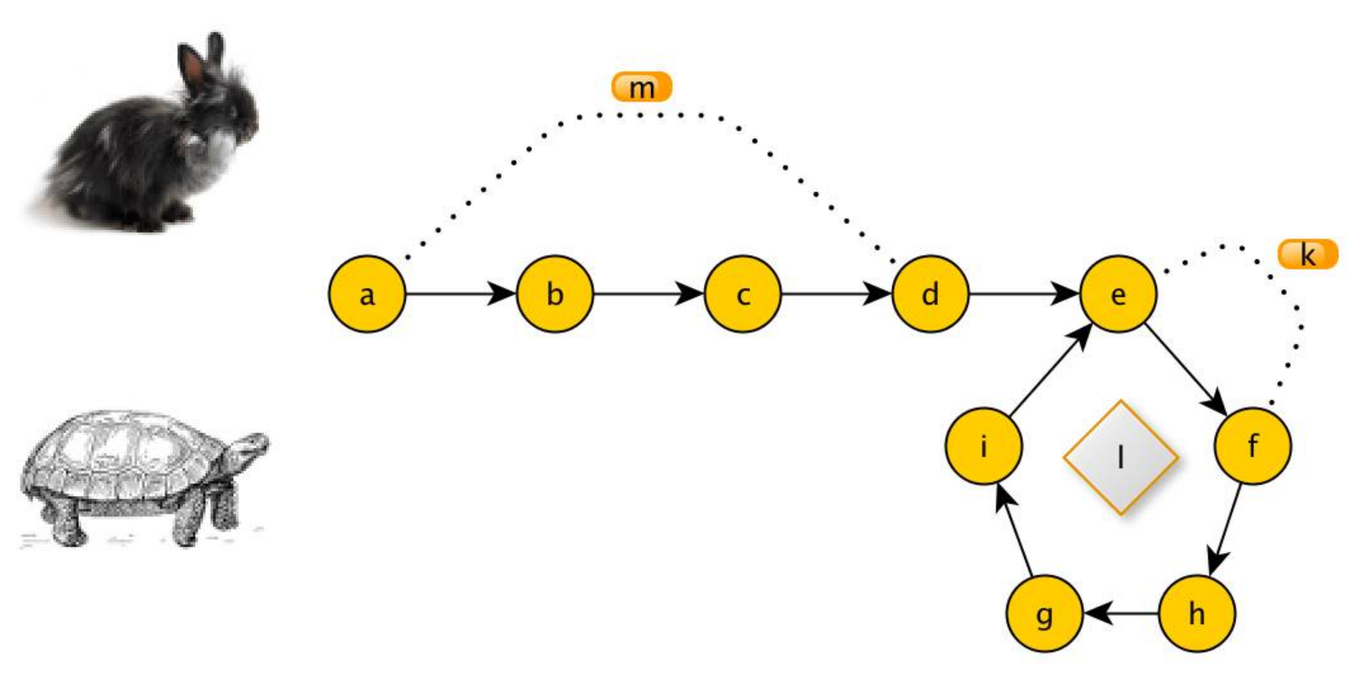
📌 8. LINKED LIST VISUAL INTUITION — Diagrams Added


Reverse a Linked List Recursively
head
Recursively reverse these (n-1) nodes (Smaller Problem)
1
2
3
4
5
6
7
8
NULL
restReversedPart
Head
8
7
6
5
4
3
2
1
NULL
restReversedPart
8
7
6
5
4
3
2
1
NULL

```
head.next.next = head
head.next = NULL
```



To apply two pointer technique, we make sure that array is sorted.



📌 9. ADVANCED TOPICS (Interview Boosters)

- **Skip Lists**
- **Unrolled Linked Lists**
- **XOR Linked List**
- **Linked List Memory Layout & Cache friendliness**
- **Graph represented with adjacency linked lists**

📌 10. Want a Linked List 100-Problems Roadmap or Linked List Patterns PDF?

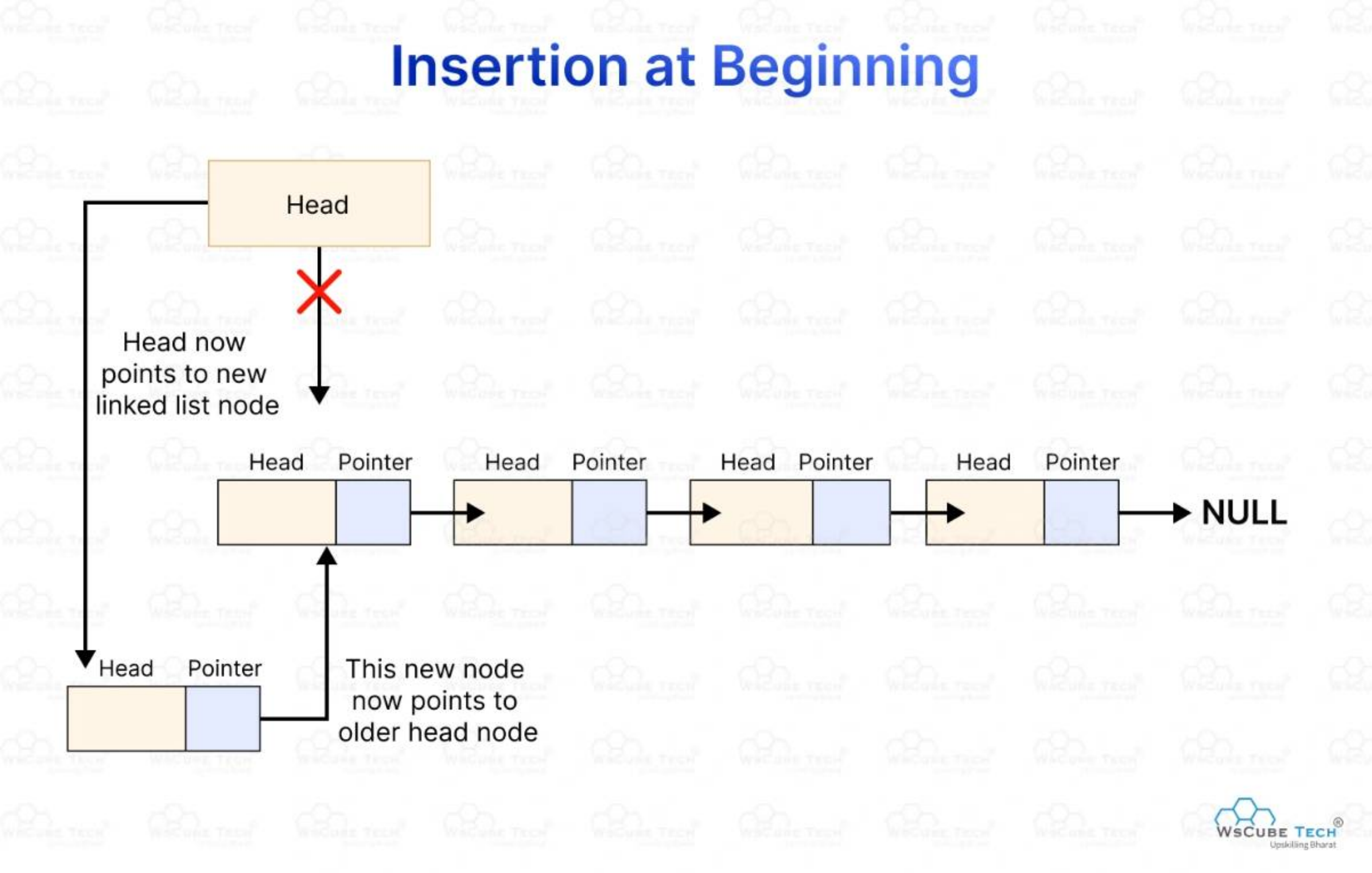
I can generate:
✅ 100 Linked List problems sorted by pattern
✅ A cheat-sheet PDF
✅ Python templates for every pattern
✅ Company-wise Linked List problem list (Google/Amazon/Meta)

Just tell me **which version you want**.

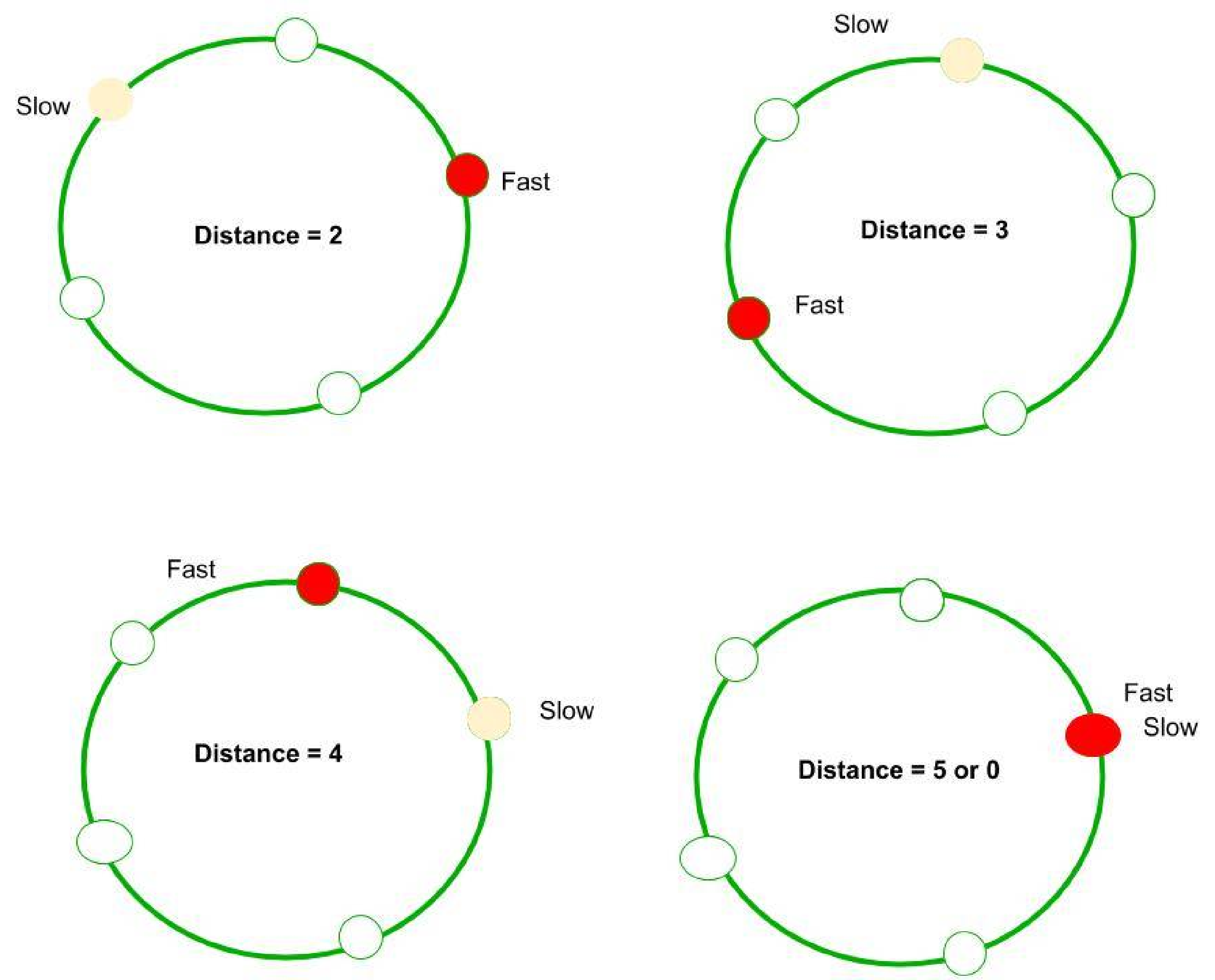
Patterns in LinkedList

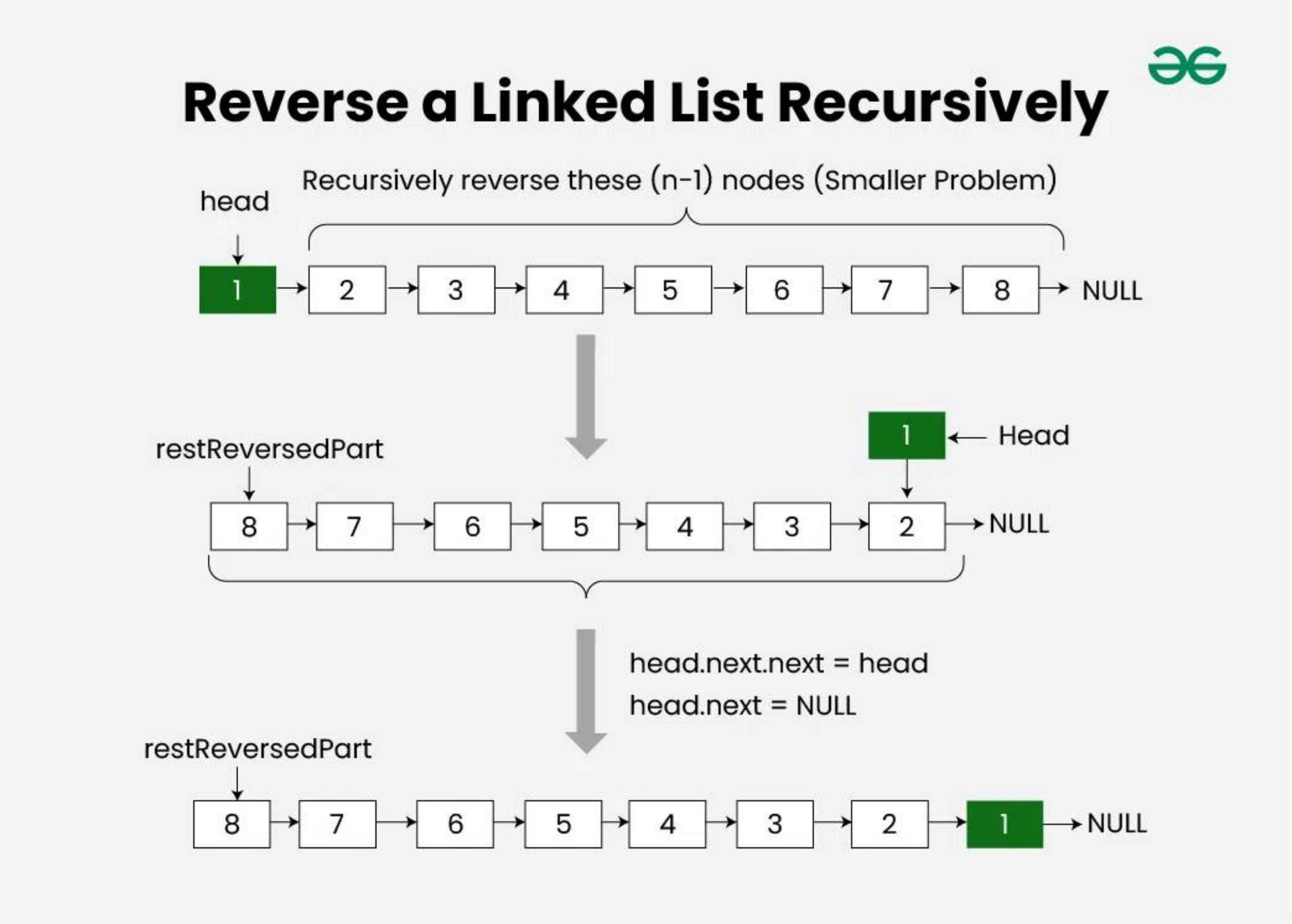
Here are the **complete Linked List Patterns** you must know for **DSA + Interviews + CP**.
This is the *master list*—every LinkedList problem maps to **one or a combination** of these patterns.

🚀 LINKED LIST PATTERNS (Ultimate Mastery Guide)



Insertion at Beginning




Reverse a Linked List Recursively
Recursively reverse these (n-1) nodes (Smaller Problem)
head
1
2
3
4
5
6
7
8
NULL
restReversedPart
Head
head.next.next = head
head.next = NULL

⭐ PATTERN 1 — Fast & Slow Pointers (Floyd's Cycle Technique)

Most important pattern.

Used for:

- Find middle
- Detect cycle
- Find cycle length
- Find cycle entry
- Check palindrome (used with reverse)
- Reorder list

Template:

```python
slow = fast = head
while fast and fast.next:
    slow = slow.next
    fast = fast.next.next
```

⭐ PATTERN 2 — Linked List Reversal (Full / Partial / Recursive)

Used for:

- Reverse entire list
- Reverse first k nodes
- Reverse nodes in k-groups
- Reverse second half
- Alternate-k reverse

Template:

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

Recursive Template:

```python
def reverse(head):
    if not head or not head.next:
        return head
    new_head = reverse(head.next)
    head.next.next = head
    head.next = None
    return new_head
```

⭐ PATTERN 3 — Dummy Node Technique

Eliminates edge cases at the head.

Used for:

- Remove Nth node
- Merge two lists
- Partition list
- Add numbers
- Delete duplicates
- Sort list

Template:

```python
dummy = ListNode(0)
dummy.next = head
curr = dummy
```

⭐ PATTERN 4 — Merge Pattern (Two Sorted Lists / Multiple Lists)

Used for:

- Merge sorted lists
- Merge sort linked list
- Merge K lists
- Insert into sorted LL

Template:

```python
dummy = ListNode(0)
tail = dummy

while l1 and l2:
    if l1.val < l2.val:
        tail.next, l1 = l1, l1.next
    else:
        tail.next, l2 = l2, l2.next
    tail = tail.next

tail.next = l1 or l2
return dummy.next
```

⭐ PATTERN 5 — Two-pass and One-pass Distance Pattern

Used when you need to operate **Nth from the end**, **length-based**, etc.

Example:

- Remove Nth node from end
- Find k-th element from end
- Split list into equal parts

One-pass Template:

```python
fast = slow = head
for _ in range(n):
    fast = fast.next

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

# slow is at (n-th from end) predecessor
```

⭐ PATTERN 6 — In-place Linked List Reconstruction

Rearranging nodes without extra arrays.

Used for:

- Odd-even linked list
- Partition list
- Segregate nodes by value
- Reorder list (L0 → Ln → L1 → L(n-1))

Example:

```
python

odd = head
even = head.next
even_head = even

while even and even.next:
    odd.next = even.next
    odd = odd.next
    even.next = odd.next
    even = even.next

odd.next = even_head
```

⭐ PATTERN 7 — Divide and Conquer (Sort List, Merge K Lists)

Linked lists don’t support random access → **merge sort** is the only efficient method.

Used for:

- Sort list
- Merge K sorted lists (divide K into halves)

Template:

```
python

def sortList(head):
    if not head or not head.next:
        return head

    mid = getMid(head)
    left = sortList(head)
    right = sortList(mid)
    return merge(left, right)
```

⭐ PATTERN 8 — Recursion Patterns

Useful when LL problems have natural recursion steps.

Used for:

- Reverse recursively
- Swap nodes in pairs
- Flatten multilevel linked list
- Add numbers (recursive)

Example:

```
python

def swapPairs(head):
    if not head or not head.next:
        return head

    nxt = head.next
    head.next = swapPairs(nxt.next)
    nxt.next = head
    return nxt
```

⭐ PATTERN 9 — Hashing Patterns (Special Cases)

Used for:

- Intersection of lists
- Cycle detection alternative
- Copy random pointer list
- Detect duplicates

Template:

```
python

seen = set()
while head:
    if head in seen:
        return True
    seen.add(head)
    head = head.next
return False
```

⭐ PATTERN 10 — Multi-pointer / Multi-list Interleaving

When manipulating **multiple lists simultaneously**.

Used for:

- Merge K sorted lists
- Flatten nested or multilevel lists
- Weaving two linked lists (copy random pointer)

⭐ PATTERN 11 — Memory Optimization / O(1) Trick

Used for:

Example:

Delete a node without head pointer

```
python

node.val = node.next.val
node.next = node.next.next
```

⭐ PATTERN 12 — Skip Pointers / Jump Pointers

Used in:

- Skip list
- Random pointer list
- LRU cache internal struct (DLL + HashMap)

📌 SUMMARY TABLE

Pattern	Problems
Fast–slow	Cycle, middle, palindrome, reorder
Reverse	K-groups, reorder, reverse halves
Dummy node	Remove, merge, partition
Merge pattern	Sort list, merge 2/K lists
Distance (Nth from end)	Remove Nth, split parts
In-place rearrange	Odd-even, partition
Divide & conquer	Sort list, K lists
Recursion	Swap pairs, reverse, flatten
Hashing	Random pointer, intersection

Pattern	Problems
Multi-list	K-lists, flatten
Memory tricks	Delete node, optimize
Skip pointers	Advanced LL structures

💬 Want the Linked List Patterns PDF or 100 problems grouped by patterns?

I can generate:

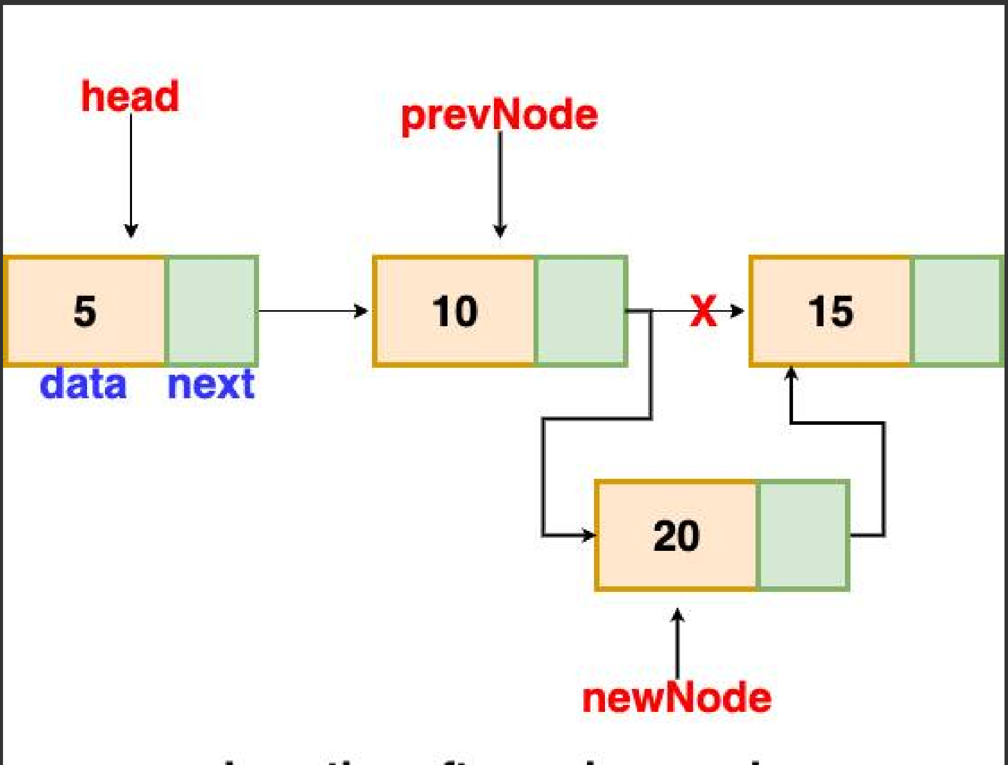
✅ A printable PDF
✅ Pattern → Template → 5 practice problems each
✅ 0–100 Linked List Roadmap

Just tell me!

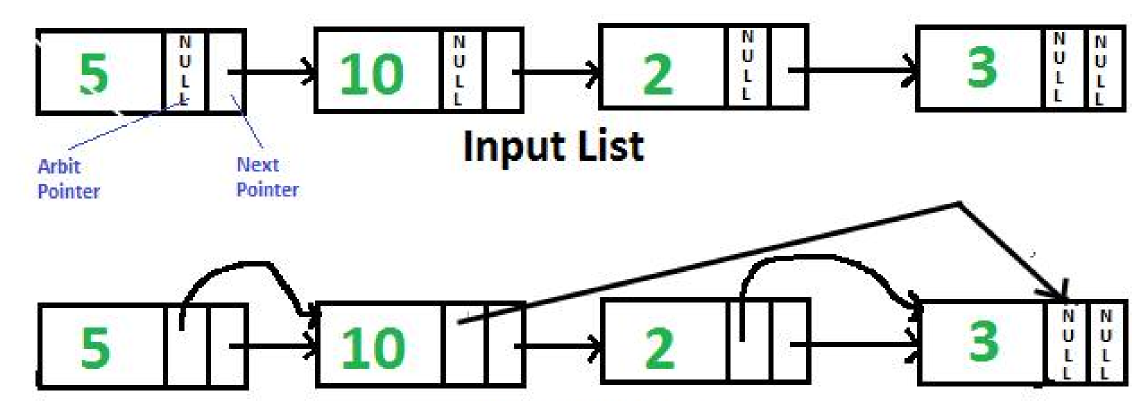
0–100 Linked List Roadmap

Here is the **0–100 Linked List Roadmap** — the ultimate progression to master every pattern, every problem type, and every trick used in interviews.
This roadmap is structured so you learn **in layers**, each building on previous skills.

🚀 LINKED LIST 0 → 100 ROADMAP (DSA MASTERY)

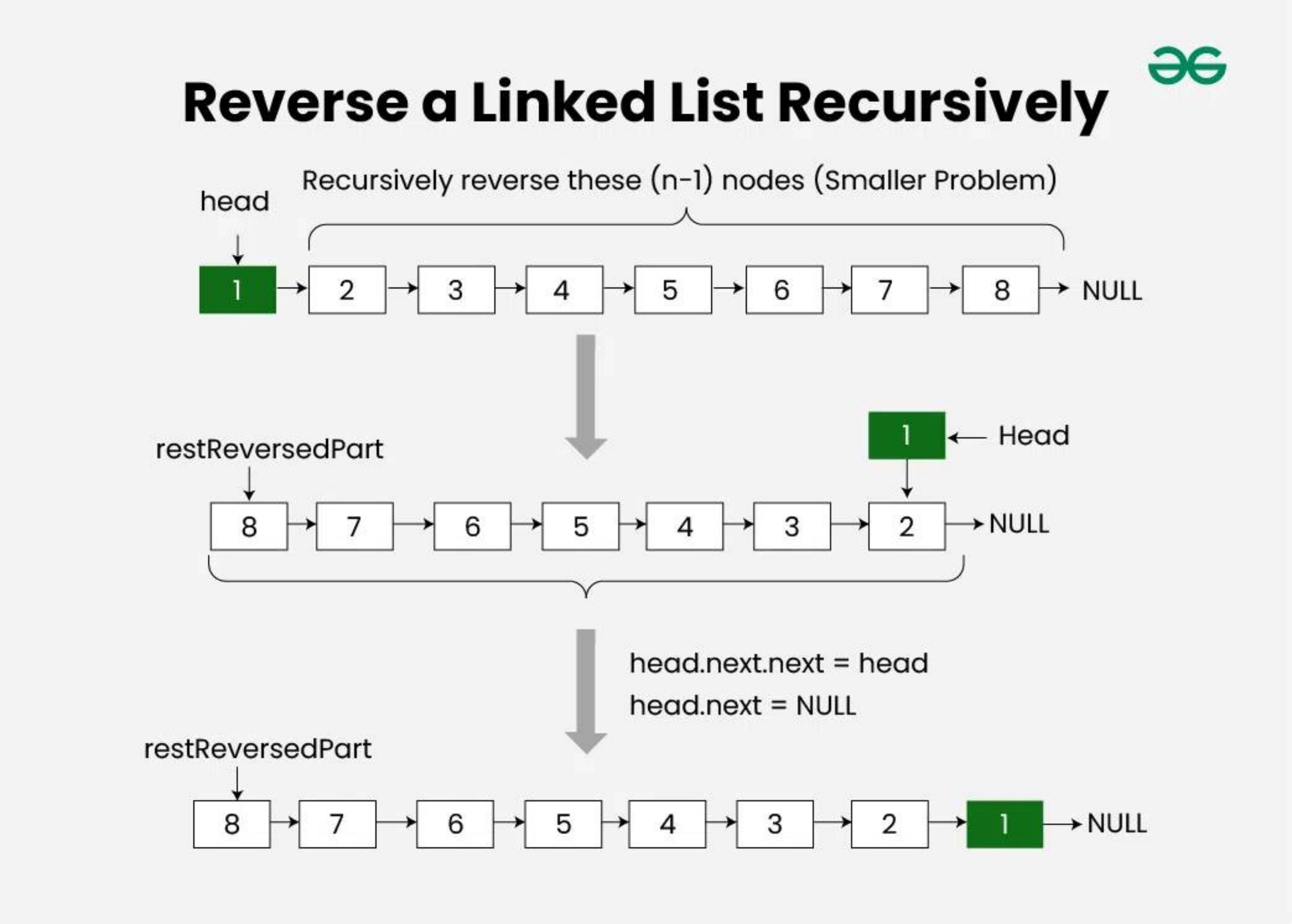


Insertion after a given node



Input List

Result List



🎯 LEVEL 0 — Fundamentals (0–5)

Understand structure, pointer movement & creation.

Learn:

- Node structure
- LL traversal
- Insert & delete (head, tail, middle)
- Dummy node concept

Problems (5)

1. Implement Linked List
2. Insert Node at Head
3. Insert Node at Tail
4. Delete Node by Value
5. Print Linked List

🎯 LEVEL 1 — Basics (6–15)

Start working with standard operations.

Learn:

- Length of list
- Search in list
- Merge simple lists
- Compare lists

Problems (10)

6. Get Length of Linked List
7. Search for a Value
8. Compare Two Linked Lists
9. Merge Two Unsorted Lists
10. Get Nth Node
11. Get Nth From End (Brute)
12. Count Occurrences
13. Check if list contains value
14. Find middle by counting
15. Maximum/minimum in LL

🎯 LEVEL 2 — Fast & Slow Pointer Mastery (16–25)

Most important pattern in linked lists.

Learn:

- Find middle (fast–slow)
- Detect cycle
- Find cycle length
- Find cycle start

Problems (10)

16. Middle of Linked List
17. Check if Linked List is Circular
18. Detect Cycle (Floyd's Algorithm)
19. Find Starting Node of Cycle

20. Find Length of Cycle
21. Check if list is palindrome (without reverse)
22. Split list in two halves
23. Find kth node from end (one-pass)
24. Detect intersection using cycle trick
25. Move last element to front

🎯 LEVEL 3 — Reversal Pattern (26–40)

Master full, partial, iterative & recursive reversal.

Learn:

- Reverse list (iterative)
- Reverse list (recursive)
- Reverse first k nodes
- Reverse between m and n
- Reverse in K groups

Problems (15)

26. Reverse Linked List (Iterative)
27. Reverse Linked List (Recursive)
28. Reverse First K Nodes
29. Reverse Last K Nodes
30. Reverse Sublist (Between m & n)
31. Reverse Nodes in K-Group
32. Reverse Even/Odd positions
33. Reverse LL in pairs
34. Reverse alternate K nodes
35. Rotate Linked List Left
36. Rotate Linked List Right
37. Check Palindrome (reverse 2nd half)
38. Reorder List (L0 → Ln → L1 → ...)
39. Swap Nodes in Pairs
40. Reverse LL keeping even nodes intact

🎯 LEVEL 4 — Dummy Node Pattern (41–50)

Eliminates boundary edge cases.

Learn:

- Remove nodes
- Insert nodes
- Merge operations
- Clean handling at head

Problems (10)

41. Remove Nth Node from End
42. Remove Duplicates from Sorted List
43. Remove Duplicates from Unsorted List
44. Delete Node in O(1) (No head given)
45. Partition List (less, greater equal)
46. Remove Elements with a Given Value
47. Insert into Sorted Linked List
48. Merge Two Sorted Lists
49. Add Two Numbers (LL addition)
50. Odd Even Linked List

🎯 LEVEL 5 — Merge + Divide & Conquer (51–65)

Key for sorting and K-list merge.

Learn:

- Merge sort on LL
- Find middle using fast–slow
- Merge K sorted lists
- Divide lists and rebuild

Problems (15)

51. Sort Linked List (Merge Sort)
52. Merge Sort Variant: Sort by Absolute Values
53. Merge K Sorted Lists (Heap)
54. Merge K Sorted Lists (Divide & Conquer)
55. Flatten a Linked List (multi-level 1D structure)
56. Flatten Linked List depth-wise

57. Merge Two Circular Linked Lists
58. Find intersection of lists
59. Split Linked List into K Parts
60. Unfold a Linked List
61. Check if two lists are identical
62. Pairwise swap without swapping data
63. Split list into odd and even lists
64. Move zeros to end in LL
65. Move negative numbers to front (stable)

🎯 LEVEL 6 — Multi-pointer & Reconstruction (66–80)

Rearranging nodes with pointer juggling.

Learn:

- Reorder nodes
- Odd-even reordering
- Segregation
- Weaving lists

Problems (15)

66. Reorder List (classic: L0 → Ln → ...)
67. Odd-Even Linked List
68. Segregate Even & Odd Nodes
69. Segregate 0s, 1s, and 2s
70. Delete alternate nodes
71. Pairwise reverse nodes
72. Move last N nodes to front
73. Alternating merge of two lists
74. Zip Merge
75. Unzip Linked List
76. Sorted Insert in Circular LL
77. Delete N nodes after M nodes
78. Clone linked list without extra memory
79. Move nodes based on frequency
80. Remove kth occurrence of element

🎯 LEVEL 7 — Advanced Linked List Problems (81–95)

Focus on tricky structures and scenarios.

Learn:

- Random pointer list
- Doubly linked list
- Circular linked structures
- Nested lists

Problems (15)

81. Copy List with Random Pointer
82. Flatten Multi-Level Doubly LL
83. Add Numbers II (digits in forward order)
84. Intersection of Two Linked Lists
85. Browser History LL Simulation
86. Implement Queue using Linked List
87. Clone complex circular list
88. Flatten N-ary Nested Lists (LL-based)
89. Delete all occurrences of a key
90. K-th smallest in LL (selection)
91. Convert LL to Balanced BST
92. Convert Sorted LL to Binary Tree
93. Find first non-repeating element in LL
94. Find first repeating element in LL
95. Rearrange LL by absolute values

🎯 LEVEL 8 — Expert Level / System Design Linked Lists (96–100)

Used in real interview systems.

Learn:

- LRU cache design
- LFU cache
- Deque with DLL
- Skip lists

- Memory-optimized linked structures

Problems (5)

96. LRU Cache (DLL + HashMap)
97. LFU Cache (DLL + HashMap + freq lists)
98. Implement Deque using Doubly LL
99. Skip List Implementation
100. XOR Linked List (memory optimized)

🔥 Bonus: Master Pattern Mapping Chart

Pattern	Roadmap Levels
Fast–Slow	16–25
Reversal	26–40
Dummy Node	41–50
Merge	48, 51–65
Divide & Conquer	51–55
In-place Rearrangement	66–80
Random Pointer	81–82
DLL/Circular LL	81–95
LRU/LFU	96–100

📄 Want this as a PDF printable roadmap with problem links?

I can export this roadmap into:

✅ **Beautiful PDF (DSA Mastery Theme)**
✅ With boxes, patterns, color-coding
✅ Includes LeetCode links
✅ Includes Python templates

Just tell me: **“Generate Linked List Roadmap PDF”**.

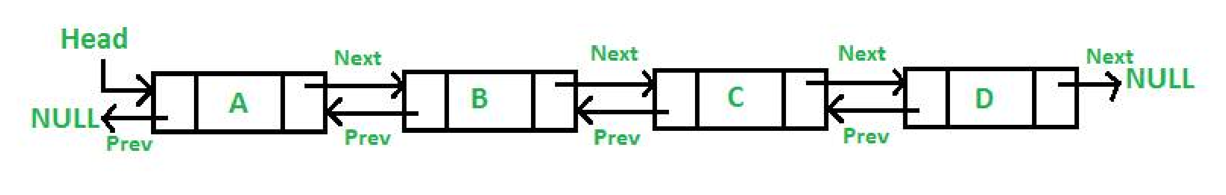
Python Templates for Doubly LL and CLL

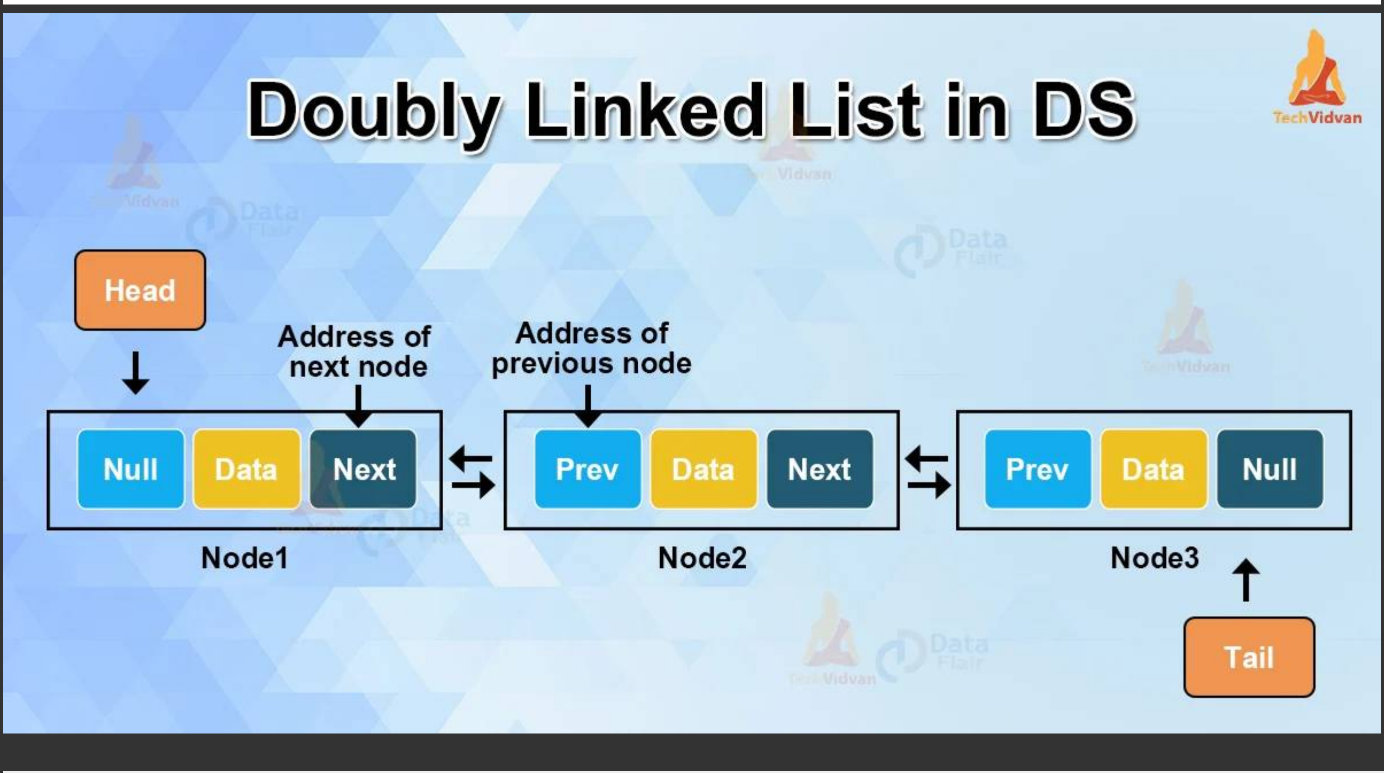
Below are **clean, production-ready Python templates** for:

✅ **Doubly Linked List (DLL)**
✅ **Circular Linked List (CLL – singly & doubly)**

All templates follow **DSA Mastery structure** → minimal, interview-ready, and extendable.

🔷 1. DOUBLY LINKED LIST (DLL) — Python Template





✅ DLL Node

```python
class DLLNode:
    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None
```

✅ Doubly Linked List Class

```python
class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None
```

🔹 Insert at Head

```python
    def insert_head(self, val):
        node = DLLNode(val)

        if not self.head:
            self.head = self.tail = node
            return

        node.next = self.head
        self.head.prev = node
        self.head = node
```

🔹 Insert at Tail

```python
    def insert_tail(self, val):
        node = DLLNode(val)

        if not self.tail:
            self.head = self.tail = node
            return

        self.tail.next = node
        node.prev = self.tail
        self.tail = node
```

🔹 Delete a Node (by value)

```python
    def delete(self, val):
        curr = self.head

        while curr and curr.val != val:
            curr = curr.next

        if not curr:
            return  # not found

        if curr.prev:
            curr.prev.next = curr.next
        else:
            self.head = curr.next

        if curr.next:
            curr.next.prev = curr.prev
        else:
            self.tail = curr.prev
```

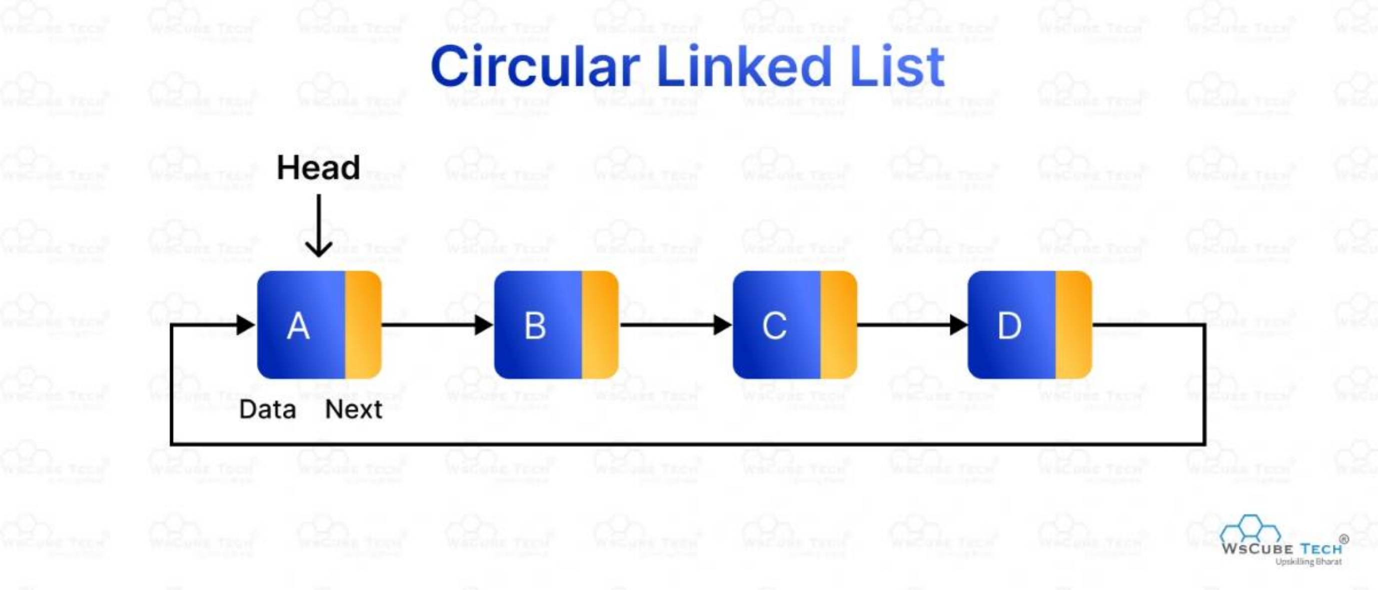
🔹 Traverse Forward

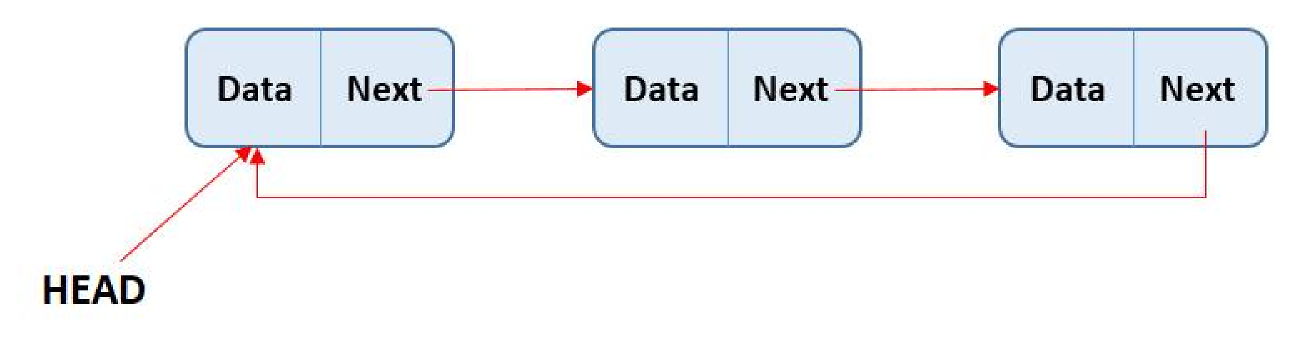
```python
    def traverse_forward(self):
        curr = self.head
        while curr:
            print(curr.val, end=" -> ")
            curr = curr.next
        print("None")
```

🔹 Traverse Backward

```python
    def traverse_backward(self):
        curr = self.tail
        while curr:
            print(curr.val, end=" -> ")
            curr = curr.prev
        print("None")
```

🔷 2. CIRCULAR LINKED LIST (CLL — Singly) — Python Template





✅ CLL Node

```python
class CLLNode:
    def __init__(self, val):
        self.val = val
        self.next = None
```

✅ Circular Linked List Class (Singly)

```python
class CircularLinkedList:
    def __init__(self):
        self.head = None
```

🔹 Insert at End

```python

    def insert(self, val):
        node = CLLNode(val)

        if not self.head:
            self.head = node
            node.next = node
            return

        curr = self.head
        while curr.next != self.head:
            curr = curr.next

        curr.next = node
        node.next = self.head
```

🔹 Delete a Node

```python

    def delete(self, val):
        if not self.head:
            return

        # 1. head to be deleted
        if self.head.val == val:
            curr = self.head
            while curr.next != self.head:
                curr = curr.next
            if curr == self.head:  # only 1 node
                self.head = None
            else:
                curr.next = self.head.next
                self.head = self.head.next
            return

        # 2. general case
        curr = self.head
        while curr.next != self.head and curr.next.val != val:
            curr = curr.next

        if curr.next.val == val:
            curr.next = curr.next.next
```

🔹 Traverse

```python

    def traverse(self):
        if not self.head:
            print("Empty")
            return

        curr = self.head
        while True:
            print(curr.val, end=" -> ")
            curr = curr.next
            if curr == self.head:
                break
        print("(HEAD)")
```

🔷 3. DOUBLY CIRCULAR LINKED LIST (DCLL) — Python Template

✅ DCLL Node

```python
class DCLLNode:
    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None
```

✅ Doubly Circular Linked List Class

```python
class DoublyCircularLinkedList:
    def __init__(self):
        self.head = None
```

🔹 Insert at End

```python
    def insert(self, val):
        node = DCLLNode(val)

        if not self.head:
            self.head = node
            node.next = node
            node.prev = node
            return

        tail = self.head.prev

        tail.next = node
        node.prev = tail
        node.next = self.head
        self.head.prev = node
```

🔹 Delete a Node

```python
    def delete(self, val):
        if not self.head:
            return

        curr = self.head
        # traverse until full loop
        while True:
            if curr.val == val:
                if curr.next == curr:  # only one node
                    self.head = None
                else:
                    curr.prev.next = curr.next
                    curr.next.prev = curr.prev
                    if curr == self.head:
                        self.head = curr.next
                return

            curr = curr.next
            if curr == self.head:
                break  # not found
```

🔹 Traverse Forward

```python

    def traverse_forward(self):
        if not self.head:
            print("Empty")
            return

        curr = self.head
        while True:
            print(curr.val, end=" <-> ")
            curr = curr.next
            if curr == self.head:
                break
        print("(HEAD)")
```

✅ YOU NOW HAVE:

✔️ Doubly Linked List (DLL)
✔️ Circular Linked List (CLL)
✔️ Doubly Circular Linked List (DCLL)
✔️ Insert, delete, traversal templates
✔️ Interview-ready clean implementations

📌 Want next?

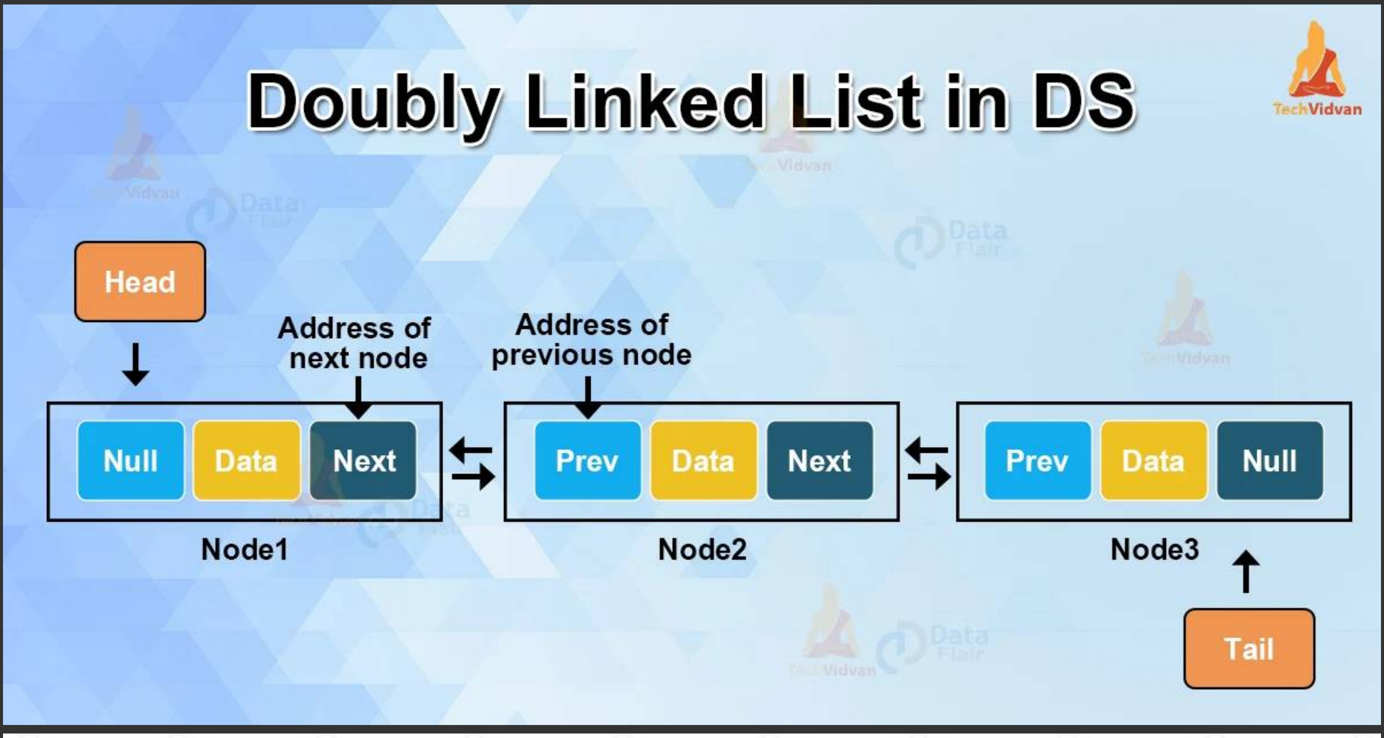
I can give you:

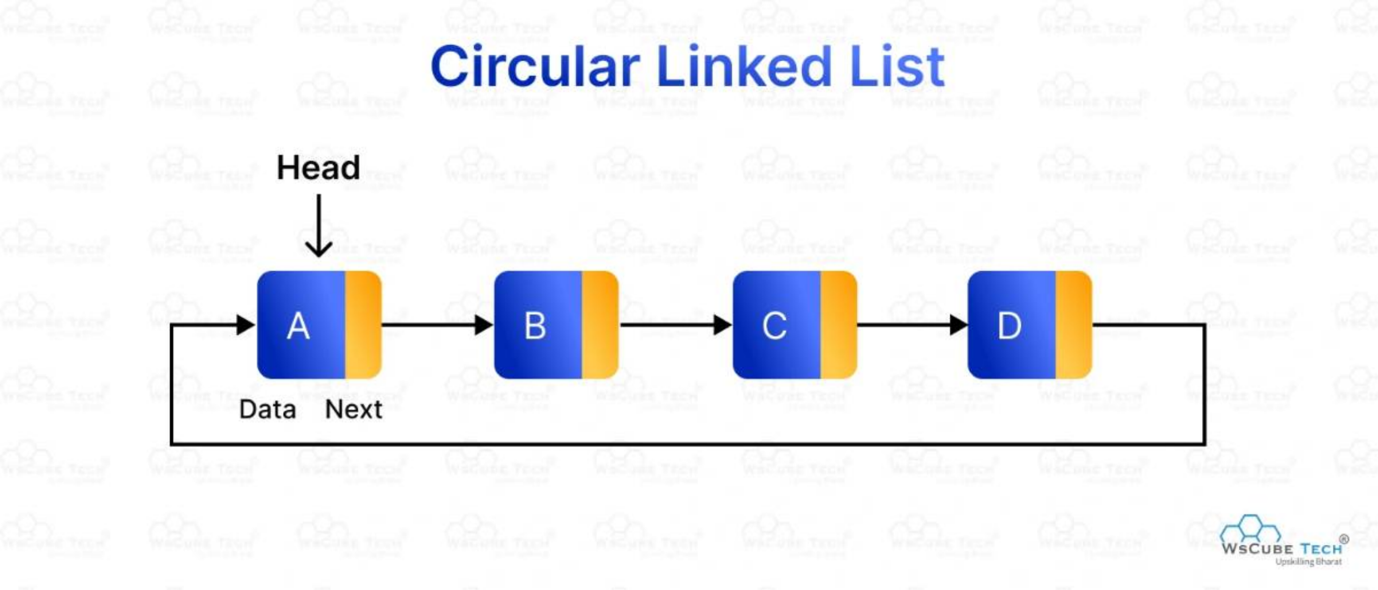
📘 **20 most important DLL & CLL problems**
🧠 **DLL/CLL pattern cheat-sheet**
⚙️ **LRU Cache using Doubly LL (production-ready)**

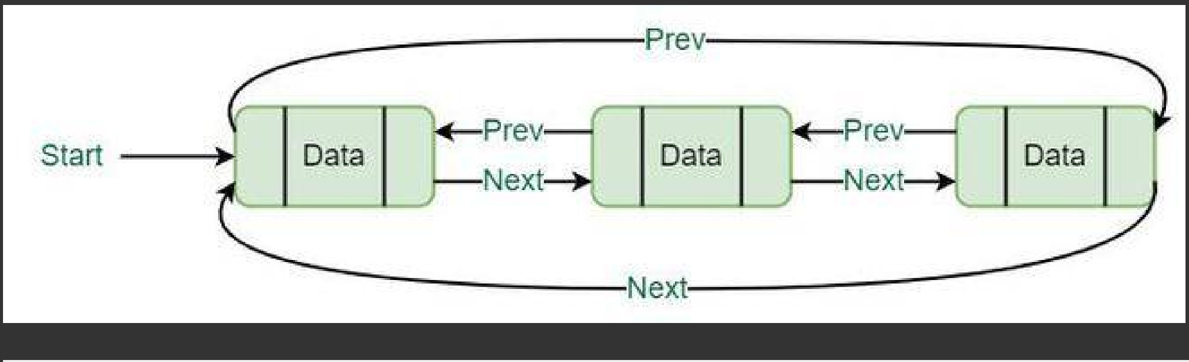
Just tell me!

Here is the **DLL + CLL Pattern Cheat-Sheet** — a compact, high-signal guide for interviews & DSA mastery.
This captures **all core operations, patterns, tricks, templates, and problem mappings** for **Doubly Linked Lists (DLL)** and **Circular Linked Lists (CLL)**.

🔷 DLL + CLL PATTERN CHEAT-SHEET (Python + DSA Mastery)







🚀 1. ESSENTIAL NODE TEMPLATES

Doubly Linked List Node

```python
class DLLNode:
    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None
```

Circular Linked List Node

```python
class CLLNode:
    def __init__(self, val):
        self.val = val
        self.next = None
```

Doubly Circular Linked List Node

```python
class DCLLNode:
    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None
```

🚀 2. DLL PATTERNS (Doubly Linked List)

⭐ DLL Pattern 1 — Bidirectional Traversal

Allows **forward & backward** movement.

Use-cases:

- Browser history
- Undo/Redo system

- Playlist navigation
- LRU/LFU caches

Template:

```python
curr = self.head      # forward
curr = self.tail      # backward
```

⭐ DLL Pattern 2 — Insert/Delete in O(1)

DLL supports **O(1) deletion of any known node**.

Delete a known node (O(1)):

```python
node.prev.next = node.next
node.next.prev = node.prev
```

⭐ DLL Pattern 3 — Sentinel (Dummy Head/Tail)

Eliminates boundary cases.

Structure:

- dummy_head
- dummy_tail
- real nodes in between

Template:

```python
self.head = Node(0)
self.tail = Node(0)
self.head.next = self.tail
self.tail.prev = self.head
```

Used in:

- LRU Cache
- LFU Cache
- Deques
- Priority queues with DLL

⭐ DLL Pattern 4 — Merge Two DLLs

Useful in:

- Merging sorted lists
- Flatten multilevel DLL

Template:

```python
def merge(a, b):
    dummy = DLLNode(0)
    tail = dummy

    while a and b:
        if a.val < b.val:
            tail.next, a.prev = a, tail
            a = a.next
        else:
            tail.next, b.prev = b, tail
            b = b.next
        tail = tail.next

    tail.next = a or b
    if tail.next:
        tail.next.prev = tail

    return dummy.next
```

⭐ DLL Pattern 5 — Flattening Multilevel DLL

Each node may have:

```lua
prev, next, child
```

Steps:

1. Recursively flatten child list
2. Insert flattened child between current node and next node
3. Fix prev pointers

⭐ DLL Pattern 6 — Deque Implementation

DLL enables:

- O(1) push front
- O(1) push back
- O(1) pop front

- O(1) pop back

Used in:

- Sliding window problems
- LFU, LRU Cache implementation

⭐ DLL Pattern 7 — Efficient Node Relocation

Reposition a node without deleting it.

Template:

```python
def move_to_front(self, node):
    self._remove(node)
    self._add_first(node)
```

Used in:

- LRU Cache
- Frequency-based queues

🚀 3. CLL PATTERNS (Circular Linked List)

⭐ CLL Pattern 1 — Loop Traversal

A CLL loops back to head.

Template:

```python
curr = self.head
while True:
    # process
    curr = curr.next
    if curr == self.head:
        break
```

⭐ CLL Pattern 2 — Tail.next = Head (Core Trick)

This enables:

- Round-robin scheduling
- Josephus Problem

- Hot Potato Game Simulation

⭐ CLL Pattern 3 — Insert at Tail in O(1)

Template:

```python
node.next = head.next
tail.next = node
```

⭐ CLL Pattern 4 — Deletion Requires Circular Maintenance

General delete:

```python
prev.next = prev.next.next
# ensure circularity remains:
if to_delete == head: head = head.next
```

⭐ CLL Pattern 5 — Split Circular List into Two Halves

Using slow–fast pointers:

Result:

```sql
First half → circular
Second half → circular
```

⭐ CLL Pattern 6 — Merge Two Circular Lists

Ensure final tail points to new head.

⭐ CLL Pattern 7 — Josephus Problem Pattern

Core formula:

```python
pos = (pos + k) % n
```

CLL implementation:

- Move k–1 steps
- Delete current node
- Repeat

🚀 4. DOUBLY CIRCULAR LINKED LIST (DCLL) PATTERNS

⭐ DCLL Pattern 1 — Constant-Time Head/Tail Rotation

You can rotate both directions:

```python
head = head.next     # rotate forward
head = head.prev     # rotate backward
```

Used in:

- Carousel UI
- Round-table simulations
- Music playlists

⭐ DCLL Pattern 2 — Easy Dequeue Operations

Deque operations:

- push_front → O(1)
- push_back → O(1)
- pop_front → O(1)
- pop_back → O(1)

⭐ DCLL Pattern 3 — Perfect for Game Simulations

Because every player has:

```lua
prev and next
```

and the structure loops forever.

Examples:

- Hot potato game
- Circular elimination games

- Multiplayer turn simulators

🚀 5. PROBLEM TYPE → PATTERN MAPPING

🎯 DLL Problems

Problem	Pattern
LRU Cache	Dummy head-tail + move_to_front
LFU Cache	DLL buckets + O(1) updates
Browser History	Bidirectional traversal
Flatten Multilevel List	Merge + recursion
Insert/Delete at both ends	Deque pattern
Sorted Insert	Merge pattern
Reverse DLL	Swap next & prev pointers

🎯 CLL Problems

Problem	Pattern
Josephus	CLL elimination loop
Round Robin Scheduling	Tail.next = head
Hot Potato Game	Circular pointer rotation
Insert/Delete in CLL	Maintain circularity
Split circular list	Fast–slow pointer
Merge circular lists	Fix final tail pointer

🎯 DCLL Problems

Problem	Pattern
Full-featured Deque	DCLL O(1) ops
Music playlist repeat	Bidirectional + circular
Circular game ring	Node.prev/next rotation
Multi-player elimination	Rotate + delete

🚀 6. QUICK REFERENCE — CORE TEMPLATES

🔹 Delete a known DLL node in O(1)

```python
if node.prev:
    node.prev.next = node.next
if node.next:
    node.next.prev = node.prev
```

🔹 CLL traversal

```python
curr = head
while True:
    curr = curr.next
    if curr == head: break
```

🔹 DCLL rotation

```python
head = head.next  # forward
head = head.prev  # backward
```

📌 Want the 20 DLL Problems + 20 CLL Problems with solutions and templates?

I can generate:

✅ *DLL Problem Set (Easy → Hard)*
✅ *CLL Problem Set (Easy → Hard)*
✅ *Full DCLL Patterns with examples*
✅ *Interview-style coding templates*

Just tell me: **"DLL/CLL problem list"**.

Interview-style coding templates --> Tips for code represenations

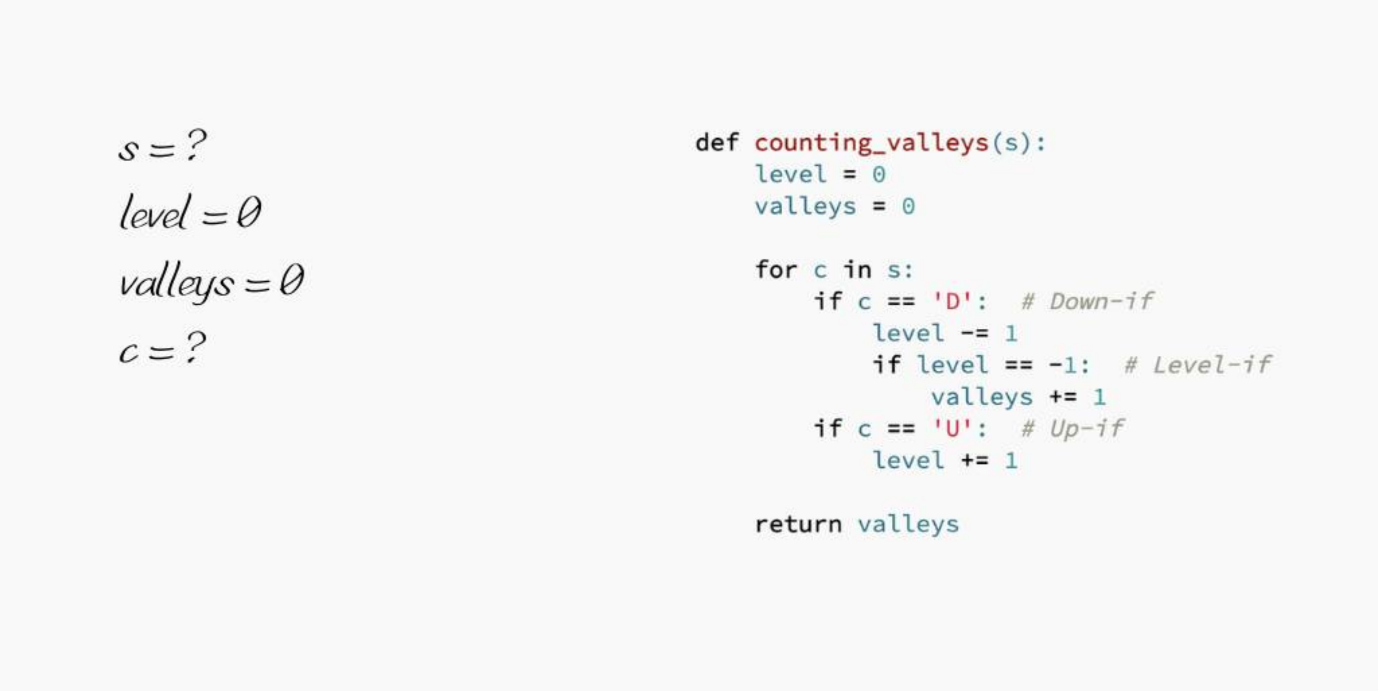
Here are **Interview-Style Coding Templates + How to Represent Code Efficiently in Interviews** (for Linked Lists, Trees, Graphs, DP, Everything).

This guide covers:

🔥 **1. Industry-standard templates**
🔥 **2. How to write clean code under interview pressure**

🔥 **3. How to represent linked list solutions clearly**
🔥 **4. Variable naming, spacing, structure tips**
🔥 **5. Patterns → template mapping**

🚀 INTERVIEW-STYLE CODE REPRESENTATION — MASTER CHEAT SHEET

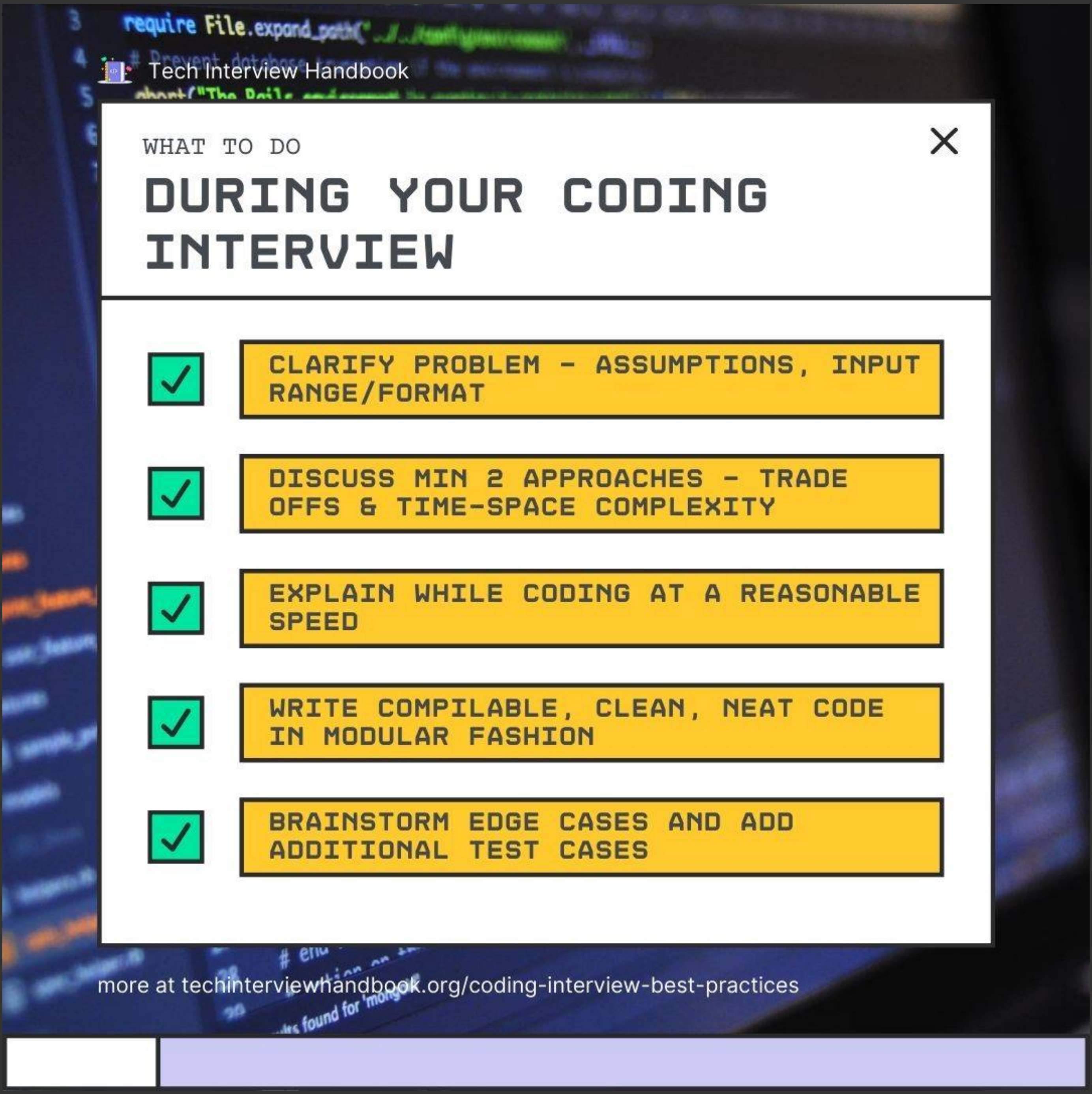


s = ?
level = 0
valleys = 0
c = ?

```
def counting_valleys(s):
    level = 0
    valleys = 0

    for c in s:
        if c == 'D':  # Down-if
            level -= 1
            if level == -1:  # Level-if
                valleys += 1
        if c == 'U':  # Up-if
            level += 1

    return valleys
```



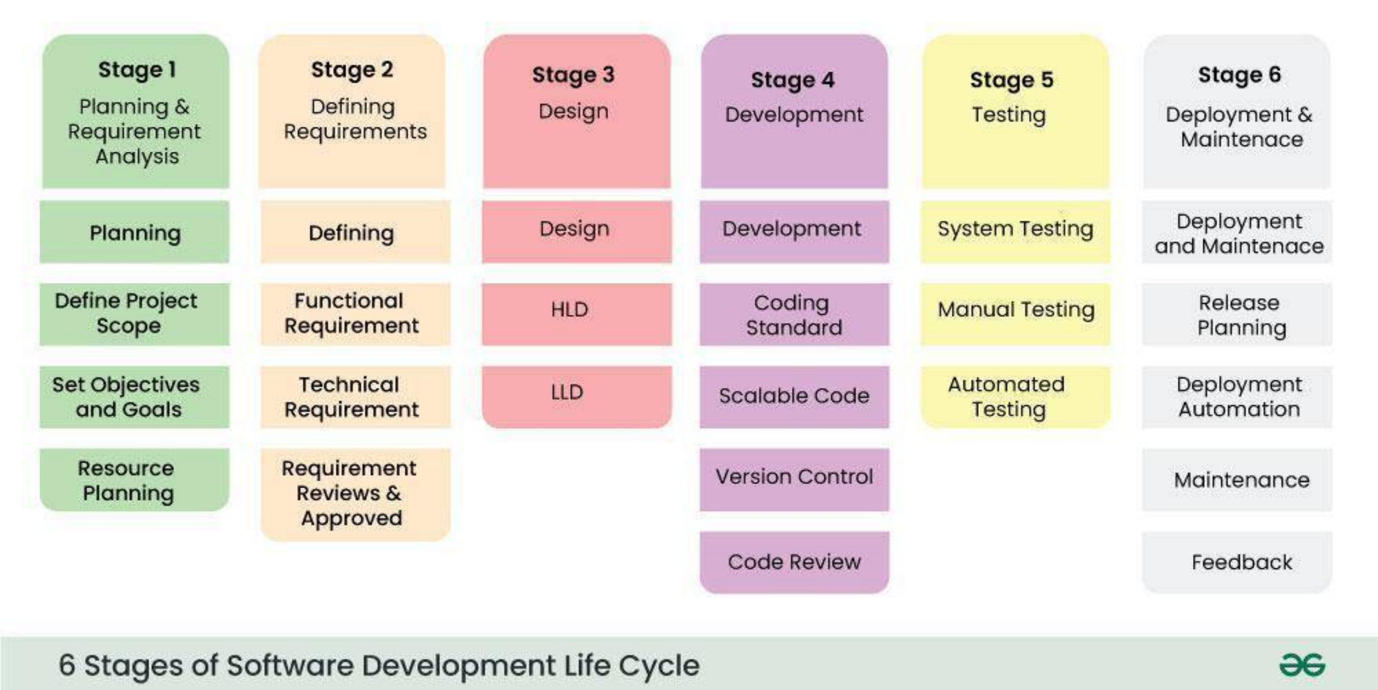
Tech Interview Handbook

WHAT TO DO

DURING YOUR CODING INTERVIEW

- ☑ CLARIFY PROBLEM – ASSUMPTIONS, INPUT RANGE/FORMAT
- ☑ DISCUSS MIN 2 APPROACHES – TRADE OFFS & TIME-SPACE COMPLEXITY
- ☑ EXPLAIN WHILE CODING AT A REASONABLE SPEED
- ☑ WRITE COMPILABLE, CLEAN, NEAT CODE IN MODULAR FASHION
- ☑ BRAINSTORM EDGE CASES AND ADD ADDITIONAL TEST CASES

more at techinterviewhandbook.org/coding-interview-best-practices



Stage 1 Planning & Requirement Analysis	Stage 2 Defining Requirements	Stage 3 Design	Stage 4 Development	Stage 5 Testing	Stage 6 Deployment & Maintenace
Planning	Defining	Design	Development	System Testing	Deployment and Maintenace
Define Project Scope	Functional Requirement	HLD	Coding Standard	Manual Testing	Release Planning
Set Objectives and Goals	Technical Requirement	LLD	Scalable Code	Automated Testing	Deployment Automation
Resource Planning	Requirement Reviews & Approved		Version Control		Maintenance
			Code Review		Feedback

6 Stages of Software Development Life Cycle

🧠 SECTION 1 — UNIVERSAL INTERVIEW CODING PRINCIPLES

✔️ Principle 1 — Start With a Minimal Node Definition

Avoid clutter. Keep class definitions short.

Linked List

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
```

Doubly LL

```python
class DLLNode:
    def __init__(self, val):
        self.val = val
        self.prev = None
        self.next = None
```

Never add unnecessary defaults (like `prev=None`) unless needed.

✔️ Principle 2 — Use Dummy Nodes (reduces bugs + makes code clean)

Interviewers *love* dummy nodes because they remove edge cases.

Example

```python
dummy = ListNode(0)
dummy.next = head
curr = dummy
```

✔️ Principle 3 — Use Short, Descriptive Pointer Names

Avoid `temp1`, `temp2`, `a`, `b`.

Better:

- `curr`
- `prev`
- `slow`, `fast`
- `left`, `right`
- `tail`

✔️ Principle 4 — Space Out Logic so Interviewer Can Read

Bad:

```python
while curr.next and curr.next.val!=x:curr=curr.next
curr.next=curr.next.next
```

Good:

```python
while curr.next and curr.next.val != x:
    curr = curr.next

curr.next = curr.next.next
```

✔️ Principle 5 — Always Separate:

(1) Find node

(2) Modify pointers

(3) Return statement

Interviewers appreciate clean segmentation.

✔️ Principle 6 — Always verbalize: “Am I modifying nodes or values?”

LL problems have two categories:

- Node manipulation
- Value manipulation

Node manipulation is harder → interviewers look for pointer safety.

✔️ Principle 7 — Use helper functions for repeated tasks

Example: reverse(), merge(), findMiddle()

It improves clarity and separates concerns.

🧩 SECTION 2 — LINKED LIST INTERVIEW TEMPLATES (CLEANEST FOR WHITEBOARD)

⭐ TEMPLATE A — Reverse Linked List (Iterative)

Important: Only 3 pointers. Clean indentation.

```python
def reverseList(head):
    prev = None
    curr = head

    while curr:
        nxt = curr.next
        curr.next = prev
        prev = curr
        curr = nxt

    return prev
```

⭐ TEMPLATE B — Fast–Slow Pointer (Middle + Cycle)

This is the gold standard representation.

```python
slow = head
fast = head

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

⭐ TEMPLATE C — Remove Nth From End (Elegant Version)

```python
def removeNthFromEnd(head, n):
    dummy = ListNode(0, head)
    slow = fast = dummy

    for _ in range(n):
        fast = fast.next

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

    slow.next = slow.next.next
    return dummy.next
```

⭐ TEMPLATE D — Merge Two Sorted Lists

Notice how clean pointer assignments are.

```python
def merge(l1, l2):
    dummy = ListNode(0)
    tail = dummy

    while l1 and l2:
        if l1.val < l2.val:
            tail.next, l1 = l1, l1.next
        else:
            tail.next, l2 = l2, l2.next
        tail = tail.next

    tail.next = l1 or l2
    return dummy.next
```

⭐ TEMPLATE E — Reverse Nodes in K-Group

Interviewers expect modular structure.

```python
def reverseKGroup(head, k):
    dummy = ListNode(0, head)
    groupPrev = dummy

    while True:
        kth = getKth(groupPrev, k)
        if not kth:
            break

        groupNext = kth.next
        prev, curr = kth.next, groupPrev.next

        while curr != groupNext:
            nxt = curr.next
            curr.next = prev
            prev = curr
            curr = nxt
        tmp = groupPrev.next
        groupPrev.next = kth
        groupPrev = tmp

    return dummy.next
```

⭐ TEMPLATE F — Doubly Linked List Insert/Delete (Interview-Ready)

Insert at Head

```python
def insert_head(self, val):
    node = DLLNode(val)

    if not self.head:
        self.head = self.tail = node
        return

    node.next = self.head
    self.head.prev = node
    self.head = node
```

Delete Node (O(1))

```python
def delete_node(self, node):
    if node.prev:
        node.prev.next = node.next
    else:
        self.head = node.next

    if node.next:
        node.next.prev = node.prev
    else:
        self.tail = node.prev
```

⭐ TEMPLATE G — Circular LL Traversal

Interviewers check if you can avoid infinite loops.

```python
def traverse(self):
    if not self.head:
        return

    curr = self.head
    while True:
        print(curr.val)
        curr = curr.next
        if curr == self.head:
            break
```

⭐ TEMPLATE H — LRU Cache (DLL + HashMap Skeleton)

This template alone gets you hired 😄.

```python
class Node:
    def __init__(self, key, val):
        self.key = key
        self.val = val
        self.prev = None
        self.next = None
```

```python
class LRUCache:
    def __init__(self, capacity):
        self.cap = capacity
        self.cache = {}      # key -> node
        self.head = Node(0, 0)
        self.tail = Node(0, 0)
        self.head.next = self.tail
        self.tail.prev = self.head
```

Add/remove/move operations are kept modular.

🎯 SECTION 3 — HOW TO REPRESENT CODE IN AN INTERVIEW

✔️ TIP 1 — Start with a Problem Skeleton

Example:

```markdown
To solve this, I'll:
1. Use fast/slow pointers to find middle
2. Reverse second half
3. Compare halves
4. Restore if required
```

Interviewer sees clarity before code.

✔️ TIP 2 — Write function signatures with clear parameters

Bad:

```python
def solve(h):
```

Good:

```python
def isPalindrome(head):
```

✔ TIP 3 — Modularize Reusable Logic

Always pull out utility functions:

- reverse()
- merge()
- findMiddle()
- getKth()

✔ TIP 4 — Keep pointer updates visually isolated

Like this:

```python
nxt = curr.next
curr.next = prev
prev = curr
curr = nxt
```

Spacing matches pointer movement.

✔ TIP 5 — Name segments

```python
# Step 1: Find kth node
# Step 2: Reverse group
# Step 3: Reconnect
```

This is HIGHLY rated by interviewers.

✔ TIP 6 — Never nest deeply

Avoid:

```python
while x:
    if y:
        if z:
```

Flatten logic. Use guard clauses.

✔️ TIP 7 — Always handle NULL cases first

Interviewers watch for this.

```python
if not head or not head.next:
    return head
```

✔️ TIP 8 — End with a clean return

No tricky nested returns.

🎁 BONUS: UNIVERSAL CLEAN TEMPLATE STRUCTURE (APPLY TO ANY LEETCODE)

```python
class Solution:
    def function(self, head):

        # 1. Edge cases
        if not head:
            return head

        # 2. Initialize pointers
        slow = head
        fast = head

        # 3. Core logic
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        # 4. Post-processing
        result = slow

        # 5. Return cleanly
        return result
```

📌 WANT NEXT?

I can generate:

🔥 **Linked List Interview Templates PDF**
🔥 **Binary Tree / DFS / BFS Interview Templates**
🔥 **DP Interview Templates (Top-Down + Bottom-Up)**
🔥 **Graphs Interview Templates**

Just tell me **which one you want next**!