# Linked List – Core Patterns Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Pattern | Core Idea | Mental Idea | Typical Scenarios |
| 1 | Traversal (Iterative / Recursive) | Traverse through nodes one by one (iteratively or recursively) to process or collect data. | Keep moving curr = curr.next until null — every node is visited exactly once. | Count nodes, Print list, Sum of values |
| 2 | Fast–Slow Pointer (Runner Technique) | Use two pointers at different speeds to find middle or detect cycles. | Let one pointer move faster to compare or meet another at a cycle/midpoint. | Find Middle Node, Detect Cycle, Palindrome Check |
| 3 | Reversal (In-Place) | Reverse the direction of links by iteratively changing next pointers. | Keep track of previous, current, next — flip one pointer at a time. | Reverse Linked List, Reverse Sublist, K-Group Reversal |
| 4 | Merge / Sort (Divide & Conquer) | Split the list in halves and merge sorted halves. | Use slow–fast split, then merge sorted halves like merge sort. | Merge Two Sorted Lists, Sort Linked List |
| 5 | Cycle Detection and Removal | Detect a loop using fast–slow pointers, then remove it by pointer tracing. | If two pointers meet, loop exists; then find loop start to remove it. | Detect and Remove Loop, Find Cycle Start |
| 6 | Intersection Detection | Use length difference or hash set to detect common node between two lists. | Align both lists from equal distance to tail; move together until they meet. | Intersection of Two Linked Lists |
| 7 | Palindrome Check | Use fast–slow pointer to reach mid, reverse second half, compare both halves. | Find mid, reverse second half, then check equality. | Palindrome Linked List |
| 8 | Clone with Random Pointers | Use a map or interleaving technique to clone a list with additional random links. | Create a mapping between original and copied nodes before linking random pointers. | Copy List with Random Pointer |
| 9 | Segregation / Partitioning | Split and merge nodes based on a condition (value < pivot). | Maintain dummy heads for each partition and connect them at the end. | Partition List, Odd–Even Linked List |
| 10 | K-Group / Segment Reversal | Reverse linked list in fixed-sized segments. | Reverse in blocks of size K; connect each reversed part correctly. | Reverse Nodes in K-Group, Pairwise Swap Nodes |
| 11 | Recursive Decomposition (Backtracking Style) | Process nodes during unwinding of recursion, useful for symmetric problems. | Let recursion handle forward motion; logic executes during backtrack. | Add Two Numbers II, Palindrome Check (recursive) |