PROBLEMS ON LINKED LIST:

1) Find Middle element of a singly linked list in one pass.

WTD: Use two pointers, one moving twice as fast as the other, to find the middle element in a single pass.

(e.g.: I/P: 1->2->3->4->5; O/P: 3)

2) Find the length of a singly linked list.

WTD: Traverse the list from head to tail, incrementing a counter to find its length.

(e.g.: I/P: 1->2->3->4; O/P: 4)

3) Reverse a linked list.

WTD: Traverse the list while reversing the next pointers of each node.

(e.g.: I/P: 1->2->3; O/P: 3->2->1)

4) Reverse a singly linked list without recursion.

WTD: Use iterative method to reverse the next pointers of each node.

(e.g.: I/P: 1->2->3; O/P: 3->2->1)

5) Remove duplicate nodes in an unsorted linked list.

WTD: Use a hash table to record the occurrence of each node while traversing the list to remove duplicates.

(e.g.: I/P: 1->2->2->3; O/P: 1->2->3)

6) Find nth node from end of a singly linked list.

WTD: Use two pointers, move one n nodes ahead, then move both until the first one reaches the end.

(e.g.: I/P: 1->2->3->4 (n=2); O/P: 3)

7) Move last element to the front of a given linked list.

WTD: Find the last node and its previous node, change their pointers to move the last node to the front.

(e.g.: I/P: 1->2->3->4; O/P: 4->1->2->3)

8) Delete alternate nodes of a linked list.

WTD: Traverse the list and remove every alternate node.

(e.g.: I/P: 1->2->3->4; O/P: 1->3)

9) Pairwise swap elements of a linked list.

WTD: Swap every two adjacent nodes by adjusting their pointers.

(e.g.: I/P: 1->2->3->4; O/P: 2->1->4->3)

10) Check if a given linked list contains a cycle and what would be the starting node?

WTD: Use Floyd's cycle-finding algorithm to detect the cycle and then find its starting node.

(e.g.: I/P: 1->2->3 (3 points back to 1); O/P: True)

11) Intersection point of two linked lists.

WTD: Use two pointers, one for each list, and traverse to find the intersection point.

(e.g.: I/P: 1->2->3 & 4->5->3; O/P: 3)

12) Segregate even and odd nodes in a linked list.

WTD: Use two pointers to rearrange nodes such that all even and odd elements are together.

(e.g.: I/P: 1->2->3->4; O/P: 2->4->1->3)

13) Merge two sorted linked lists.

WTD: Use a temporary dummy node to hold the sorted list, compare each node and attach the smaller one to the dummy.

(e.g.: I/P: 1->3->5 & 2->4->6; O/P: 1->2->3->4->5->6)

14) Add two numbers represented by linked lists.

WTD: Traverse both lists, sum the corresponding nodes, and manage the carry.

(e.g.: I/P: 2->4 & 5->6 (24 + 56); O/P: 8->0)

15) Find sum of two linked list using stack.

WTD: Use two stacks to hold the numbers from each list, then pop and sum them, storing the result in a new list.

(e.g.: I/P: 2->4 & 5->6 (24 + 56); O/P: 8->0)

16) Compare two strings represented as linked lists.

WTD: Traverse both lists, comparing each node's value. If they are equal throughout, the lists are equal.

(e.g.: I/P: 'a'->'b'->'c' & 'a'->'b'->'c'; O/P: Equal)

17) Clone a linked list with next and random pointer.

WTD: Create a deep copy of the linked list including the random pointers using a hash table to map original nodes to their copies.

(e.g.: I/P: 1->2->3 (random pointers set randomly); O/P: Cloned list with same structure and random pointers)

18) Merge sort on a linked list.

WTD: Implement the Merge Sort algorithm on a linked list, splitting the list into halves and merging them back in sorted order.

(e.g.: I/P: 3->1->2; O/P: 1->2->3)

19) Detect and remove loop in a linked list.

WTD: Use Floyd's algorithm to detect the loop and then remove it by setting the next pointer of the last node in the loop to NULL.

(e.g.: I/P: 1->2->3 (3 points back to 1); O/P: 1->2->3)

20) Flatten a multi-level linked list.

WTD: Use a stack or recursion to flatten the list so that all nodes are at the same level.

(e.g.: I/P: 1->2->3 (2 has child 4->5); O/P: 1->2->4->5->3)

21) Partition a linked list around a given value.

WTD: Traverse the linked list, creating two separate lists - one for values less than the partition value and another for values greater than or equal to the partition value. Finally, merge these lists.

(e.g.: I/P: 1->4->3->2->5->2, Partition Value: 3; O/P: 1->2->2->4->3->5)

22) Remove all nodes in a linked list that have a specific value.

WTD: Traverse the linked list and remove any node that has a value matching the specified value. Make sure to properly update the next pointers and free any removed nodes.

(e.g.: I/P: 1->2->6->3->4->5->6, Value to Remove: 6; O/P: 1->2->3->4->5)

23) Convert a binary number represented by a linked list to an integer.

WTD: Traverse the linked list and convert the binary number represented by the linked list nodes to an integer. Use bit manipulation for the conversion.

(e.g.: I/P: 1->0->1; O/P: 5)

24) Find the common ancestor of two nodes in a binary tree represented as a doubly linked list.

WTD: Traverse the binary tree represented as a doubly linked list and find the common ancestor of the given two nodes. Utilize parent pointers in the doubly linked list to backtrack.

(e.g.: I/P: Nodes 6 and 9 in Binary Tree 4->5->6->7->8->9; O/P: 7)

25) Determine if a linked list is a palindrome.

WTD: Use a slow and fast pointer to find the middle of the list. Reverse the second half and compare it with the first half to determine if the linked list is a palindrome.

(e.g.: I/P: 1->2->2->1; O/P: True)