1. **Reverse a Linked List (both Iterative and Recursive)**

**//Iterative**

**ListNode\* reverseList(ListNode\* head) {**

**ListNode \*curr=head,\*prev = NULL,\*next;**

**while(curr!=NULL)**

**{**

**next = curr->next;**

**curr->next = prev;**

**prev = curr;**

**curr = next;**

**}**

**return prev;**

**}**

**//recursive**

**ListNode\* reverseList(ListNode\* head) {**

**if(head == NULL or head->next == NULL) return head;**

**ListNode \*newhead = reverseList(head->next); //down down down**

**head->next->next = head; // reverse**

**head->next = NULL; //reverse**

**return newhead; //up**

**}**

1. **Reverse a linked list in a group of given size**
2. **Detect a cycle in linked list**

**bool hasCycle(ListNode \*head) {**

**if(head == NULL) return false;**

**ListNode \*slow = head, \*fast = head;**

**while(fast->next != NULL && fast != NULL)**

**{**

**slow = slow->next;**

**fast = fast->next->next;**

**if(slow == fast) return true;**

**}**

**return false;**

**}**

1. **Return the starting point of loop**

**ListNode \*detectCycle(ListNode \*head) {**

**if(head == NULL) return NULL;**

**ListNode\*slow=head,\*fast=head;**

**bool iscycle = false;**

**while(fast != NULL && fast->next!=NULL)**

**{**

**slow = slow->next;**

**fast = fast->next->next;**

**if(slow == fast){**

**iscycle = true;**

**break;**

**}**

**}**

**if(!iscycle) return NULL;**

**slow = head;**

**while(slow != fast)**

**{**

**slow = slow->next;**

**fast = fast->next;**

**}**

**return slow;**

**}**

1. **Delete Node from LL**

**/\***

**while(fast->next != slow)**

**fast = fast->next;**

**fast->next = nullptr;**

**\*/**

1. **Remove Duplicates in sorted Linked List**

**ListNode\* deleteDuplicates(ListNode\* head) {**

**ListNode \*first=head;**

**while(first!=NULL && first->next!=NULL){**

**if(first->val==first->next->val){**

**first->next=first->next->next;**

**}else{**

**first=first->next;**

**}**

**}**

**return head;**

**}**

1. **Remove duplicates from unsorted list**

**void removeDuplicates(struct Node \*start)**

**{**

**// Hash to store seen values**

**unordered\_set<int> seen;**

**/\* Pick elements one by one \*/**

**struct Node \*curr = start;**

**struct Node \*prev = NULL;**

**while (curr != NULL)**

**{**

**// If current value is seen before**

**if (seen.find(curr->data) != seen.end())**

**{**

**prev->next = curr->next;**

**delete (curr);**

**}**

**else**

**{**

**seen.insert(curr->data);**

**prev = curr;**

**}**

**curr = prev->next;**

**}**

**}**

1. **Swap first and last node of linked list**

**void moveToFront(Node \*\*head\_ref)**

**{**

**if (\*head\_ref == NULL || (\*head\_ref)->next == NULL)**

**return;**

**/\* Initialize second last and last pointers \*/**

**Node \*secLast = NULL;**

**Node \*last = \*head\_ref;**

**/\*After this loop secLast contains address of second last node and**

**last contains address of last node in Linked List \*/**

**while (last->next != NULL)**

**{**

**secLast = last;**

**last = last->next;**

**}**

**/\* Set the next of second last as NULL \*/**

**secLast->next = NULL;**

**/\* Set next of last as head node \*/**

**last->next = \*head\_ref;**

**/\* Change the head pointer to point to last node now \*/**

**\*head\_ref = last;**

**}**