1. Delete node in a linked list

```
void deletenode(ListNode *node) {
    if(node->next == NULL) {
        delete node;
    }
    swap(node->val, node->next->val);
    ListNode *temp = node->next;
    node->next = node->next->next;
    delete temp;
}
```

Time Complexity - O(n)
Space Complexity - O(1)

2. Remove linked list elements

```
ListNode *remove_elements(ListNode *head, int target) {
    while(head != NULL) {
        if(head->val == target) {
            head = head->next;
        }
        else {
            break;
        }
    }
    if(head == NULL) {
        return head;
    }
    ListNode *temp = head;
    while(temp->next != NULL) {
        if(temp->next->val == target) {
            temp->next = temp->next->next;
        }
        else {
            temp = temp->next;
        }
    }
    return head;
}
```

Time Complexity - O(n)
Space Complexity - O(1)

3. Merge two sorted linked lists

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if(list1 == NULL) {
        return list2;
    }
    if(list2 == NULL) {
        return list1;
    }
    if(list1->val <= list2->val) {
        list1->next = mergeTwoLists(list1->next, list2);
        return list1;
    }
    else {
        list2->next = mergeTwoLists(list2->next, list1);
        return list2;
    }
}
```

Time Complexity - O(nlogn)
Space Complexity - O(n)

4. Linked list cycle II

```
ListNode *detectCycle(ListNode *head) {
    if (head == NULL || head->next == NULL) {
        return NULL;
    }
    ListNode *slow  = head, *fast = head, *entry = head;
    while (fast->next && fast->next->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) {
            while(slow != entry) {
                slow  = slow->next;
                entry = entry->next;
            }
            return entry;
        }
    }
    return NULL;
}
```

Time Complexity - O(n)

Space Complexity - O(1)

5. Remove nth node from the end of the linked list

```
ListNode* removeNthFromEnd(ListNode* head, int n) {
    if(!head){
        return NULL;
    }
    ListNode *temp = new ListNode(),* slow = temp, *fast = temp;
    temp->next = head;
    for(int i = 0; i < n; i++) {
        fast = fast->next;
    }
    while(fast->next != NULL) {
        slow = slow->next;
        fast = fast->next;
    }
    slow->next=slow->next->next;
    return temp->next;
}
```

Time Complexity - O(n)
Space Complexity - O(1)

6. Given a singly linked list of size N. The task is to left-shift the linked list by k nodes, where k is a given positive integer smaller than or equal to the length of the linked list.

```
Input: N = 5
value[] = {2, 4, 7, 8, 9}
k = 3
Output: 8 9 2 4 7
Explanation:Rotate 1:4 -> 7 -> 8 -> 9 -> 2
Rotate 2: 7 -> 8 -> 9 -> 2 -> 4
Rotate 3: 8 -> 9 -> 2 -> 4 -> 7
```

```
ListNode* rotate_list(ListNode* head, int k) {
    if(!head) {
        return head;
    }
    int size = 1;
    ListNode *temp = head;
    while(temp->next){
        temp = temp->next;
        size++;
    }
```

```
    k = size - (k % size);
    temp->next = head;
    while(k--) {
        temp = temp->next;
    }
    head = temp->next;
    temp->next = NULL;
    return head;
}
```

Time Complexity - O(n)
Space Complexity - O(1)

7. Given the head of a linked list, we repeatedly delete consecutive sequences of nodes that sum to 0 until there are no such sequences. After doing so, return the head of the final linked list. You may return any such answer. (Note that in the examples below, all sequences are serializations of ListNode objects.)

```
Input: head = [1,2,-3,3,1]
Output: [3,1]
Note: The answer [1,2,1] would also be accepted.
```

```
ListNode* remove_zero_sum_sublists(ListNode* head) {
    ListNode *dummy = new ListNode(0);
    dummy->next = head;
    ListNode *curr = dummy;
    while (curr) {
        ListNode *start = curr->next;
        int sum = 0;
        while (start) {
            sum += start->val;
            if (sum == 0) {
                curr->next = start->next;
                break;
            }
            start = start->next;
        }
        if(!start) {
            curr = curr->next;
        }
    }
    return dummy->next;
}
```

Time Complexity - O(n)

Space Complexity - O(n)

8. Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list. The first node is considered odd, and the second node is even, and so on. Note that the relative order inside both the even and odd groups should remain as it was in the input. You must solve the problem in O(1) extra space complexity and O(n) time complexity.

```
Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]
```

```
ListNode* odd_even_list(ListNode* head) {
    if(head == NULL || head->next == NULL) {
        return head;
    }
    ListNode* odd = head, *even = head->next, *even_head = even;
    while(even != NULL && even->next != NULL) {
        odd->next = even->next;
        even->next = even->next->next;
        odd->next->next = even_head;
        odd = odd->next;
        even = even->next;
    }
    return head;
}
```

Time Complexity - O(n)
Space Complexity - O(1)