1. Merge Two Sorted linked list and sort it

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode temp1 = list1;
        ListNode temp2 = list2;
        ListNode temp3 = new ListNode(0);  // Dummy node
        ListNode current = temp3;  // Pointer to form the new list

        if (list1 == null) {
            return list2;
        }
        if (list2 == null) {
            return list1;
        }
        while (temp1 != null && temp2 != null) {
            if (temp1.val <= temp2.val) {
                current.next = temp1;
                temp1 = temp1.next;
            } else {
                current.next = temp2;
                temp2 = temp2.next;
            }
            current = current.next;
        }
        // Append the remaining nodes of temp1 or temp2
        if (temp1 != null) {
            current.next = temp1;
        } else if (temp2 != null) {
            current.next = temp2;
        }
        return temp3.next;  // Return the merged list, skipping the dummy node
    }
}
```

1. Rotate List

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

Output: [4,5,1,2,3]

```
class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if(k<=0 || head==null || head.next==null){
```

```
        return head;
      }
      ListNode last=head;
      int length=1;
      while(last.next!=null){
        last=last.next;
        length++;
      }
      last.next=head;
      int rotation=k%length;
      int skip=length-rotation;
      ListNode newLast=head;
      for(int i=0;i<skip-1;i++){
        newLast=newLast.next;
      }
      head=newLast.next;
      newLast.next=null;
      return head;
    }
}
```

1. Remove Duplicates

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
      if (head == null) {
        return head;
      }
      ListNode node = head;
      while (node != null && node.next != null) {
        if (node.val == node.next.val) {
          node.next = node.next.next;
        } else {
          node = node.next;
        }
      }
      return head;
    }
}
```

1. Reverse LL II

Input: head = [1,2,3,4,5], left = 2, right = 4

Output: [1,4,3,2,5]

```
class Solution {
    public ListNode reverseBetween(ListNode head, int left, int right) {
        if (head == null) return head;
        // Create a dummy node to handle edge cases where the head is part of the reversed section
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode prev = dummy;
        // Move prev to the node right before the section to reverse
        for (int i = 1; i < left; i++) {
            prev = prev.next;
        }
        // The start node of the section to reverse
        ListNode start = prev.next;
        ListNode then = start.next;
        // Reverse the sublist between left and right
        for (int i = 0; i < right - left; i++) {
            start.next = then.next;
            then.next = prev.next;
            prev.next = then;
            then = start.next;
        }
        return dummy.next;
    }
}
```

1. Linked List Cycle

https://assets.leetcode.com/uploads/2018/12/07/circularlinkedlist.png

Input: head = [3,2,0,-4], pos = 1

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

```
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode fast=head;
        ListNode slow=head;
        while(fast!=null && fast.next!=null){
            fast=fast.next.next;
            slow=slow.next;
            if(fast==slow){
                return true;
```

```
        }
      }
      return false;
    }
}
```

1. Sort List

Input: head = [4,2,1,3]

Output: [1,2,3,4]

```
class Solution {
    public ListNode sortList(ListNode head) {
        if(head==null || head.next==null) return head;
        ListNode mid=findMid(head);
        ListNode left=head, right=mid.next;
        mid.next=null;
        left=sortList(left);
        right=sortList(right);
        return marge(left,right);
    }
    ListNode findMid(ListNode head){
        ListNode i=head, j=head.next.next;
        while(j!=null && j.next!=null){
            i=i.next;
            j=j.next.next;
        }
        return i;
    }
    ListNode marge(ListNode left, ListNode right){
        ListNode dummy=new ListNode(-1);
        ListNode temp=dummy;
        while(left!=null && right!=null){
            if(left.val<right.val){
                temp.next=left;
                left=left.next;
            }
            else{
                temp.next=right;
                right=right.next;
            }
            temp=temp.next;
```

```
        }
        if(left!=null) temp.next=left;
        else temp.next=right;
        return dummy.next;
    }
}
```

1. Reverse Linked List

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode pres = head;
        ListNode next = null;
        while (pres != null) {
            next = pres.next;
            pres.next = prev;
            prev = pres;
            pres = next;
        }
        return prev;
    }
}
```

1. Middle of a LL

```
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode fast=head;
        ListNode slow=head;
        ListNode ans=slow;
        while(fast!=null && fast.next!=null){
            fast=fast.next.next;
            slow=slow.next;
        }
        ans=slow;
        return ans;
    }
}
```

1. Delete middle node of a Linked List

```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if (head == null || head.next == null) {
```

```
        return null;
    }
    ListNode fast = head;
    ListNode slow = head;
    ListNode prev = null;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        prev = slow;
        slow = slow.next;
    }
    if (prev != null) {
        prev.next = slow.next;
    }
    return head;
  }
}
```