DAY 3

Reverse Linked List

```
   ListNode* reverseList(ListNode* head) {

       //Curr node = head
       ListNode* prev = NULL, * curr = head;
       //Curr node value is there
       while (curr) {
           // NextNode = stored
           ListNode* nextNode = curr -> next;
           // Curr next to be pointed to prev
           curr -> next = prev;
           // Change the prev to current
           prev = curr;
           // Set the front to nextNode;
           curr = nextNode;
       }
       return prev;
   }
```

Middle of the Linked List

```
ListNode* middleNode(ListNode* head) {

       ListNode *temp = head;
       int count = 0;
       while(temp){
           count ++;
           temp = temp -> next;
       }
       if(count < 2)    return head;
       count = (count / 2);
       temp = head;
```

```
        while(count --){
            temp = temp -> next;
        }
        return temp;
    }
```

Merge Two Sorted Lists

```
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode * p1 = list1, *p2 = list2;
        ListNode * res = new ListNode(-1), *answer = res;
        while(p1 and p2){
            if(p1 -> val == p2 -> val){
                answer -> next = new ListNode(p1 -> val);
                answer = answer -> next;
                answer -> next = new ListNode(p1 -> val);
                answer = answer -> next;
                p1 = p1 -> next;
                p2 = p2 -> next;
            }else if(p1 -> val > p2 -> val){
                answer -> next = new ListNode(p2 -> val);
                answer = answer -> next;
                p2 = p2 -> next;
            }else{
                answer -> next = new ListNode(p1 -> val);
                answer = answer -> next;
                p1 = p1 -> next;
            }
        }
        while(p1){
            answer -> next = new ListNode(p1 -> val);
            answer = answer -> next;
            p1 = p1 -> next;
        }
        while(p2){
            answer -> next = new ListNode(p2 -> val);
            answer = answer -> next;
            p2 = p2 -> next;
        }
        return res -> next;
    }
```

Remove Nth Node From End of List

Optimal

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

100% runtime and 91% memory solution(**hacky solution**)

```
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode *temp = head;
        if(!head or !head -> next){
            return NULL;
        }
        int count = 0;
        while(temp){
            count ++;
            temp = temp -> next;
        }
        if(count - n == 0){
            head = head -> next;
            return head;
        }
        int i = 0;
        temp = head;
        while(i < count - n - 1){
            temp = temp -> next;
            i++;
        }

        if(temp -> next){
            temp -> next = temp -> next -> next;
        }
        return head;
    }
```

Add Two Numbers

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* answer = new ListNode(-1), *temp = answer;
        int carry = 0;
        while (l1 or l2 or carry) {
            int sum = carry;
            if (l1) {
                sum += l1 -> val;
                l1 = l1 -> next;
            }
            if (l2) {
                sum += l2 -> val;
                l2 = l2 -> next;
            }
            carry = sum / 10;
            temp -> next = new ListNode(sum % 10);
            temp = temp -> next;
        }
        return answer -> next;
```

```
    }
```

Delete Node in a Linked List

```
void deleteNode(ListNode* node) {
        node -> val = node -> next -> val;
        node -> next = node -> next -> next;
    }
```