DSA SERIES

- Learn Coding

Topic to be Covered today

Linked List Leetcode Problems

LETS START TODAY'S LECTURE

Convert Binary Number in a Linked List to Integer (1290)

```
class Solution {
public:

    ListNode* reverse(ListNode* head){
        ListNode* prev = NULL;
        ListNode* curr = head;
        ListNode* forward = NULL;

        while(curr!=NULL){
            forward = curr->next;
            curr->next = prev;
            prev = curr;
            curr= forward;
        }
        return prev;
    }

    int getDecimalValue(ListNode* head) {

        ListNode* newHead = reverse(head);
```

```
        int ans = 0;
              int P = 0;

              while(newHead!=NULL){
                  ans += newHead -> val * pow(2,P);
                  P++;
                  newHead = newHead->next;
              }

              return ans;

          }
      };
```

Finding middle node of the linked list(876)

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        if(head == NULL || head-> next==NULL){
            return head;
        }

        ListNode* slow = head;
        ListNode* fast = head;

        while(fast !=NULL && fast->next!=NULL){
            slow = slow->next;
            fast = fast ->next->next;
        }
        return slow;
    }
};
```

Linked List Cycle I (141)

```
class Solution {
public:
    bool hasCycle(ListNode *head) {

        if(head == NULL) return false;

        ListNode* slow = head;
        ListNode* fast = head;

        while(fast!=NULL && fast->next!=NULL){
            slow = slow->next;
            fast = fast->next->next;

            if(slow==fast){
                return true;
            }
        }
        return false;
    }
};
```

Palindrome linked list (234)

```
class Solution {
public:
    ListNode* getMiddle(ListNode* head){
        ListNode* slow = head;
        ListNode* fast = head->next;

        while(fast!=NULL && fast->next!=NULL){
            fast=fast->next->next;
            slow = slow->next;
        }

        return slow;
    }
```

```
ListNode* reverse(ListNode* head){
    ListNode* curr = head;
    ListNode* prev = NULL;
    ListNode* forward = NULL;

    while(curr != NULL){
        forward = curr->next;
        curr->next   = prev;
        prev =  curr;
        curr=forward;
    }
    return prev;

}

bool isPalindrome(ListNode* head) {

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

```
        ListNode* mid = getMiddle(head);

              ListNode* temp = mid->next;
              mid->next = reverse(temp);

              ListNode* head1 = head;
              ListNode* head2 = mid->next;

              while(head2!=NULL){
                  if(head1->val != head2->val){
                      return false;
                  }

                  head1 = head1->next;
                  head2 = head2->next;
              }

              return true;
          }
      };
```

Remove duplicates from the sorted linked list(83)

```
Node* removeDuplicates(Node* head) {

        Node *curr = head;

        while(curr!=NULL && curr->next !=NULL){
            if(curr->data == curr->next->data){
                Node* duplicate = curr->next;
                curr->next = curr->next->next;

                delete duplicate;
            } else{
                curr = curr->next;
            }
        }

        return head;

}
```

Add one to a linked list number (GFG)

```
class Solution {
  public:

  Node* reverse(Node* head){
      Node* prev = NULL;
      Node* curr = head;
      Node* forward = NULL;

      while(curr!=NULL){
          forward = curr->next;
          curr->next=prev;
          prev=curr;
          curr=forward;
      }

      return prev;
  }
```

```
    Node* addOne(Node* head) {
        head = reverse(head);

        Node* curr = head;
        int carry = 1;
        while(curr && carry){
            int sum = curr->data + carry;
            curr->data = sum%10;
            carry= sum/10;

            if(!curr->next && carry){
                curr->next = new Node(0);
            }
            curr=curr->next;
        }
        return reverse(head);


    }
};
```

Learn coding

THANK YOU