DAY-30

(1) Search element in a Linked list

(i) Iterative solution:-

→ initialize a node pointer, current = head.

→ Do following while current is not NULL.

⇒ current → key is equal to the key being searched return ~~true~~ true.

⇒ current = current → next

→ Return false.

for search Implementation part:-

```
def search(self, n):
    current = self.head
    while current != None:
        if current.data == n:
            return True
        current current.next
    return False.
```

(ii) Recursive Solution :-

bool search (head, x)
→ if head is NULL,
return False.
→ if head's key is same as x,
return true
→ Else,
return search (head->, x)

code part → implementation :-

```
...
def search (~~list~~ self, llist, key):
    if (not llist ):
        return False
    if (li.data == key ):
        return True.
    return self.search(llist.next, key)
```

(2). Find middle of a linked list.

(i) Traverse the whole linked list and count the number of nodes.

Now, traverse the list again, and return the node at count/2.

(ii) → Traverse the linked list using two pointers.
→ move one pointer by one and another by two.
→ when the fast pointer (second one) reached end
- slow pointer will reach middle of the linked list.

code part - implementation

```
def middle(self):
    slow_ptr = self.head
    fast_ptr = self.head
    if self.head is not None:
        while (fast_ptr is not None and
               fast_ptr.next is not None):
            fast_ptr = fast_ptr.next.next
            slow_ptr = slow_ptr.next
        return slow_ptr.data
```

(iii) → initialize the temp var. as head.
→ initialize count to Zero.
→ Take till head will become Null (means the end of the list) and ~~increment~~ increment the temp node when count is odd only, in this way *temp will traverse till mid element and *head will traverse all linked list.
→ Print the data of temp.

Code part - implementation.

```
def Middle(self):
    temp = self.head
    count = 0
    while self.head:
        if (count & 1): # if count is
            temp = temp.next          # odd.
        self.head = self.head.next
        count += 1
    return( temp.data )
```