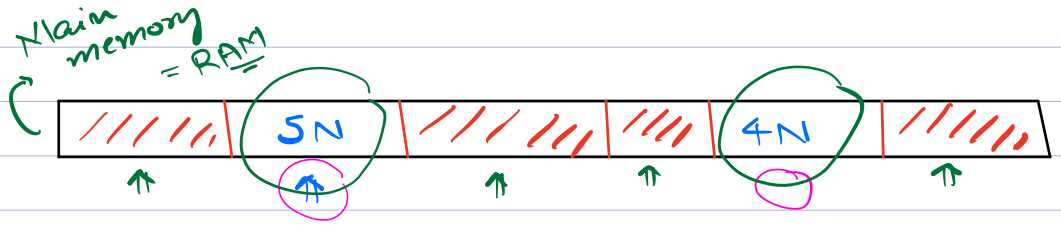
Array ⇒ Integer array of size N

4 Bytes × N = 4N Bytes.



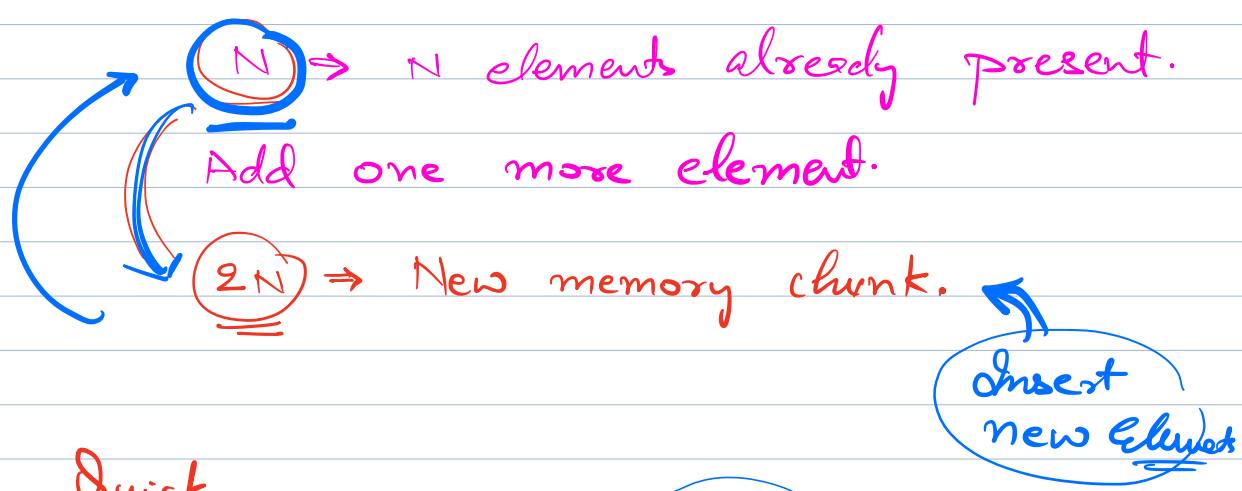
1) Integer array of size N = 4N ✓

2) Integer array of size 2N = 8N Bytes.

9N Bytes available <

Size ✓

Dynamic arrays ⇒ continuous.



Quick

Random access ⇒ O(1) ⇒ Arrays.

Search ⇒ Key ⇒ Entire List of values.

⇒ Optimise on memory.

Linked List

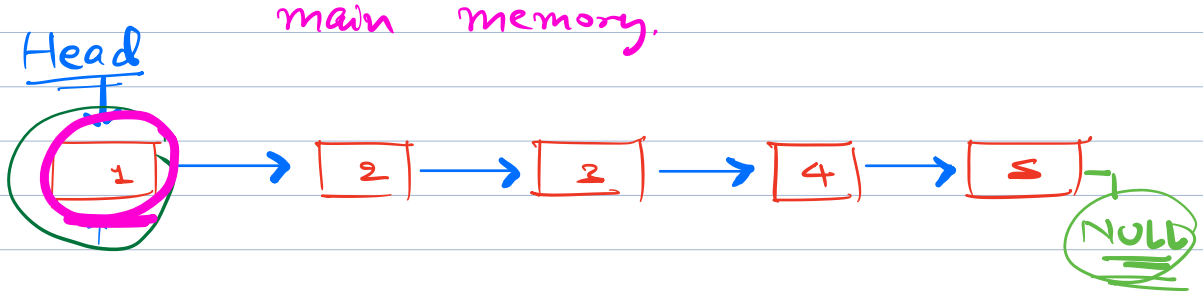
⇒ Linear DS

⇒ Not stored in contiguous memory locations.

List of Nodes.

LL of 5 elements.

Assume 1 node takes 1 block in main memory.



⇒ Store the pointer of each node. ✗

Structure of a Node

1) Data
2) Address / Reference of next node.

```
class Node {
    int data;
    Node next;
}
```

C/C++

```
struct Node {
    int data;
    Node* next;
};
```

```
Node node1 = new Node();
node1.data;
node1.next;
```

Q Given an integer array of size N.
Create a Linked List & return head.

```
Node head = new Node();
head.data = A[0];

Node temp = head;
```

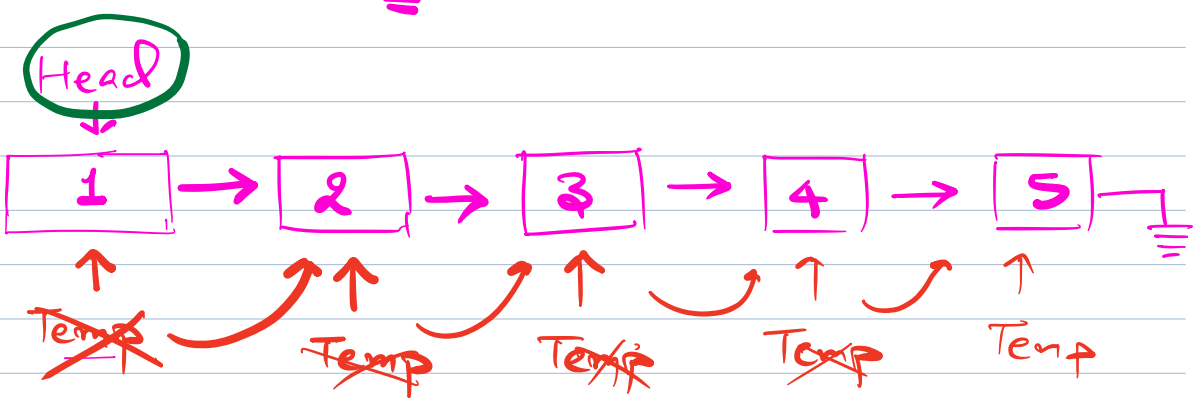
⇒ for (i=1; i<N; i++){

temp.next = new Node()

temp = temp.next;
temp.data = Ali;
temp.next = NULL;

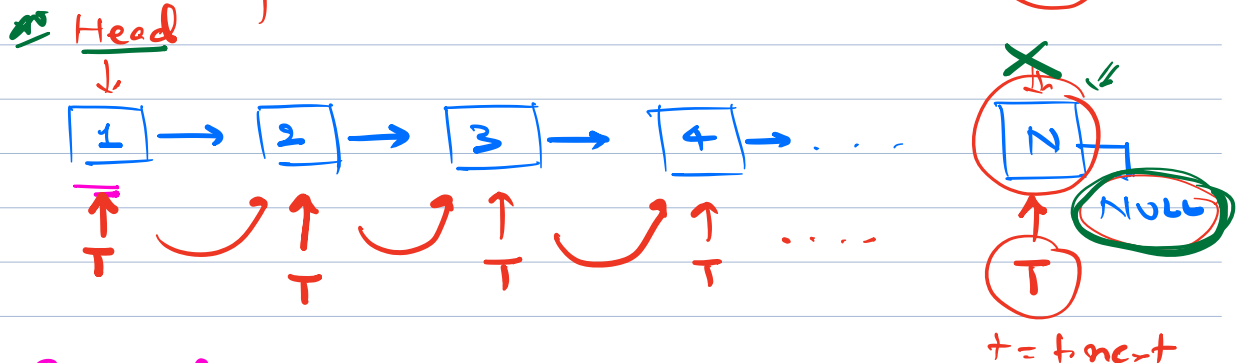
}

A: 1, 2, 3, 4, 5 (i: 0, 1, 2, 3, 4)



Basic Operations on LL

356



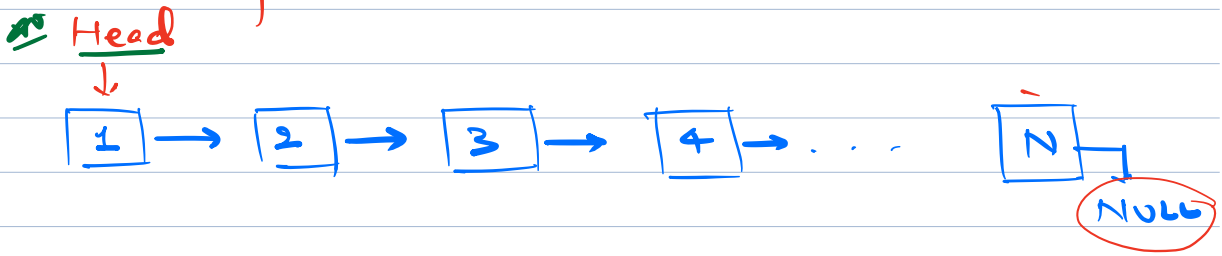
1) Search ⇒ O(N)

K

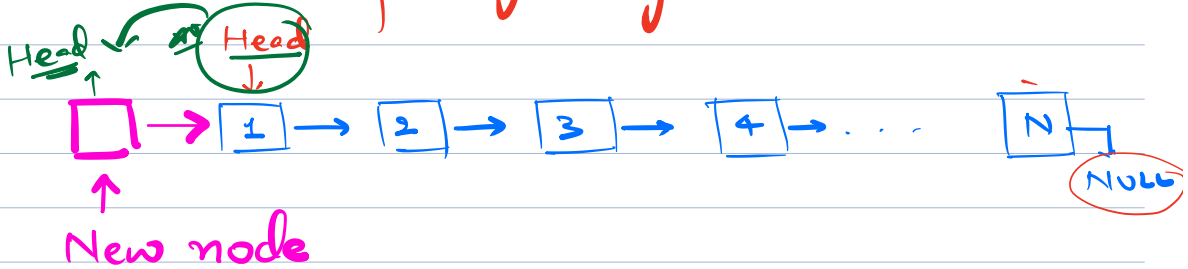
temp = head;

```
while ( temp != NULL ) {
    if ( temp.data == k){
        return true;
    }
    temp = temp.next;
}
return false;
```

2) Insert



(i) Insert at beginning.

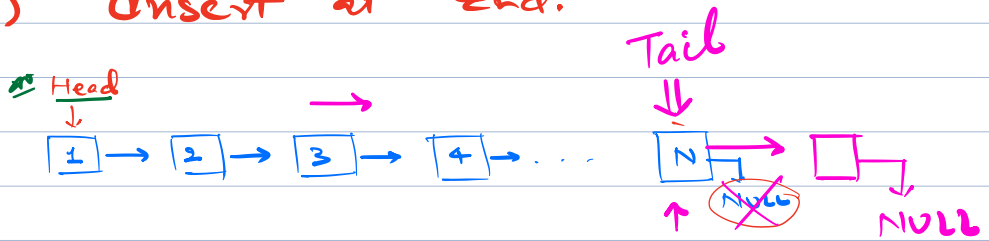



```
Node newNode = new Node()
newNode.data = ///;

newNode.next = Head;
```

Head = new Node;

T.C. = O(1)

(ii) Insert at End.




```
temp = head;
while ( temp.next != null) {
        temp = temp.next;
}
temp.next = new Node();
temp = temp.next();
temp.next = NULL;
```

T.C. = O(N)

⇓

O(1) using tail pointer.

H.W. (iii) Insert at K^{th} position from

start.

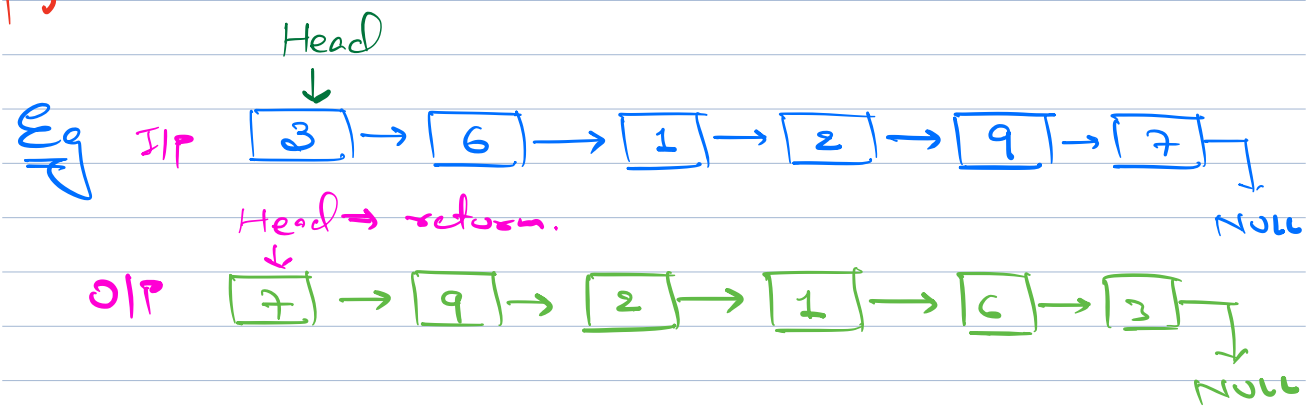
Q Find the length of the LL.

Q Reverse a given LL & return the head.

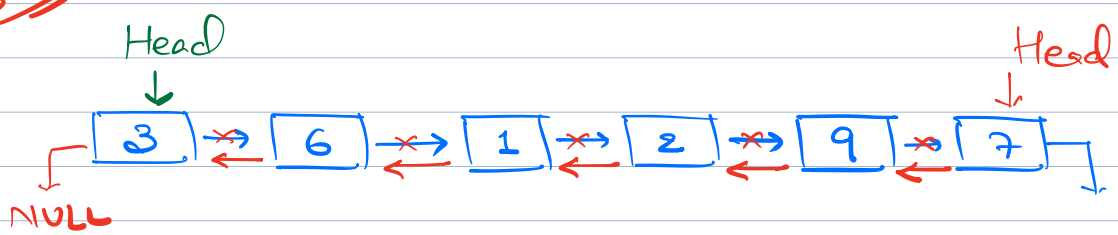
Amazon
MS
Adobe
Paytm.

⇒ O(1) extra space.

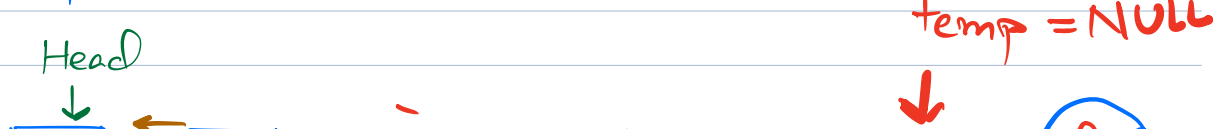
Eg



Sol^n



1) Reverse the direction of each pointer.



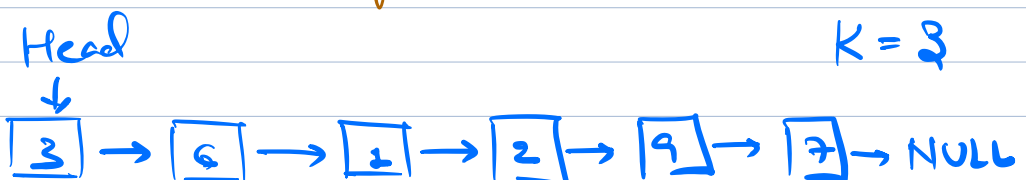
$h_2 = NULL$

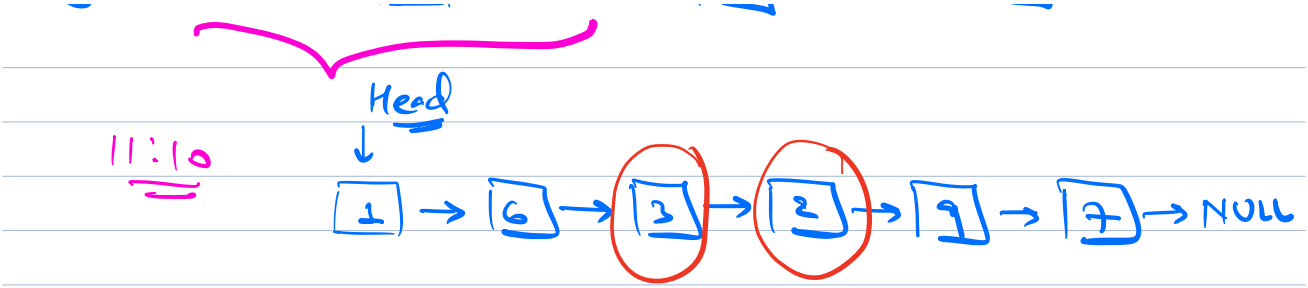
```
h1 = Head;
h2 = h1.next;
h1.next = NULL;

while ( h2 != NULL ) {
    temp = h2.next
    h2.next = h1;
    h1 = h2;
    h2 = temp;
}
return h1;
```

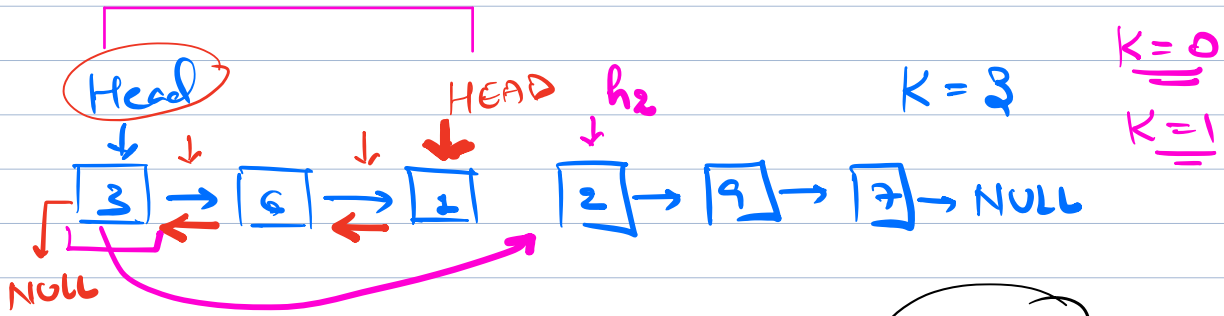
T.C. = O(N)
S.C. = O(1)

Q Given a linked list & an integer K. Reverse the first K nodes. S.C. = O(1)

Eg







K > N

temp = head;

temp.next = h_2

Head of the updated LL

Node reverseFirstK (head, k) {

if (K < 2 || head == NULL) {

return head;

}

h_1 = Head;

h_2 = h_1.next

h_1.next = NULL;

K--;

while ((h_2 != NULL) && (K > 0)) {

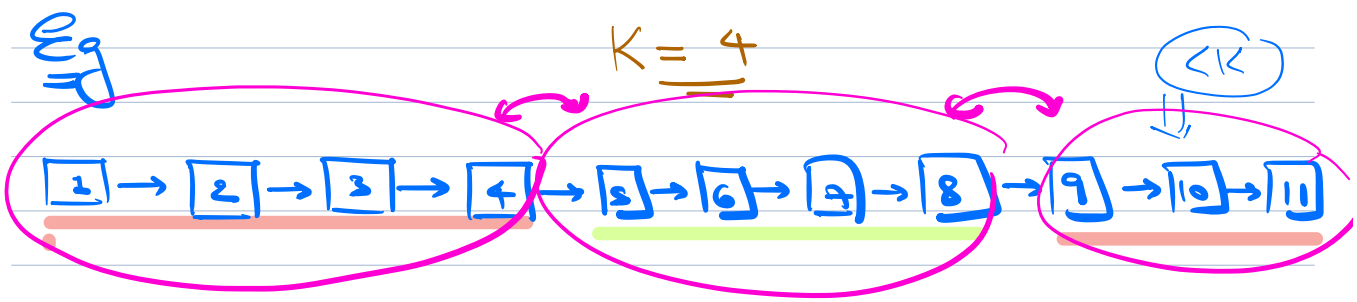
temp = h_2.next

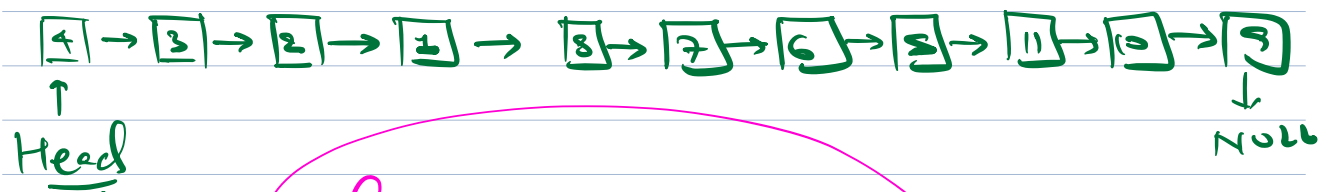
```
        h2.next = h1;

        h1 = h2;
        h2 = temp;
        k--;
    }
    Head.next = h2;
    return h1;
}
```

H.W. Google

Reverse a Linked List in groups of K.

Eg





Can use recursion