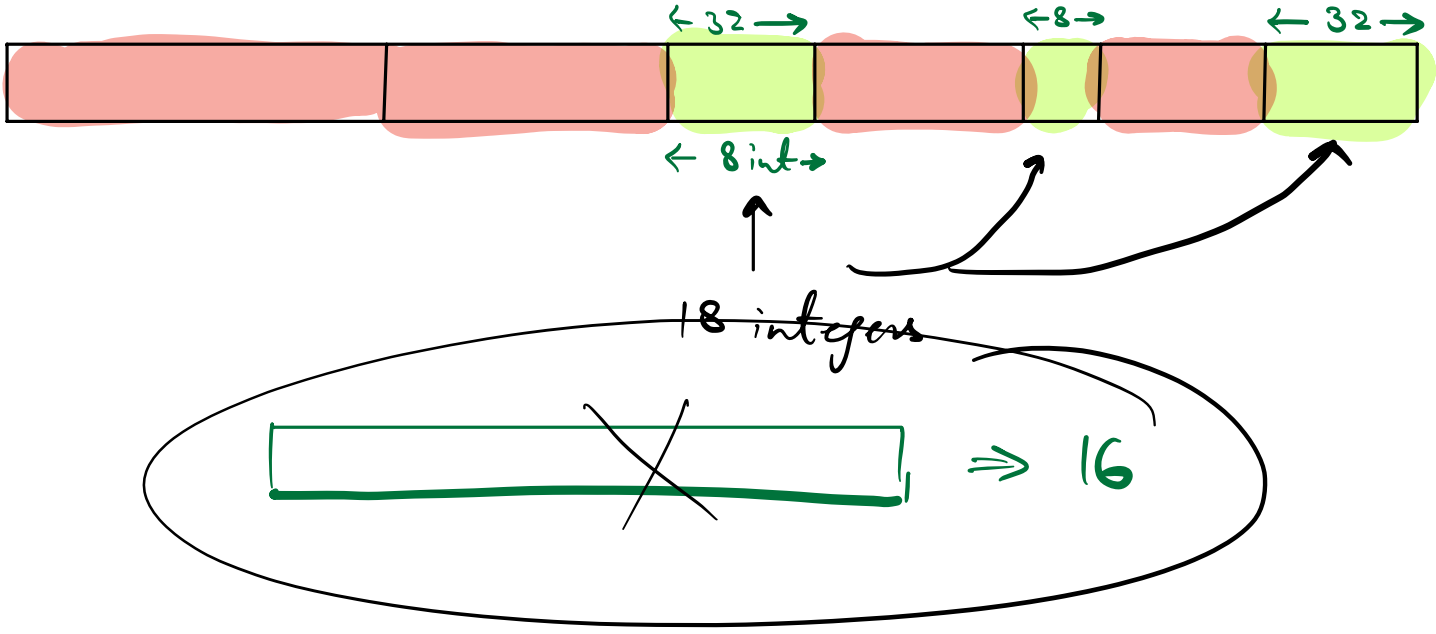
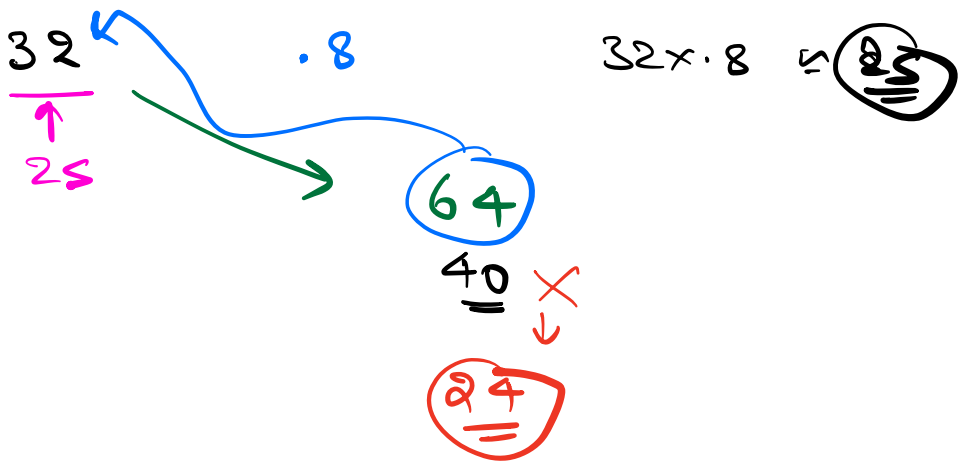


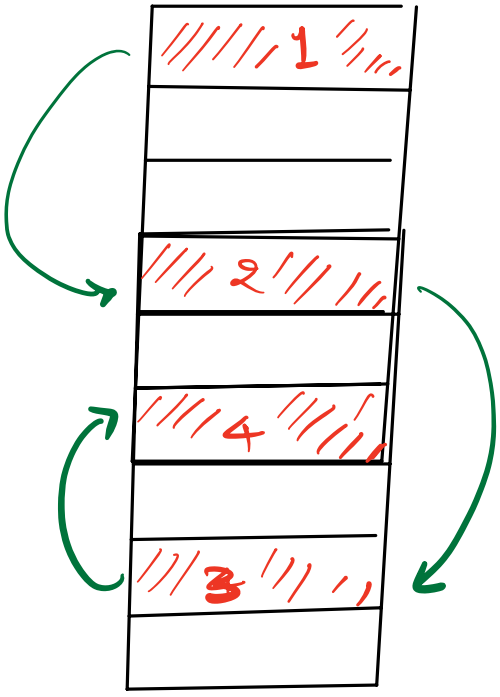
Non contiguous DS required.

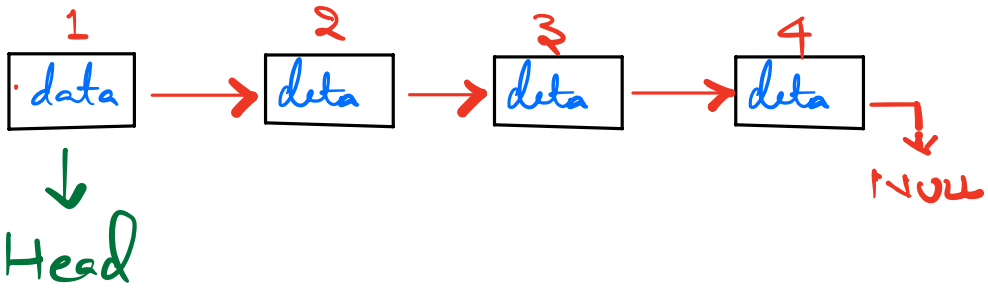
2) Size.



Linked List

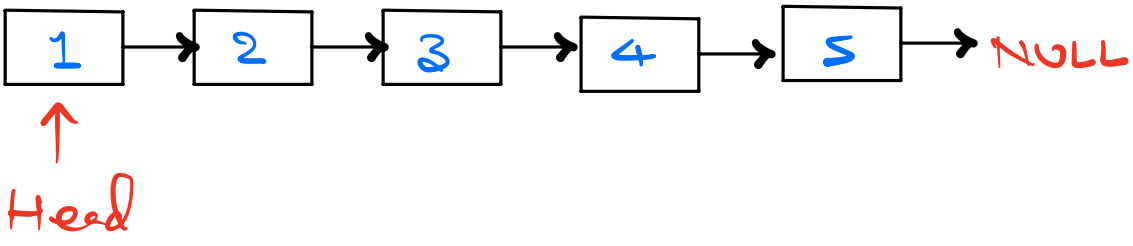
1) Linear DS
2) Non contiguous.





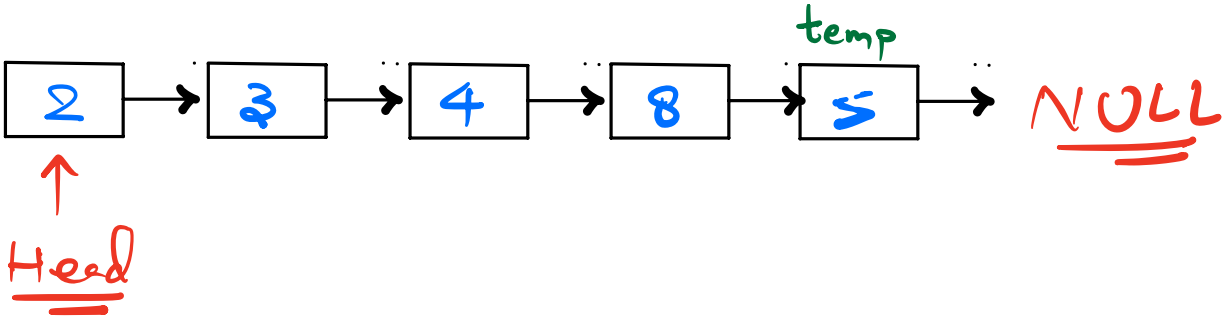

```
class Node {
    int data;
    Node next;
    Node(int x) {
        data = x;
        next = NULL;
    }
}
```

```
Node head = new Node(4);
```



Given an integer array of size N.
Create a LL & return its Head.

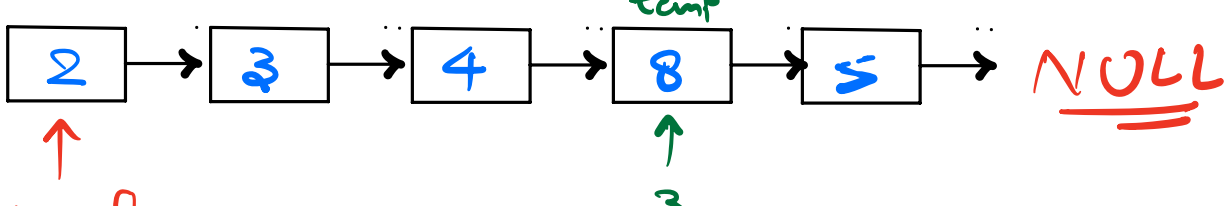
A = [2, 3, 4, 8, 5]
0 1 2 3 4




```
Node createLL ( A , N) {

    Node Head = new Node (A[0]);

    Node temp = Head;

    for (i = 1; i < N; i++) {

        temp.next = new Node (A[i]);
        temp = temp.next
    }
    return Head;
}
```

Given the Head Node of a LL. Return the element on Kth index. (K<N)

K = 3

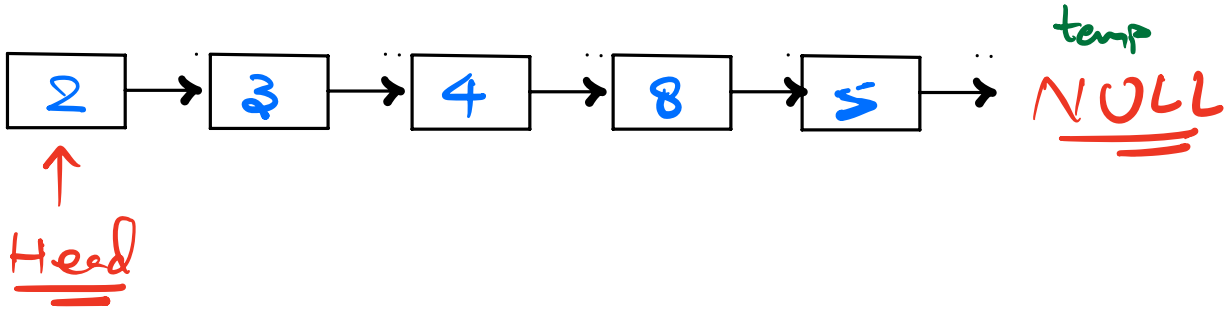


Head

Ans = 8

```
int getKthElement (Head, K) {
    Node temp = Head;
    for (i=0; i<K; i++) {
        temp = temp.next;
    }
    return temp.data;
}
```

Given a LL. Return true if there is a data K present in any node of LL.



K = 3 ⇒ true

K = 7 ⇒ false;

Code

```
boolean find (Head, K) {
    Node temp = Head;
    while (temp != NULL) {
        if (temp.data == K) {
            return true;
        }
        temp = temp.next;
    }
    return false;
}
```

T.C. = O(length of LL)

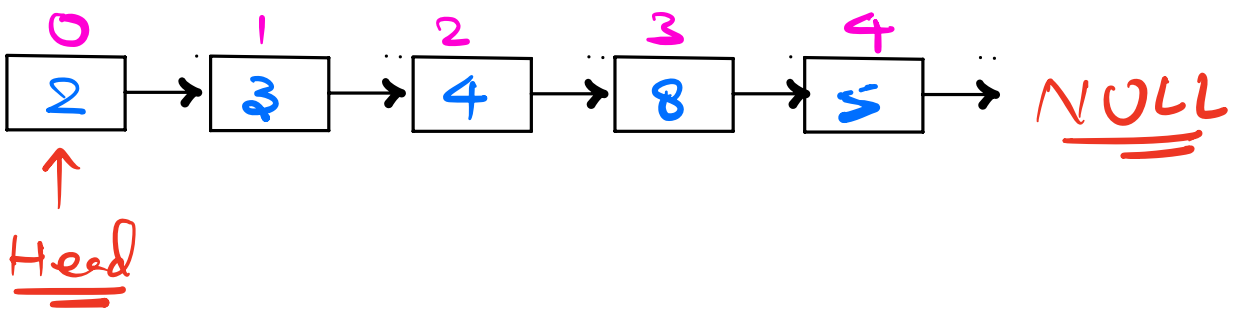
Given a LL. Return its size.

Code

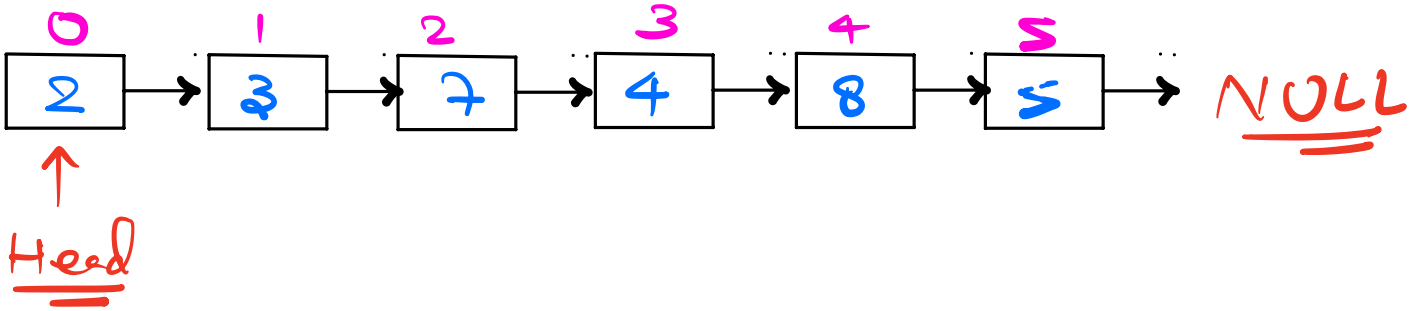
```
int getLength (Node Head) {
```

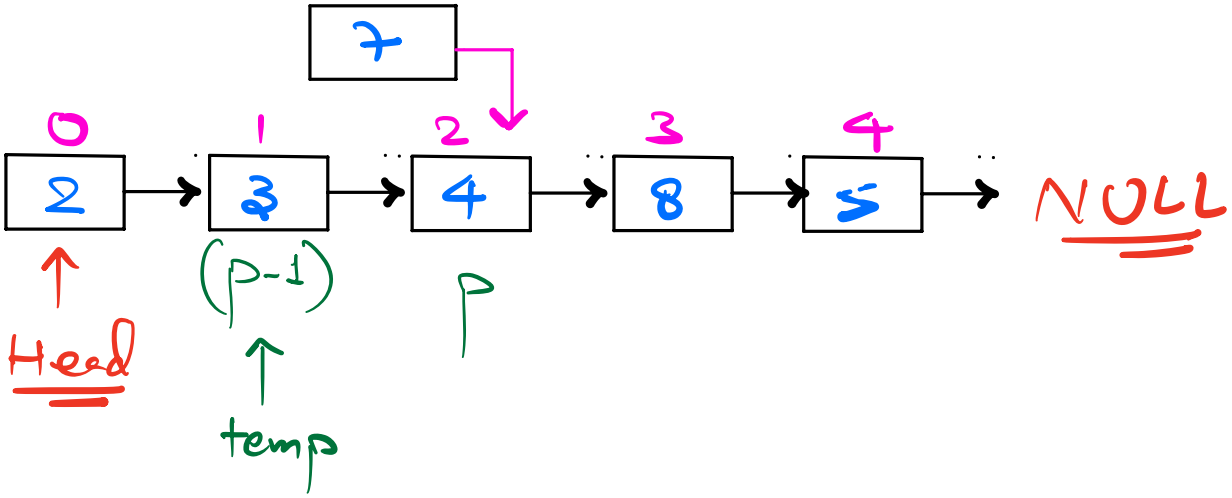
```
Node temp = Head;
int count = 0;

while (temp != NULL) {

    count ++;
    temp = temp.next;
}
return count;
}
```

Given a LL. Insert a new node with data K at the index p return Head;

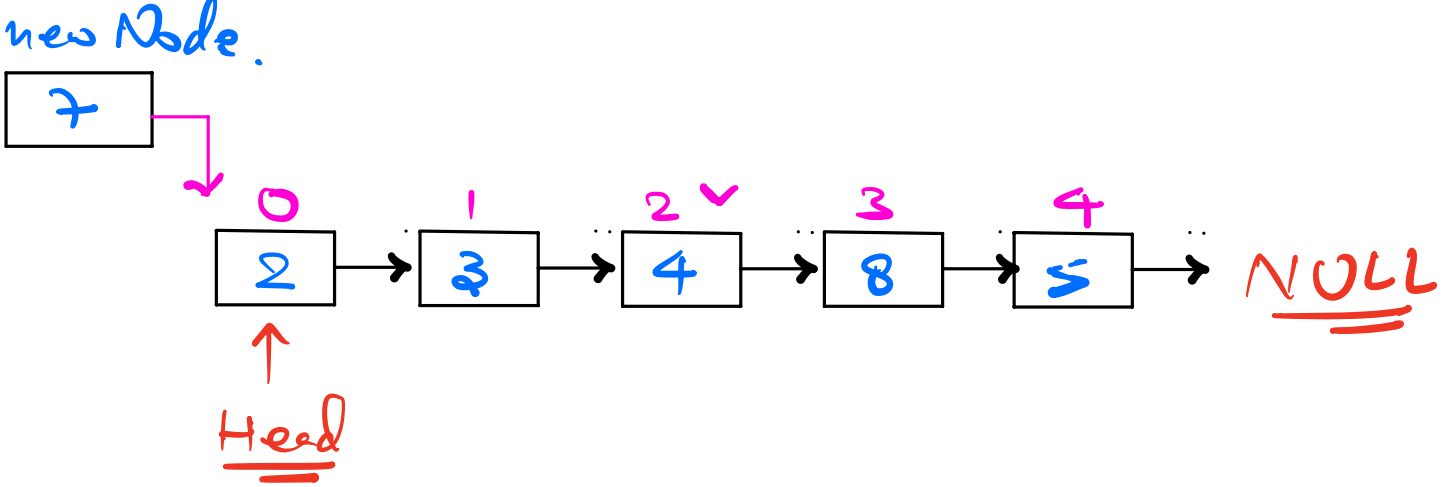


p = 2
K = 7



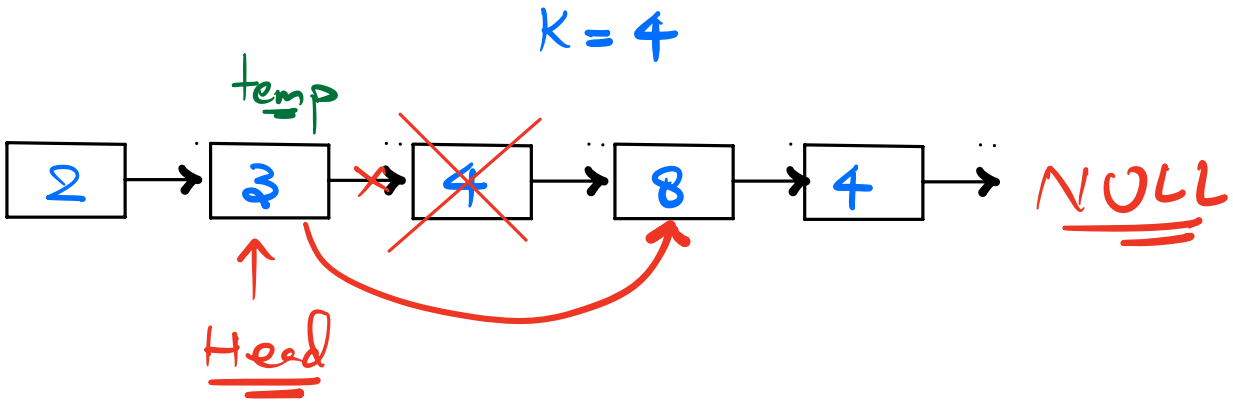


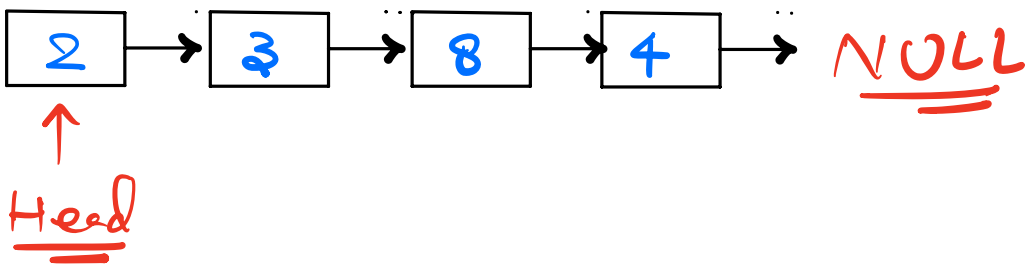

```
Node insert (Head, K, P) {
    Node newNode = new Node(K);
    if (P == 0) {
        newNode.next = Head;
        Head = newNode;
        return Head;
    }

    Node temp = Head;
    for (i=0; i<(p-1); i++) {
        temp = temp.next;
    }
    newNode.next = temp.next;
    temp.next = newNode;

    return Head;
}
```



$$T.C. = O(P) = O(N)$$

Q Given a LL. Delete the first occurrence of K in the LL.

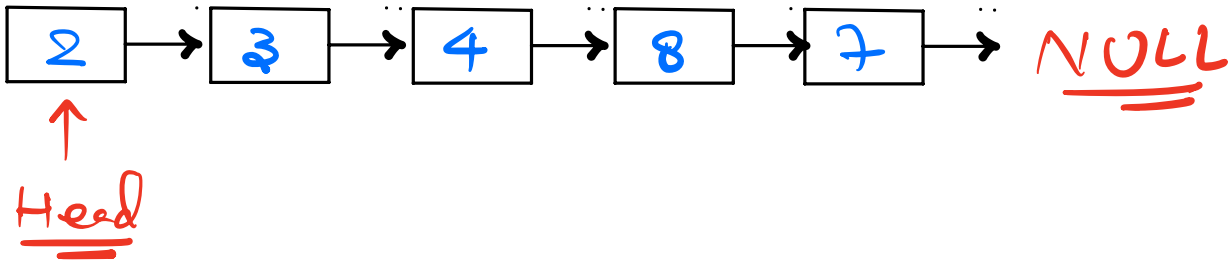


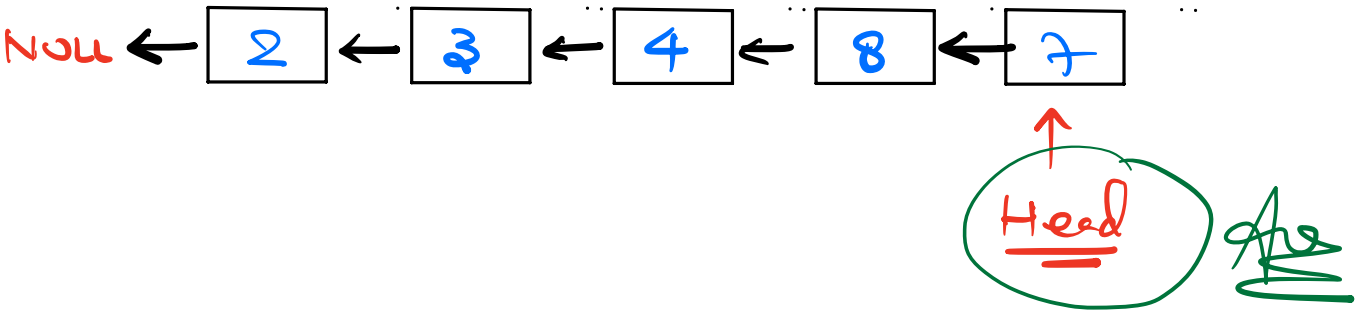


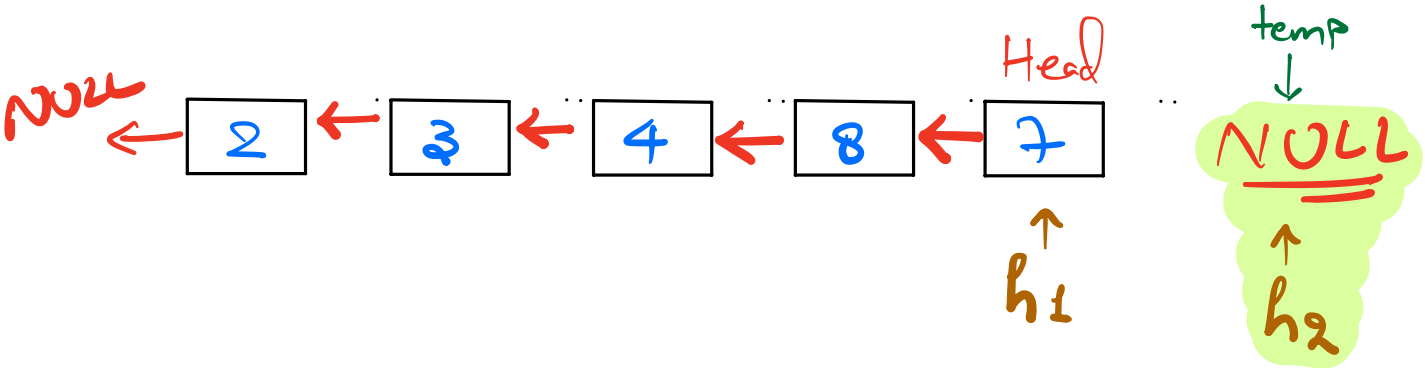
Node delete (Head, K) {

```
if (Head == NULL) {
        return Head;
}

if (Head.data == k) {
        temp = Head.next;
        Head.next = NULL;
        Head = temp;
        return Head;
}
temp = Head;

while (temp.next != NULL) {

    if (temp.next.data == k) {

        temp.next = temp.next.next;
        return Head;
    }
    temp = temp.next;
}
return Head;
}
```

Given a LL. Reverse & return the new Head.







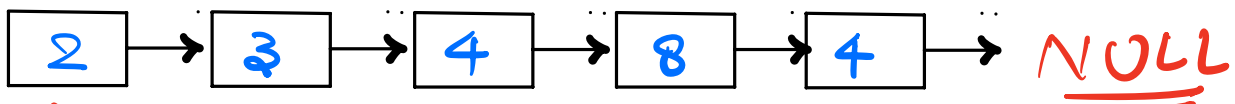
Code

```
Node reverse (Node Head) {

if (Head == NULL) { return Head; }

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

```
        h2.next = h1
        h1 = h2;
        h2 = temp;
    }

    Head.next = NULL;
    Head = h1;
    return Head;
}
```

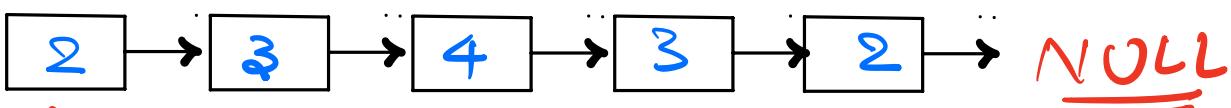
T.C. = O(N)

Q) Given a LL. Return true if it is a pallindrome.



Head

False



Head

True

Soln 1) Reverse of LL

O(N) ← 1) Create a copy of LL

O(N) ← 2) Reverse the copy

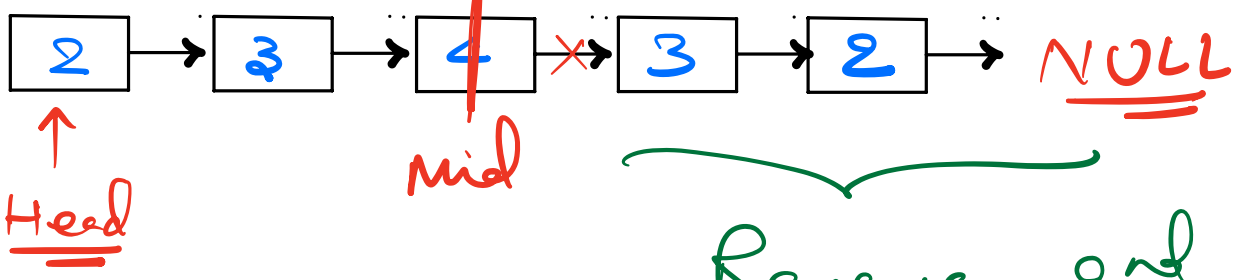
O(N) ← 3) Compare both list.

T.C. = O(3N) = O(N)

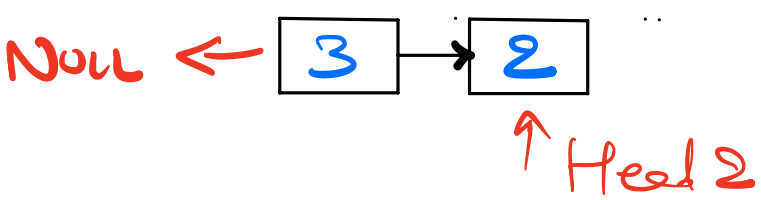
S.C. = O(N) ⇒ Copy LL.

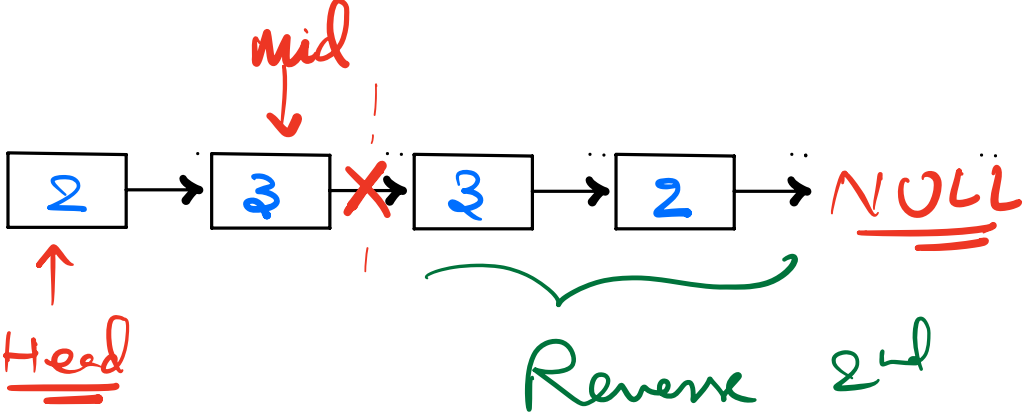
2)

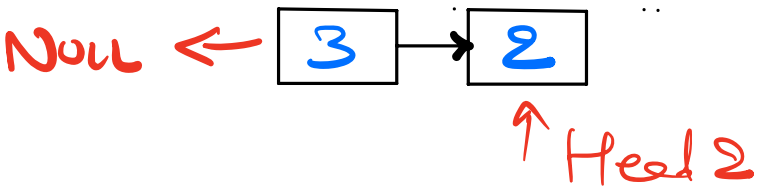
S.C. → O(1)



Reverse 2nd Half







H.W.

How to find mid ??

↓

Hint : Length

Saturday afternoon

↓

PS → BS + Sorting

+
