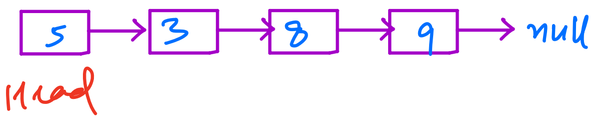
Linked List 1 : Introduction



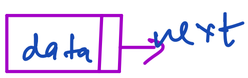
Arrays/ Dynamic array

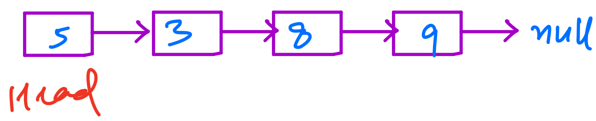
continous memory allocation

linear DS that utilize free memory in best way is Linked List.

```
class A{

}
```

```
A a = new A()
```






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

Ques → Access K^{th} element of linked list (K starts from 0)

K = 0 1 2 3

5 → 3 → 8 → 9 → null

Head // never update Head unless required

Array → A[K] TC = O(1)

```
Node access (Node head, int K) {
    Node temp = head;

    for ( i = 0 to K-1) {
        if (temp == null)
            return null
        temp = temp.next;
    }
    return temp
}
```

temp

5 → 3 → 8 → 9 → null

Head

if K >= n ⇒ null pointer exception

TC = O(K)

Ques → Check if the value X is present in LL.

Searching. Array → unorganised TC = O(N)

Array → organised TC = O(log N)

Linear Search

```
Node temp = Head
while (temp != null) {
    if (temp.data == X)
        return true

    temp = temp.next
```

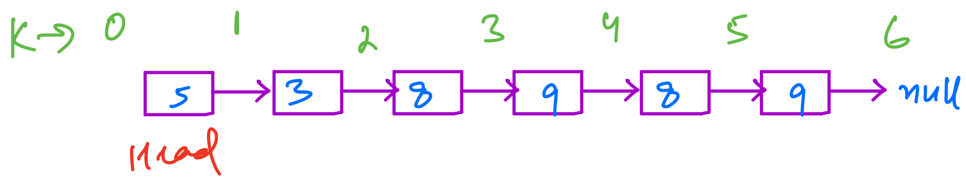
TC = O(N)

}

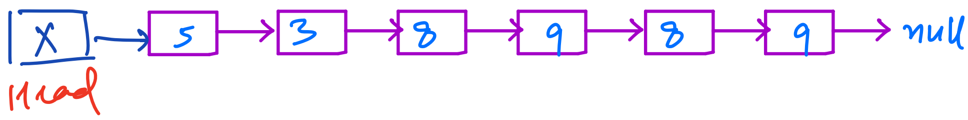
return false

Ques → Insert a new node with data x at position K,

where K=0 ⇒ head node. (0 <= K <= n)

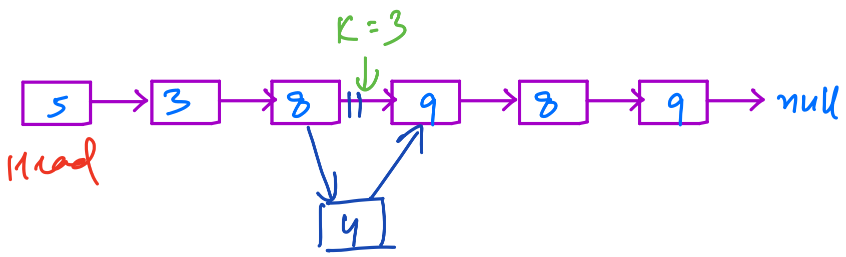


K=0 ⇒



K=3

x=4



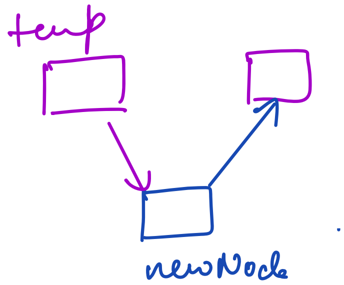
→ K=0

1. Head = null // empty list ✓
2. K=0 ⇒ Head will update ✓

```
Node newNode = new Node(x)

if (K==0){
    newNode.next = Head
    Head= newNode   // update Head
}
```

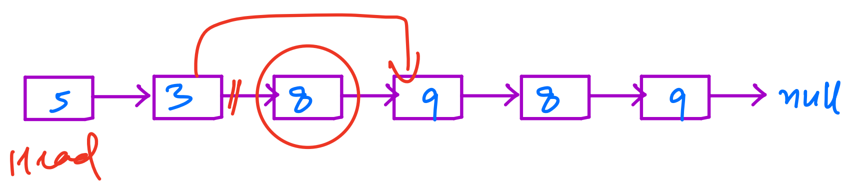
```
else {
    Node temp = Head
    for (i=0 to k-2) { // K-1 times
        temp = temp.next
    }
    newNode.next = temp.next
    temp.next = newNode
}
```

TC = O(K)



Ques → Delete the first occurance of value X from LL.

If not present ⇒ no change



X=8

Cases

1. Head = null // empty LL ✓
2. Head.data = X ⇒ delete Head node ✓
3. No change // X not present ✓

```
if (Head == null) return
if (Head.data == X) {
    Head = Head.next
}
```

```
Node temp = Head

while ( temp.next != null) {
    if( temp.next.data == X ) {
        temp.next = temp.next.next;
        break;
    }
    temp = temp.next
}
```

TC = O(N)

H.W → delete all occurance of X

Ques → find the length of given LL.

5 → 3 → 8 → 9 → null

Head

ans = 4

```
temp = Head
n = 0
while (temp != null) {
    n++
    temp = temp.next
}
print(n)
```

TC = O(N)

Ques → Find middle element of LL.

Array → A[n/2]

5 → 3 → 8 → 9 → null

Head

5 → 3 → 8 → 9 → 9 → null

Head

TC = O(N + N/2)

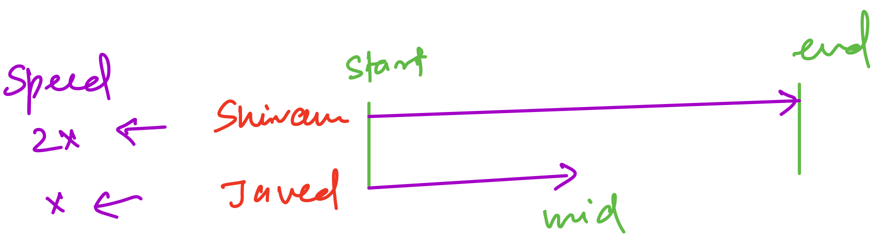
= O(N)

```
temp = Head
n = 0
while (temp != null) {
    n++
    temp = temp.next
}
temp = Head
for (i = 1 to n/2) {
    temp = temp.next
}
print(temp.data)
```

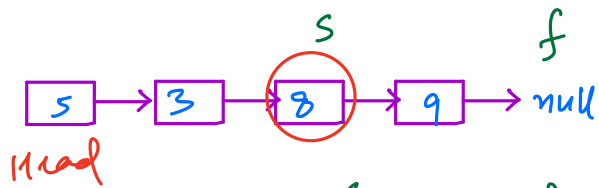
start → end

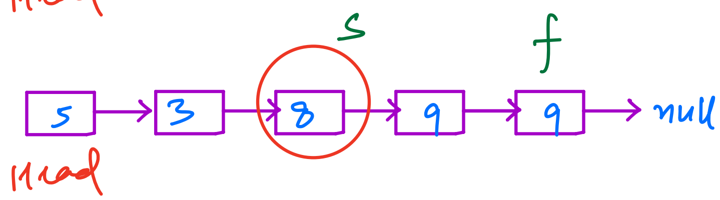
start → mid

Solve in 1 traversal

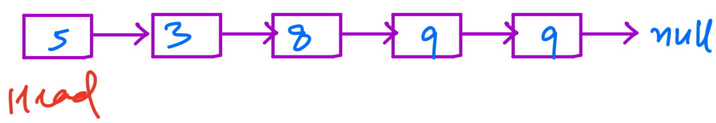


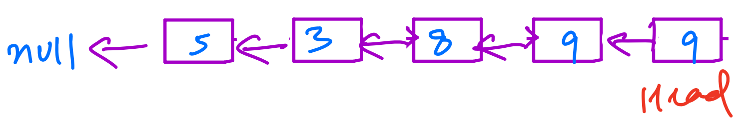
```
slow = Head
fast = Head

while( fast != null && fast.next != null) {
    fast = fast.next.next;
    slow = slow.next
}

return slow
```

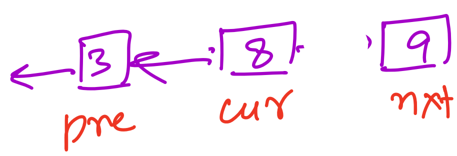


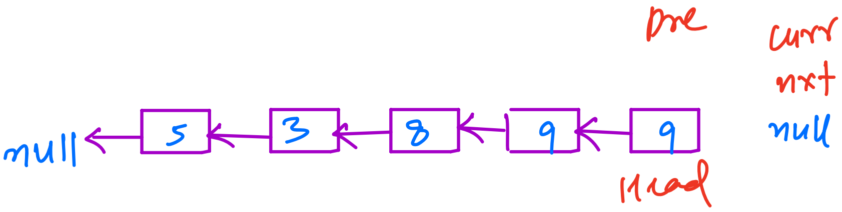


Ques → Reverse the given LL.





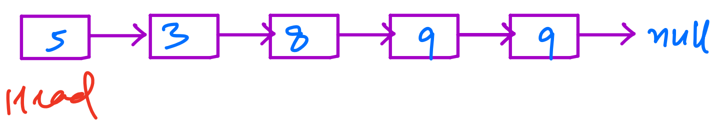




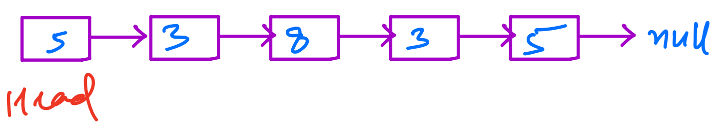

```
pre = null
cur = Head

while ( cur != null ) {
    nxt = curr.next
    cur.next = pre
    pre = cur
    cur = nxt
}

Head = pre
```

TC = O(N)

Ques → Check if the given LL is palindrome.



ans = false



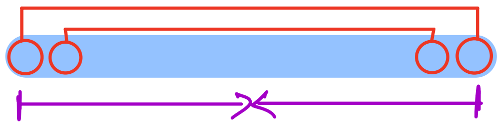
ans = true

Idea1 : 1. Clone the list & reverse it

2. Compare both

TC = O(N)

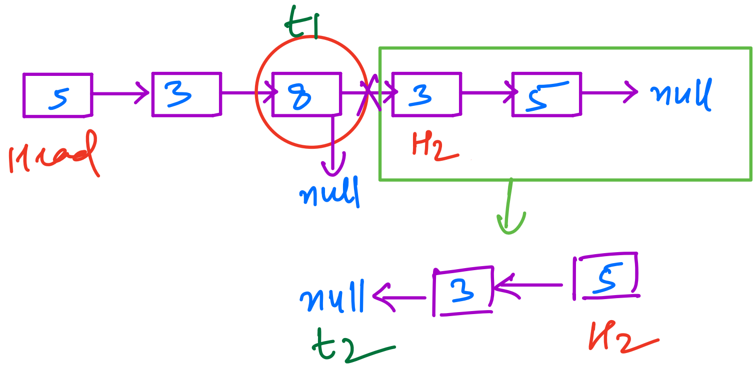
SC = ~~O(N)~~ ⇒ O(1)



Solution :

1. find the middle
2. Reverse second half LL
3. compare first & second-half

TC = O(N)
SC = O(1)



if (t1 == null || t2 == null)
⇒ true

if (t1.data != t2.data)
⇒ false