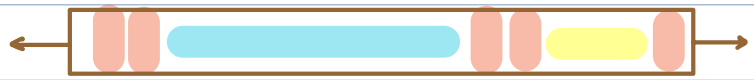
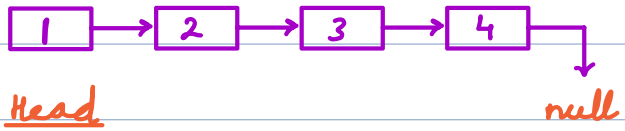
Arrays / Dynamic Array



continuous memory allocation

Linked List → 1) Linear data structure.

2) Need not have continuous memory allocation.

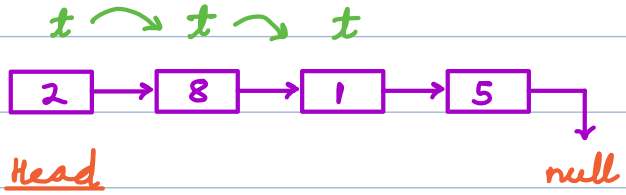



```
class Node {
    int data ; // any datatype
    Node next ;
    Node (int x) {
        data = x ;
        next = null ;
    }
}
```

Operations on Linked List

1) Access K^{th} element (K=0 → first data)

Array → A[K] TC = O(1)




```
Node temp = Head
for i → 1 to K {
    temp = temp.next
```

```
}
return temp.data          TC = O(K)
```

---

2) Check if X is present (Searching)

Array → Linear Search TC = O(N)

Array (sorted) → Binary Search TC = O(log N)

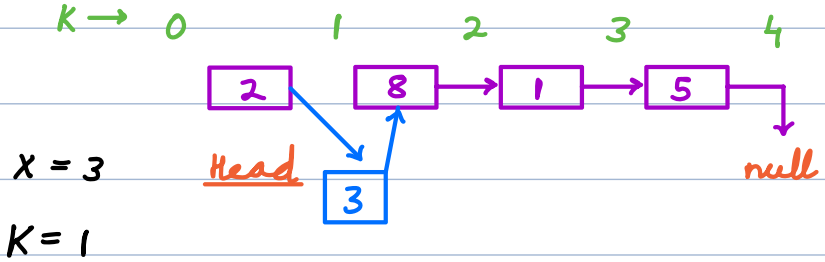
```
Node temp = Head
while (temp != null) {
    if (temp.data == X)  return true
    temp = temp.next
}
return false              TC = O(N)
```

Cannot use BS → ∵ we cannot jump to middle element.

---

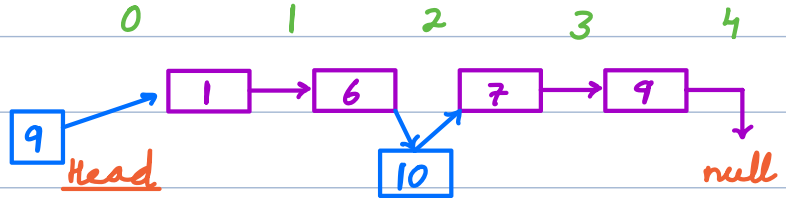
3) Insert a node with data x at position K (0 → start)

[0 <= K <= N]



X = 3

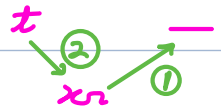
K = 1



X = 10 K = 2

X = 9 K = 0

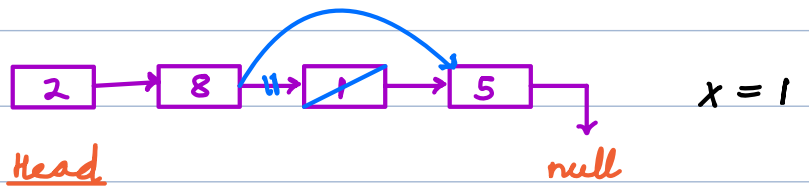
```
xr = new Node(x)
if (K == 0) {
    xr.next = Head
    Head = xr
    return Head
}
temp = Head
for i → 1 to (K-1) {
    temp = temp.next
}
xr.next = temp.next
temp.next = xr
return Head
```



TC = O(K)

---

4) Delete the first occurrence of x.
(If not present → leave as it is)

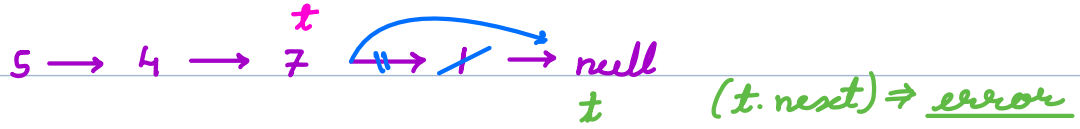


x = 1

X = 4

cases → 1) Empty list (Head == null)
nothing to do.

2) (Head.data == x)
Head = Head.next



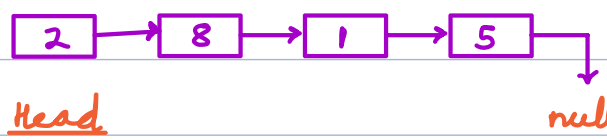

```
if (Head == null) return Head
if (Head.data == x) {
        Head = Head.next
        return Head
}

temp = Head
while (temp.next != null) {
        if (temp.next.data == x) {
                temp.next = temp.next.next
                break
        }
        temp = Temp.next
}
return Head
```

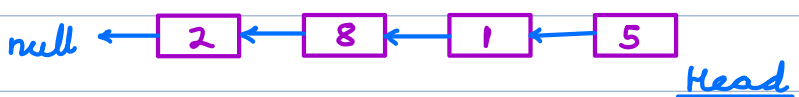
TC = O(N)

H.W → Delete all occurences of X.

K = 2 

Q → Reverse the given linked list.



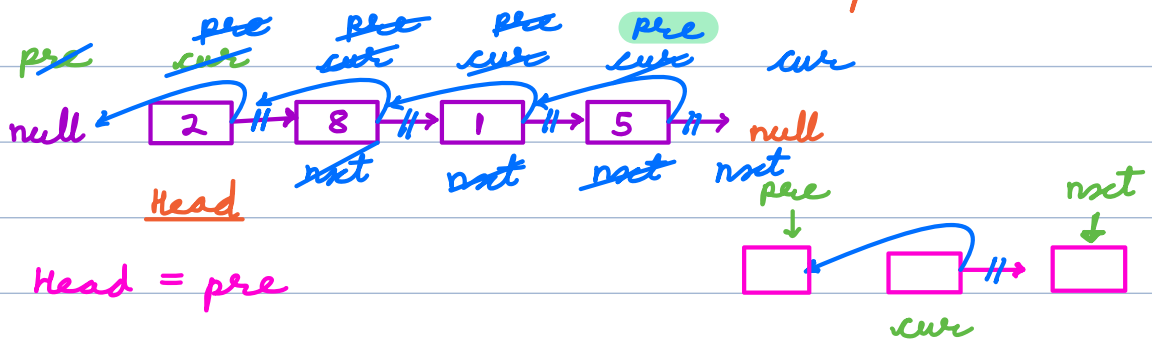


Cases

1) Empty list → if (Head == null) return Head

2) Single Node → if (Head.next == null) return Head

3) Else → Reverse the links & update Head



Head = pre


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

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

Q → Check if the given linked list is palindrome. ⟹

2 → 5 → 8 → 7 → 2 → null
Head

Ans = false

2 → 5 → 8 → 5 → 2 → null
Head

Ans = true

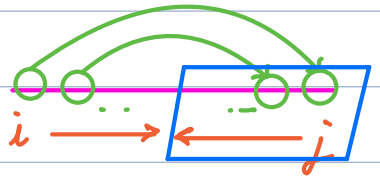
1 → null Ans = true

Head

Sol 1 → 1) create a copy list
2) Reverse the copy
3) compare copy with original.

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

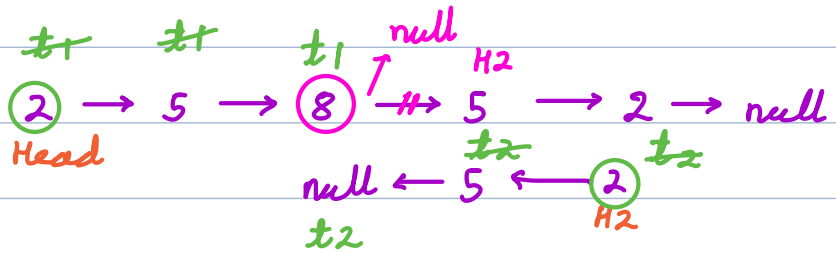
Sol 2 →



reversed

1) Find middle element
2) Reverse second half
3) compare reverse with first half

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



Ans = true

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