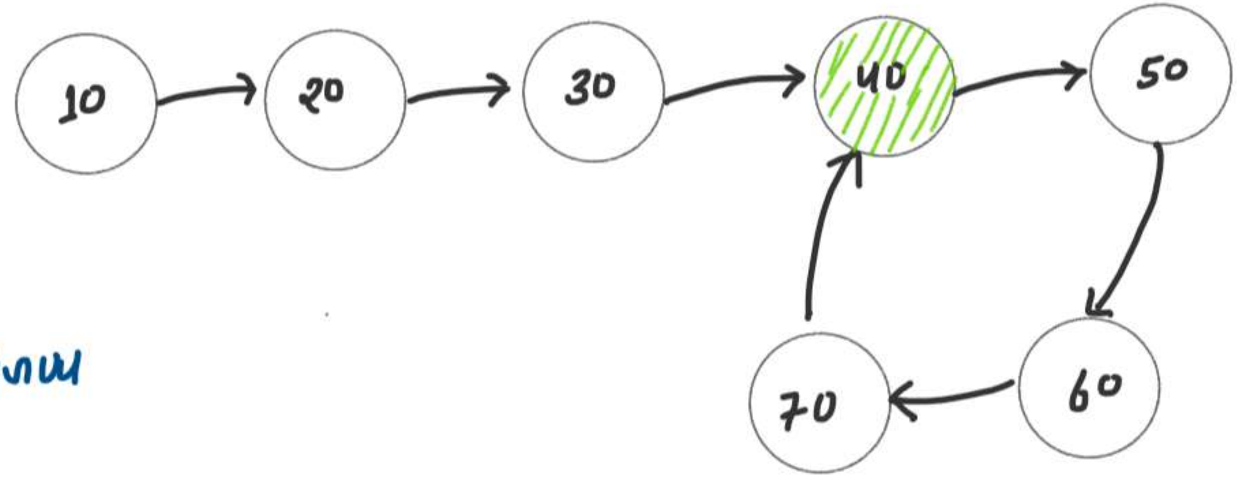
6/11/2023

LINKED LIST CLASS - 4

Problem 1: Linked List Cycle (Leetcode-141)

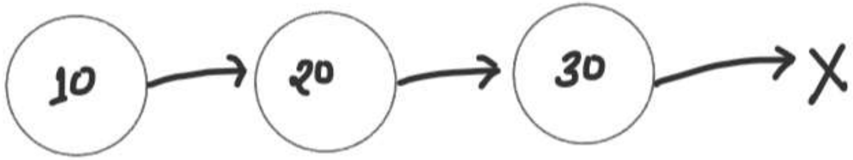
Ex:1



Loop Present

Output true

Ex:2



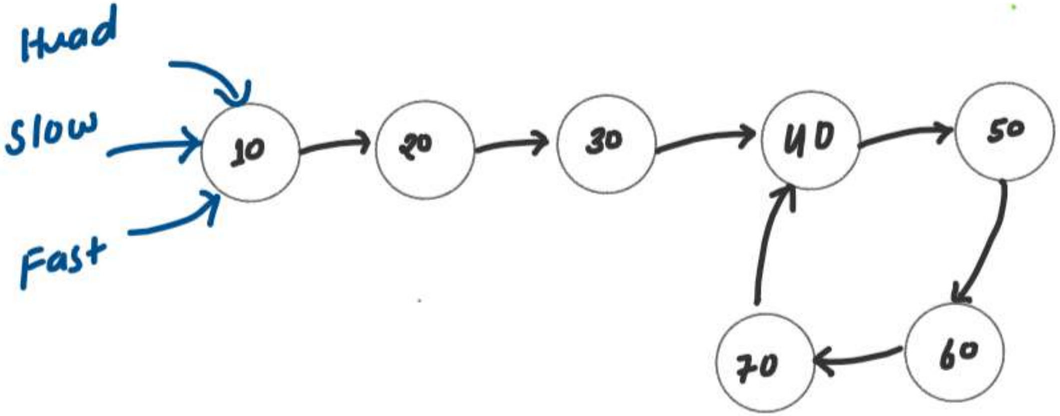
Loop Not Present

Output false

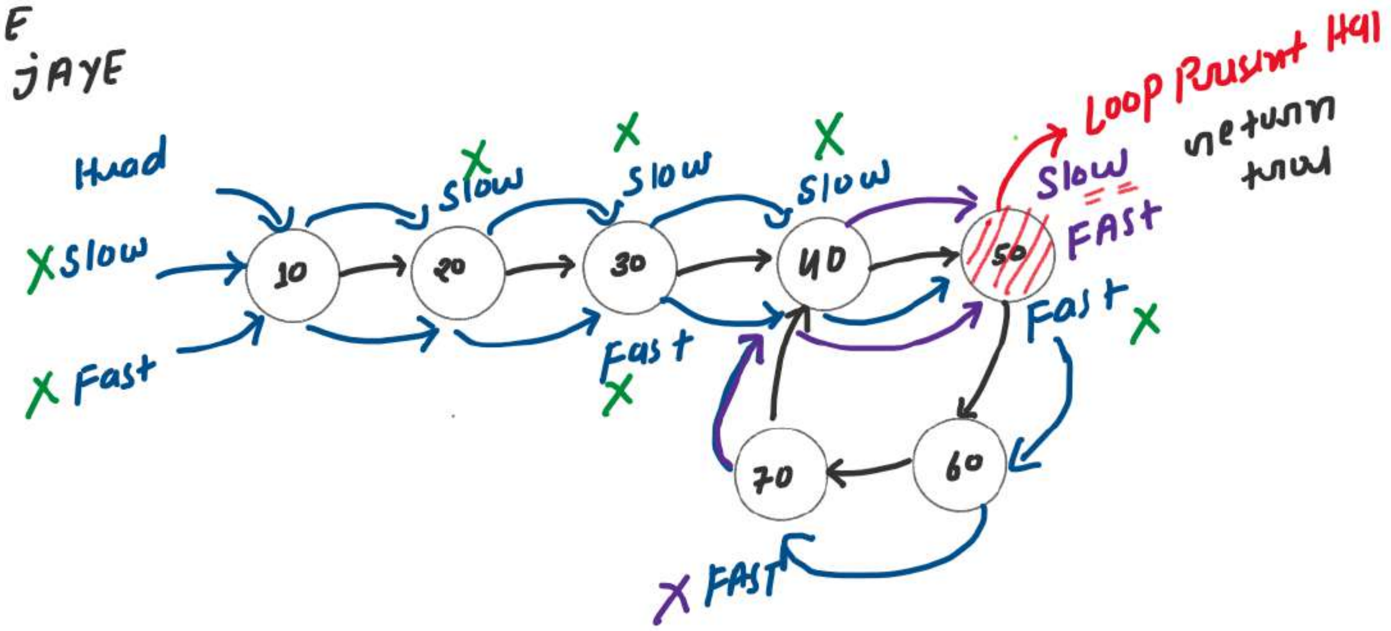
Approach 1: *Fast and slow algorithm*
(Jab fast two step move hoga tabhi slow one step move ho payega)

Step:1

Node* slow = head;
Node* fast = head;



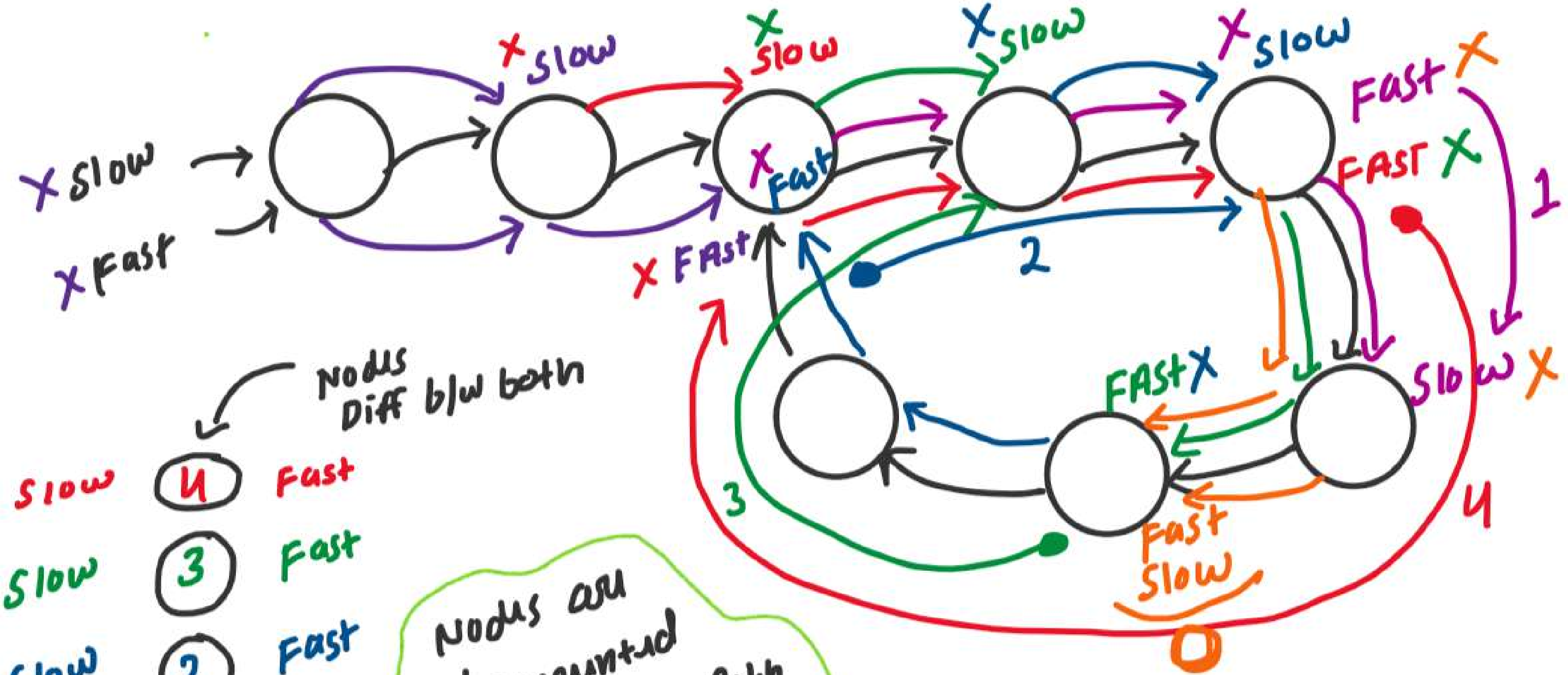
Step:2 MOVE SLOW AND FAST
JAB TAK DONO SAME
POSITION PAR NA AA JAYE



```
while ( fast != Null ){
    fast = fast -> next;
    if( fast != Null ){
        fast = fast -> next;
        slow = slow -> next;
    }
    if ( fast == slow ){
        return true;
    }
} -> return false;
```

-> Check for loop

Reasoning: *Why this algorithm work*

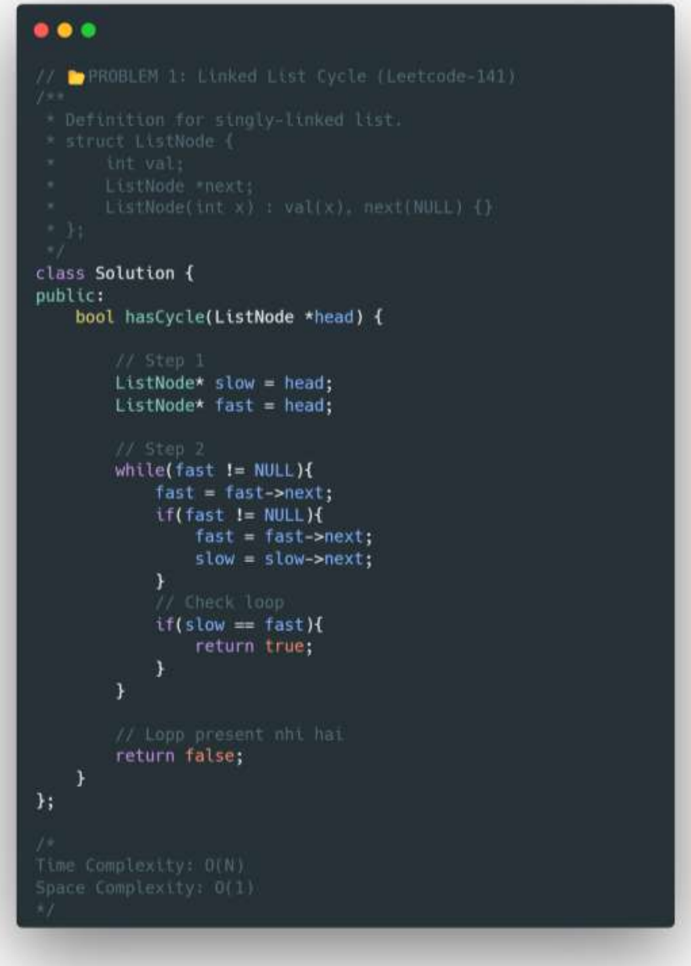


Nodes Diff b/w both

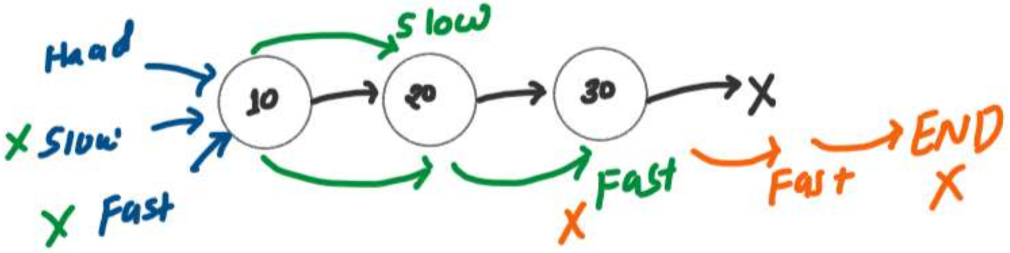
Slow	4	Fast
Slow	3	Fast
Slow	2	Fast
Slow	1	Fast
Slow	0	Fast

Nodes are decremented by 1 b/w Both S and F

Slow == Fast } Loop Present Hai



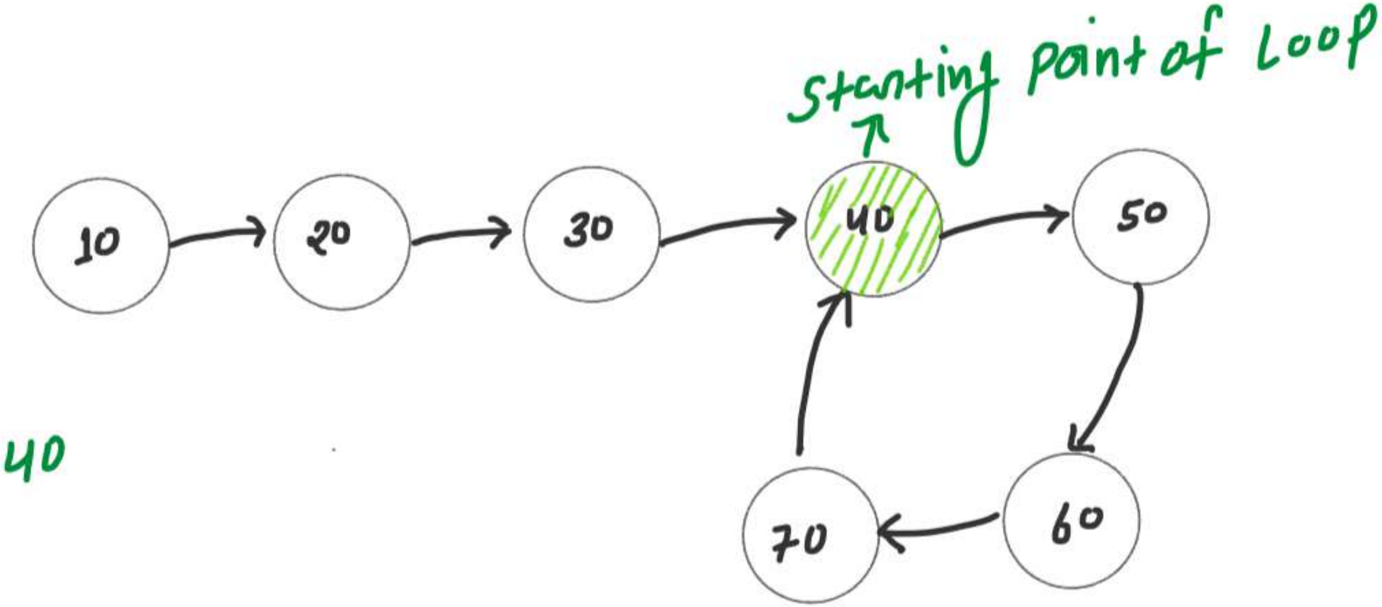
```
// 📁PROBLEM 1: Linked List Cycle (Leetcode-141)
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {

        // Step 1
        ListNode* slow = head;
        ListNode* fast = head;

        // Step 2
        while(fast != NULL){
            fast = fast->next;
            if(fast != NULL){
                fast = fast->next;
                slow = slow->next;
            }
            // Check loop
            if(slow == fast){
                return true;
            }
        }

        // Lopp present nhi hai
        return false;
    }
};

/*
Time Complexity: O(N)
Space Complexity: O(1)
*/
```



return false } Loop Present nahi Hai

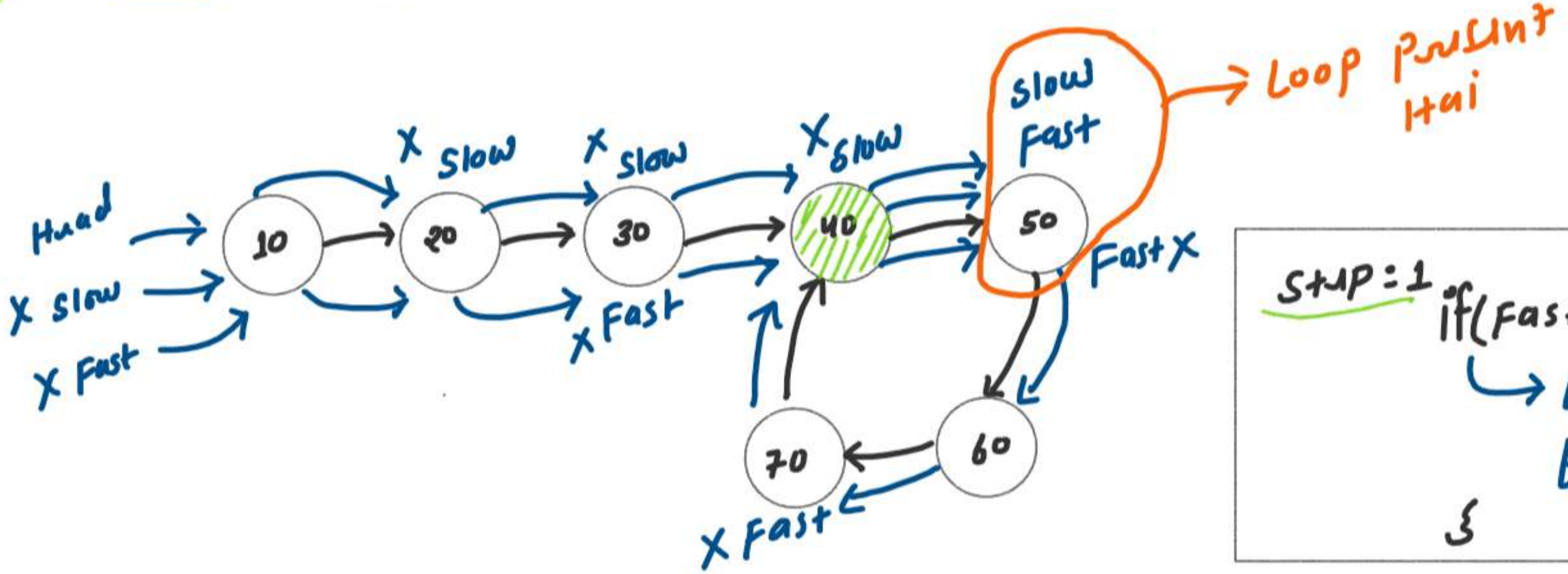
📂 Problem 2: Starting point of loop (Leetcode-142)

Ex:1



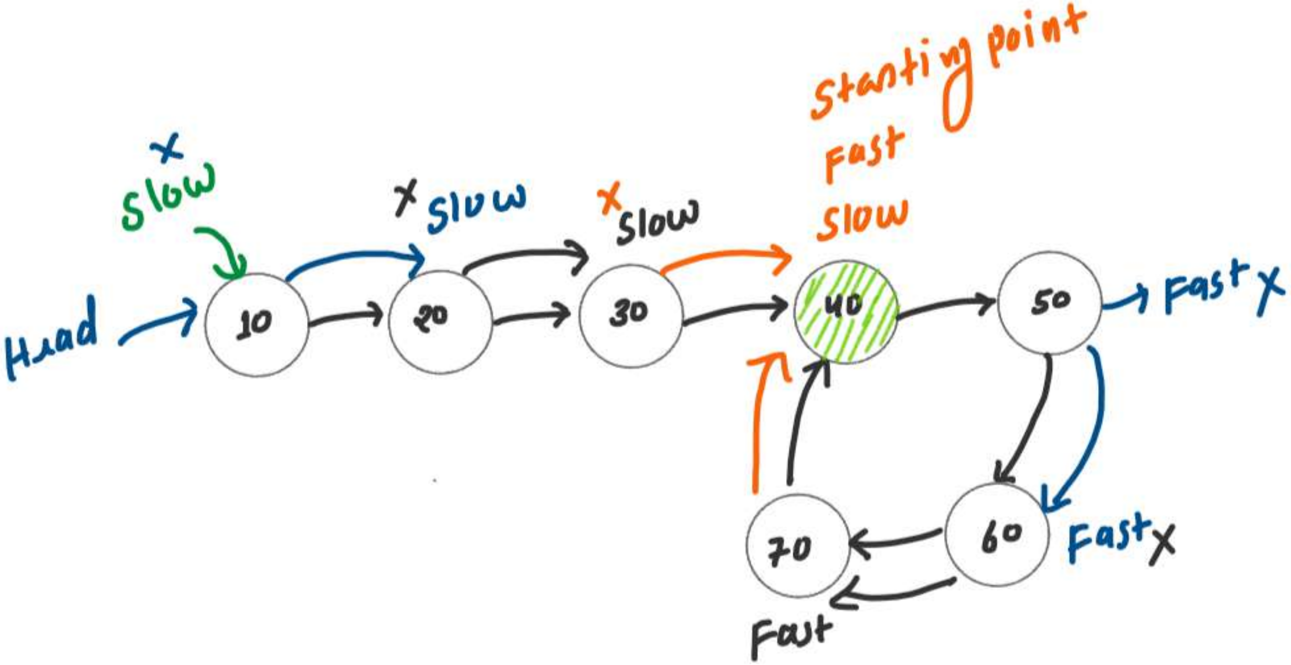
Output 40

Approach 1: *Fast and slow algorithm*



Step:1

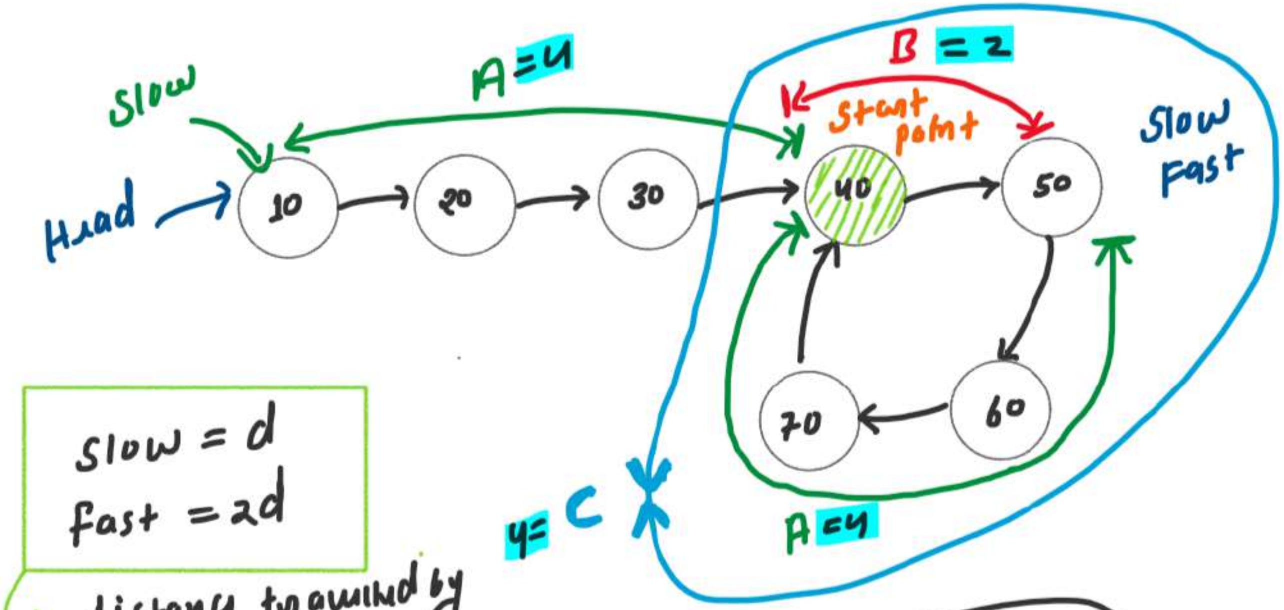
```
if(Fast == slow) {
    ↳ Loop Present
      break;
}
```



step:2

```
slow = head

|| slow & Fast -> 1 step
while ( slow != Fast)
  {
     Fast = Fast -> next;
     slow = slow -> next;
  }
|| return starting point
   return slow;
```

Reasoning: Why this algorithm work



slow = d
fast = 2d

distance travelled by Both in step 1

K => Number of cycles
A, B, C => Distances

slow = d
fast = 2d

distance travelled by Fast = 2(distance travelled by slow)

$$A + K_1C + B = 2 * (A + K_2C + B)$$

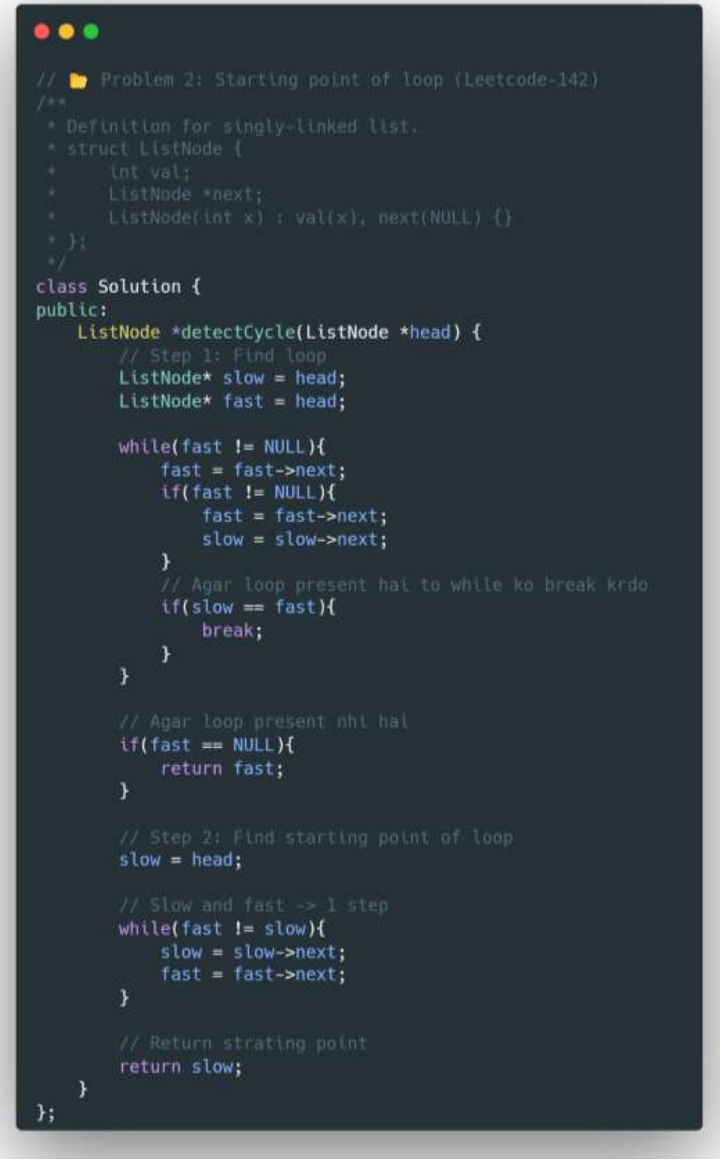
$$A + B + K_1C = 2A + 2B + 2K_2C$$

$$(K_1 - K_2)C = A + B$$

let $K_1 - K_2 = K$

$$KC = A + B$$

$$\boxed{A = KC - B}$$



```
// 📁 Problem 2: Starting point of loop (Leetcode-142)
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        // Step 1: Find loop
        ListNode* slow = head;
        ListNode* fast = head;

        while(fast != NULL){
            fast = fast->next;
            if(fast != NULL){
                fast = fast->next;
                slow = slow->next;
            }
            // Agar loop present hai to while ko break krdo
            if(slow == fast){
                break;
            }
        }

        // Agar loop present nhi hai
        if(fast == NULL){
            return fast;
        }

        // Step 2: Find starting point of loop
        slow = head;

        // Slow and fast -> 1 step
        while(fast != slow){
            slow = slow->next;
            fast = fast->next;
        }

        // Return strating point
        return slow;
    }
};
```

T.C.

Step1 O(N)

Step2 O(N)

Overall T.C. = O(N) + O(N)

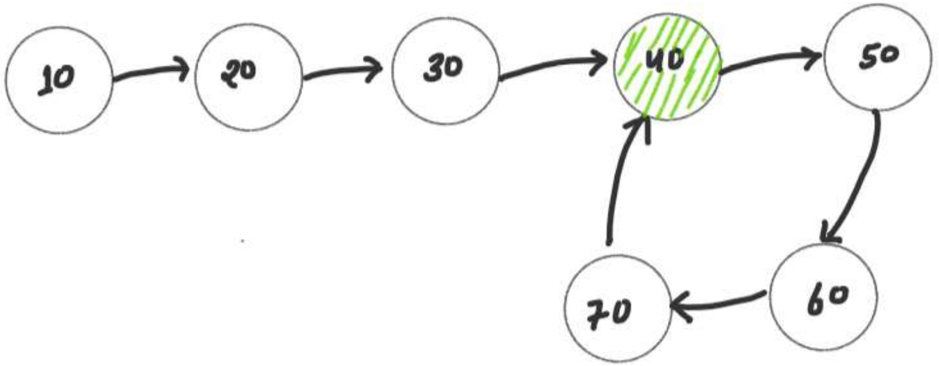
= O(N)

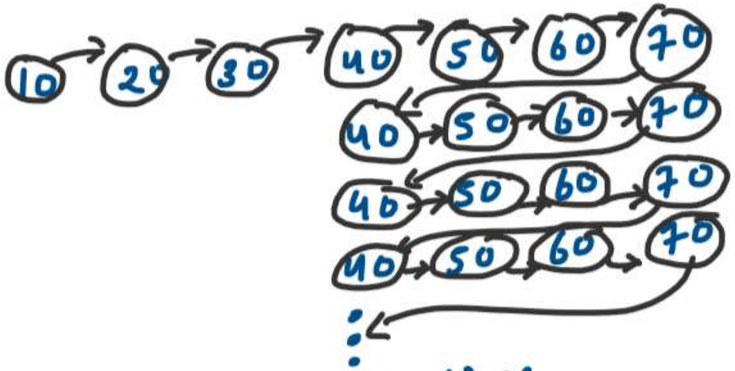
S.C.

O(1)

📂 Problem 3: Remove loop (GFG)

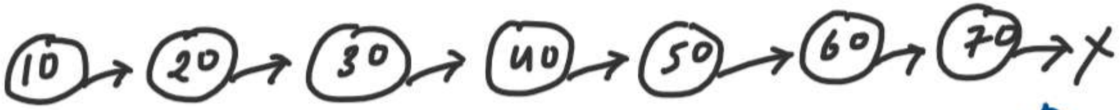
Ex:1





∞ time
loop chalta Rahega
to ISSI Loop ko
REMOW Karna Hai

Output



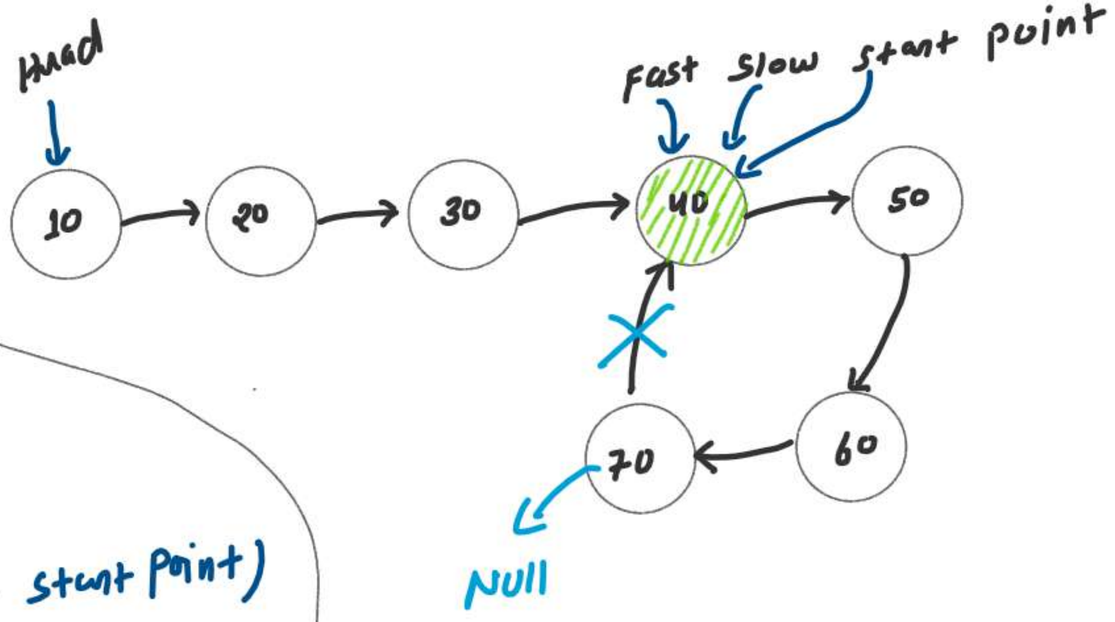
Approach 1: *Fast and slow algorithm*

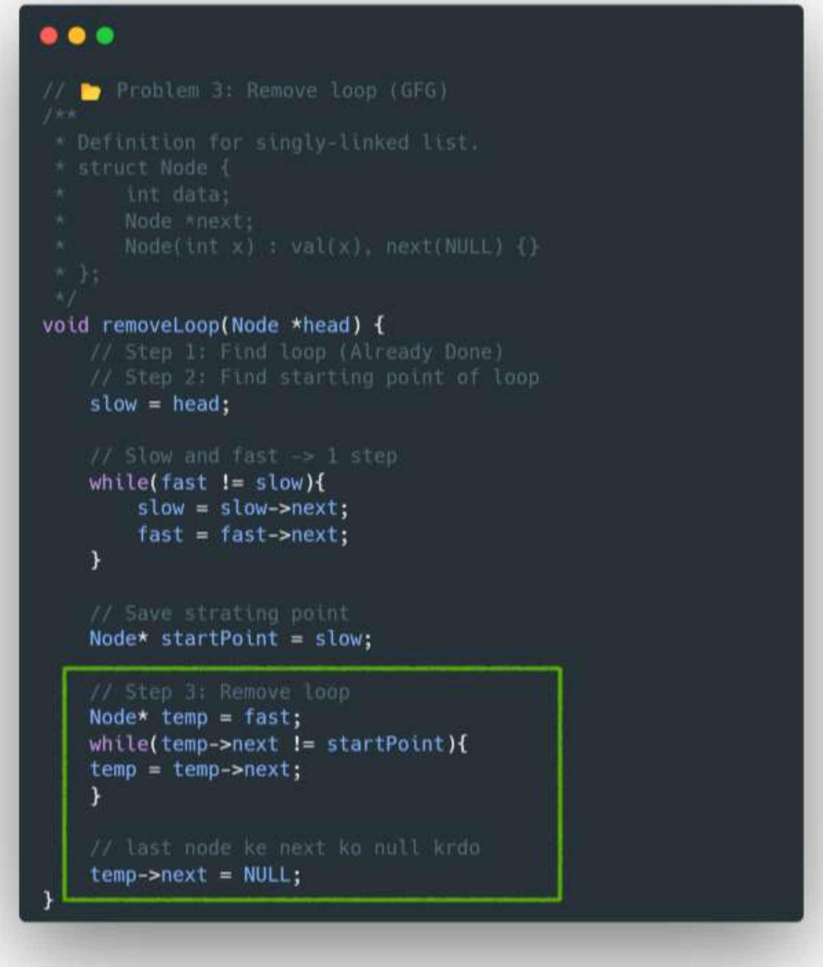
Step:1 Check loop

Step:2 Find starting point

Step:3

```
Node* startPoint = slow;
Node* temp = Fast;
while( temp->next != startPoint)
  {        temp = temp->next;
  }
temp->next = NULL;
```





```
// 📁 Problem 3: Remove loop (GFG)
/**
 * Definition for singly-linked list.
 * struct Node {
 *     int data;
 *     Node *next;
 *     Node(int x) : val(x), next(NULL) {}
 * };
 */
void removeLoop(Node *head) {
    // Step 1: Find loop (Already Done)
    // Step 2: Find starting point of loop
    slow = head;

    // Slow and fast -> 1 step
    while(fast != slow){
        slow = slow->next;
        fast = fast->next;
    }

    // Save strating point
    Node* startPoint = slow;

    // Step 3: Remove loop
    Node* temp = fast;
    while(temp->next != startPoint){
    temp = temp->next;
    }

    // last node ke next ko null krdo
    temp->next = NULL;
}
```

T.C.

step:1 O(N)

step:2 O(N) → + => Overall T.C. **O(N)**

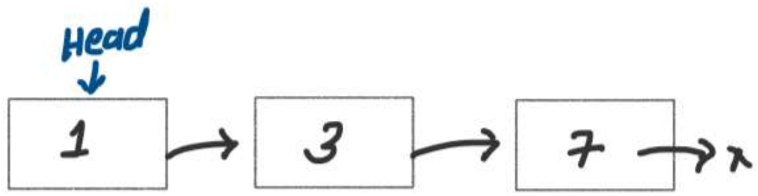
step:3 O(N)

S.C.

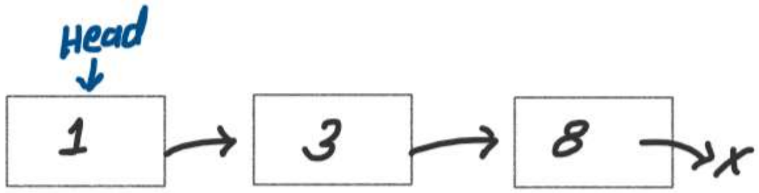
O(1)

Problem 4: Add 1 to a linked list (GFG)

Ex 1

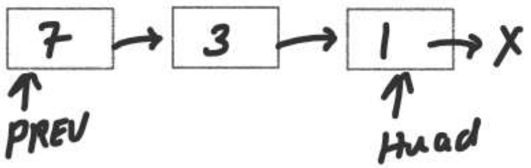


output

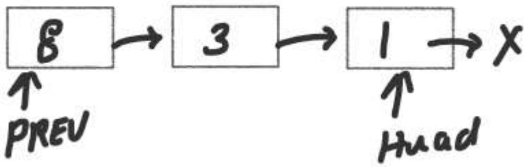


Approach

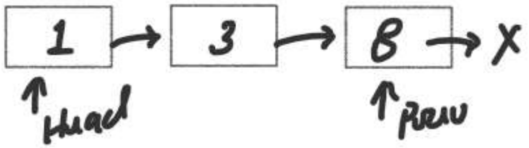
Step 1 Reverse



Step 2 Add 1



Step 3 Reverse

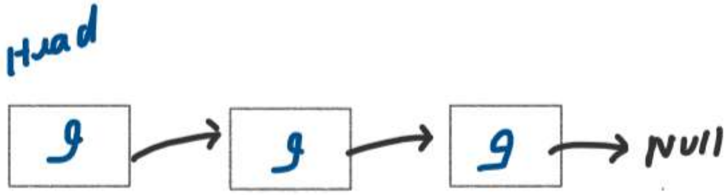


$$\begin{array}{r} 1\,3\,7 \\ +\ \boxed{1} \\ \hline 1\,3\,8 \\ \hline \end{array}$$

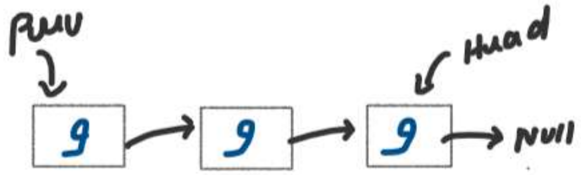
$$\begin{array}{r} 1 \\ 9\,9 \\ +\ 1 \\ \hline 1\,0\,0 \\ \hline \end{array}$$

$$\begin{array}{r} 1\,1 \\ 1\,9\,9 \\ +\ \ 1 \\ \hline 2\,0\,0 \\ \hline \end{array}$$

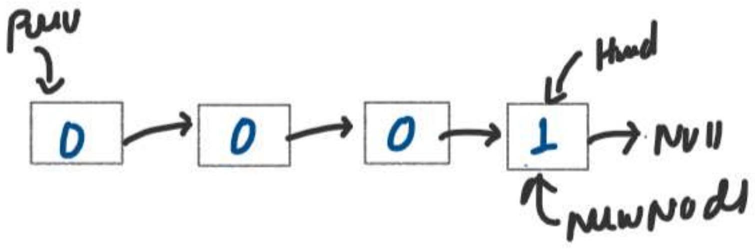
EX2



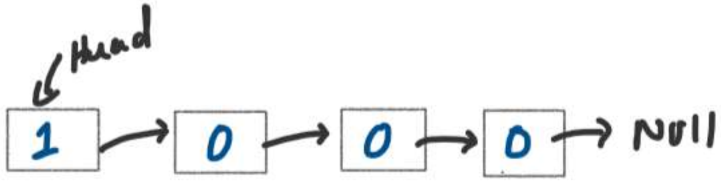
Step:1



Step:2

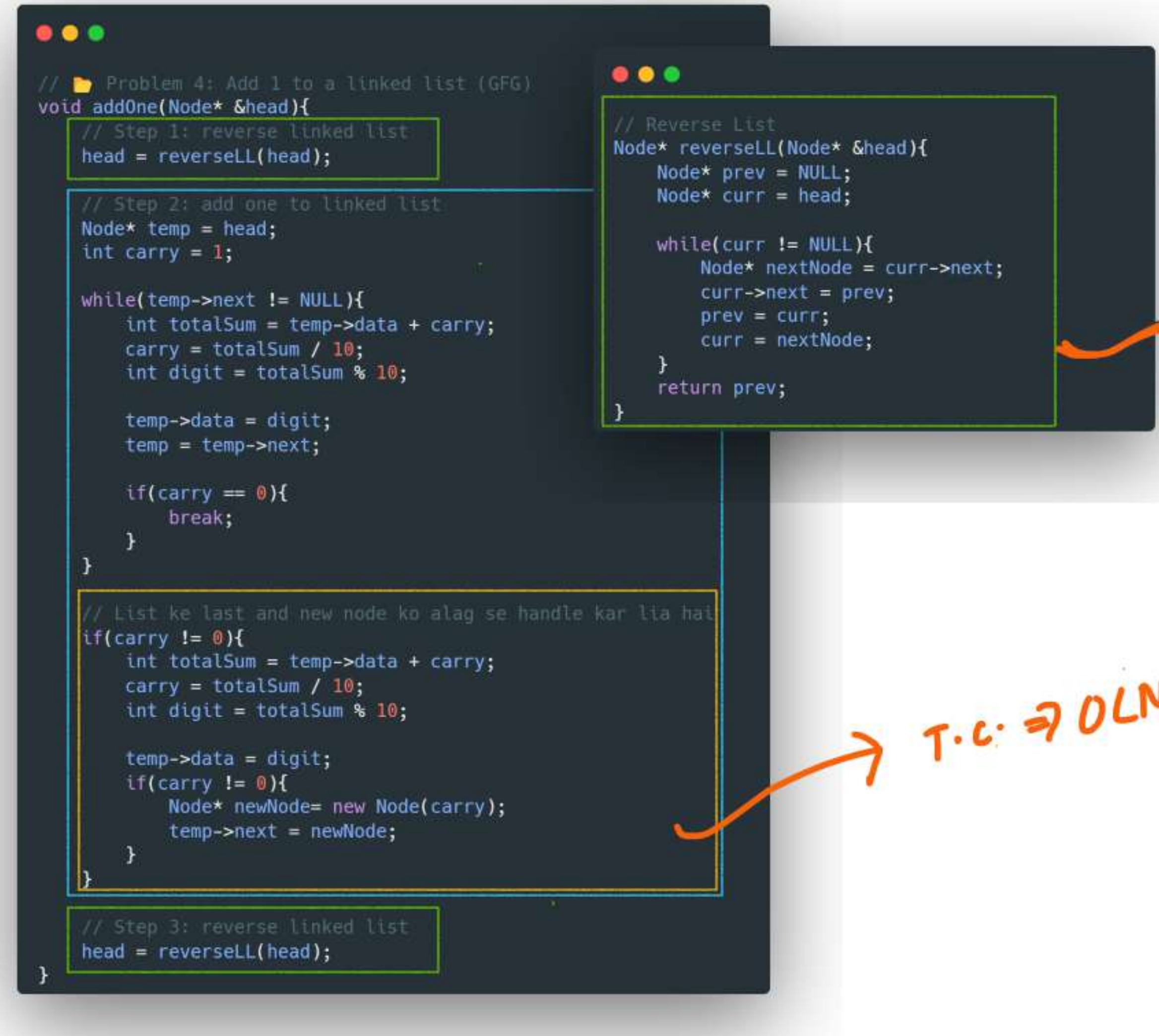


Step:3



if(carry != 0 && List Khatam Hogyi Ho)

→ NewNode Create Karenge



```
// 📁 Problem 4: Add 1 to a linked list (GFG)
void addOne(Node* &head){
    // Step 1: reverse linked list
    head = reverseLL(head);

    // Step 2: add one to linked list
    Node* temp = head;
    int carry = 1;

    while(temp->next != NULL){
        int totalSum = temp->data + carry;
        carry = totalSum / 10;
        int digit = totalSum % 10;

        temp->data = digit;
        temp = temp->next;

        if(carry == 0){
            break;
        }
    }

    // List ke last and new node ko alag se handle kar lia hai
    if(carry != 0){
        int totalSum = temp->data + carry;
        carry = totalSum / 10;
        int digit = totalSum % 10;

        temp->data = digit;
        if(carry != 0){
            Node* newNode= new Node(carry);
            temp->next = newNode;
        }
    }

    // Step 3: reverse linked list
    head = reverseLL(head);
}
```

```
// Reverse List
Node* reverseLL(Node* &head){
    Node* prev = NULL;
    Node* curr = head;

    while(curr != NULL){
        Node* nextNode = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextNode;
    }
    return prev;
}
```

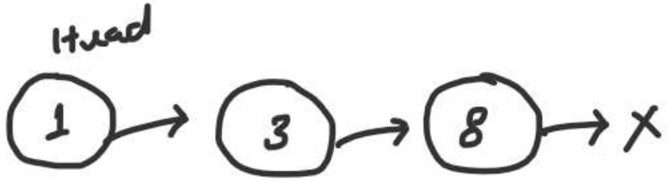
T.C. = O(N)

T.C. ⇒ O(N)

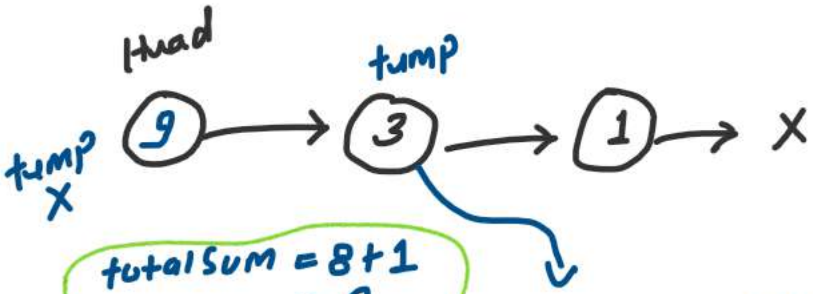
T.C. ⇒ O(N) + O(N)

⇒ O(N)

Ex 1



Step1



Step2

+ (carry=1)

totalSum = 8+1
= 9

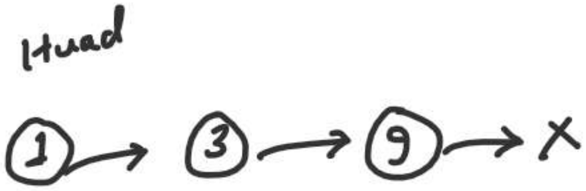
Digit = 9 % 10
= 9

carry = 9 / 10
= 0

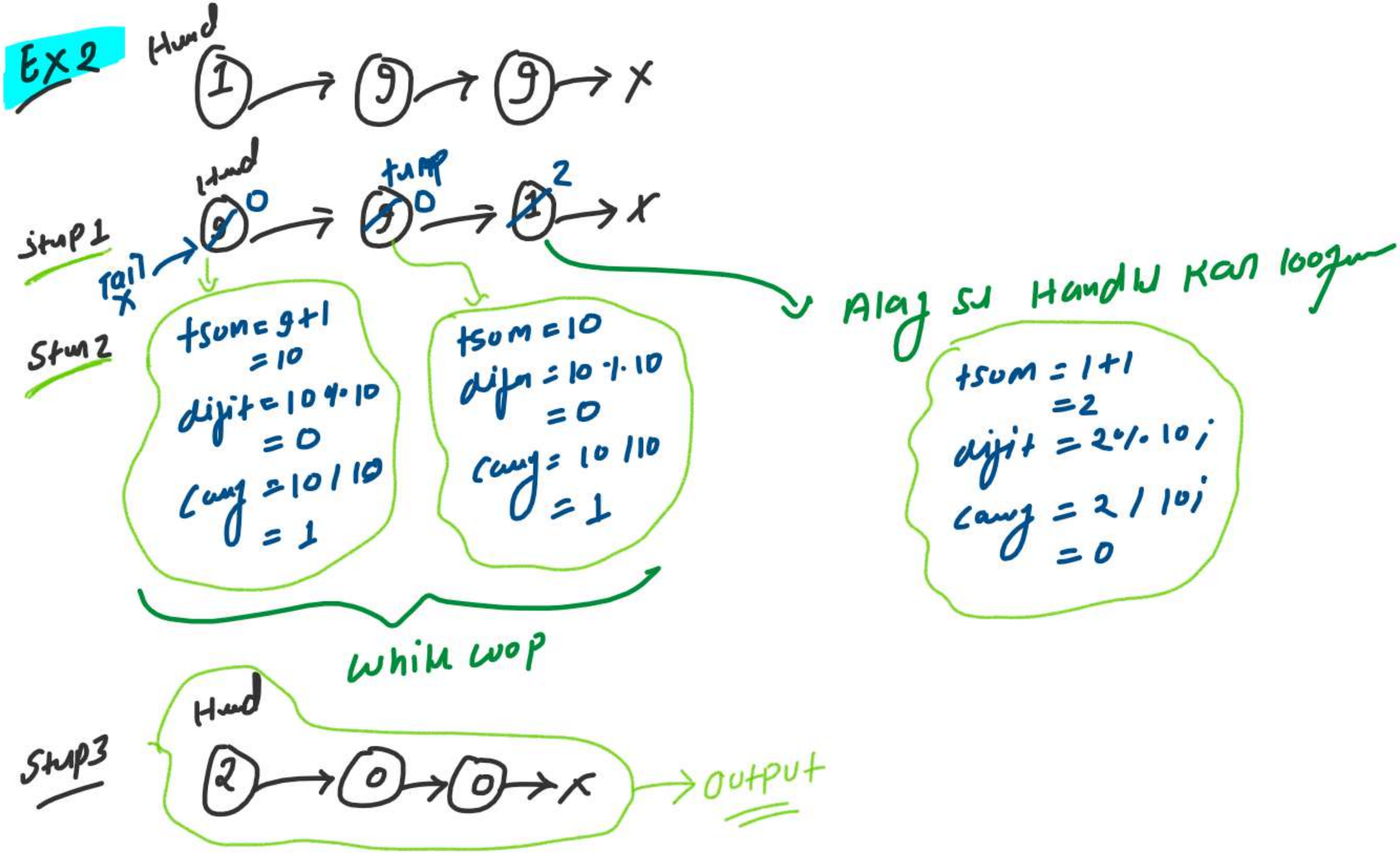
(carry == 0)

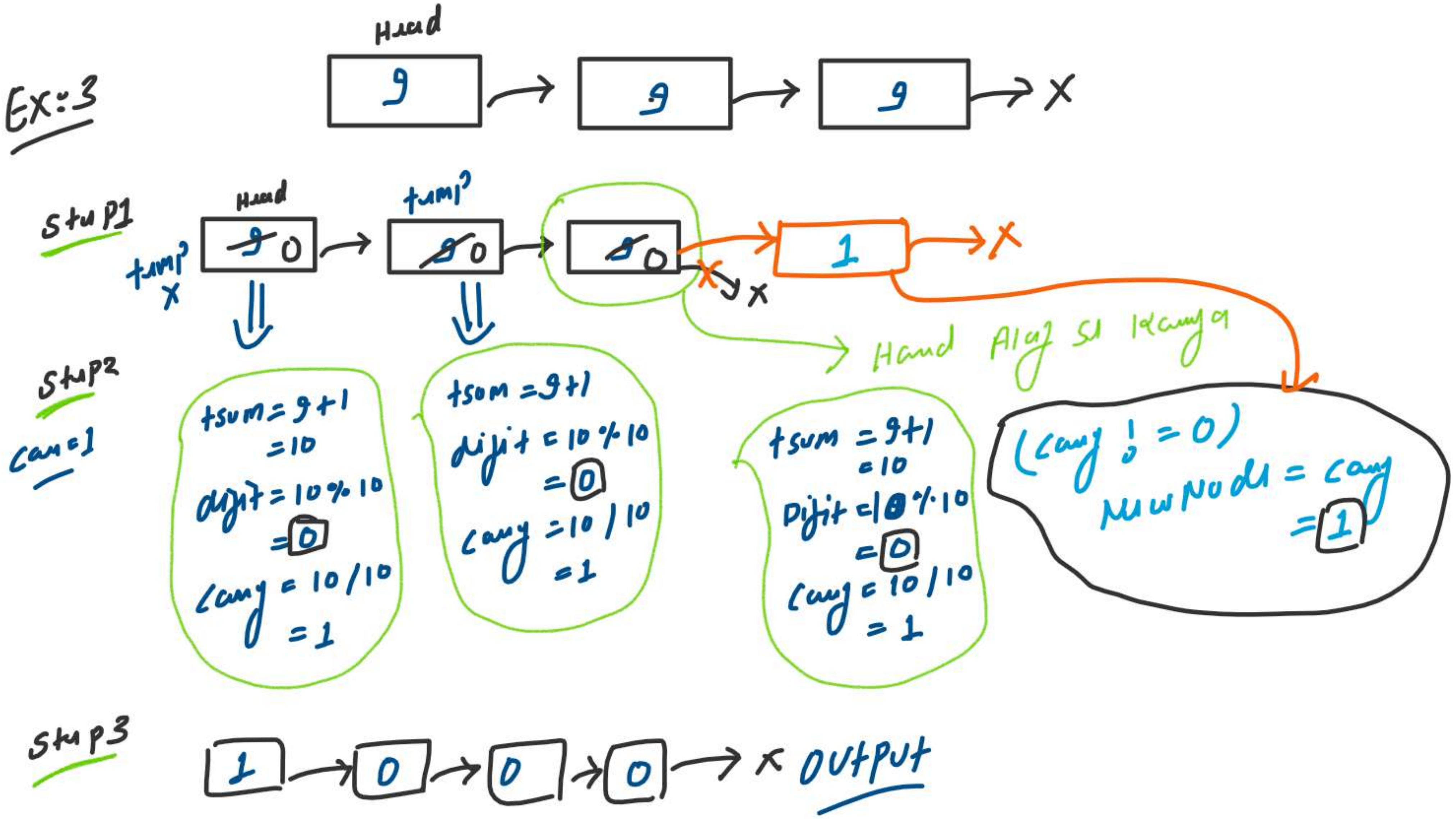
Break

Step3



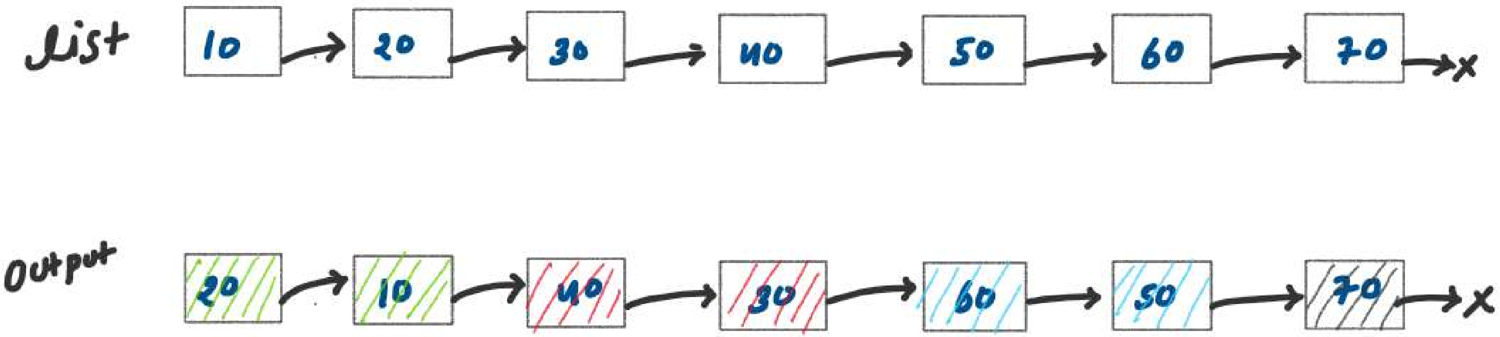
Output


Ex 2
Head
1 → 9 → 9 → X
step 1
Head
temp
~~9~~ 0 → ~~9~~ 0 → ~~1~~ 2 → X
tail
X
Step 2
tsum = 9+1
= 10
digit = 10 % 10
= 0
carry = 10 / 10
= 1
tsum = 10
digit = 10 % 10
= 0
carry = 10 / 10
= 1
while loop
Alag se Handle kar loop me
tsum = 1+1
= 2
digit = 2 % 10;
carry = 2 / 10;
= 0
Step 3
Head
2 → 0 → 0 → X
output


Ex:3
Head
9
9
9
X
Step1
Head
temp
temp
X
~~9~~ 0
~~9~~ 0
~~9~~ 0
1
X
X
Head Alag Se Karega
Step2
carry=1
tsum = 9+1
=10
digit = 10%10
=0
carry = 10/10
=1
tsum = 9+1
digit = 10%10
=0
carry = 10/10
=1
tsum = 9+1
=10
Digit = 10%10
=0
carry = 10/10
= 1
(carry != 0)
NewNode = carry
=1
Step3
1
0
0
0
X
output

Problem 5: Reverse Nodes in K-Group (Leetcode-25)

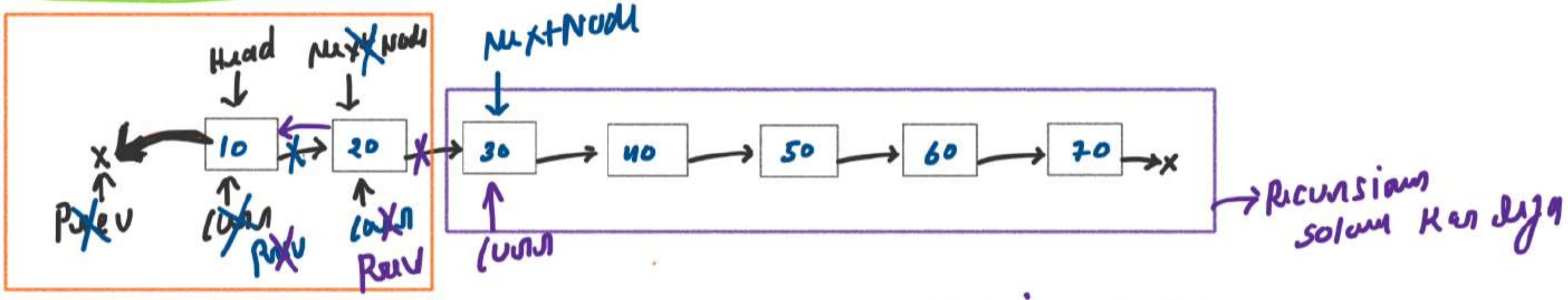
Input   k = 2



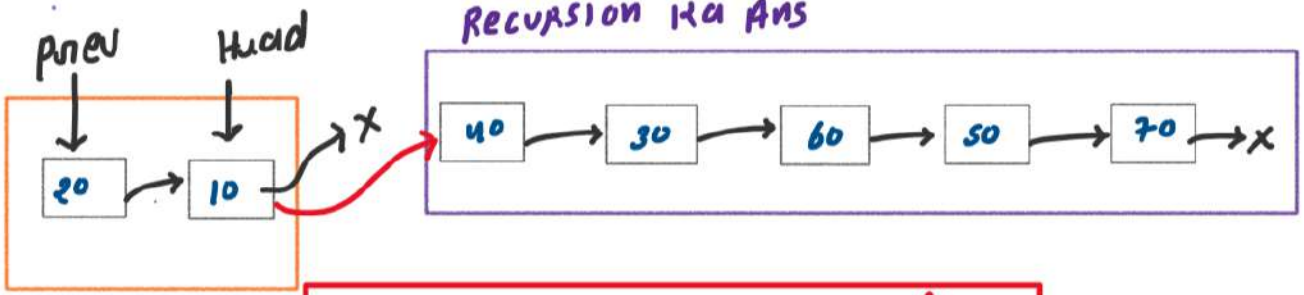
Approach 1: Recursive approach

$\boxed{K=2}$

1 call hum solve kar lenge →



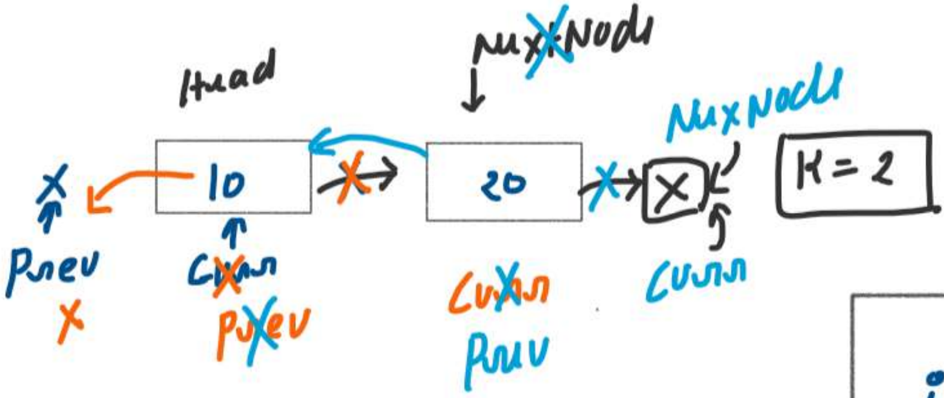
```
Node* prev = Null;
Node* curr = Head;
Node* nextNode = curr→next;
int position = 0;
while( position < K) {
        nextNod = curr→next;
        curr→next = prev;
        prev = curr;
        curr = nextNode;
        position++;
}
```

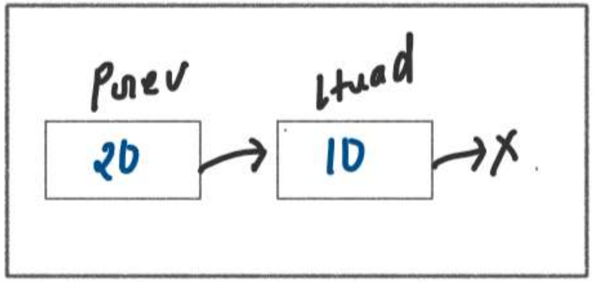


Bhaiya NE In Do lines MAIN 4 Times GALTI KI thi

Head → next = Recursion ka Ans
return prev

Ex





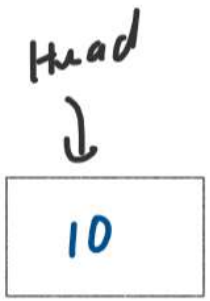
1 case solve by us

RECUR-SION

```
if ( nextNode != Null) {
    recursionKaAns = f(nextNode , K);
  ** head -> next = recursionKaAns;
}
** return Prev
```

Ex

single



K=2

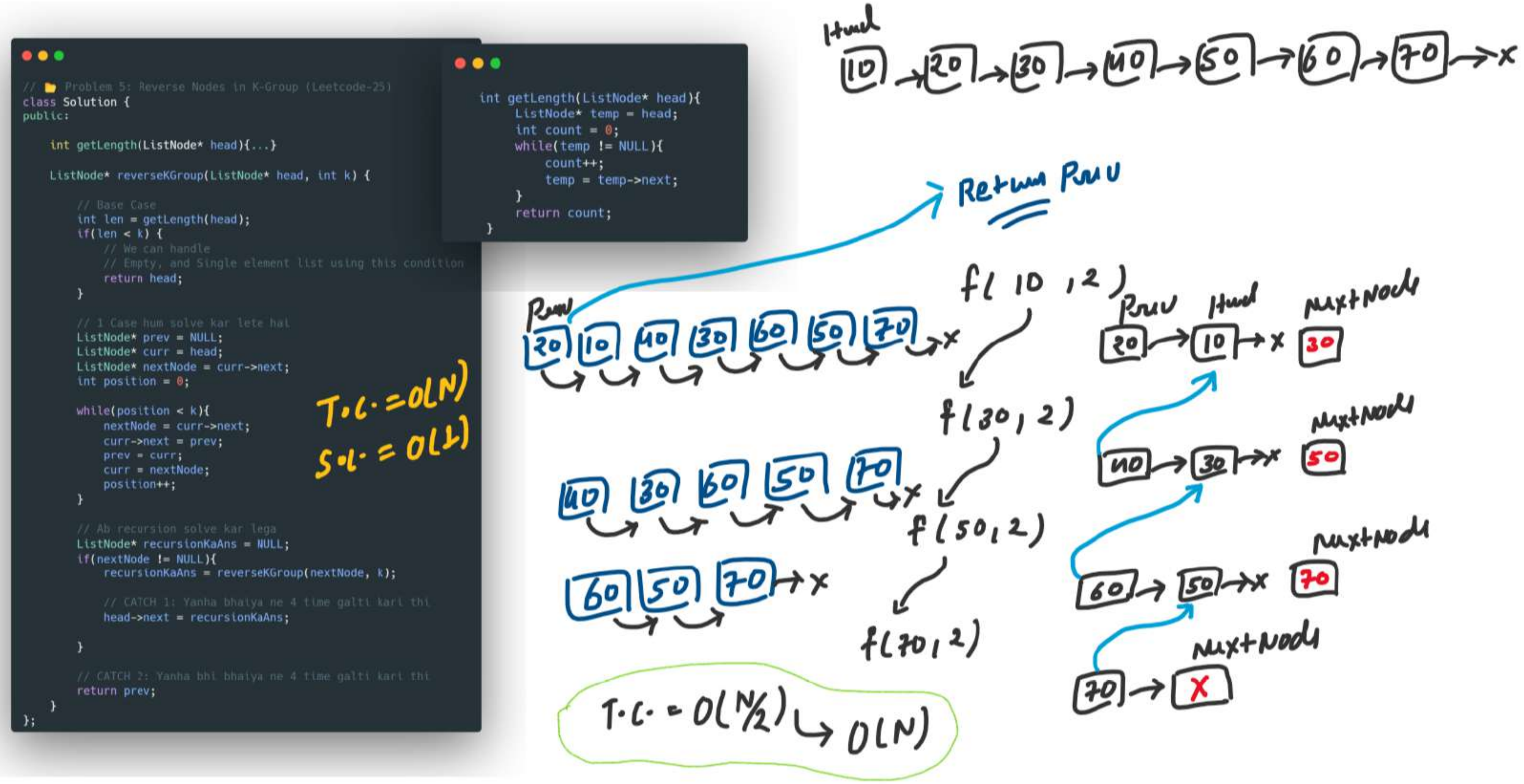
length of list = 1

K-group = 2

BASE CASE

```
if ( length < K )
    ↳ return Head
```

we can handle single and Empty list so dont we need to write separately.



```
// 📁 Problem 5: Reverse Nodes in K-Group (Leetcode-25)
class Solution {
public:

    int getLength(ListNode* head){...}

    ListNode* reverseKGroup(ListNode* head, int k) {

        // Base Case
        int len = getLength(head);
        if(len < k) {
            // We can handle
            // Empty, and Single element list using this condition
            return head;
        }

        // 1 Case hum solve kar lete hai
        ListNode* prev = NULL;
        ListNode* curr = head;
        ListNode* nextNode = curr->next;
        int position = 0;

        while(position < k){
            nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
            position++;
        }

        // Ab recursion solve kar lega
        ListNode* recursionKaAns = NULL;
        if(nextNode != NULL){
            recursionKaAns = reverseKGroup(nextNode, k);

            // CATCH 1: Yanha bhaiya ne 4 time galti kari thi
            head->next = recursionKaAns;

        }

        // CATCH 2: Yanha bhi bhaiya ne 4 time galti kari thi
        return prev;
    }
};
```

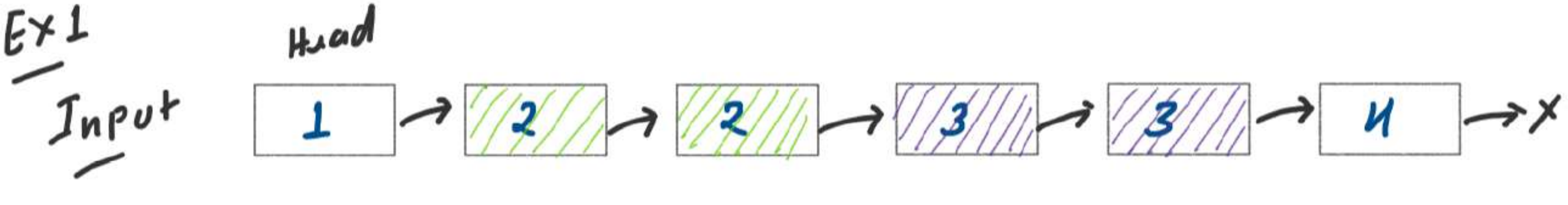
```
    int getLength(ListNode* head){
        ListNode* temp = head;
        int count = 0;
        while(temp != NULL){
            count++;
            temp = temp->next;
        }
        return count;
    }
```

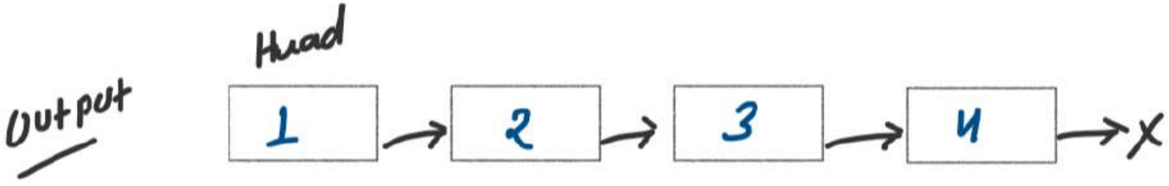
T.C. = O(N)
S.C. = O(1)

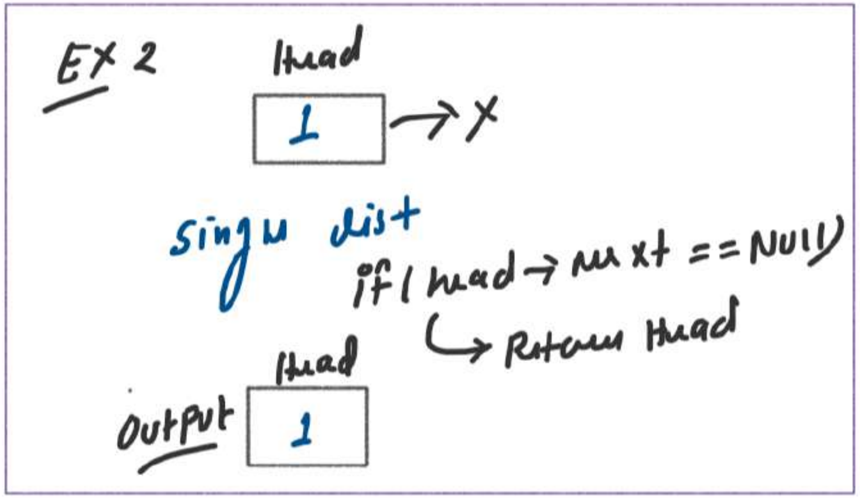
T.C. = O(N/2) ↳ O(N)

Problem 6: Remove Duplicates from sorted linked list (Leetcode-83)

Ex 1

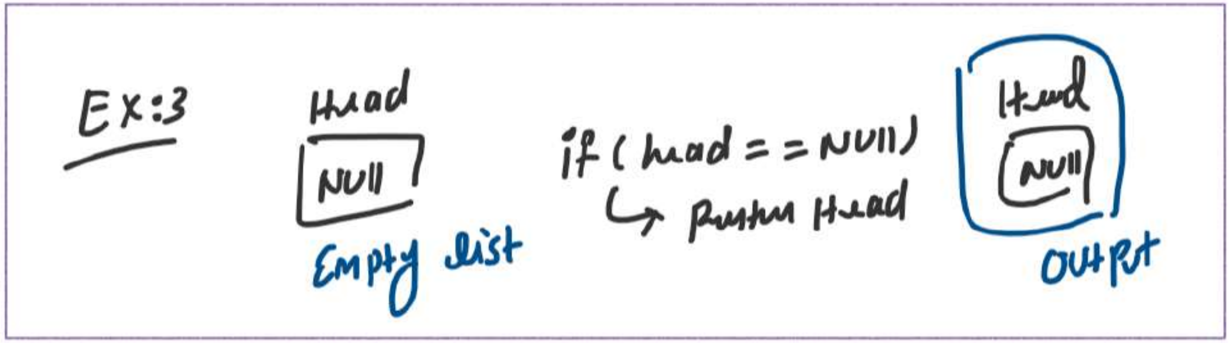






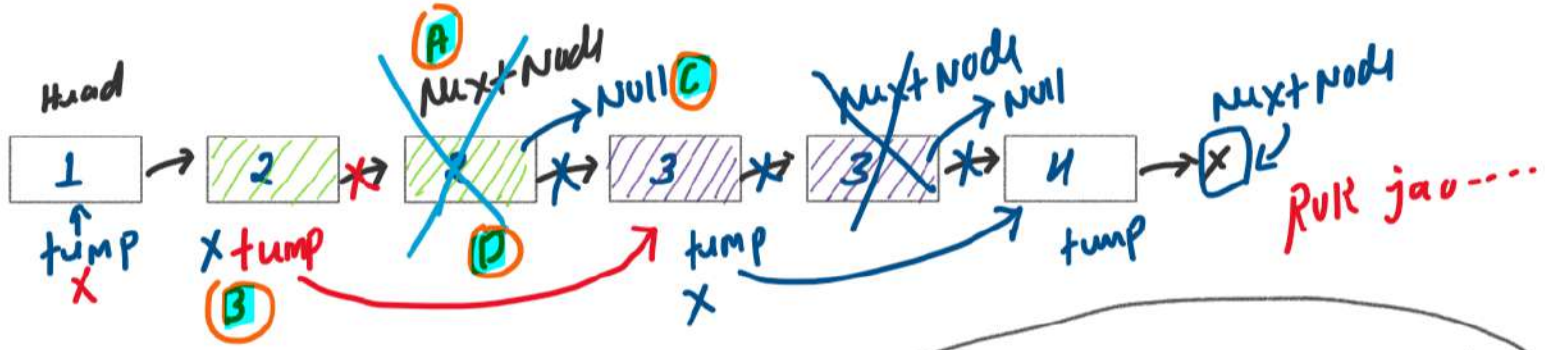
Single list

if (head → next == NULL)
↳ Return Head



if (head == NULL)
↳ Return Head

Ex1



```
Node* temp=head;
while(temp != Null){

  if(temp->next != Null &&
     temp->Data == temp->next->data)
   {
      // Remove Duplicate
   }
  else{
       temp = temp->next;
   }
}
return Head;
```

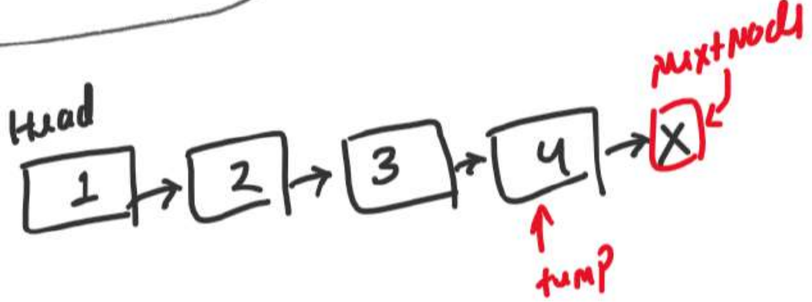
A Node* nextNode = temp -> next;

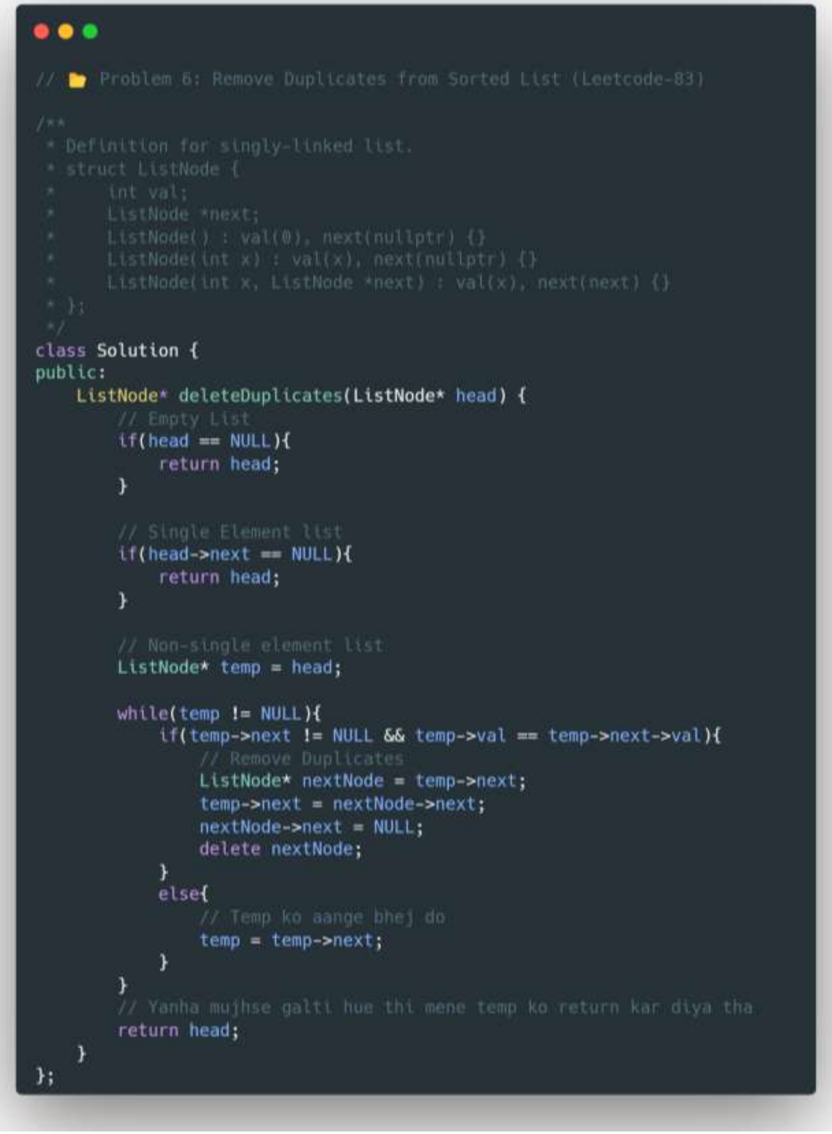
B temp = nextNode -> next;

C nextNode -> next = Null;

D delete nextNode;

Output





```
// 📁 Problem 6: Remove Duplicates from Sorted List (Leetcode-83)

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        // Empty List
        if(head == NULL){
            return head;
        }

        // Single Element list
        if(head->next == NULL){
            return head;
        }

        // Non-single element list
        ListNode* temp = head;

        while(temp != NULL){
            if(temp->next != NULL && temp->val == temp->next->val){
                // Remove Duplicates
                ListNode* nextNode = temp->next;
                temp->next = nextNode->next;
                nextNode->next = NULL;
                delete nextNode;
            }
            else{
                // Temp ko aange bhej do
                temp = temp->next;
            }
        }
        // Yanha mujhse galti hue thi mene temp ko return kar diya tha
        return head;
    }
};
```

T.C. = O(N)

where, N is number of nodes

S.C. = O(1)