- When doing curr->next->next
  See if NULL->next ही नही हो रहा ना ! (occurs when curr is at last node)

Points to Remember

- edge case checking is very important
  if(head == NULL || head->next == NULL) return head;
- NULL->next Segmentation fault
- last pointer का next NULL है तो उस edge case को देखो
- Dummy Node बनाओ तो Question Solve करो करो, edge case में edge होगा ।
  (dum->next करोगे Question easy होगा)
- Search for Pattern

(I) Delete a node in SLL (See the edge case Written)

1→3→4 x=3 (position)
o/p 1→3

Format for all Question

```
Node * f(Node *head, int x)
{
    if(x==1)
    {  head = head->next;
       return head;
    }                                  } when 1st node is deleted

    Node *curr = head;
    for(int i=1; i<=(x-2) && curr!=NULL; i++)
        curr = curr->next;

    if(curr==NULL)                     } x>(size of LL)
        return head;
    if(curr->next->next == NULL)
    {   curr->next = NULL;             } when last node is deleted
        return head;
    }
    curr->next = curr->next->next;     } others
    return head;
}
```

when x>(size of LL)

(II) Remove every K^{th} Node

1-2-3-4-5-6-7-8 k=3

o/p : 1→2→4→5→7→8

```
Node *f(Node *head, int k)
{   if(k==0)                          } Write edge case at top
        return head;
    if(k==1)
        return NULL;
    Node *cur = head;
    int c=1;
    while(cur!=NULL && cur->next!=NULL)
    {
        if(c== k-1)
        {   cur->next = cur->next->next;
            c=1;
        }
        else
            c++;
        cur = cur->next;
    }
    return head;
}
```

(III) Find middle element in a LL

Slow & fast pointer

1→2→3→4→5 | 1→2→3→4→5→6

o/p 3 | 4

```
int f(Node *head)
{   Node *slow = head, *fast = head;
    while(fast!=NULL && fast->next!=NULL)
    {   slow = slow->next;
        fast = fast->next->next;
    }
    return slow->data;
}
```

2nd Method

simply count size of LL (say c=5) $5/2$ tak chala dt for loop aur pahl $(5/2)^{th}$ element.

(IV) N^{th} Node from end of LL

1→2→3→4→5→6 n=2
o/p → 5

1→2→3→4 n=1
o/p → 4

1 n=4
o/p 1

```
int f(Node *head, int n)
{
    int c=1;
    Node *run=head;
    while(run->next!=NULL)
    {   c++;
        run=run->next;
    }
    if(n>c) return -1;
    for(int i=0; i<c-n; i++)
        head=head->next;
    return head->data;
}
```

O(2n)

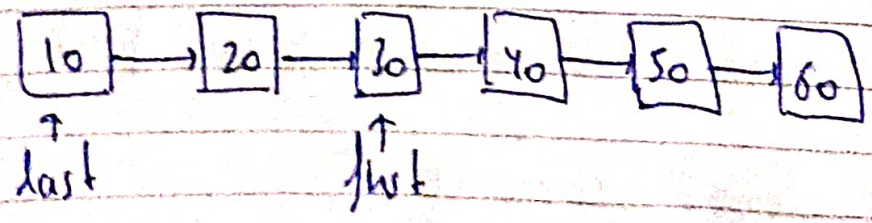
```
int f(Node *head, int n)
{                          last
    Node *fast=head, *second=head;
    for(int i=0; i<n; i++)
    {   if(fast==NULL) return -1;
        fast=fast->next;
    }
    while(fast!=NULL)
    {   fast=fast->next;
        last=last->next;
    }
    return last->data;
}
```

O(n)

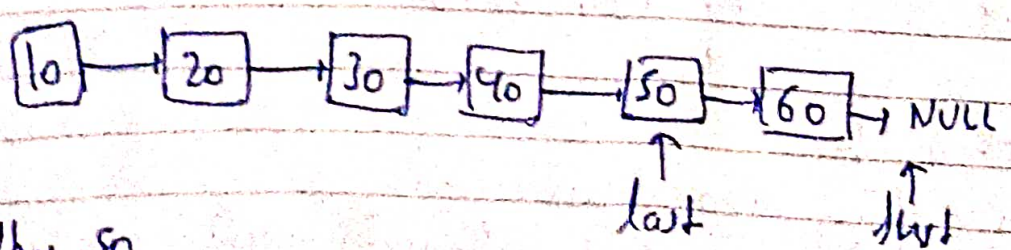
2 pointer

say n=2

after fast for loop



when while loop complete



o/p → 50

Imp

(V) Reverse a SLL :

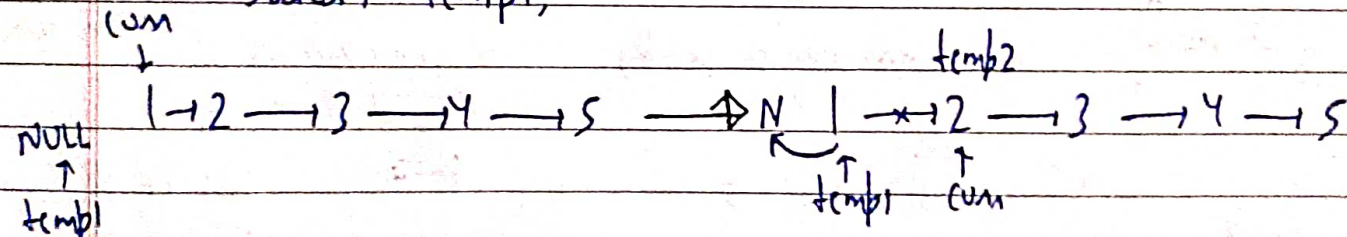
1→2→3→4→5→6 | o/p → 6→5→4→3→2→1

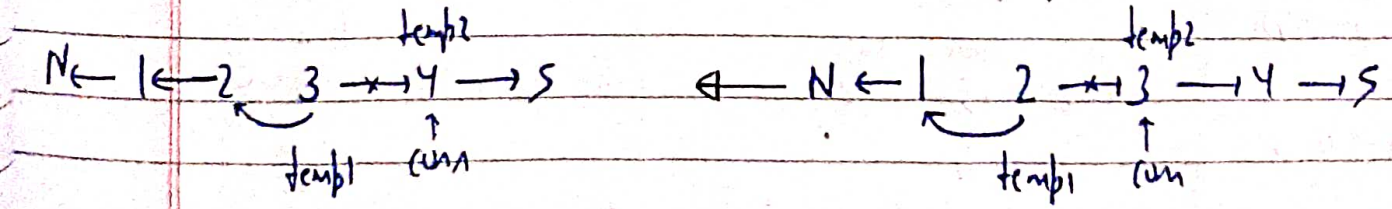
naive → use vector, store value → now use * store last value of vector

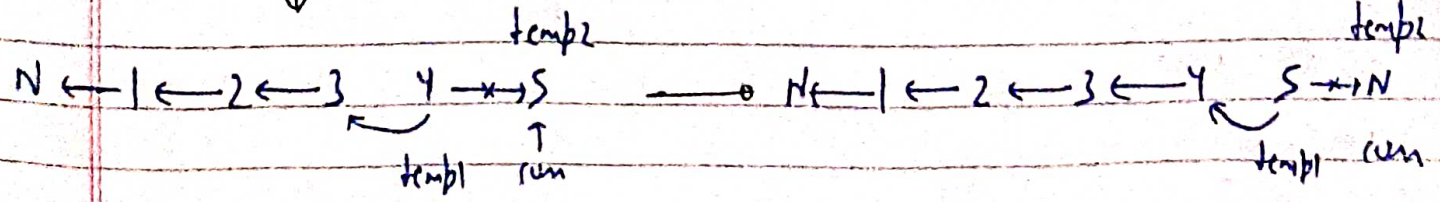
a) Iterative :

```
Node *f(Node *head)
{
    Node *temp1 = NULL, *curr = head, *temp2;
    while (curr != NULL)
    {
        temp2 = curr->next;
        curr->next = temp1;
        temp1 = curr;
        curr = temp2;
    }
    return temp1;
}
```

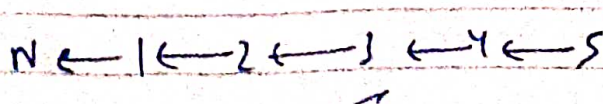
You may use edge case
if(head == NULL or head->next == NULL)
return head;







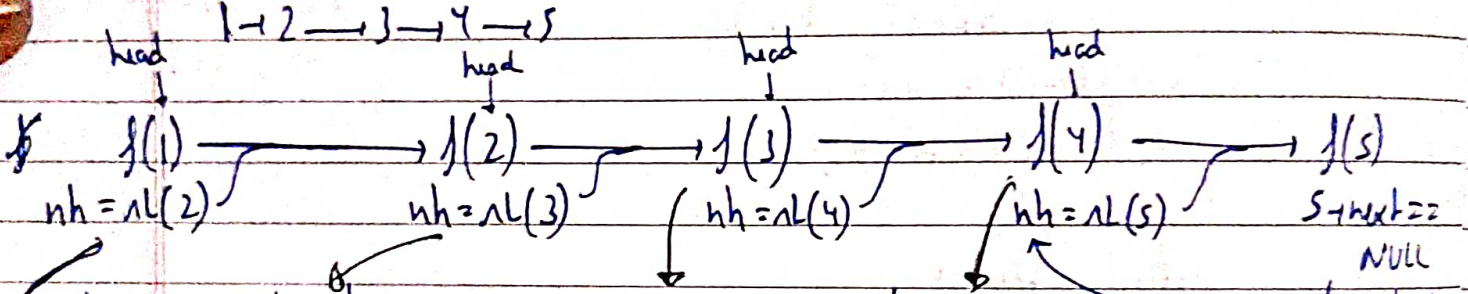
now curr == NULL
return temp



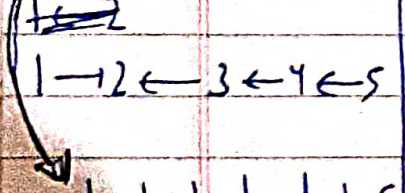
nh = new head
nl = newlist
lt = left tail

b) Recursive (खान से)

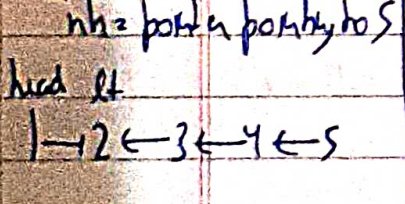
```
Node *newtelist(Node *head)
{
    if(head==NULL || head->next==NULL) return head;
    Node *new_head = newreselist(head->next);
    Node *left_tail = head->next;
    left_tail->next = head;
    head->next = NULL;
    return new_head;
}
```

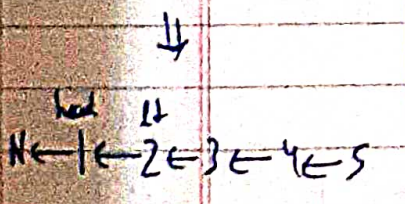


Similarly 3 will bond to 2



nh = pointer pointing to 5

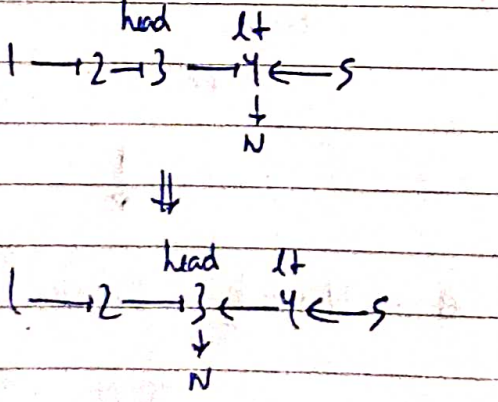




return nh

finally returns

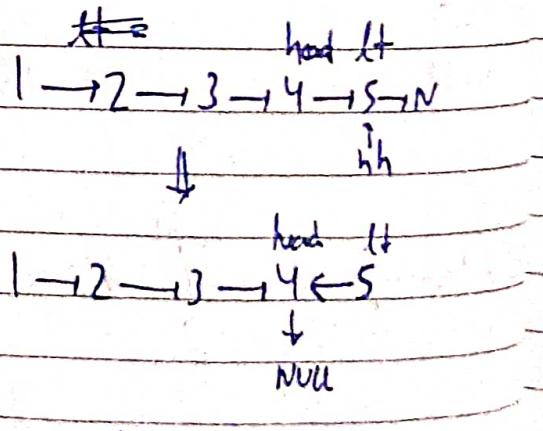
nh = pointer pointing to 5



return nh

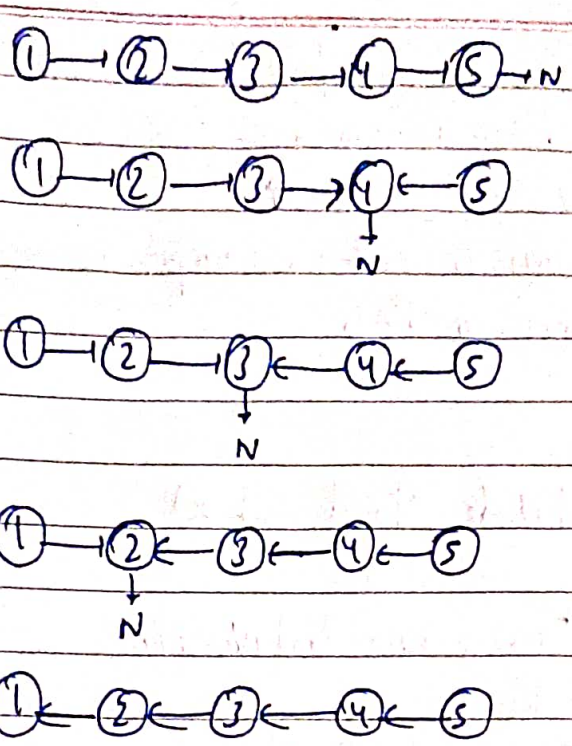
now nh = pointer pointing to 5

return head



return nh.

In Short



(VI) Remove duplicate element from Sorted LL :

2→2→4→4→5→6

o/p 2→4→5→6

```
Node *fn(Node *head)
{
    Node *curr = head;
    while(curr->next != NULL)
    {
        if(curr->data == curr->next->data)
            curr->next = curr->next->next;
        else
            curr = curr->next;
    }
    return head;
}
```

ऐसा करते वक्त एक बात ध्यान रखना (To delete not used node using temp pointer)

eg: Node * temp = curr->next;
curr->next = curr->next->next;
delete(temp);

curr तभी बढ़ाओ जब (curr->val != curr->next->val)

Hard

(VII) Reverse Nodes in k-Group:

1→2→3→4→5→6→7→8 k=3

o/p 3→2→1→6→5→4→7→8

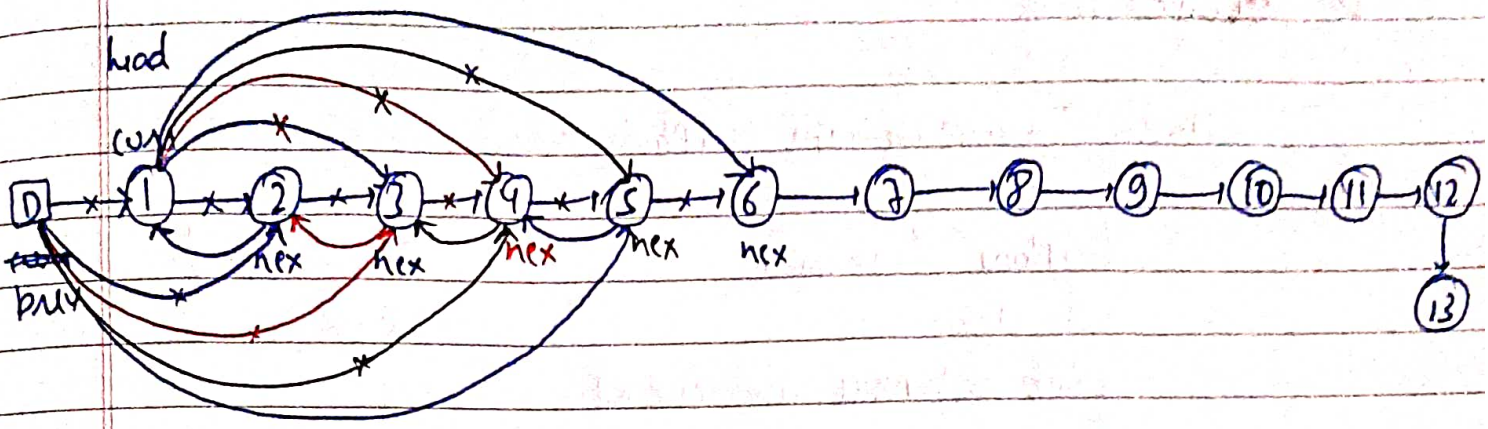
- if the no. of nodes is not multiple of k then left out nodes in the end remain as it is.

a) Iterative

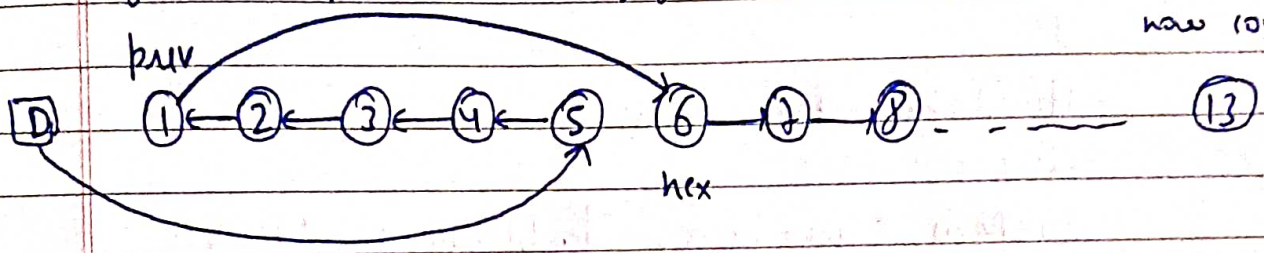
```
ListNode * f(ListNode *head, int k)
{
    ListNode *dum = new ListNode(0);
    dum->next = head;
    ListNode *cur = head, *prev = dum, *nex = dum;
    int count=1;
    while(cur->next!=NULL)
    {   count++;
        cur=cur->next;
    }
    while(count>=k)
    {
        cur = prev->next;
        nex = cur->next;
        for(int i=1; i<k; i++)
        {
            cur->next = nex->next;
            nex->next = prev->next;
            prev->next = nex;
            nex = cur->next;
        }
        prev=cur;
        count -= k;
    }
    return dum->next;
}
```

count = 13 k = 5

while(count >= k) ~~do~~ chal raha तबतक

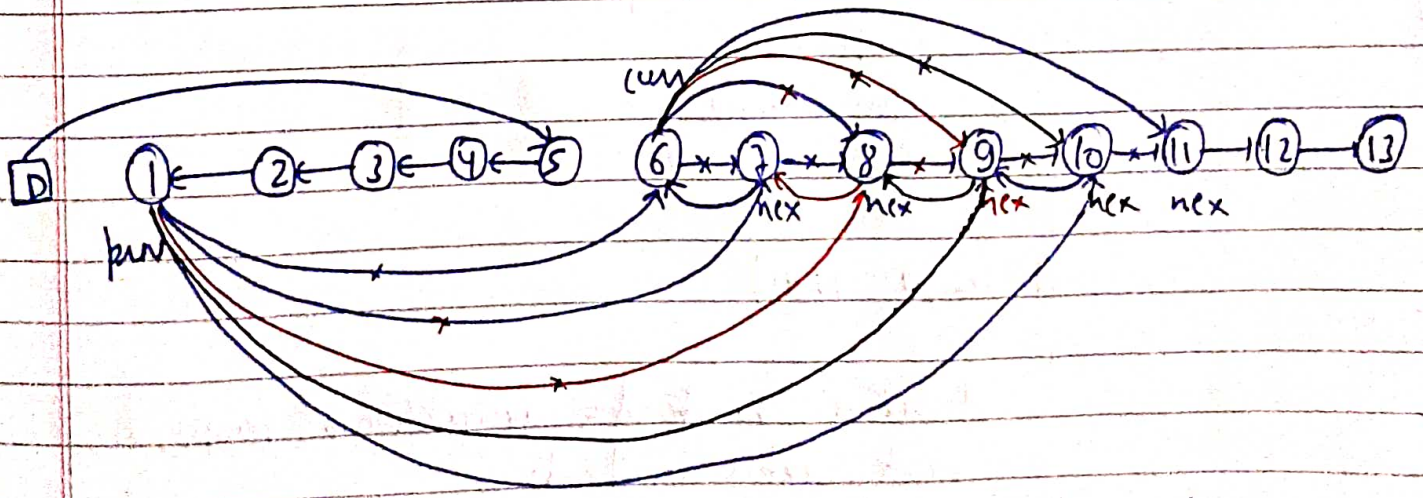


after 1 complete execution of for loop तब prev = cur तब count -= k;
now count = 8

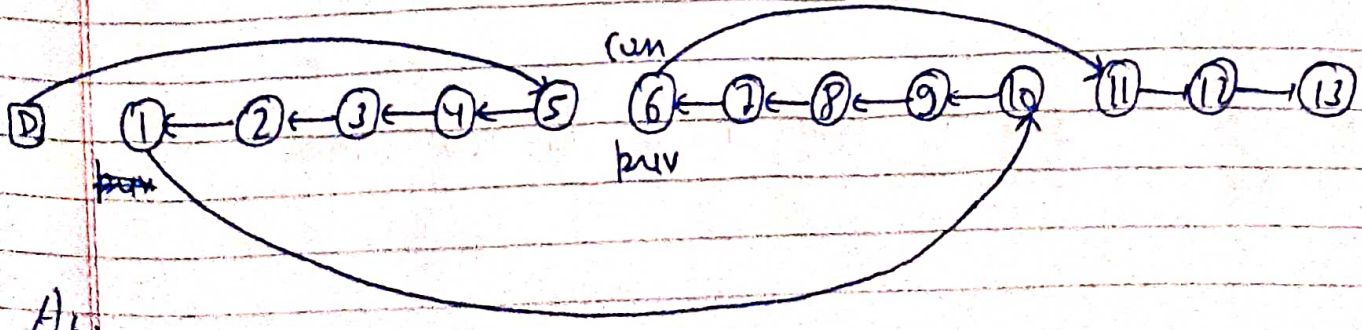


देख लो dum->next answer है।

→ while(count >= k) chal raha तबतक



after 1 complete execution of for loop तब prev = cur तब count -= k
now count = 3



Ans

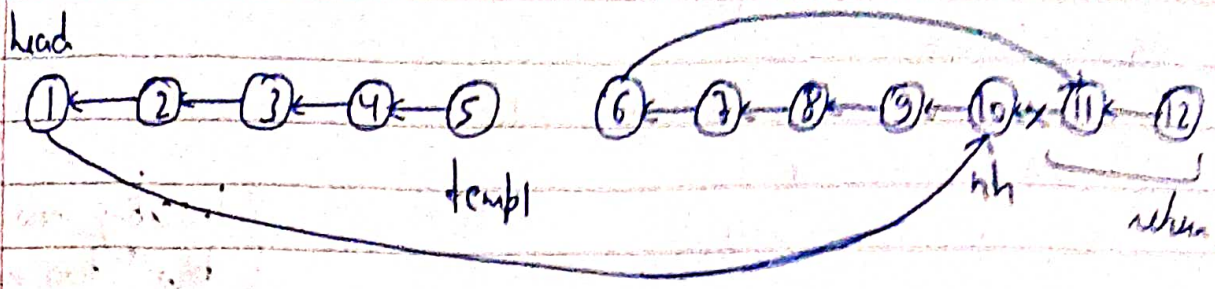
b) Recursive

```
ListNode *reverseKGroup(ListNode *head, int k)
{
        ListNode *t=head;                          } no. of node है उतने
        int cl=0;                                  } k है, तो return head
        while(t!=NULL && cl<k)                     }
        {   cl++;                                  }
            t=t->next;                             }
        }                                          }
        if(cl<k) return head;                      }
        int c=0;
        ListNode *curr=head, *temp1=NULL, *temp2;  } simply reverse all
        while(curr!=NULL && c<k)                   }
        {                                          }
            temp2 = curr->next;                    }
            curr->next = temp1;                    }
            temp1= curr;                           }
            curr= temp2;                           }
            c++;                                   }
        }
        if(temp2!=NULL)
        {
            ListNode *newhead= reverseKGroup(temp2, k);
            head->next = newhead;
        }
        return temp1;
}
```

इसमें पहले यह दो line देख जाएगा

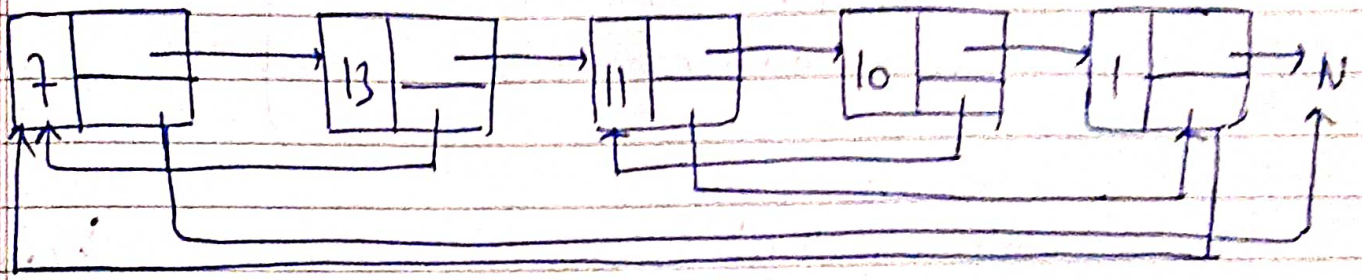
K=5

head → next = nh



★

VIII Copy list with Random Pointer :



return deep copy of LL

a) Hashing (O(n) , S(→ O(n))

```
Node *f(Node *head)
{   unordered_map<Node*, Node*> m;
    int i=0;
    Node *ptr = head;
    while(ptr != NULL)
    {
        m[ptr] = new Node(ptr->val);
        ptr = ptr->next;
    }
    ptr = head;
    while(ptr)
    {
        m[ptr]->next = m[ptr->next];
        m[ptr]->random = m[ptr->random];
        ptr = ptr->next;
    }
    return m[head];
}
```

```
Node *f(Node *head)
{   if(head==NULL) return NULL;
    Node *curr= head;
    while(curr!= NULL)
    {
        Node *temp= new Node(curr->val);
        temp->next = curr->next;
        curr->next = temp;
        curr = curr->next->next;
    }
```
} inserting new node b/w two node

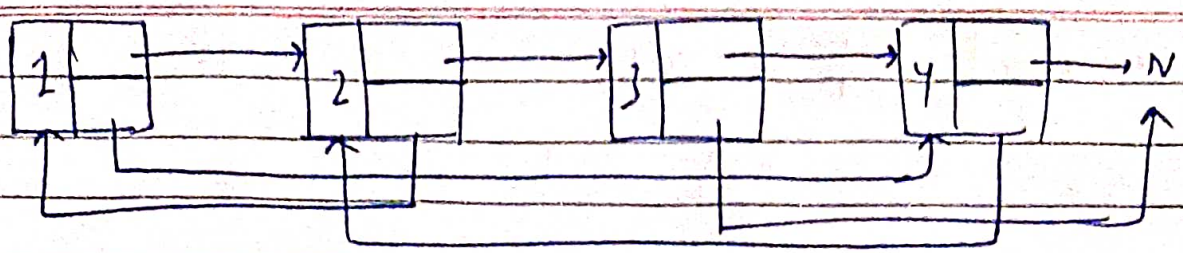
```
    curr= head;
    while(curr!=NULL)
    {
        if(curr->random!=NULL)
            curr->next->random = curr->random->next;
        curr =curr->next->next;
    }
```
} connecting random of newly inserted node

इस line को एक बार dala के देखना

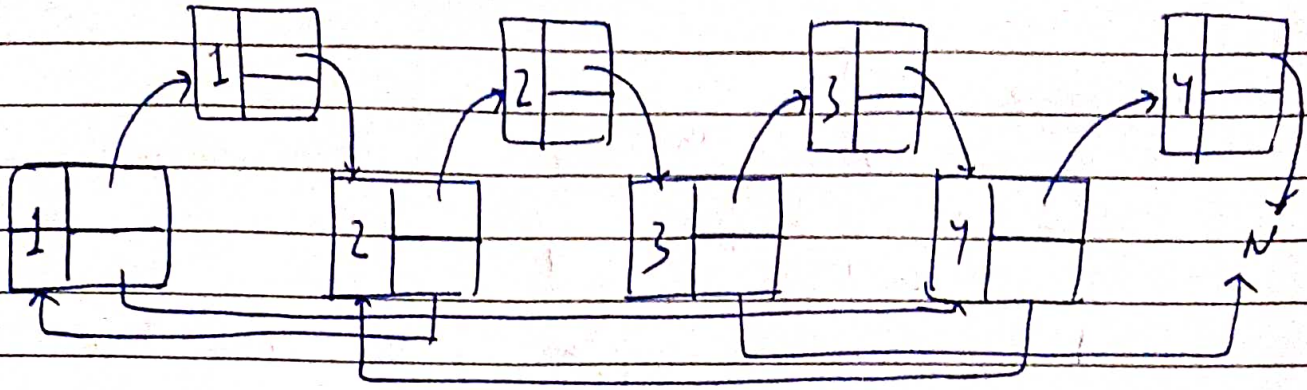
```
    Node *dum= head->next;
    Node *temp= head->next, *front = head->next->next, *back=head;
    curr = head;
    while(front!= NULL)
    {
        back->next = front;
        temp->next = front->next;
        back = front;
        temp= front->next;
        front= front->next->next;
    }
```
} separating original ll & newly ll

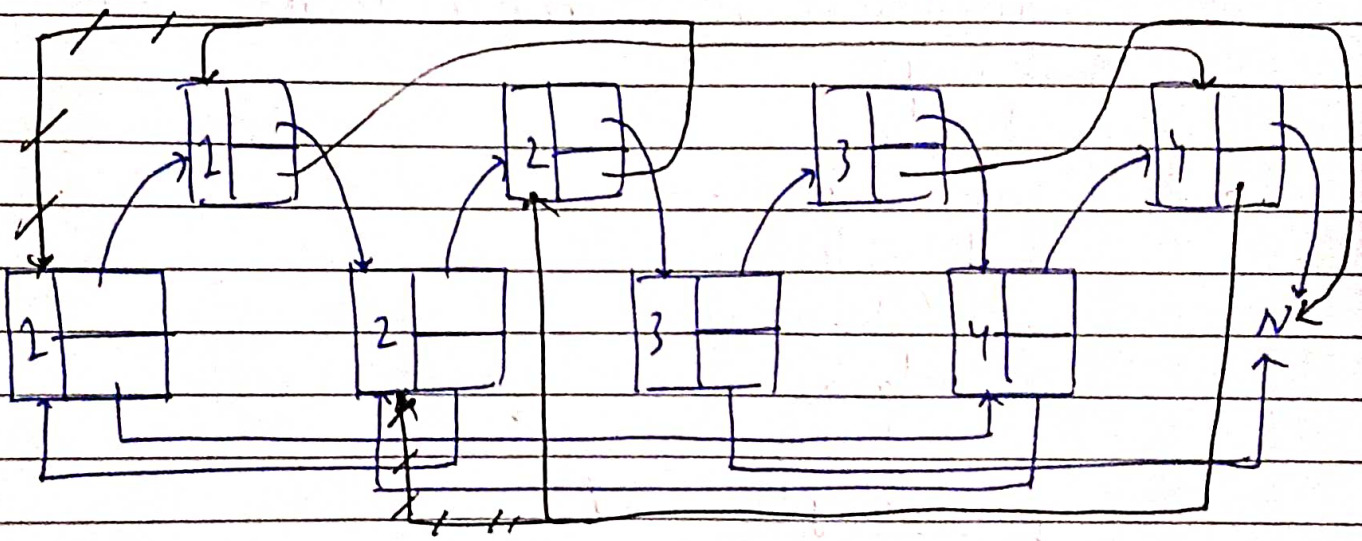
```
    back->next = NULL;
    return dum;
}
```

Original LL

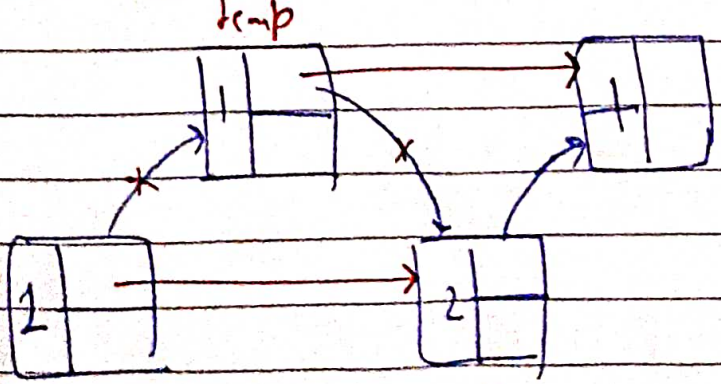


inserting new node in two node



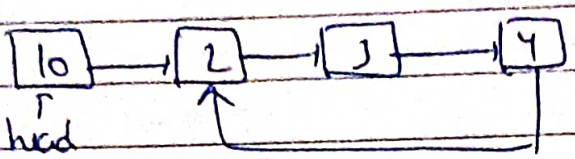


Separation Simply दोनों list separate कर दो



Similarly others

→ Detect loop in a LL

I/P 

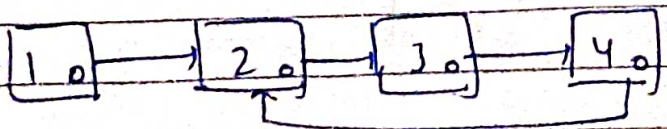
o/p Yes

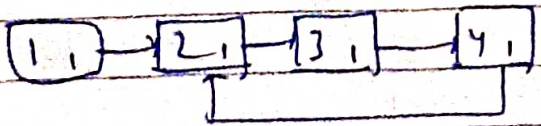
a) Naive $O(n^2)$
- Say we are at i^{th} Node, we store address of ~~next of i~~ (i→next) into temp
- In 2nd for loop we iterate from j=0 to j=i & check if (j address == temp) ~~ad~~ if yes return True.

b) If modification to LL structure is allowed O(n)

```
struct Node{
    int data;
    Node *next;
    bool visited;
    Node(int x)
    { data=x;
      next=NULL;
      visited=false;
    }
};
```

→ Traverse the LL from beginning if any node visited is True then loop exist, while traversing mark the node visited as True.

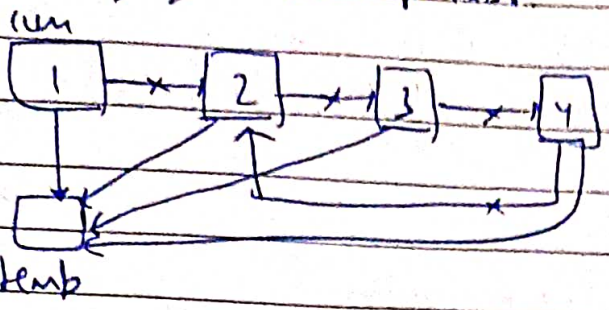
0→false 

after while traversing 

4→next is visited is True ∴ loop exist

c) Modification to linked list pointer O(n)

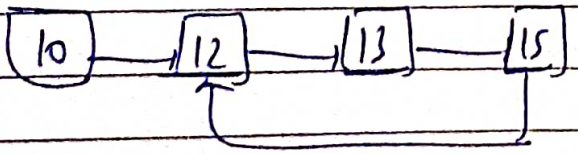
- We use temp as dummy node
We traverse the LL & check if (cur->next) is already pointing to temp node if yes then loop exist.



when cur come at 2 again then 2->next points to temp. ∴ Loop exist.

d) Marking O(n) S(n) → O(n)

```
bool isloop(Node *head)
{   unordered_set<Node *> s;
    for(Node *cur=head; cur!=NULL; cur=cur->next)
    {
        if(s.find(cur) != s.end())
            return true;
        s.insert(cur);
    }
    return false;
}
```



set = { addrss(10), add(12), add(13), add(15) }

Now when cur points to 12 its address is present in set ∴ loop exist.

Floyd's Cycle Detection / Tortoise & hare Algorithm

```
bool f(Node *head)
{   if(head == NULL || head->next == NULL)
        return 0;
    Node *slow = head, *fast = head;
    while(fast != NULL && fast->next != NULL)
    {
        slow = slow->next;
        fast = fast->next->next;
        if(slow == fast)
            return 1;
    }
    return 0;
}
```