Dummy Node Concept :

→ Also called "Sentinel node"

→ ek extra temporary node hota hai, jo usually linked list k start m lagaya jata hai

→ helper node

→ final answer mai dummy return ni hota.

dummy → 1 → 2 → 3 → 4 → NULL

[actual head = dummy → next]

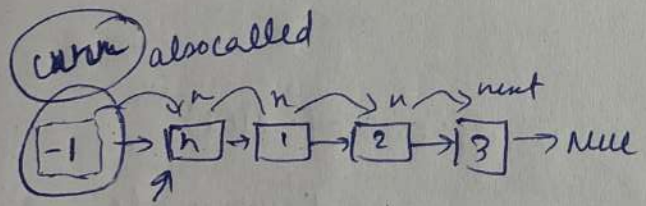
Advantages : head special cases ni likhna
Code short & clean
Edge cases automatically handled.

Eg → without

```
if(head->val == x)
{
    head = head->next;
}
```

with ;

```
ListNode* dummy = new ListNode(-1);
dummy->next = head;
LinkedList* cur = dummy;
while(cur->next)
{
    if(cur->next->val == x)
    {
        cur->next = c->next->next;
        break;
    }
    cur = cur->next;
}
return dummy->next;
```



dummy->next = head

(cur->next)

```
if(cur->next->val = x)
{
}
```

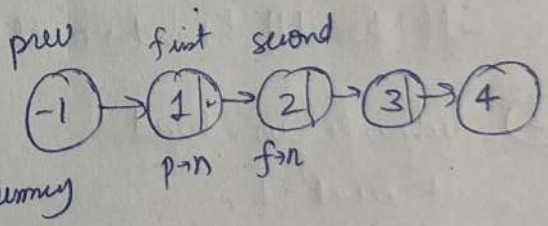
= { dummy helps to remove the edge case for removing the head }

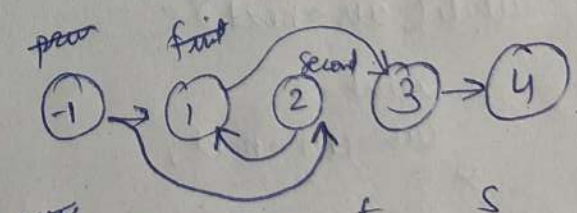
Swap nodes in pair :

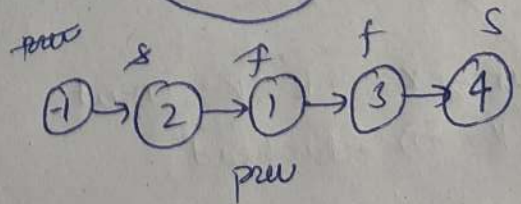
I/p 1 → 2 → 3 → 4

O/p → 2 → 1 → 4 → 3

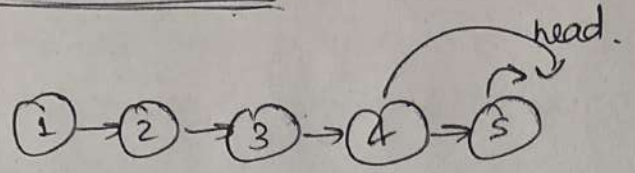
```
* ListNode dummy = (-1)
  dummy -> next = head;
  ListNode* prev = dummy;
  while (prev -> next && prev -> next -> next)
  {
      LN* first = prev -> next;   // assign.
      LN* second = first -> next;

      //swapping.
      first -> next = second -> next;
      second -> next = first;
      prev -> next = second;

      prev = first;
  }
  return dummy -> next;
};
```







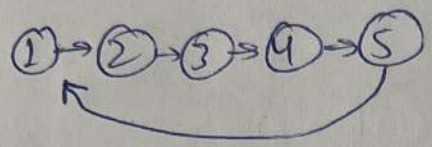
Rotate linked list LC-61



n = 2.

O/p → 4 → 5 → 1 → 2 → 3.

Approach → make it circular. → keep count of length.



→ keep another pointer in head.

newhead = length - n;

newhead = 5 - 2 = (3)

iterate to (3) node, & break, 3→next = head;

3→next = NULL;

Code :

(1) edge cases

```
if(!head || !head->next || k==0) return head;
```

(2)
```
listnode* cur = head;
int count = 1;
while(cur->next){
    count++;
    cur = cur->next;
}
```

(3) optimize k → k = k % count;

```
if(k==0) return head;
```

(4) make circular

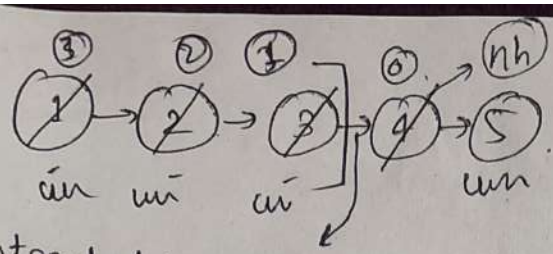
```
cur->next = head;
```

(5) move to new tail

```
int steps = count - k;
while(step--)
```

→ break cycle (6)

```
ListNode* newhead = cur->next;
cur->next = NULL;
return newhead;
}
```



int cout = 1;

length → 1+4 = (5)

k = 2 % 5
⇒ (2)

steps = 5 - 2 ⇒ (3) , (mtlb (3) steps chaia)

while (steps--)
{
cur = cur → next;
}

ListNode* newhead = cur → next;
cur → next = NULL;
return newhead;

LC → 92 Reverse Linked List - II

[cur (head k node ko prev) k bad chipkana hai]

I/P (1) → [(2) → (3) → (4)] → (5) [left = 2 , right = 4]

O/P (1) → (4) → (3) → (2) → (5)

Approach → main concept hai, left → right tk ka reversal

so, phele prev ko left-1 tk
→ cur → next of prev
→ next = cur → next

} same bs inside a loop with for(0 → k - l) times
5 - 3 = (2) times
0, 1, 2,

#

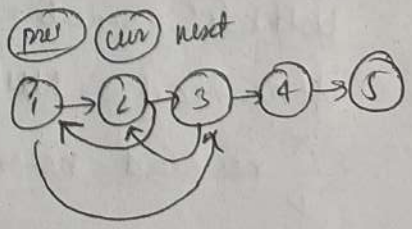
if (!head || left == right) return head;

LN* dummy = NULL;
dummy → next = head;

} dummy

LN* prev = dummy;
for(i = 1 , i < left ; i++)
{
prev = prev → next;
}

LN* cur = prev → next;
for(int i = 0; i < R - l; i++)
{
next = cur → next;
cur → next = prev;
next → next = prev → next;
prev → next = next;
}
return dummy → next



(2) → (1) → (3)

LC→148: Sort list:

I/P → 4 → 2 → 1 → 3

O/P → 1 → 2 → 3 → 4

basic sorting perform krnihai;

① Brute

→ store in vector. O(n)

→ sort(ans.begin(), ans.end()); O(nlogn)

→ store back to linked list. O(n) + O(n) SC

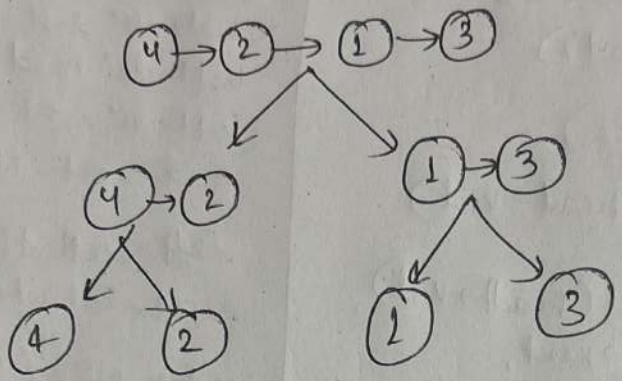
TC → O(nlogn) + O(n) SC

CODE:

```
vector<int> ans;
ListNode* newhead = NULL, tail = NULL;
ListNode* temp = head;
while(temp)                          // stored
  {
    ans.push_back(temp->val);
    temp = temp->next;
  }
sort(ans.begin(), ans.end());        // sorted
for(int i=0; i<ans.size(); i++)
  {
    ListNode* newnode = new ListNode(ans[i]);
    if(newhead == NULL)              // melb new node
      {
        newhead = newnode = tail;
      }
    else
        tail->next = newnode;
        tail = newnode;
  }
return newhead;
}
```

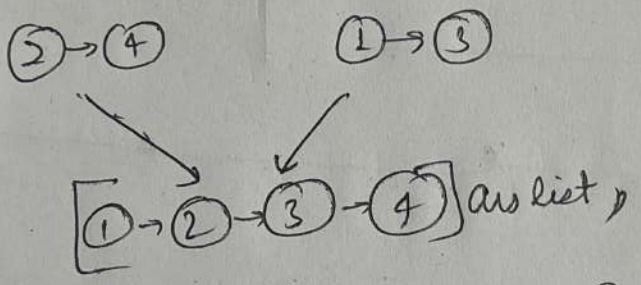
OPTIMAL :

→ Use divide and conquere like merge sort;
→ find mid of linked,
→ go for getting it divide (till single node is present)
→ merge sorting the elements/ node



Single element left

Now merge them



as list,

Sorted : Code → mid = head (ko dalke).
→ right = mid → next;
→ mid → next = NULL; break;
→ main (left)
→ main (right)
→ return head.

mid → Slow / fast;
merge; → head1, head 2
while (! = !)
{
node → age process;
}

→ Ques solved by recursion
→ complex
→ O(n)
→ T(n) → O(1)

```
ListNode* mid(ListNode* head)
{
    s, f = head;
    while(f && f->next)
    {
        s = s->next;
        f = f->next->next;
    }
    return slow;
}
```

middle finding.

```
ListNode* merge(head1, head2)
{
    while(head1 && head2)
    {
        while(head1->val < head->val)
        {
            newnode = LN(head1->val);
            head1->head1->next;
        }
    }
}
```

```
main()
{
    ListNode* left = head
    ListNode* mid = mid(head);
    ListNode* right = mid->right;
    mid->next = NULL;
    left = sortlist(left);
    right = sortlist(right);
    return merge(left, right);
}
```

sorted