PEPCODING

PURSUIT OF EXCELLENCE AND PEACE

MODULE

LINKED LIST

+91 11 4019 4461

PepCoding, 3rd Floor, 15 Vaishali,
Pitampura Opposite Metro Pillar 347,
Above Karur Vysya Bank, New Delhi,
Delhi 110034

www.pepcoding.com

/pepcoding

Linked Lists

Q Reverse a linked list (Reverse Pointer Iterative)

What?

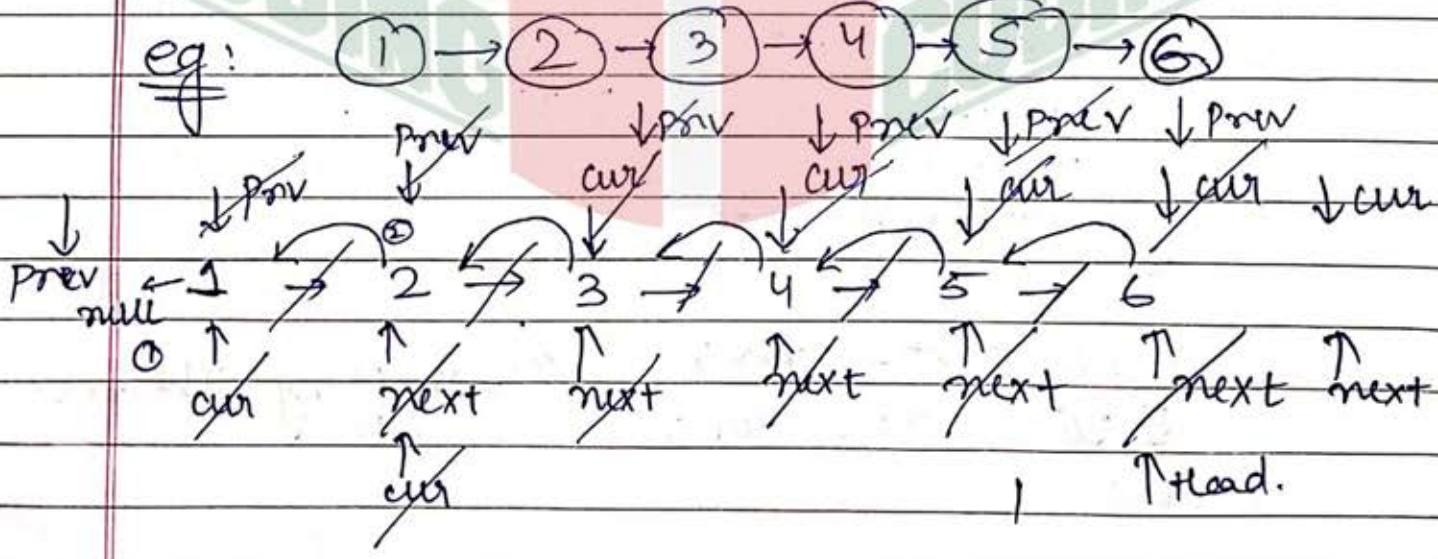
Given a linked list. Reverse the linked list without using extra space.

How?

We will maintain 3 pointers prev, cur, next.
Initially prev = null, cur = head, next = cur.next.

```
While (cur != null)
{   next = cur.next.        (rpi)
    cur.next = prev;
    prev = cur;
    cur = next;
}
head = prev;
```

eg:



Ans 6 → 5 → 4 → 3 → 2 → 1 → null.

Q Reverse linked list m to n

What?
Given a singly linked list and 2 no:s m & n. Reverse the linked list from position m to n in one pass.

How?
Move cur till m.
From m till cur is equal to n apply rpi
Attach the reverse list of elderlier list.

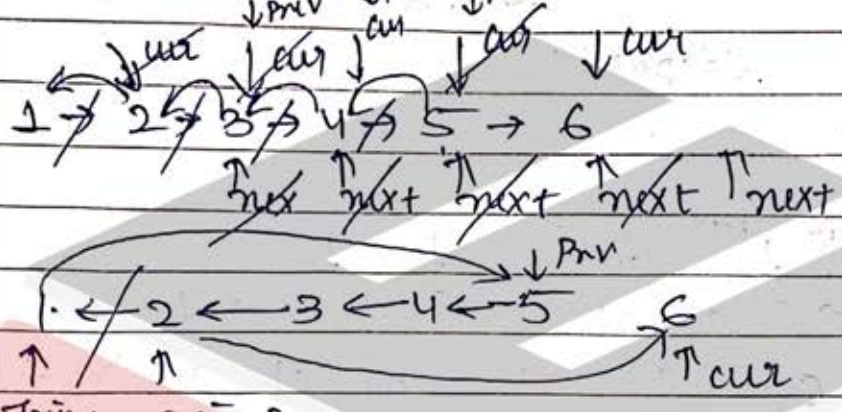
```
k=1                                  //cur = head , prev=null.
while (k<m)
    prev = cur;
    cur = cur.next;
    k++;

Join1 = prev;
Join2 = cur;

while (k<=n)
{ next = cur.next
   cur.next = prev    //rpi
    prev = cur
    cur = next   k++
}

Join1.next = prev;          //connecting the reverse lists.
Join2.next = cur;
head
```

eg: 1 → 2 → 3 → 4 → 5 → 6 m=2, n=5

1 → 2 → 3 → 4 → 5 → 6 K=1 (K<m)

prev cur

Join1 Join2.



K <= n.

Join1 Join2.

1 → 5 → 4 → 3 → 2 → 6 Ans

Reverse linked list in group of K

What?

Given a singly linked list of N nodes and a num K. reverse the linked list K at a time.

How?

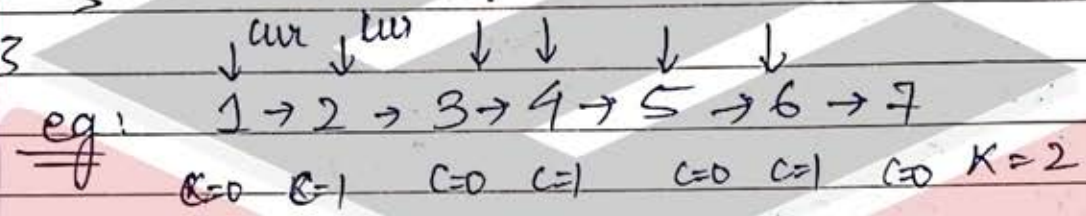
We will use stack<Node>. when count < K push in stack atlast pop and add to list

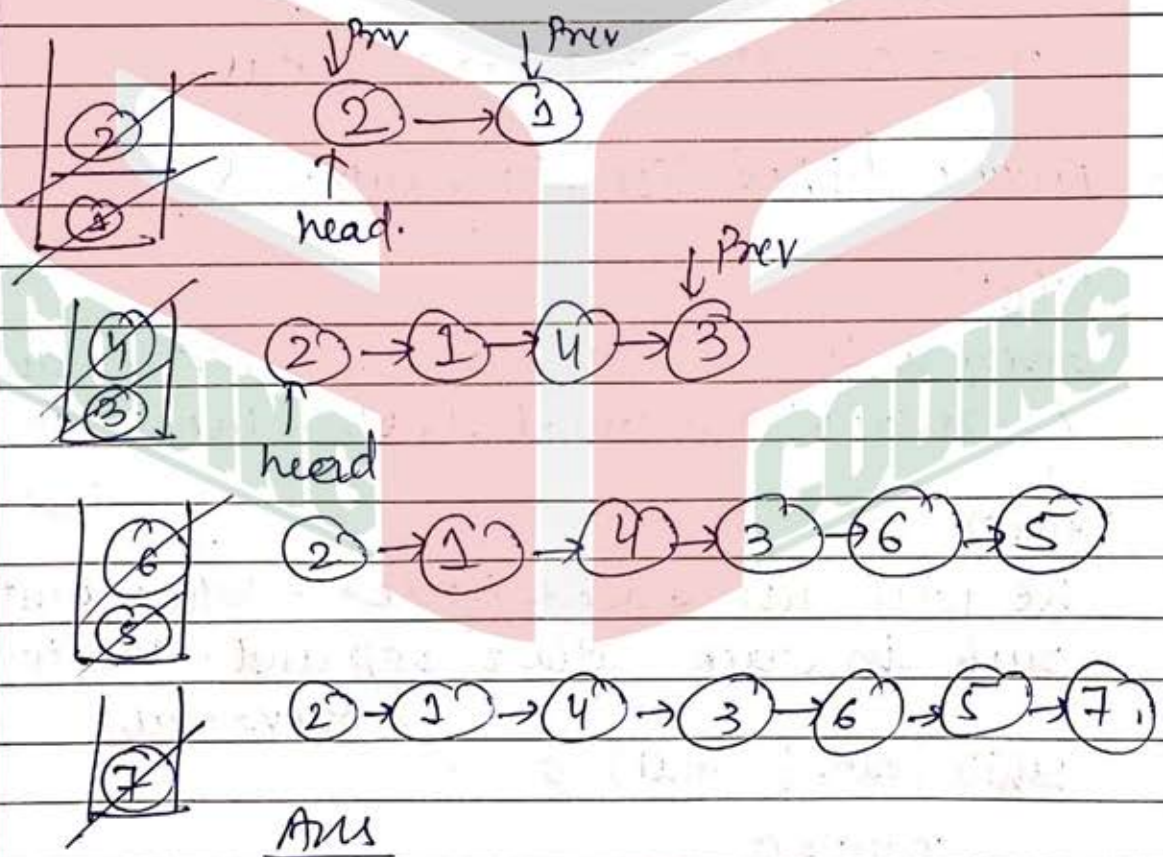
```
//prev = null.
while (cur != null) {
    count = 0
    while (cur != null && count < K)
    {   st.push(cur)
        cur = cur.next;
        K++;
    }
```

~~prev=~~

```
while (st.size()>0)
{ if (prev==null)          head=prev;
  { prev=st.peek();        st.pop();
  }
  else
  { prev.next=st.peek()
    st.pop();
  }
}
```

eg:





Ans

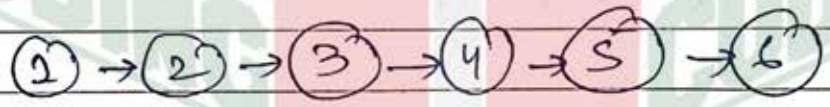
Q Modular Node

What?

Given a linked list of N nodes and a number K Find the modular node ie a node which is last node in linked list whose index (1 based) is divisible by K ie i%K==0.

How?

1) Reverse the linked list using rpi
2) Calculate size of LL
3) while (cur != null) // i=1
{
    if (size-i)% K==0)
        return cur.data;
    else
        cur = cur.next;
    i++;
}



K = 4

size = 6

(6-1)%4 != 0 (6-2)%4 = 0

Ans = 5

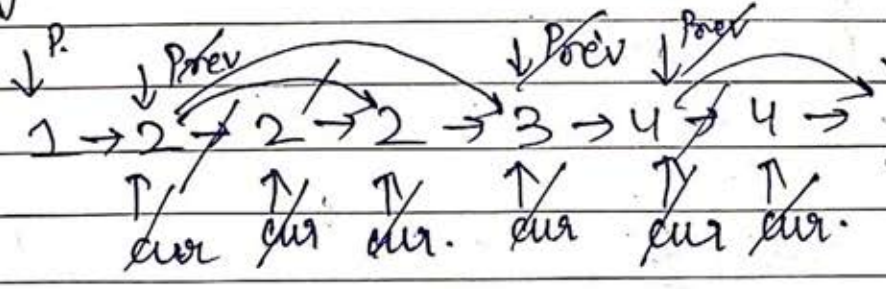
Q Remove Duplicates from Sorted LL.

What?
Given a singly linked list remove all the duplicates from the linked list and preserve only one occurrence of each element.

How?
Check if data of the cur node is equal to prev node or not If data is same. remove cur from the list.

```
prev = head , cur = head.next ;
while (cur != null)
{
    if (prev.data = cur.data)
    { prev.next = cur.next ;
      cur = cur.next ;
    }                                  // prev doesn't Change.
    else {
        prev = cur ;
        cur = cur.next .
    }
```

eg: 1 → 2 → 2 → 2 → 3 → 4 → 4 → 5



Ans 1 → 2 → 3 → 4 → 5 Ans

Date
DELTA Pg No.

Q Remove Duplicates from Unsorted LL

What?

Given an unsorted LL remove all the duplicates from the list such that there is only one occurence of each item.

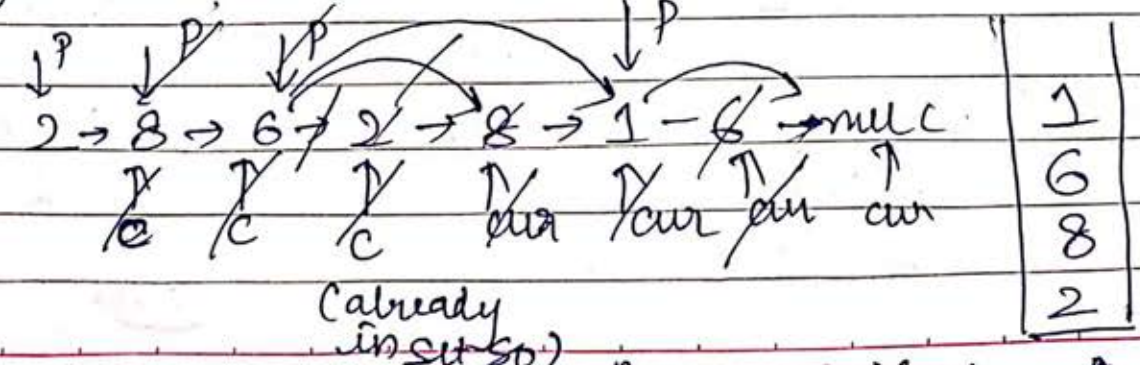
How?

Use a hashset to store the elements of linked list.

Keep prev at head initially and cur at head.next and add prev.data to hashset.

```
while (cur != null)
{
    if (set.contains(cur.data))
    {   prev.next = cur.next;
        cur = cur.next;
    }else{
        prev = cur;
        cur = cur.next;
        set.add(prev);
    }
}
```

eg: 2 → 8 → 6 → 2 → 8 → 1 → 6



Ans 2 → 8 → 6 → 1

Q Alternatingly Merge lists

What?

Given 2 singly linked list with N and M nodes respectively. Insert the nodes of second list in first list alternatingly

How?

We will breaks the links ~~nen~~ and rejoin it in first list using 2 pointers cur1 which points to the element of 1st list and cur2 which points to element of 2nd list.

cur1 = head1, cur2 = head2.

```
while (cur1 != null && cur2 != null) {
  Node next1 = cur1.next;
  Node next2 = cur2.next;
  cur1.next = cur2;
  cur2.next = next1;
  cur1 = next1 ; cur2 = next2;
}
```

eg: L1 = 1→2→3

L2 → 4→5→6→8.

(I) 1→2→3 with 4→5→6→8; cur1 at 1, cur2 at 4

(II) 1→4→2→3 with 5→6→8; cur1 at 2, cur2 at 5

(III) 1→4→2→5→3→null with 6→8; cur1 at 3, cur2 at 6

(IV) L1 = 1→4→2→5→3→6→8

L2 = 8

Ans

Date
DELTA Pg No.

8 Arranging Vowels & Consonants

What?

Given a Linked list of letters in English alphabet (a to z). Arrange the vowels and consonant such that all the vowel nodes come before the consonants while maintaining the order of arrival

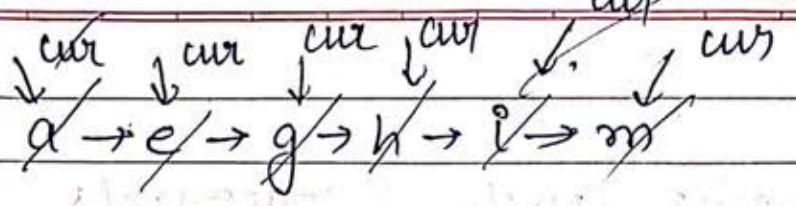
How?

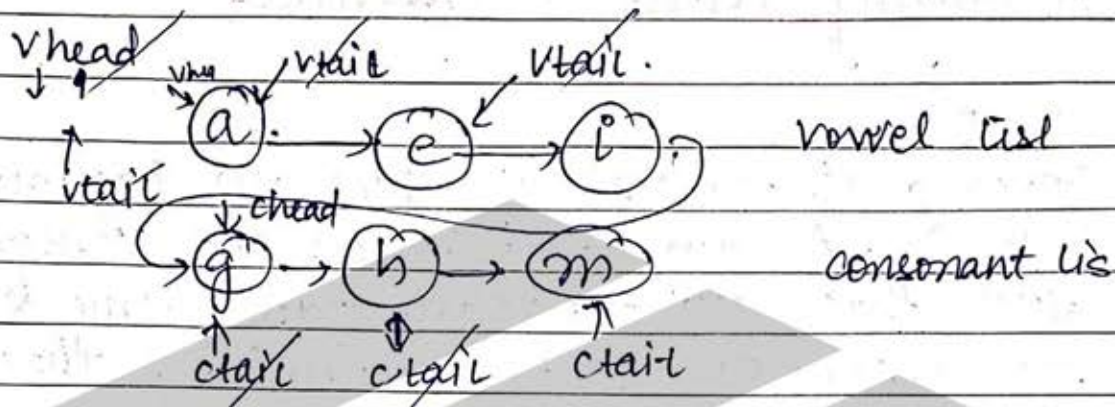
We will do this on the basis of 4 pointer approach

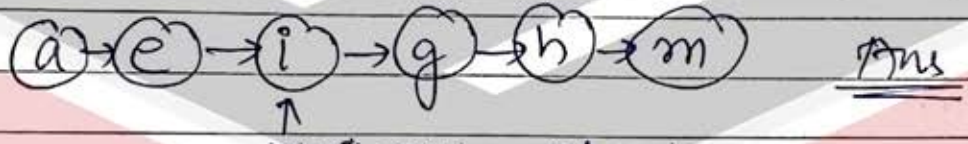
vtail → vowel list tail
vhead → vowel list head.
ctail → consonant list ~~head~~ tail
chead → consonant list head.

```
while (cur != null)
{   Node = removeFirst () → from cur ll.
    if (cur.data == vowel)
        addlast in vowel list (Node)
    else
        addlast in consonant list (Node)
}
if(vhead != null)
    vtail.next = chead;
    return vhead;
else
    return chead;
```

eg:







Q Rearrange linked list odd Even

~~How?~~ what?

Given a singly linked list of N nodes. Arrange the nodes in list in such a way that all odd nodes appear at beginning of the list. After that all even nodes appear. (Index Based not value First node is odd)

How?

Same 4 pointer approach is used as above.

ohead otail ehead etail

counter = 1

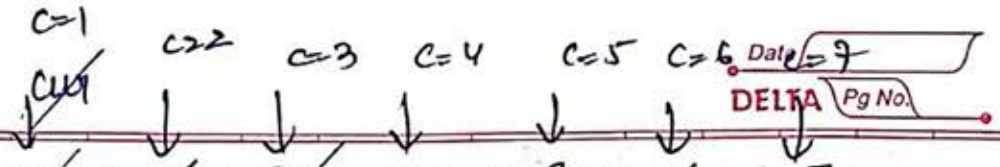
While list is not empty.

If (counter % 2 == 0)

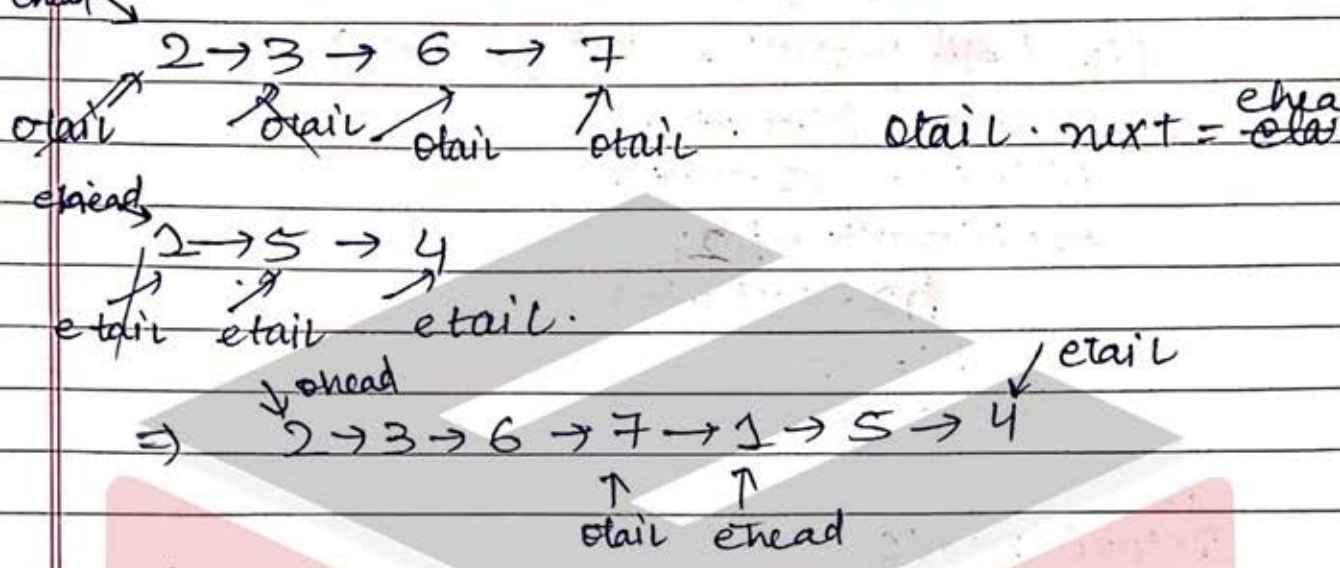
addLast in even list

else

addLast in odd list.



eg: $2 \to 1 \to 3 \to 5 \to 6 \to 4 \to 7$



head = ohead.
tail = etail

8) FOLD

What?

Given a linked list $L_0 \to L_1 \to L_2 \cdots L_{n-1} \to L_n$.
Arrange linked list so the new linked list formed is $L_0 \to L_n \to L_1 \to L_{n-1} \to L_2 \cdots$ and so on.

How?

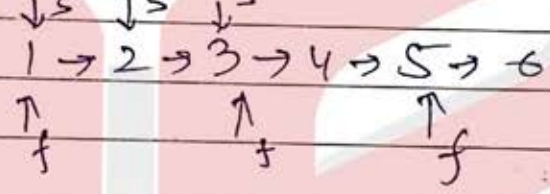
1) Find the middle of linked list and split it into 2 halves.
2) Reverse the second half using reverse Pointer Iterative technique
3) Now alternatingly merge 2 linked list.

Node h1 = head
Node h2 = slow.next
slow.next = null.
while (h1 != null && h2 != null)
{ Node on = h1.next;
Node tn = h2.next;
h1.next = h2;
h2.next = on;
h1 = on;
h2 = tn;
}
return head1;

Node head2 = slow.next

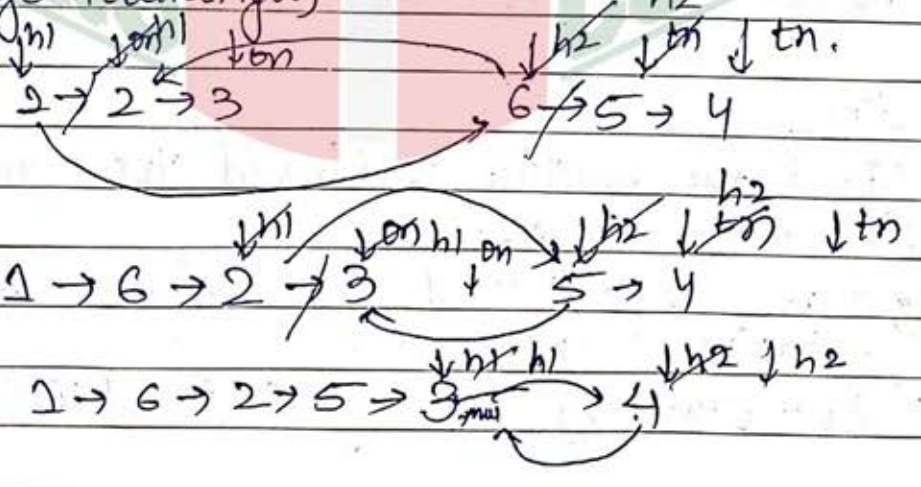
eg: 1 → 2 → 3 → 4 → 5 → 6

1) Find middle



2) Reverse 2nd half 1→2→3 6→5→4

3) Merge Alternatingly



1→6→2→5→3→4 Ans.

Q Unfold

What?

Given a linked list in form $L_0 \to L_n \to L_1 \to L_{n-1} \to L_2$ and so on rearrange in linked is following form $L_0 \to L_1 \to L_2 \to L_3 \to \dots \to L_{n-1} \to L_n$

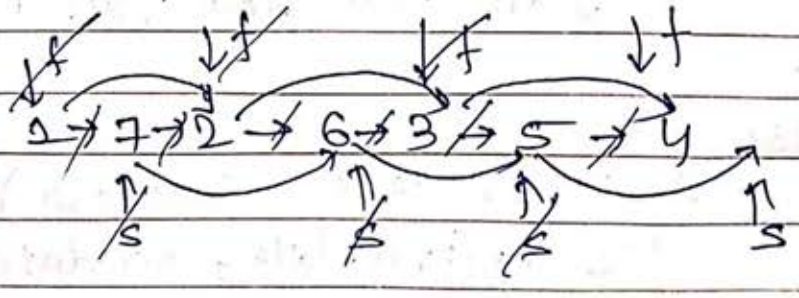
How?

1) We will place first pointer at head and second pointer at head → next.
2) Now applying following steps.

```
//head1 = head , head2 = head->next
while ( first!=null && second!=null)
{
    first->next = first->next->next;
    second->next = second->next->next;
     first = first->next;
     second = second->next;
}    first.next = null ;
//when size less than equal to 2 don't do anything.
```

3) Reverse the head2 list and combine both list.

eg: $1 \to 7 \to 2 \to 6 \to 3 \to 5 \to 4$



$1 \to 2 \to 3 \to 4$ $\quad$ $7 \to 6 \to 5$

Reverse second half

5 → 6 → 7

Combine

1 → 2 → 3 → 4 → 5 → 6 → 7 Ans

first → next = head2;

8 Zig zag linked list

What?

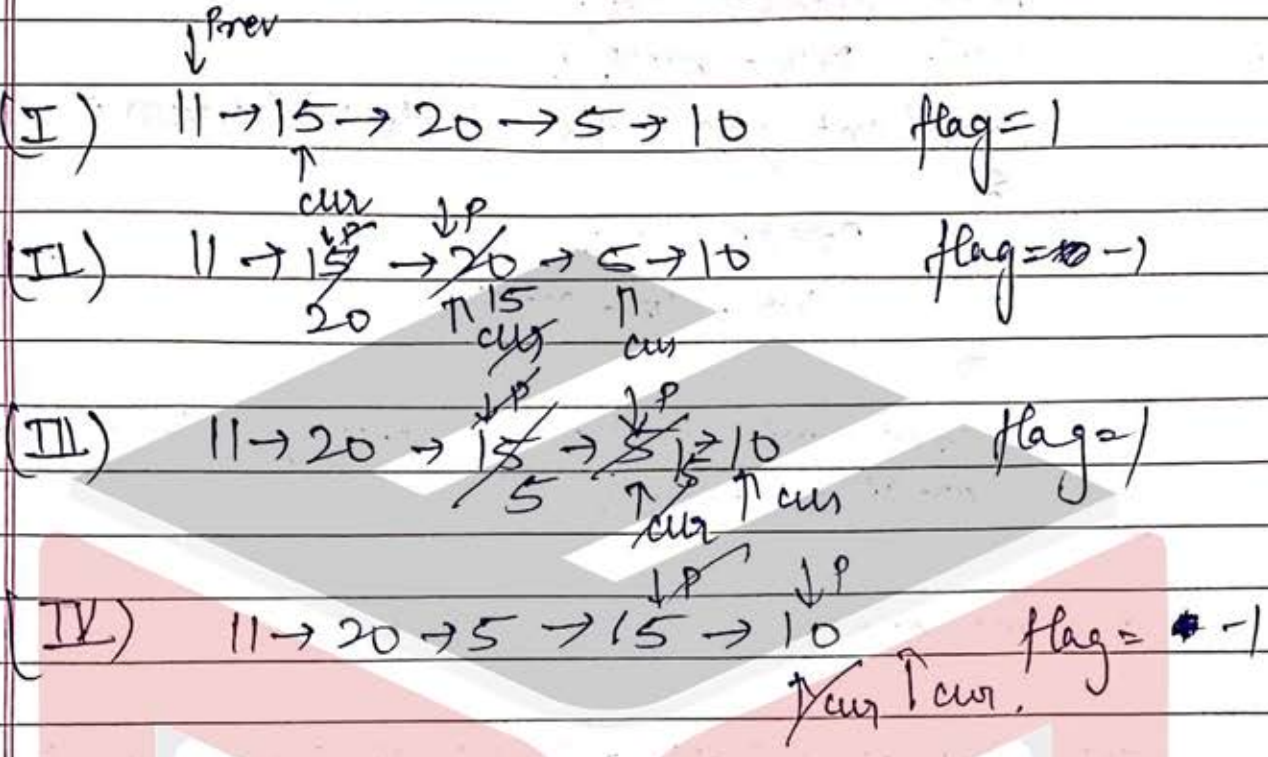
Given a linked list. Rearrang it in such a way that converted list should be of form a < b > c < d > e < f ... where a, b, c ... are consecutive data node in LL and order of it is sustained

How?

We will keep a flag which indicates our element should be greater or smaller.

```
prev = head, cur = head->next;
flag = 1;
while (cur != null)
{
    if (flag == 1)
    {   if (prev.data > cur.data)
            swap (prev.data, cur.data);
    }
    else {
        if (prev.data < cur.data)
            swap (prev.data, cur.data);
    }
    flag = flag * -1;
}
```

eg: 11 → 15 → 20 → 5 → 10



Q Split Linked list in K parts.

What?
Given a linked list and an Integer K. Split linked list in K parts and return head of each part of linked list

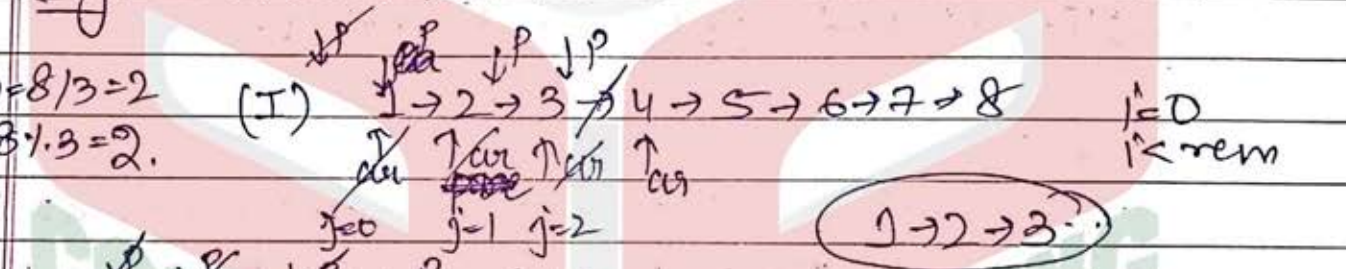
How!
1) Count the no. of nodes in the linked list.
2) Calculate width ie no. of nodes in each part. width = count / k;
3) Remainder of the nodes will be distributed in first rem parts rem = count % k.

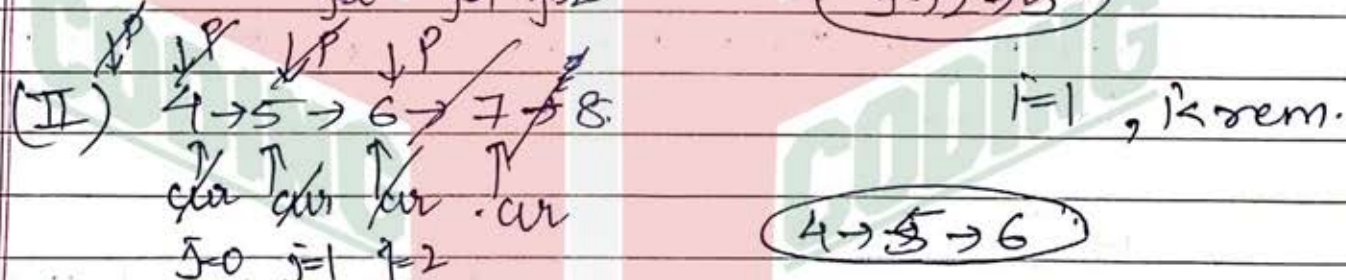
```
Node[] arr = new Node[K] //To store heads.
cur = head;
```

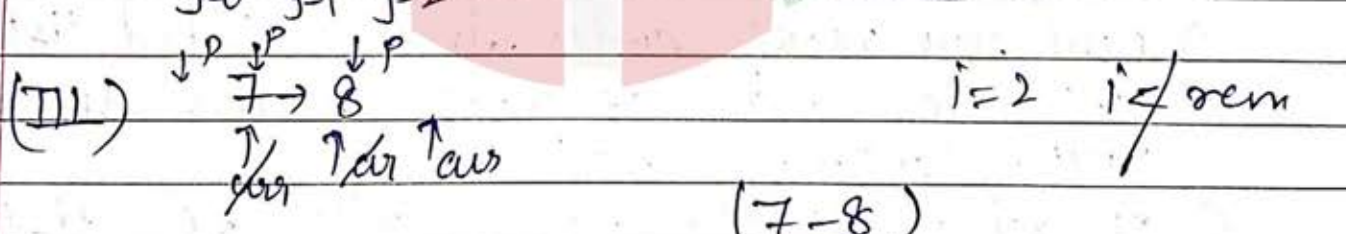
```
for(int i=0; i<k; i++) {
    Node head= cur;
    Node prev= null;
    for(int j=0; j< width+(i<rem ? 1:0); j++)
    {
        prev= cur;
        cur= cur.next;
    }
    prev.next= null;  // setting tail next null.
    arr[i]= head;
}
return arr;
```

eg: $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8$ $K=3$.

width = 8/3 = 2
rem = 8%3 = 2.







(7-8)

(1→2→3) (4→5→6) (7-8)

K parts

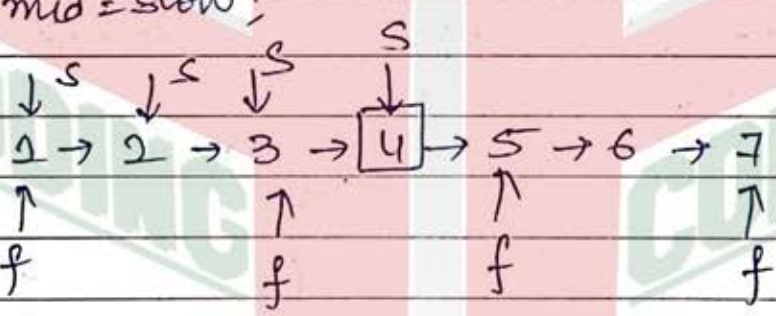
Date
DELTA Pg No.

Q Find Middle element In Linked list

What?
Given a singly linked list, find the middle element of the list.

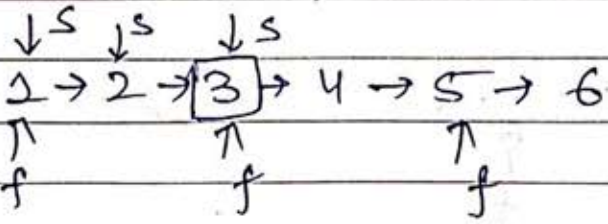
How?
2 pointers are use slow & ~~fast~~ fast.
Slow moves 1 step at time & fast move 2 steps at time.
When fast is at end slow will be in middle
// slow = head, fast = head.
while (fast->next != null && fast->next->next != NULL)
{
slow = slow->next;
fast = fast->next->next;
}
mid = slow;

eg:



mid = slow;

eg:



mid = s.

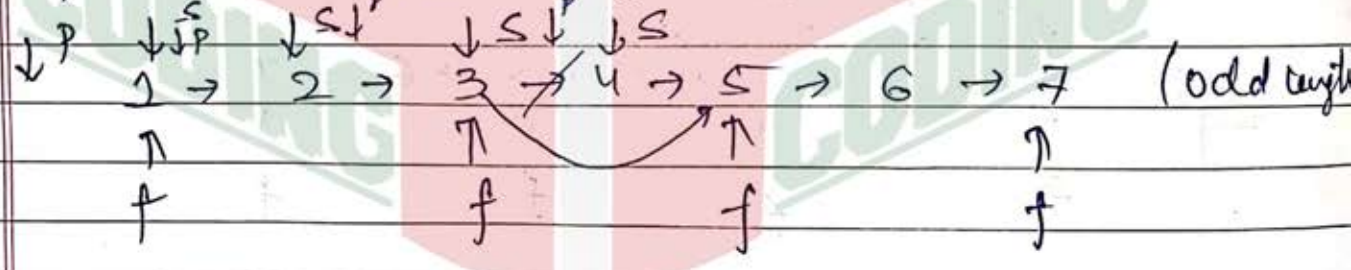
Q Delete Middle Node

What?
Given a linked list. Delete the middle node of the linked list.

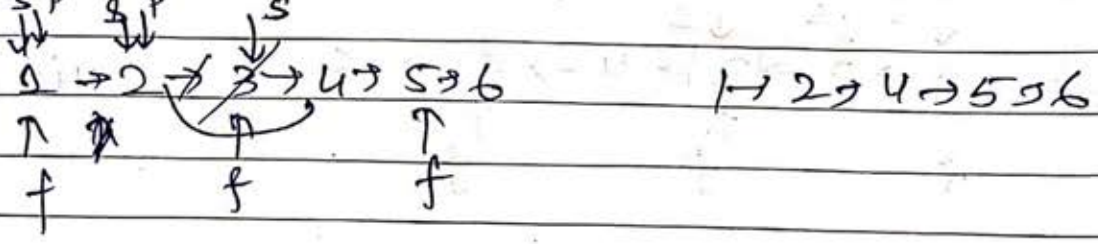
How?
Find the middle of the element by previous logic. Keep a pointer at middle prev.

prev = null, slow = ~~null~~ head, fast = head.

```
while (fast.next != null && fast.next.next != null)
{
    prev = slow;
    slow = slow.next;
    fast = fast.next.next;
}
prev.next = slow.next;   // deletion.
```



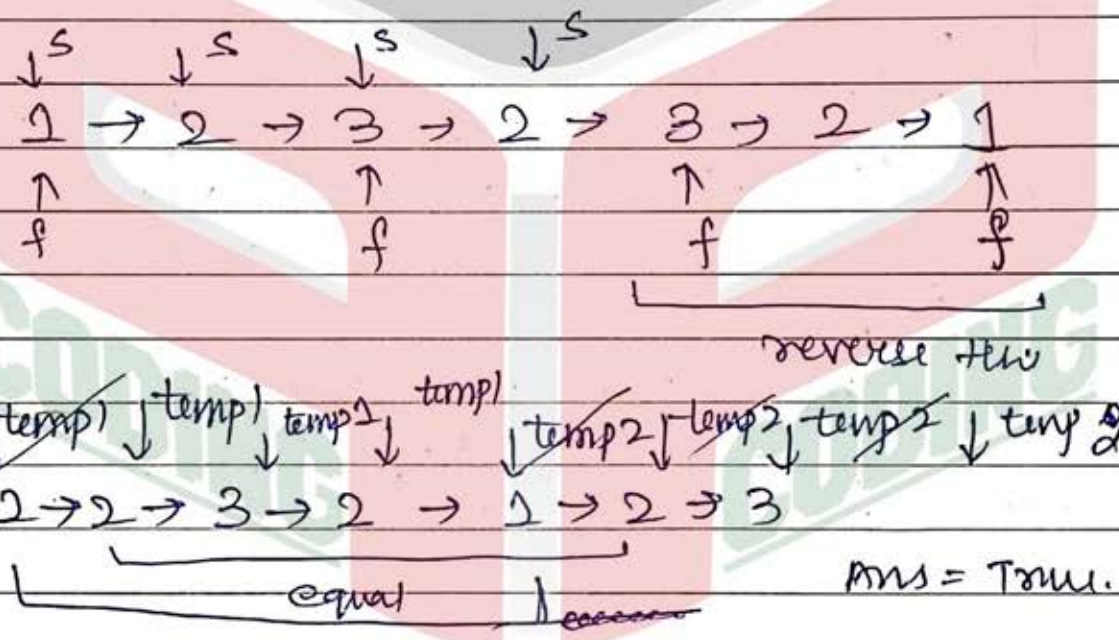
⇒ 1 → 2 → 3 → 5 → 6 → 7 ..



Q Is linked list Pallindrome

What?
Given a linked list. find if the content of the linked list form a pallindrome or not.

How?
1) Find the middle of the linked list (opt)
2) Reverse second half of linked ~~linked~~ list
3) Now compare till any of the list is not ended.
4) If no character match return false.




```
While (temp2 != null)
{ if (temp1.data != temp2.data)
      return false;
  else
      temp1 = temp1.next;
      temp2 = temp2.next;
}
```

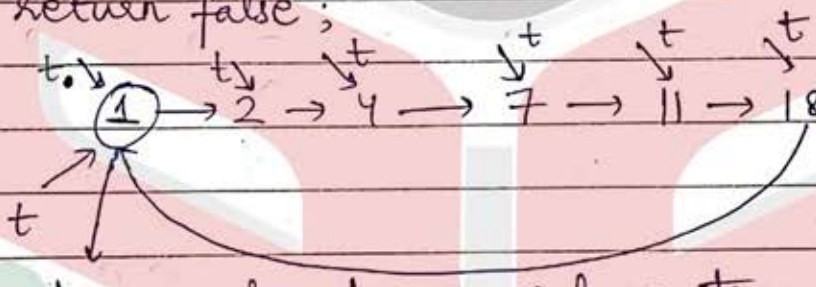
Q. Check for Circular linked list.

What?
Check if the given linked list is circular.

How?
* Keep a temp pointer, initially at head.
* Then iterate it till the tail to check if tail next is head, tail is last node in linked list.

```
temp = head;
while( temp != NULL)
  {   temp = temp.next
      if (temp == head)
          return true;
  }
return false;
```

ex:



temp = head ⇒ return true.

ex: 1 → 5 → 7 → 8 → 2 → NULL ⇒ return false.

Q. Insert into sorted cyclic linked list.

What?
Insert a node at its right position in a sorted cyclic linked list, if it's empty, make it head, create a new cyclic list and return reference to that single node.

How?

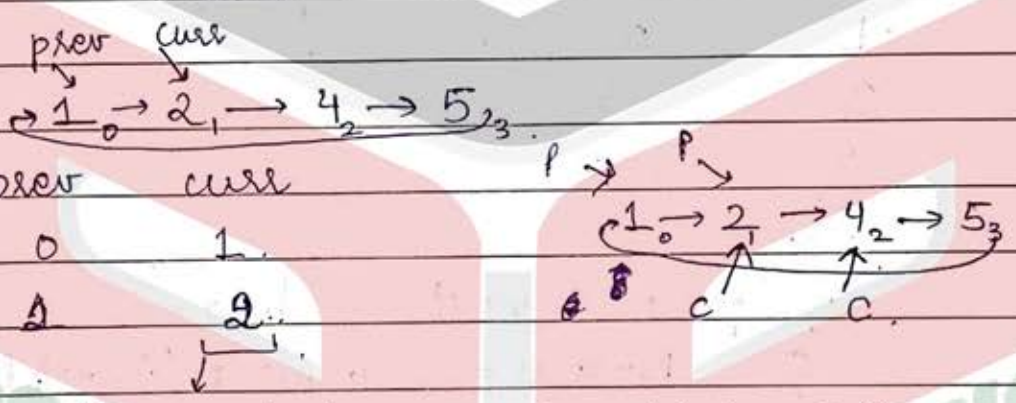
* Consider multiple cases:

1) If(head == NULL) // empty list.
{ // create new node with given data
// node → next = node.
head = node;
return head;
}

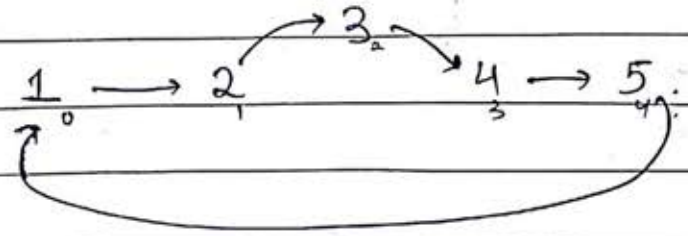
2) If the list is not empty,

I) When it is inserted somewhere in between.

ex: 1 → 2 → 4 → 5.
insert : 3.



if(curr.data >= insertdata && prev.data <= insertdata)
{ Node n = new Node(insertdata, curr)
prev.next = n;
n.next = curr;
return head;
}

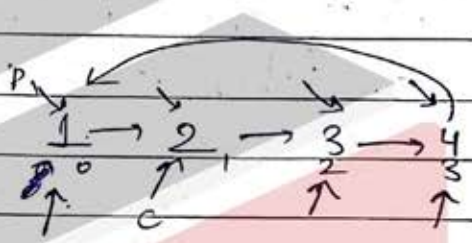


II) When ~~either~~ the node has to be inserted at tail because the insertdata is largest value / least value.

ex: 1 → 2 → 3 → 4

insertdata = 5.

Check that when ~~prev~~ → next == head, insert.



prev	curr
0	1
1	2
2	3
3	0

```
if (prev → next == head)
{   Node n = new Node (insertdata, head);
    prev → next = n;
    n → next = head;
    if (insertdata > head → data)
        return head
    else
    {   head = n;
        return head;
    }
}
```

Q. Circular linked list split in two halves.

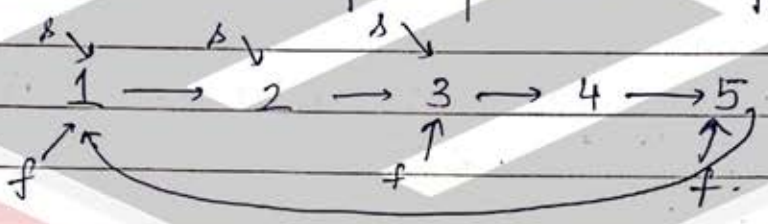
What?

Split the given circular list in two halves such that:
a) If N is odd, first half has one node extra.
b) Resultant lists should also be circular.

How?

* Use slow and fast pointers to find mid node.

ex:



* If (fast → next != head)
  fast = fast → next

* Now simply break two linked lists as:
  LL 2: fast → next = slow → next.

  LL 1: slow → next = head.

* Now return the two new heads:
  head1 = head;
  head2 = head != fast → next ? fast → next : NULL
  ↓
  when list can't be broken.

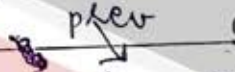
Q. Clone a linked list with random & next pointers.

What?

You are given a linked list with N nodes, and each node with 2 pointers, one pointing to next node, other to any random node in list. You need to clone the list and return the head of cloned list.

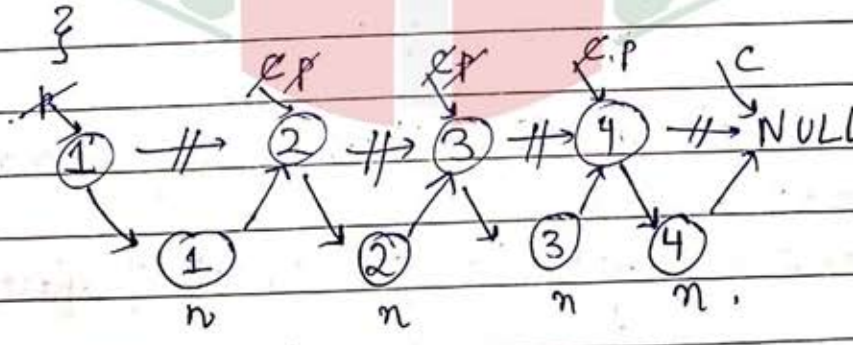
How?

1) Create copy of each node and insert it at its original's next position.



ex: 1 → 2 → 3 → 4.

```
prev = head;
curr = NULL;
while (prev != NULL)
{   curr = prev → next;
    Node n = new Node (prev → data)
    prev → next = n;
    n → next = curr;
    prev = curr;
}
```



2) Correct random pointers as:

original → next → random = original → random != NULL?

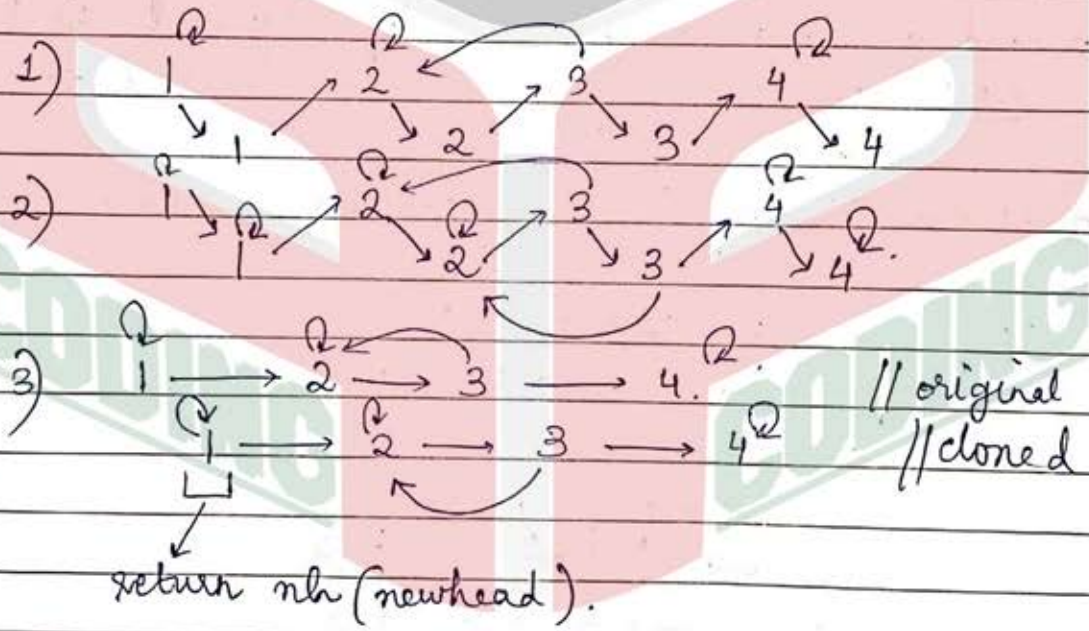
original → random → next : original → random.

// for all nodes.

3) Make original and copy lists different.

```
prev = head;
curr = head → next;
Node* newhead = curr;
while (prev != NULL && curr != NULL)
    // if prev → next is not null
        prev → next = prev → next → next.
    // if curr → next is not null
        curr → next = curr → next → next
    // prev = prev → next, curr = curr → next.
return newhead; // cloned list.
```

ex: 1 → 2 → 3 → 4.



Q. Add one to linked list

What?

You are given a linked list having nodes each with data 0-9. You need to add 1 in the linked list and then return head of the linked list. The head node is the most significant digit node of list.

How?

* Perform addition in list, first reverse the list using reverse pointer iterative method.
* Add 1 to least significant node as it is in the head now, and perform carry operations.
* Add a new carry node if required in the end.

ex: in case of $9 \rightarrow 9$, we need to return $1 \rightarrow 0 \rightarrow 0$.

* Reverse list again and return head.

ex: $1 \rightarrow 2 \rightarrow 9 \rightarrow 9$

rpi: $9 \rightarrow 9 \rightarrow 2 \rightarrow 1$

\+ 1

$0 \rightarrow 0 \rightarrow 3 \rightarrow 1$

c=0, c=1, c=1, c=0

rpi: $1 \rightarrow 3 \rightarrow 0 \rightarrow 0$.

↳ return head.

Similiar: Add 2 linked lists.

* rpi both, perform addition, return rpi of resultant list formed.

Q. Multiplication of 2 linked lists.

What? given

You are, 2 linked lists with N and M nodes. Multiply the two linked lists and return the head of new list. The given lists represent numbers.

How?

* Create a final list which stores result.
* reverse two given lists.
* while you don't traverse all digits of second list,
  // take first number from list 2.
  // Multiply it with entire list1.
  // Add a zero corresponding to a 0 in next iteration of multiplication, adding a zero would be moving head ahead for product to be formed.
  // add result after each number multiplication in result list, return result list head.

ex: L1 → 4 → 5 → 6

L2 → 2 → 3.

rpi L1: 6 → 5 → 4

rpi L2: 3 → 2

result list = empty initially.

1) At first node, count = 0 (lets keep it for adding zeroes)
   ∴ Multiply by 3 first:

L1: 6 → 5 → 4.

temp list:

8 → 6 → 3 → 1.

L2: ×3

8 → 6 → 3 → 1

Since count = 0,

result = rpi (templist)

∴ result = 1 → 3 → 6 → 8.

2) count = 1, add 1 0 node in templist.

templist

0 → 2 → 1 → 9

l1: 6 → 5 → 4.

l2: × 2

① ①

2 → 1 → 9

addition of 2 linked lists

count = 1, result = result + rpi (templist).

⇒ result = 1 → 3 → 6 → 8.

\+ 9 → 1 → 2 → 0

1 → 0 → 4 → 8 → 8.

rpi.

rpi l1: 8 → 6 → 3 → 1

rpi l2: 0 → 2 → 1 → 9

8 → 8 → 4 → 0 → 1.

∴ ans = 1 → 0 → 4 → 8 → 8.

↓

return head.

Q Addition of polynomial

What?

Given 2 polynomials in form of linked list perform addition of these polynomial list and return the head of resultant LL./Print the result

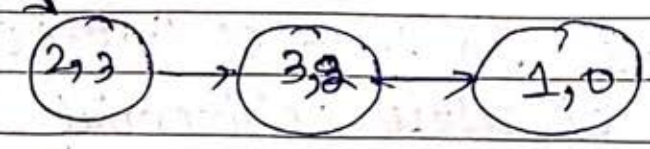
How?

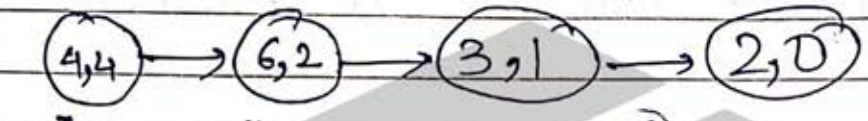
Two pointers p1 & p2 are pointing to the heads of polynomials

Since we can add only those terms that have same ~~coefficient~~ power. So we will compare powers and then add.

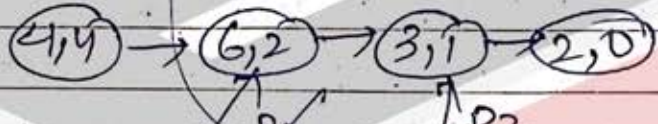
```
while (p1 != null && p2 != null) {
    coeff = 0, pow = 0;
    if (p1.pow > p2.pow) {
        coff = p1.coeff ; pow = p1.pow;
        p1 = p1.next;
    } else if (p1.pow < p2.pow) {
        coff = p2.coeff ; pow = p2.pow;
        p2 = p2.next;
    } else {
        coeff = p1.coeff + p2.coeff;
        pow = p1.pow ;  p1 = p1.next; p2 = p2.next;
    }
    sysо( p1.coeff + "x^" + pow );
}
// If any of p1 or p2 is not null
print rest of the polynomial
```

eg: $2x^3 + 3x^2 + 1$
$4x^4 + 6x^2 + 3x + 2$





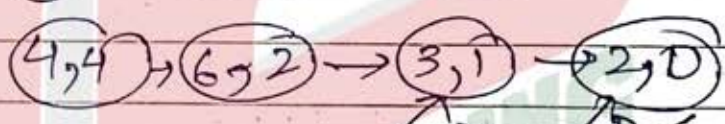
(I) $4x^4$.



(II) $4x^4 + 2x^3$

(III) $4x^4 + 2x^3 + 9x^2$

(IV) $4x^4 + 2x^3 + 9x^2 + 3x$



(V) $4x^4 + 2x^3 + 9x^2 + 3x + 3$

Ans

Q Merge Sort of linked list

What?

Given a linked list of N node. Sort the linked list using mergeSort.

How?

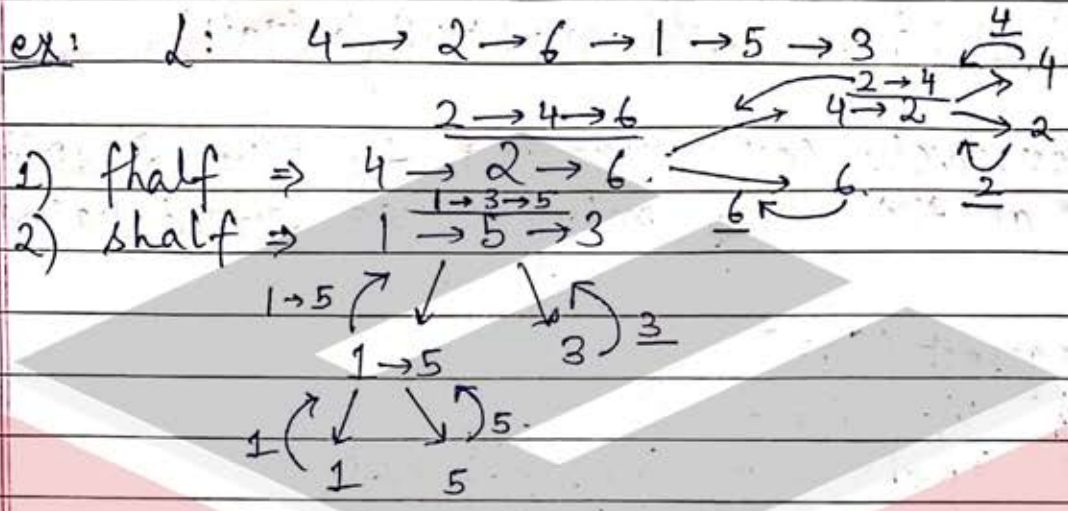
Simple like merge sort. First get the mid node of the given linked list. and break it into 2 halves. First half is from head to mid and second half is from mid.next to high (tail).

Recursively call mergeSort for the 2 split halves. and at the end merge these sorted halves together

```
Mergesort (list)
{
  if (list.size == 1)
  { // make new linkedlist blist.
    blist.addLast (list.head.data);
    return blist;
  }
  fhalf = new LinkedList();
  shalf = new LinkedList();
  mid = list.mid();        // gives you middle node of list
  midn = mid.next;
  fhalf.head = list.head;
  fhalf.tail = mid;
  fhalf.tail.next = NULL;
  fhalf.size = list.size % 2 == 0 ? list.size / 2 :
               list.size / 2 + 1;
  shalf.head = midn;
  shalf.tail = list.tail;
  shalf.size = list.size - fhalf.size;

  fhalf = Mergesort (fhalf);
  shalf = Mergesort (shalf);
  res = mergesortedlists (fhalf, shalf);
        // (merges 2 sorted lists).
```

mid.next = midx; // to maintain original list
return res;
}

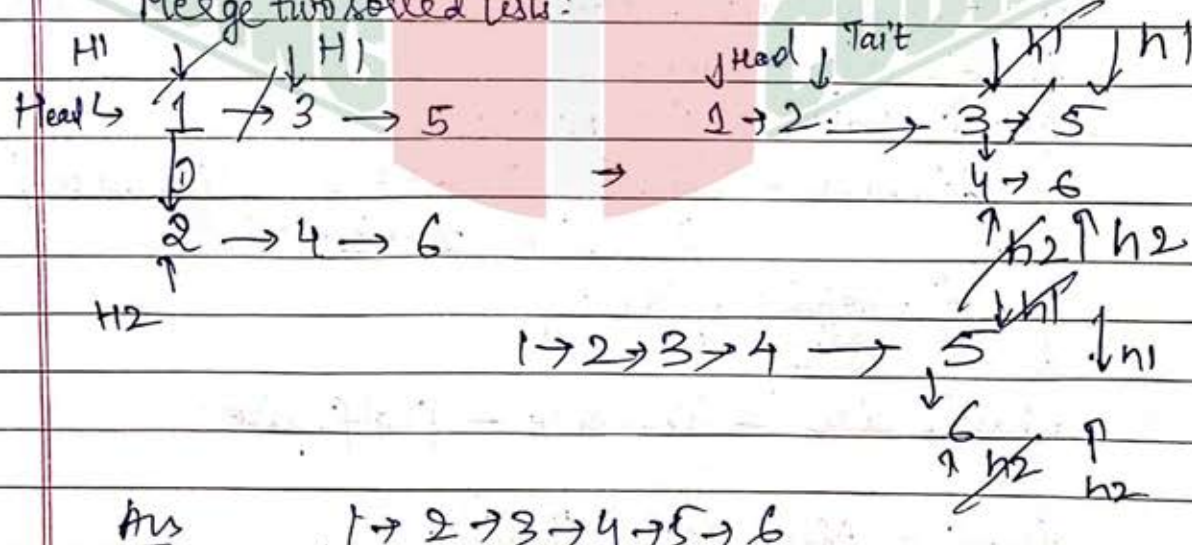
ex: L: 4 → 2 → 6 → 1 → 5 → 3



1 → 3 → 5 2 → 4 → 6

ans: 1 → 2 → 3 → 4 → 5 → 6.

Merge two sorted lists:



Ans 1 → 2 → 3 → 4 → 5 → 6

```
Node getmidnode (Node lo, Node hi)
{
    slow = lo;
    fast = lo;
    while (fast != high);          //conditions as get mid function
    {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}

void mergesort (Node lo, Node hi)
{
    Node mid = getmidnode (lo, hi);
    mergesort (lo, mid);
    mergesort (mid.next, hi);
    mergesortedlist (lo, mid, high);
}
// Inplace mergesort.
```

Date
DELTA Pg No.

Q. Connected components in a list.

What?

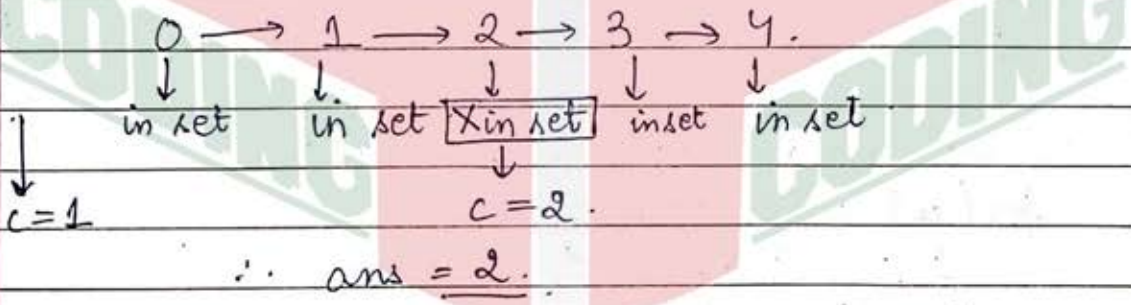
You are given singly linked list containing N unique integer nodes. You are also given G numbers which are subset of values in linked list.
Return the number of connected components in G, where two values are connected if they appear consecutively in the list.

How?

* Create a hashset of the set given.
* For each node, if node exists in set and its next does not, count++ where count is the number of connected components.

ex: 0 → 1 → 2 → 3 → 4 : list.

set : 0, 3, 1, 4



∴ ans = 2.

Q. Kth node from last.

What?

Given a list, return Kth node from last.

How?

* Keep a pointer fast and do fast = fast→next k times. Now keep pointer slow at head to maintain

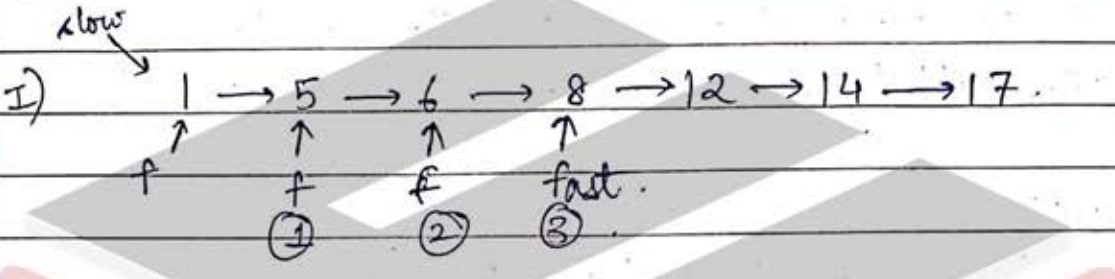
a difference of k between slow and fast.

* Now move both one by one, when fast points to NULL, slow is at kth from last node.

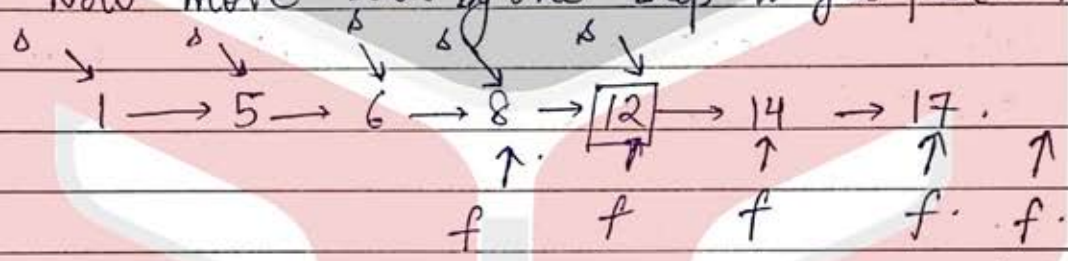
ex: 1 → 5 → 6 → 8 → 12 → 14 → 17.

k = 3.

I)



II) Now move both by one step to get fast at NULL.



when fast == NULL, slow = 12 which is kth node from last (k=3).

ans = 12

→
```
Node* slow = head, *fast = head;
for(int i=0; i<k; i++)
    fast = fast→next;
while (fast != NULL)
{   slow = slow→next;
    fast = fast→next;
}
return slow→data;
```

Page No.
Date

Loops in linked lists

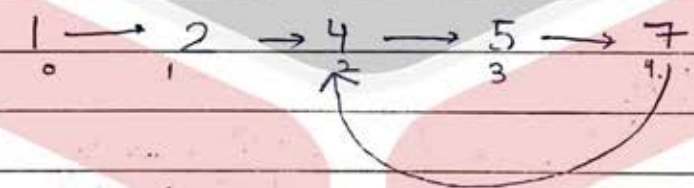
Q. Loop detection in linked lists.

What?

You are given a linked list with N nodes. Check if the list has a loop?

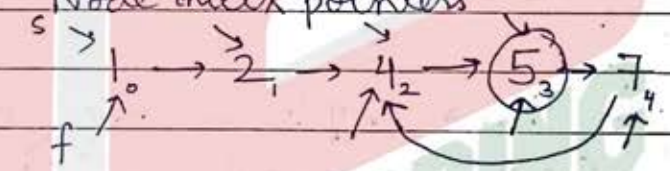
How?

* Keep two pointers slow and fast, both pointing to head initially.
* Move slow by one & fast by two at each step, if they meet at a point, loop exists.

ex: 

slow	fast
0	0
1	2
2	4
3	3

Node index pointers



⇒ detected loop at 5, return true.

```
while (fast → next != NULL && fast → next → next != NULL)
    slow = slow → next;
    fast = fast → next → next;
    if (slow == fast)
        {return true;}
```

Page No.
Date

Q. Cycle starting Node.

What?

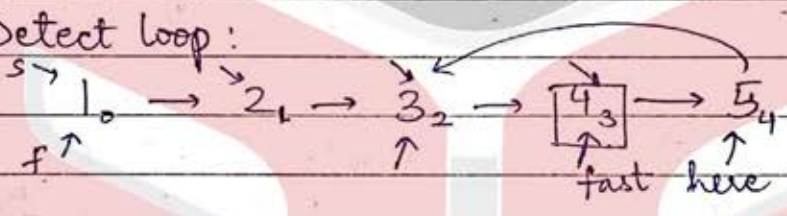
You are given a linked list which has N nodes. Return the cycle starting node.

How?

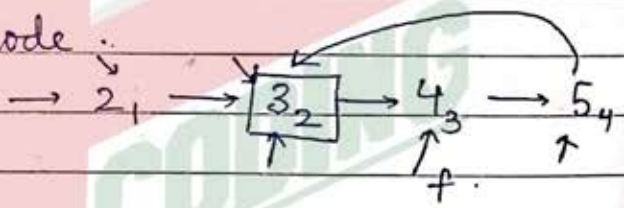
* First check if the list has cycle. If yes, break the loop where slow == fast.
* Now keep slow at head and move both slow and fast by one step until slow == fast.
* Return slow.

ex: 1 → 2 → 3 → 4 → 5

I) Detect loop:



II) Find cycle starting node:



Slow	Fast
0	3
1	4
2	2

⇒ return 3_2 as the node at which cycle starts.

```
// detect loop.
if (slow == fast)
    break;
slow = head;
while (slow != fast)
    slow = slow → next.
    fast = fast → next.
return slow;
```

Page No.
Date

Q. Detecting length of loop.

What?

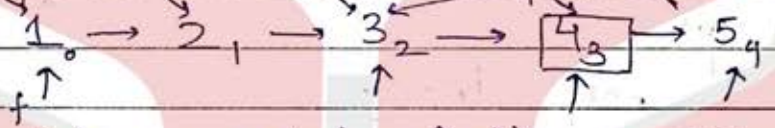
You are given a linked list with N nodes, return the number of nodes in the loop in list.

How?

* First detect if there is a loop in the list, if no, return 0.
* If yes, take a counter variable and simply increment count by 1 each time slow → next != fast.

ex: $1_0 \rightarrow 2_1 \rightarrow 3_2 \rightarrow 4_3 \rightarrow 5_4$.

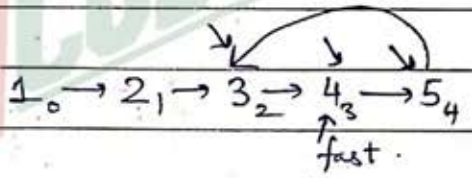
I) Loop exists at slow == fast at 4_3.



II)
```
While (slow → next != fast)
    count++;
    slow = slow → next.
```

Counter is initially 1

slow	fast	count
3	3	1
4	3	2
2	3	3



→ here slow → next = fast.

∴ length of loop = 3.

Page No.
Date

Q. Remove loop in singly linked list.

What?

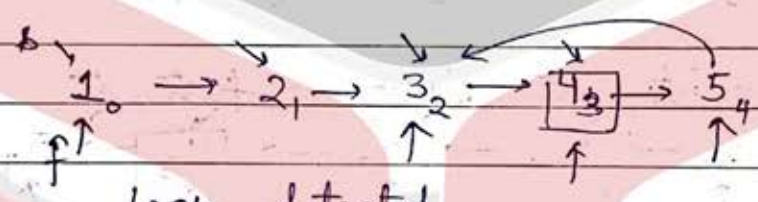
You are given a linked list with N nodes, remove the loop from list if present.

How?

* Detect if loop exists.
* Keep a prev pointer (NULL initially) and move slow at head.
* Just link the node's next pointing to cycle starting node as NULL to break loop.

ex: 1 → 2 → 3 → 4 → 5

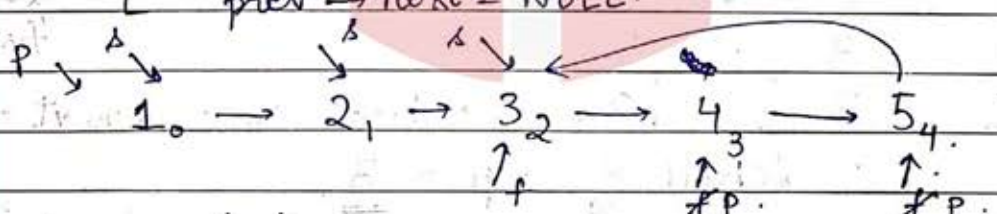
I)



Loop detected

II)

```
if (slow == fast)
    slow = head;   prev = NULL;
    while (slow != fast)
        slow = slow → next;
        prev = fast;
        fast = fast → next;
    prev → next = NULL.
```



slow	fast	prev
0	3	NULL
1	4	3
2	2	4

⇒ slow == fast, ∴ p → next = NULL.

Results in:—

1 0 → 2 1 → 3 2 → 4 3 → 5 4