



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?

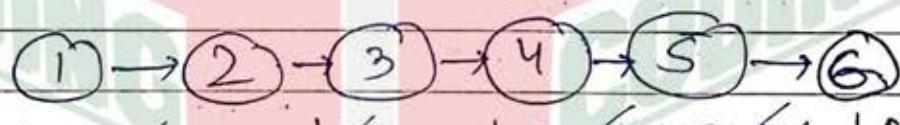
We will maintain 3 pointers `prev`, `cur`, `next`. Initially `prev = null`, `cur = head`, `next = cur.next`.

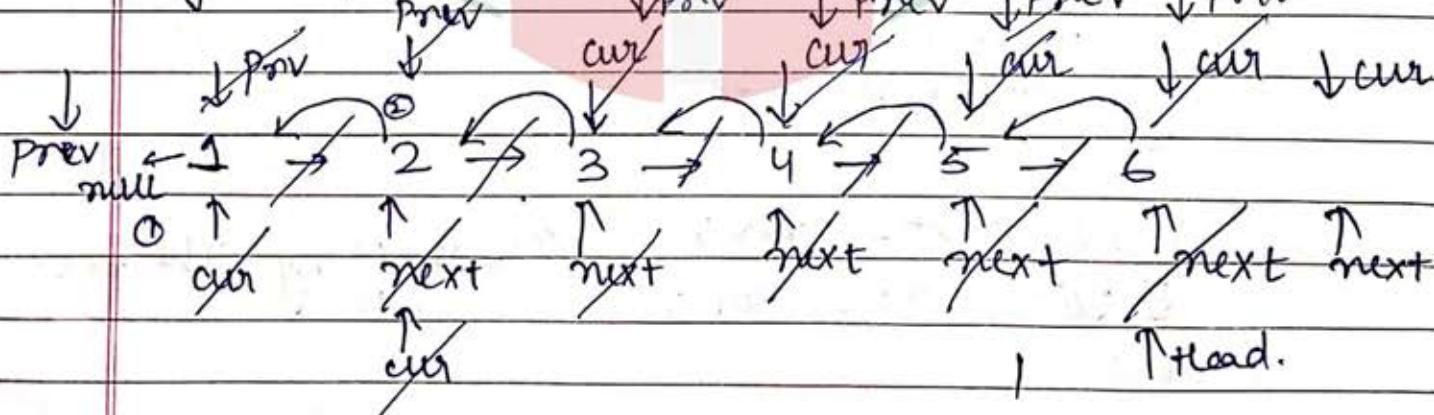
`while (cur != null)`

```
{   prev = cur.next;
    cur.next = prev;
    prev = cur;
    cur = next;
```

3

`head = prev;`

eg: 



Ans $6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow \text{null}$.

Q) Reverse linked list m to n

What?

Given a singly linked list and 2 nos 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 earlier list.

K=1

//cur = head, prev = null.

while ($K < m$)

prev = cur;

cur = cur.next;

K++;

Join1 = prev;

Join2 = cur;

while ($K \leq n$)

{ next = cur.next

cur.next = prev

//rpi

prev = cur

cur = next K++

3

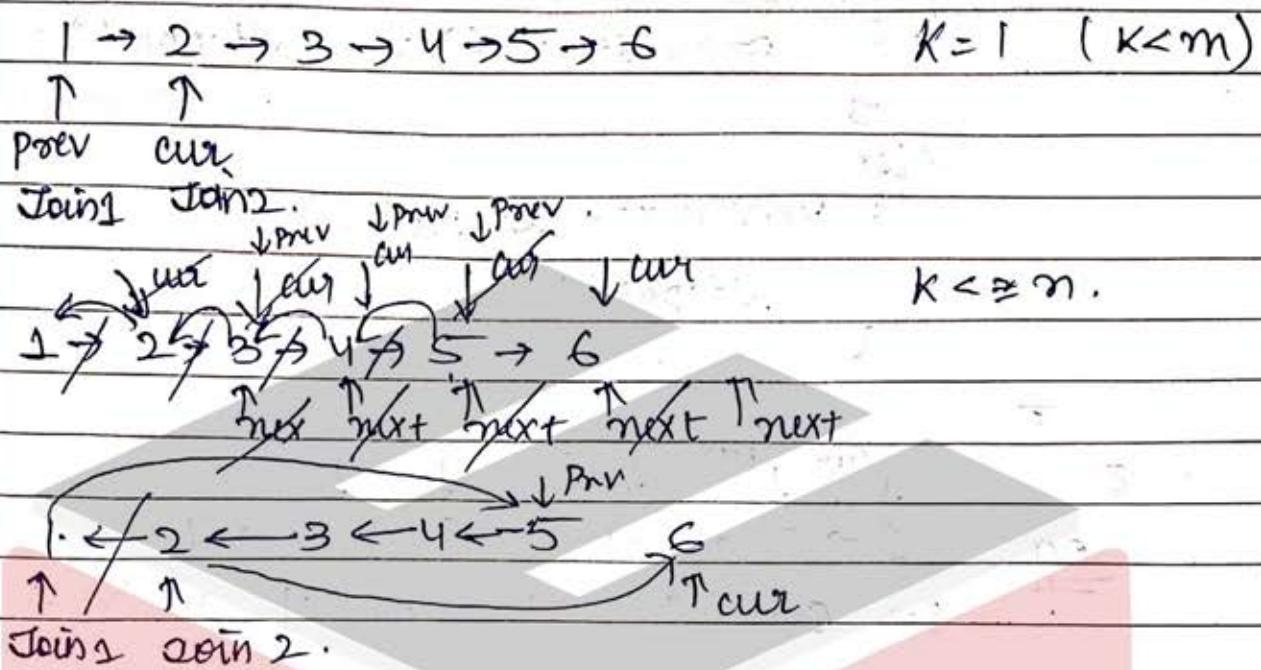
Join1.next = prev;

Join2.next = cur;

//connecting the reverse lists.

head

e.g.: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$ $m=2, n=5$



$1 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 6$ Ans

Q) Reverse linked list in group of k.

What?

Given a singly linked list of N Nodes and a number k. reverse the linked list in k at a time.

How?

We will use stack<Node>. when count < k push in stack atleast pop and add to list if prev = null.

while (cur != null) {

 count = 0;

 while (cur != null && count < k)

 st.push(cur);

 cur = cur.next;

 K++;

}

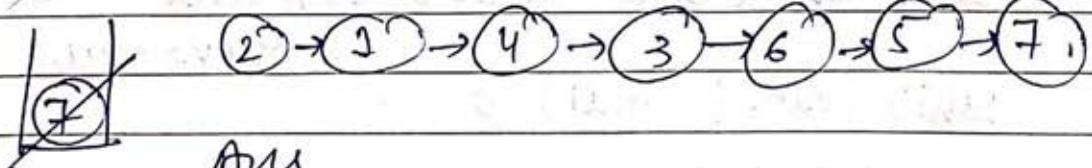
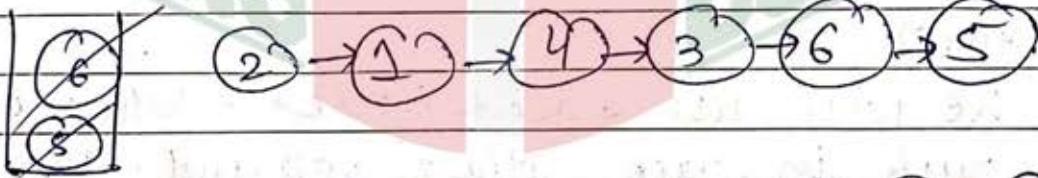
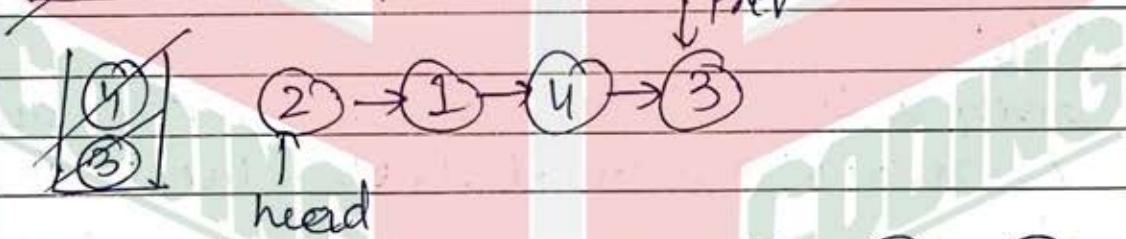
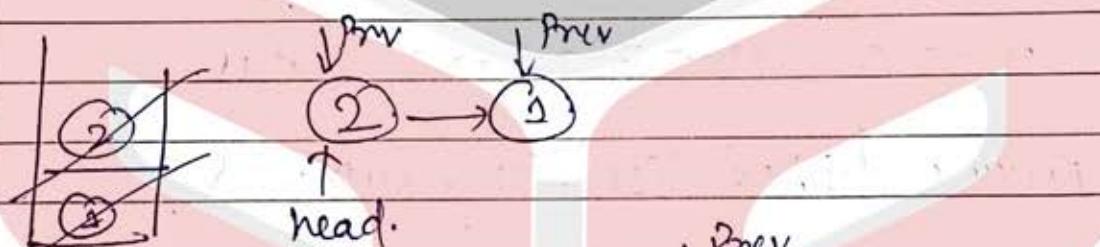
~~prev =~~

```

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

```

eg: $\begin{matrix} \downarrow \text{cur} \\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \end{matrix}$
 $\begin{matrix} \text{c=0} & \text{c=1} & \text{c=0} & \text{c=1} & \text{c=0} & \text{K=2} \end{matrix}$

Ans

Q Modular Node

what)

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

How?

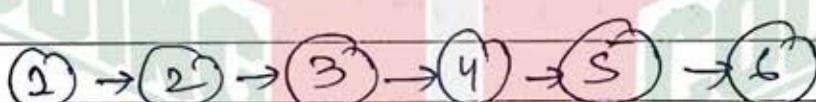
- 1) Reverse the linked list using rpi
 - 2) Calculate size of LL
 - 3) while ($\text{cur} \neq \text{null}$)
 {
 // i = }

```
if (size-2)%k == 0  
    return cur.data;
```

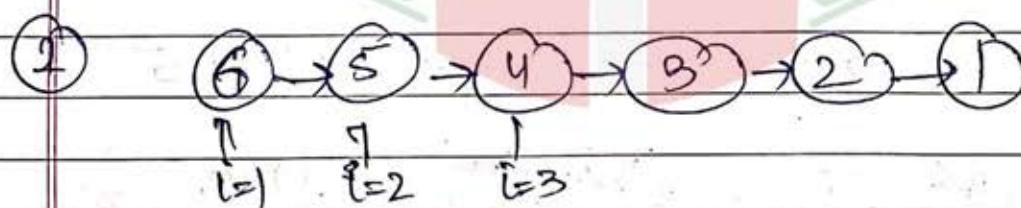
else
 $\text{cur} = \text{cur}.\text{next_}$

1++

1++



$$k = 4$$



size = b

$$(G \rightarrow Y \cdot y) \models_0 (G \rightarrow Y \cdot y) = 0$$

$$\text{Ans} = \underline{5}$$

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)

 if prev.next == cur.next;

 cur = cur.next;

 // pointer doesn't change.

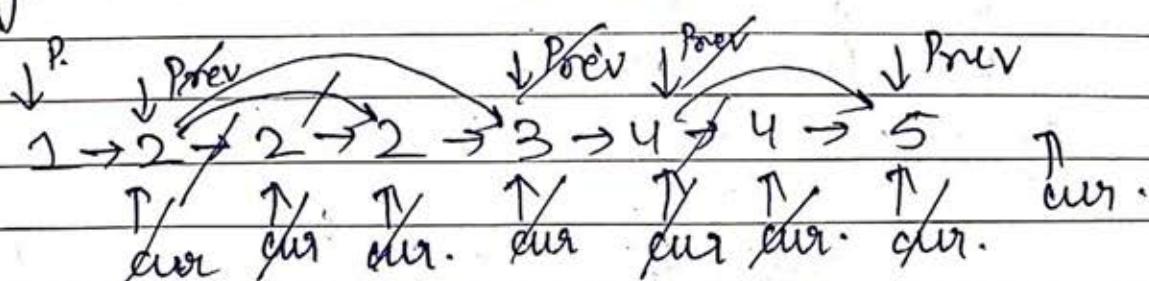
 } else {

 prev = cur;

 cur = cur.next;

}

eg: 1 → 2 → 2 → 2 → 3 → 4 → 4 → 5



Ans 1 → 2 → 3 → 4 → 5 Ans

Q Remove Duplicates from Unsorted LL

What?

Given an unsorted LL remove all the duplicates from the list such that there is only one occurrence 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;

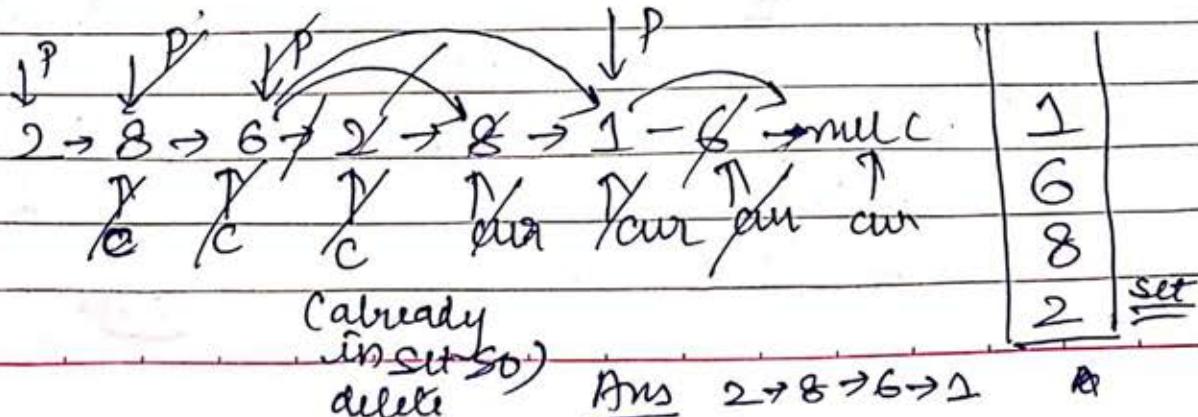
 else {

 prev = cur;
 cur = cur.next;
 set.add (prev);

}

}

eg: 2 → 8 → 6 → 2 → 8 → 1 → 6



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.

flow?

We will break the links 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;

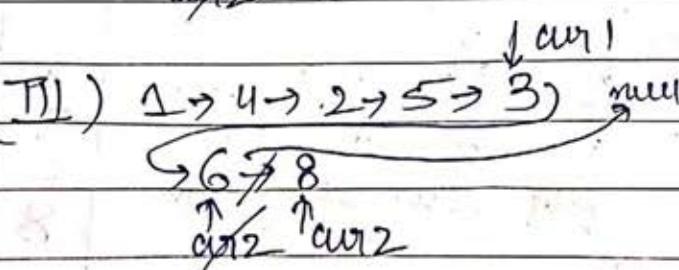
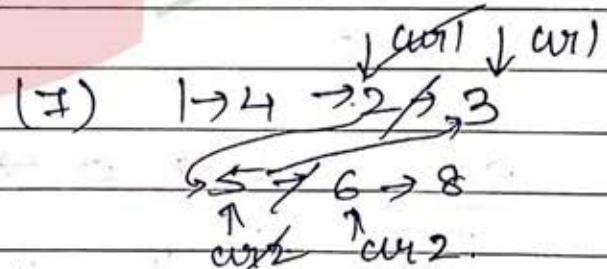
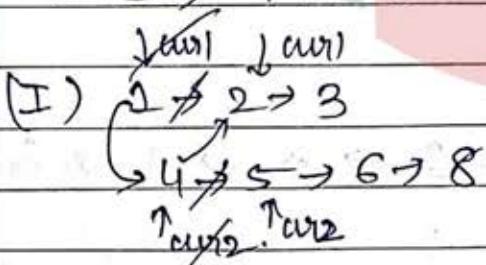
 cur2.next = next1;

 cur1 = next1; cur2 = next2;

}

eg: L1 = 1 → 2 → 3

L2 = 4 → 5 → 6 → 8.



(IV) L1 = 1 → 4 → 2 → 5 → 3 → null
 L2 = 8

Ans

Q 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)

 2 node = removeFirst () → from cur list

 if (cur.data == vowel)

 addLast in vowel list (node)

 else

 addLast in consonant list (node)

3

if (vhead != null)

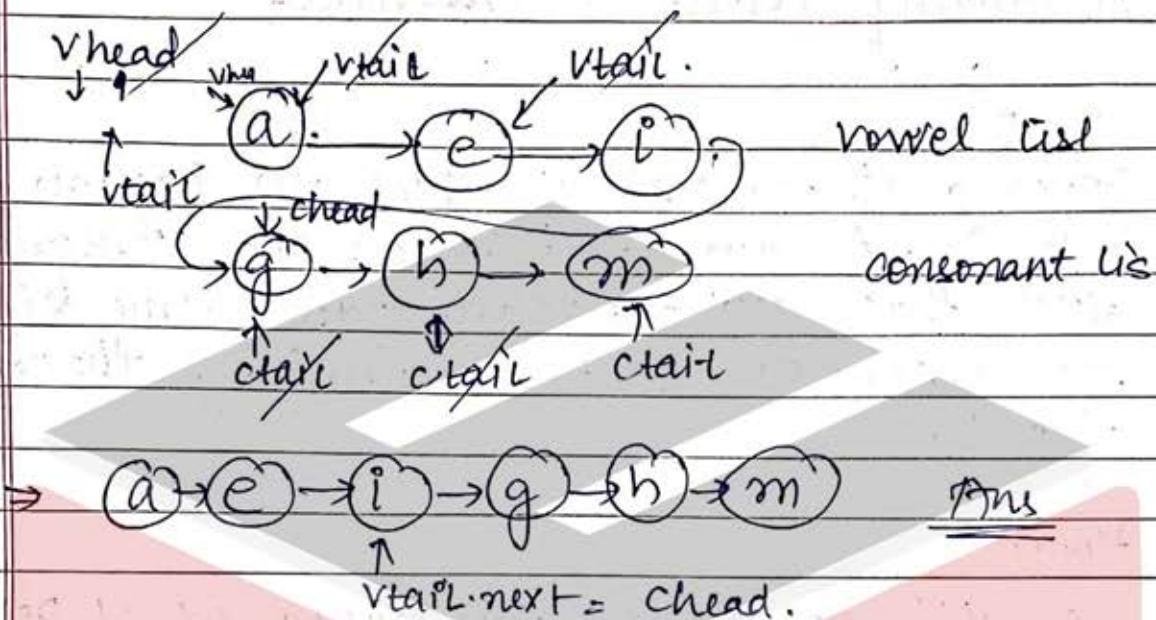
 vtail.next = chead;

 return vhead;

else

 return chead;

eg: $\alpha \rightarrow e \rightarrow g \rightarrow h \rightarrow i \rightarrow m$



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

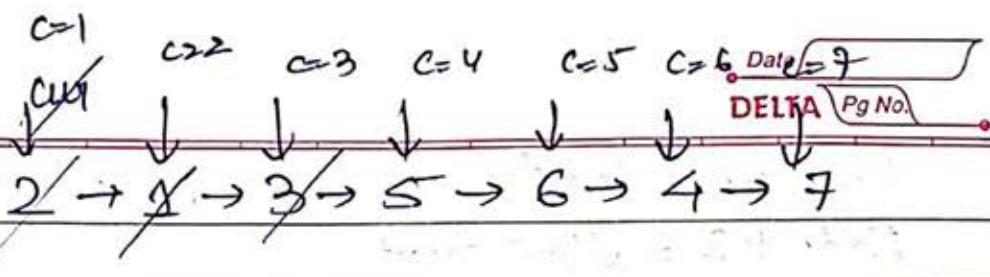
while list is not empty.

If (counter % 2 == 0)

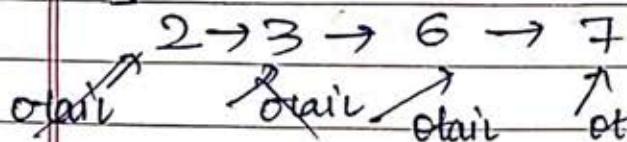
 add last in even list

else

 add last in odd list.

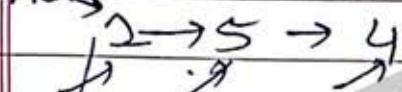


ohead

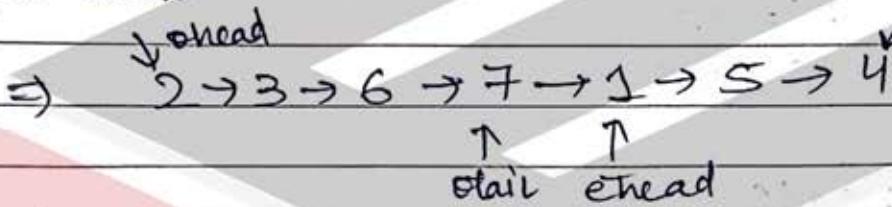


$$\text{otail} \cdot \text{next} = \text{ohead}$$

ohead



ohead otail otail



head = ohead.

tail = otail

Q FOLD

what?

Given a linked list $l_0 \rightarrow l_1 \rightarrow l_2 \rightarrow \dots \rightarrow l_{n-1} \rightarrow l_n$.

Arrange linked list so the new linked list formed is $l_0 \rightarrow l_n \rightarrow l_1 \rightarrow l_{n-1} \rightarrow l_2 \rightarrow \dots$ and soon.

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 alternately merge 2 linked list.

Node $h_1 = \text{head}$

Node $h_2 = \text{slow} \cdot \text{next}$

$\text{slow} \cdot \text{next} = \text{null}$.

while (~~h_1~~ $= \text{null} \& \& h_2 \neq \text{null}$)

{ Node $on = h_1 \cdot \text{next}$;

Node $tn = h_2 \cdot \text{next}$;

~~$h_1 \cdot \text{next} = h_2$~~ ;

~~$h_2 \cdot \text{next} = on$~~ ;

$h_1 = on$;

$h_2 = tn$;

return head.

eg: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

1) Find middle.

$\downarrow^s \downarrow^s \downarrow^s$

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

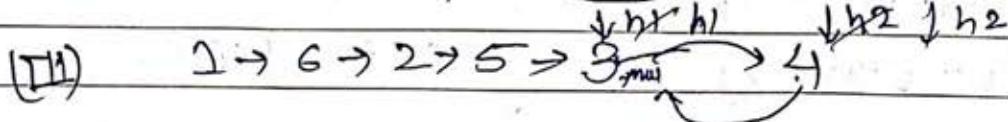
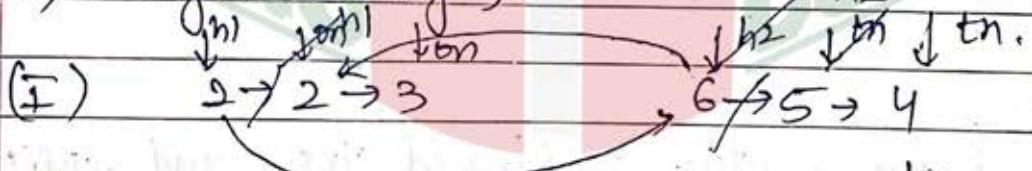
\uparrow_f

\uparrow_f

\uparrow_f

2) Reverse 2nd half $1 \rightarrow 2 \rightarrow 3$ $6 \rightarrow 5 \rightarrow 4$

3) Merge Alternately



$1 \rightarrow 6 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4$

Ans.

Q Unfold

What?

Given a linked list in form $l_0 \rightarrow l_n \rightarrow l_1 \rightarrow l_{n-1} \rightarrow l_2$ and so on rearrange in linked list in following form $l_0 \rightarrow l_1 \rightarrow l_2 \rightarrow l_3 \dots l_{n-1} \rightarrow l_n$

How?

We will place first pointer at head and second pointer at head \rightarrow next.

2) Now applying following steps.

//head1 = head, head2 = head \rightarrow next.

while (first != null && second != null)

{

 first \rightarrow next = first \rightarrow next \rightarrow next;

 second \rightarrow next = second \rightarrow next \rightarrow next;

 first = first \rightarrow next;

 second = second \rightarrow next;

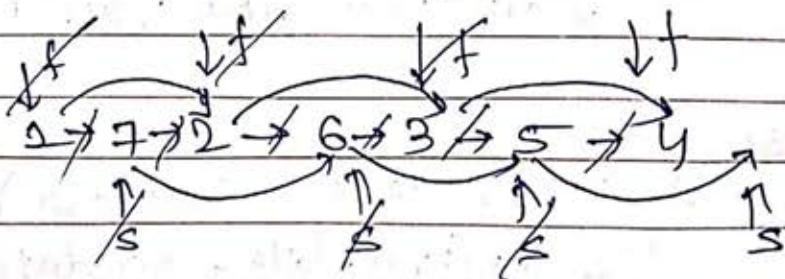
3 first.next = null;

// when size less than equal to 2 don't do anything.

3) Reverse the head2 list and combine both list.

eg:

$1 \rightarrow 7 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 5 \rightarrow 4$



$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ $7 \rightarrow 6 \rightarrow 5$

Reverse second half

$5 \rightarrow 6 \rightarrow 7$

Combine

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$ Ans.

first \rightarrow next = head2;

Q) Zig Zag linked list

What?

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

How?

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

prev = head, cur = head \rightarrow next;

flag = 1;

while (cur != null)

{

 if (flag == 1)

 if (prev.data > cur.data)

 swap (prev.data, cur.data);

 3

 else {

 if (prev.data < cur.data)

 swap (prev.data, cur.data);

 3

 3 flag = # = -1;

eg: $11 \rightarrow 15 \rightarrow 20 \rightarrow 5 \rightarrow 10$

\downarrow Prev

(I) $11 \rightarrow 15 \rightarrow 20 \rightarrow 5 \rightarrow 10$ flag = 1

(II) $11 \xrightarrow{\downarrow P} \cancel{15} \rightarrow 20 \rightarrow 5 \rightarrow 10$ flag = 0 - 1

(III) $11 \rightarrow 20 \rightarrow \cancel{15} \rightarrow \cancel{5} \rightarrow 10$ flag = 1

(IV) $11 \rightarrow 20 \rightarrow 5 \rightarrow 15 \rightarrow 10$ flag = 0 - 1
 $\cancel{15}$ cur, cur.

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++) {
```

Note headⁱ = arr,

Node prev = null;

```
for (int j=0; j<width + (i<rem ? 1 : 0); j++)
```

prev = cur;

cur = cur->next;

3

prev->next = null; // setting tail's next null.

arr[i] = headⁱ;

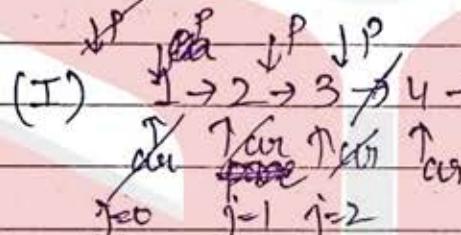
3

return arr;

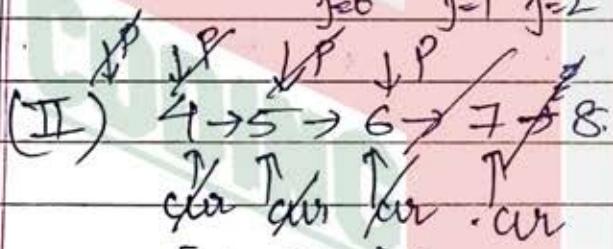
eg: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 K=3.

$$\text{width} = 8/3 = 2$$

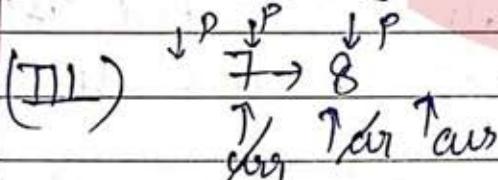
$$\text{rem} = 8 \% 3 = 2.$$



1 → 2 → 3



i=1, i<rem.



4 → 5 → 6

i=2 i<rem

(7-8)

(1 → 2 → 3)

(4 → 5 → 6)

(7 → 8)

K parts

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.

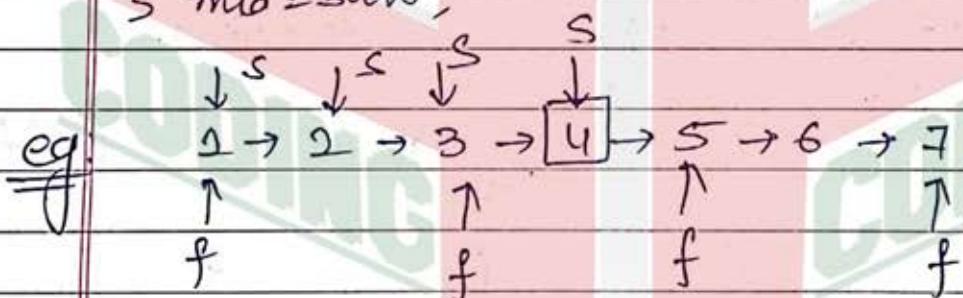
-while (fast->next) = null || fast->next->next) = NUL)

{

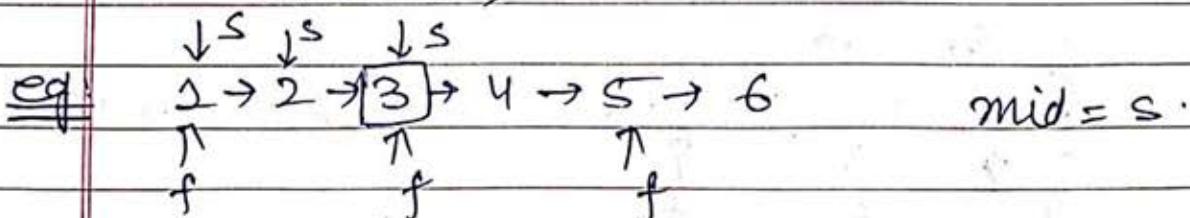
slow = slow->next;

fast = fast->next->next;

3 mid = slow;



mid = slow;



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 posn.

prev=null, slow= ~~head~~^{head}, fast=head.

while (fast.next != null & fast.next.next != null)

2

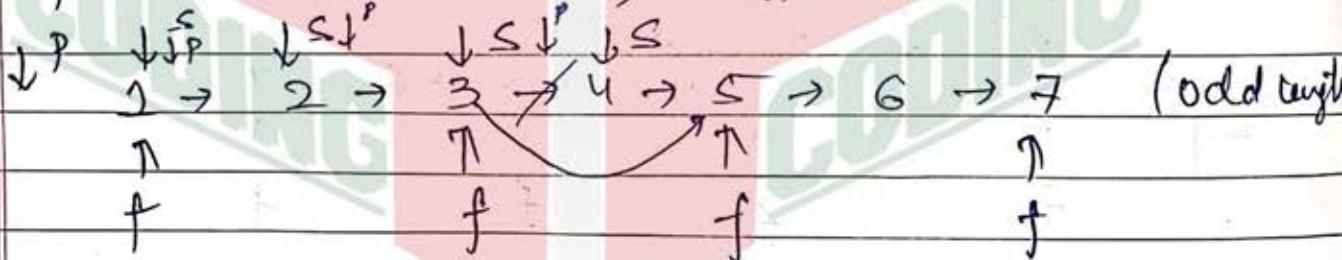
prev=slow;

slow=slow.next;

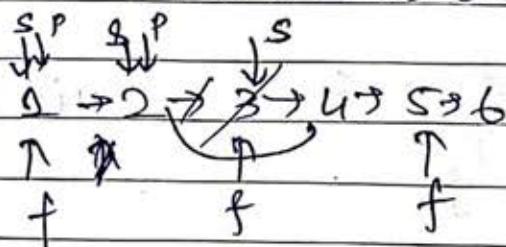
fast=fast.next.next;

3

prev.next=slow.next; // deletion.



$\Rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \dots$



$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6$

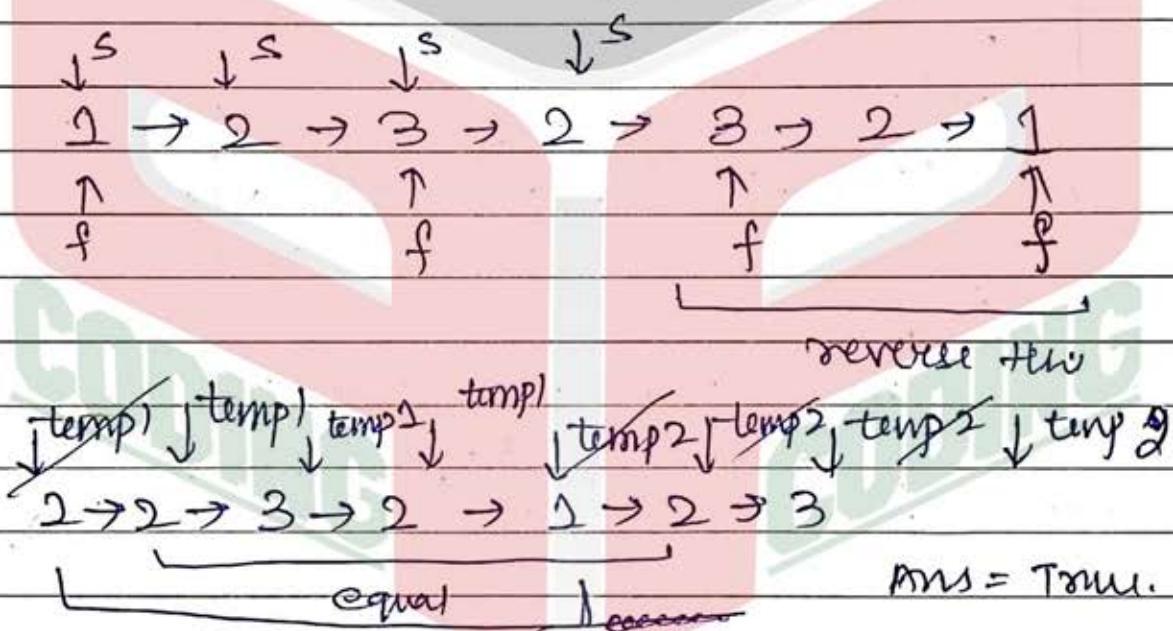
Q) Is linked list Palindrome

What?

Given a linked list . find if the content of the linked list form a palindrome or not.

How?

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



while ($\text{temp2} \neq \text{null}$)

{ if ($\text{temp1} \cdot \text{data} \neq \text{temp2} \cdot \text{data}$)
return false;

else

$\text{temp1} = \text{temp1} \cdot \text{next};$

$\text{temp2} = \text{temp2} \cdot \text{next};$

3

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.

$\text{temp} = \text{head};$

$\text{while}(\text{temp} \neq \text{NULL})$

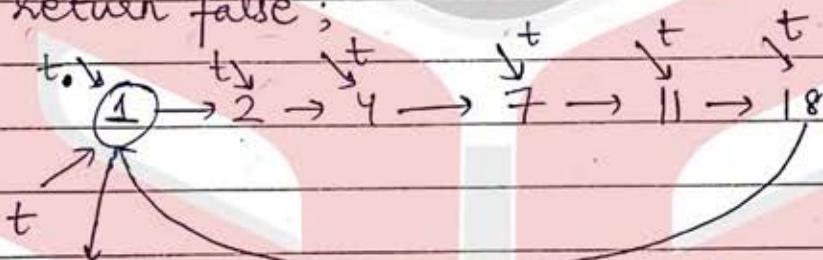
{ $\text{temp} = \text{temp} \cdot \text{next}$

 if ($\text{temp} == \text{head}$)

 return true;

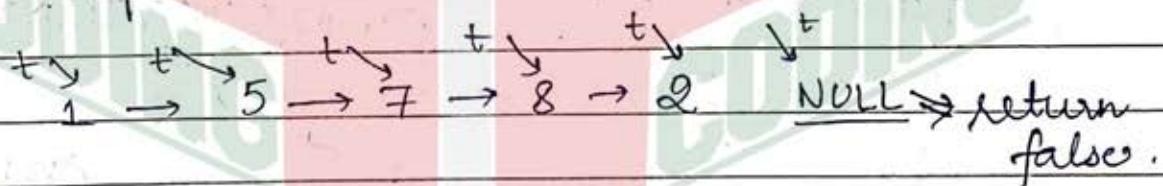
}
 return false;

ex:



$\text{temp} = \text{head} \Rightarrow \text{return true}.$

ex:



NULL $\Rightarrow \text{return false}.$

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;

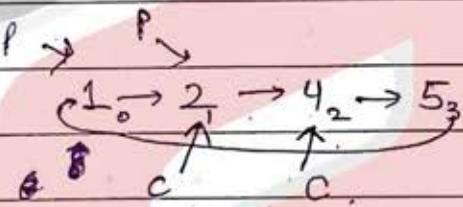
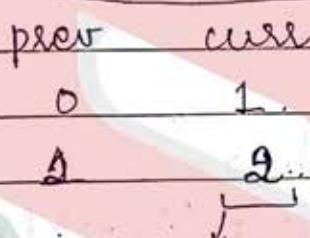
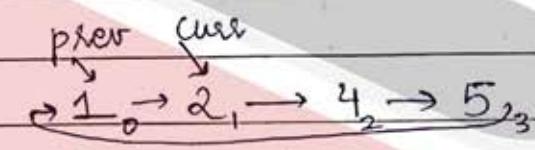
{

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)
```

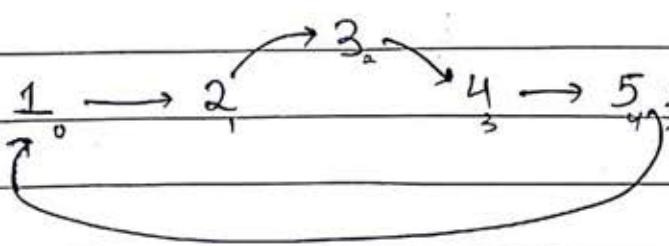
{ Node n = new Node(insertdata, curr);

prev.next = n;

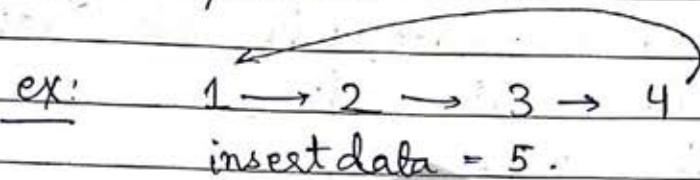
n.next = curr;

return head;

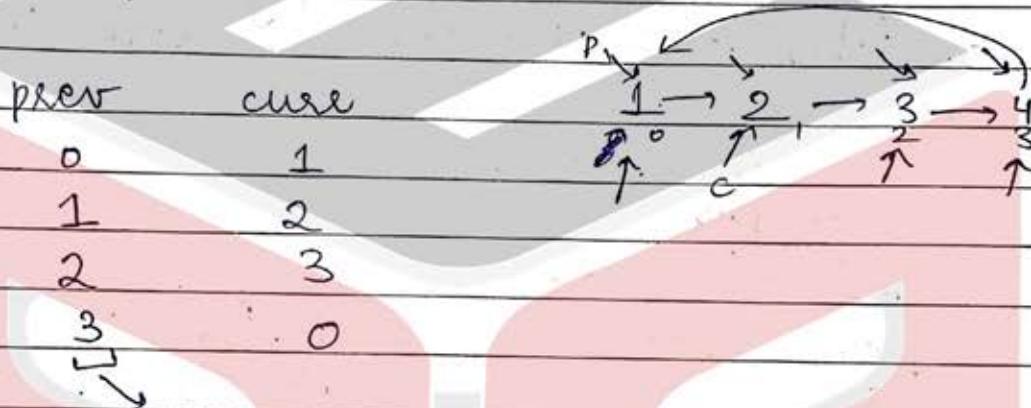
{



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



Check that when $\text{prev} \rightarrow \text{next} == \text{head}$, insert.



```

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;
}
    
```

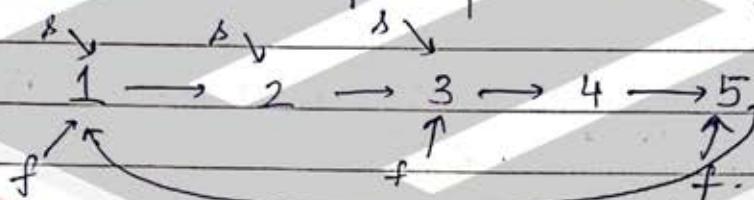
- Q. Circular linked list split in two halves.
What?

circular
 Split the given 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?

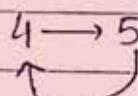
- * Use slow and fast pointers to find mid node.

ex:

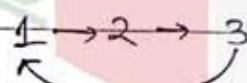


- * If ($\text{fast} \rightarrow \text{next} != \text{head}$)
 $\text{fast} = \text{fast} \rightarrow \text{next}$

- * Now simply break two linked lists as:
 L1 : $\text{fast} \rightarrow \text{next} = \text{slow} \rightarrow \text{next}$



- L1 : $\text{slow} \rightarrow \text{next} = \text{head}$.



- * Now return the two new heads:

$\text{head1} = \text{head}$;

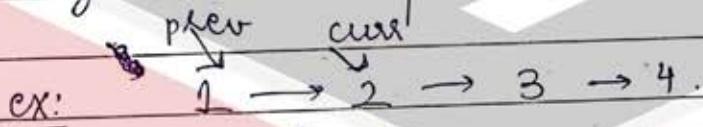
$\text{head2} = \text{head} != \text{fast} \rightarrow \text{next}$? $\text{fast} \rightarrow \text{next} : \text{NULL}$
 ↓
 when list can't be broken.

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.



prev = head;
curr = NULL;

while (prev != NULL)

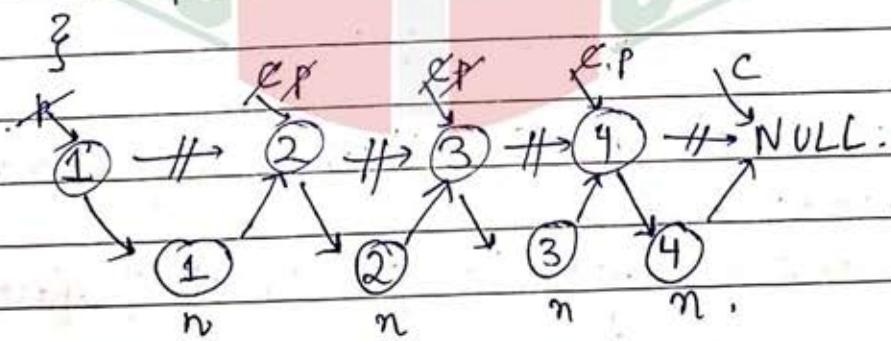
{ curr = prev → next;

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`

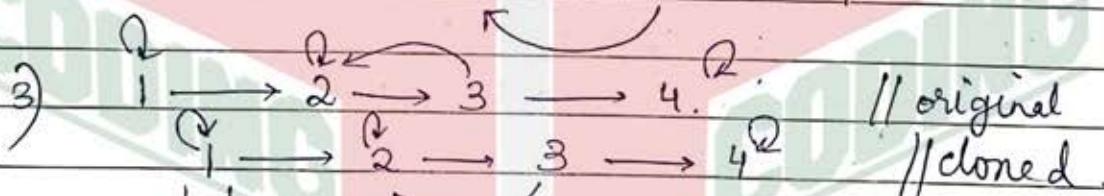
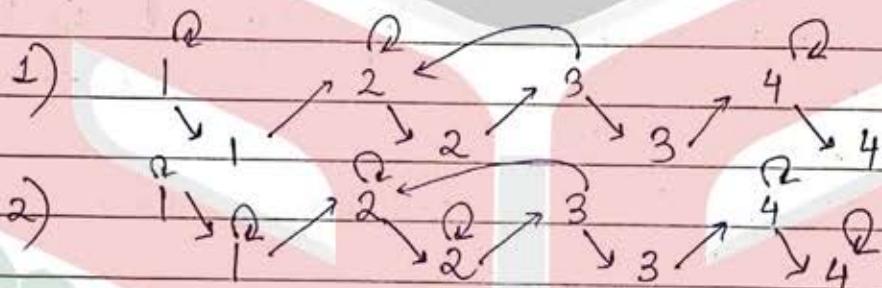
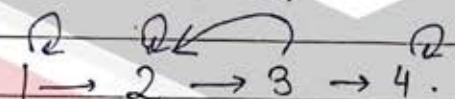
`// if curr→next is not null`

`curr → next = curr → next → next`

`// prev = prev → next , curr = curr → next`

`return newhead; // cloned list.`

ex:



`return nh (newhead).`

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:

$$1 \rightarrow 2 \rightarrow 9 \rightarrow 9$$

$$\text{rpi: } 9 \rightarrow 9 \rightarrow 2 \rightarrow 1$$

$$+ \quad 1$$

$$\overbrace{\begin{array}{cccc} 0 & \rightarrow & 0 & \rightarrow \\ c=0 & \rightarrow & c=1 & \rightarrow \\ 3 & \rightarrow & 1 & \rightarrow \\ c=1 & \rightarrow & c=0 & \end{array}}$$

$$\text{rpi: } \underbrace{1}_{\downarrow} \rightarrow 3 \rightarrow 0 \rightarrow 0$$

return head.

Similar : Add 2 linked lists.

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

Q. Multiplication of 2 linked lists.

What? given

You are given 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: $L_1 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow \dots$
 $L_2 \rightarrow 2 \rightarrow 3 \rightarrow \dots$

Repr L1: $6 \rightarrow 5 \rightarrow 4$

Repr L2: $3 \rightarrow 2 \rightarrow 1$

result list = empty initially.

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

$L_1: 6 \rightarrow 5 \rightarrow 4$.

temp list:

$8 \rightarrow 6 \rightarrow 3 \rightarrow 1$.

$L_2: \times 3$

$\begin{array}{r} ① \\ 8 \rightarrow 6 \rightarrow 3 \rightarrow 1 \end{array}$

Since count = 0,

result = spi(templist)

$$\therefore \text{result} = 1 \rightarrow 3 \rightarrow 6 \rightarrow 8,$$

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

$$11: \quad 6 \rightarrow 5 \rightarrow 4.$$

$$\overline{0 \rightarrow 2 \rightarrow 1 \rightarrow 9}$$

12 : x 2

$$\frac{1}{2 \rightarrow 1 \rightarrow 9}$$

addition of 2 linked lists

```
count = 1, result = result + rpi(template) -
```

$\Rightarrow \text{result} = 1 \rightarrow 3 \rightarrow 6 \rightarrow 8$

$$+ \frac{9 \rightarrow 1}{2 \rightarrow 0}$$

$$\overbrace{1 \rightarrow 0 \rightarrow 4 \rightarrow 8 \rightarrow 8}^{\text{ }} \leftarrow$$

$$\text{NPI 21: } 8 \rightarrow 6 \rightarrow 3 \rightarrow 1$$

api 12: 0 → 2 → 1 → 9

$$\overline{8 \rightarrow 8 \rightarrow 4 \rightarrow 0 \rightarrow 1}$$

api

$$\therefore \text{dns} = 1 \rightarrow 0 \rightarrow 4 \rightarrow 8 \rightarrow 8$$

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

flow?

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) {
        coeff = p1->coeff ; pow=p1->pow;
        p1 = p1->next;
```

```
    else if (p1->pow < p2->pow) {
        coeff = p2->coeff ; pow=p2->pow;
        p2 = p2->next;
```

else

3

else {

```
    coeff = p1->coeff + p2->coeff;
```

```
    pow = p1->pow ; p1=p1->next; p2=p2->next;
```

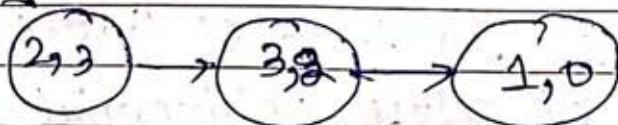
3

```
    sys0( #coeff + "x^" + pow );
```

3

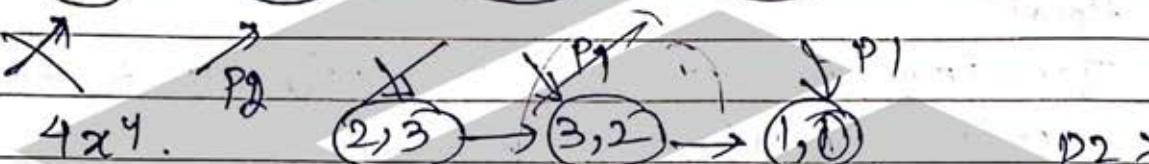
// If any of p1 or p2 is not null
 print rest of the polynomial

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

 $\downarrow P_2$ 

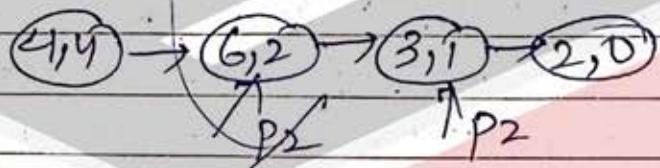
(I)

$4x^4$



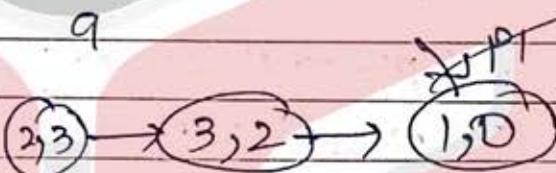
(II)

$4x^4 + 2x^3$



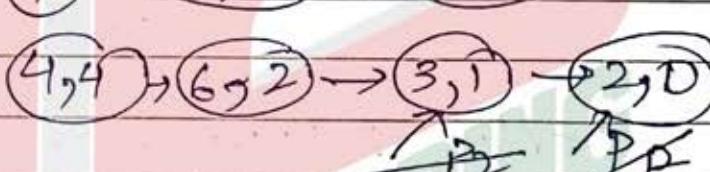
(III)

$4x^4 + 2x^3 + 9x^2$

~~+ 3x~~

(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 nodes. Sort the linked list using merge sort.

flow?

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 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 linked list blist
 blist.addLast(list.head.data);
 return blist;

}

fhalf = new LinkedList();

shalf = new LinkedList();

mid = list.mid();

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;

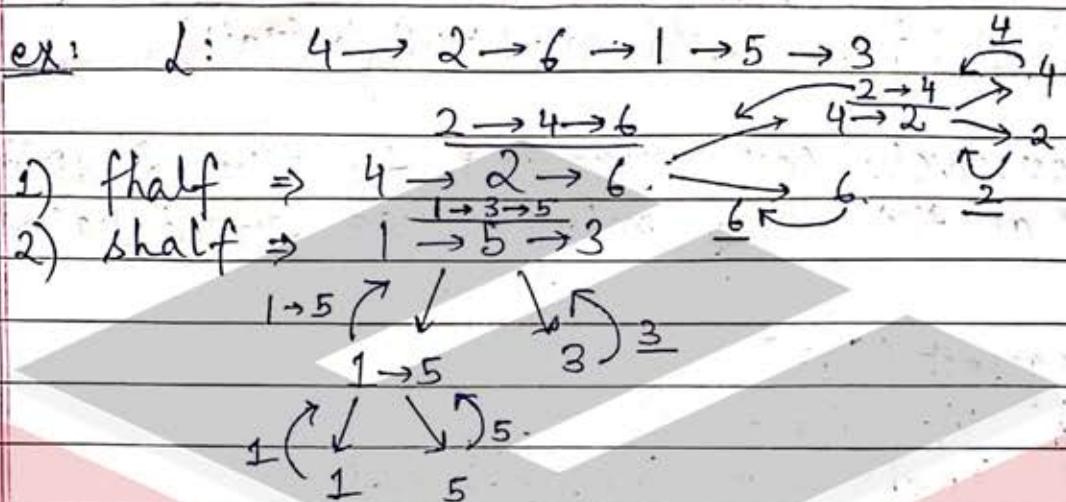
shalf.size = list.size - fhalf.size;

fhalf = Mergesort(fhalf);

shalf = Mergesort(shalf);

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

$\text{mid}.\text{next} = \text{mid} \&;$ // to maintain original list
 return res;

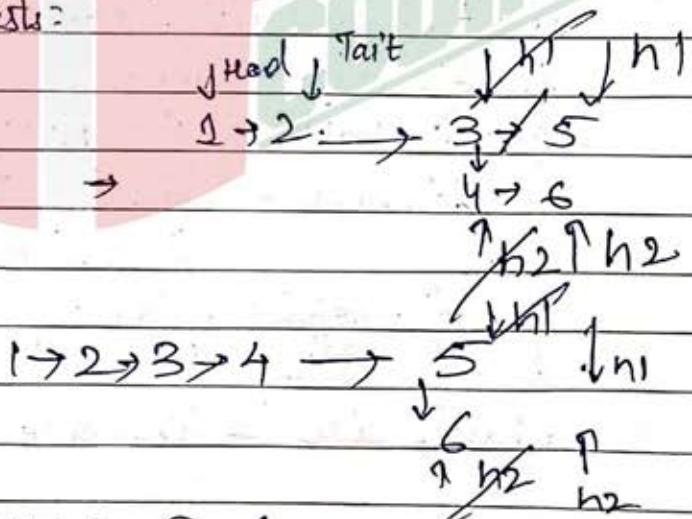
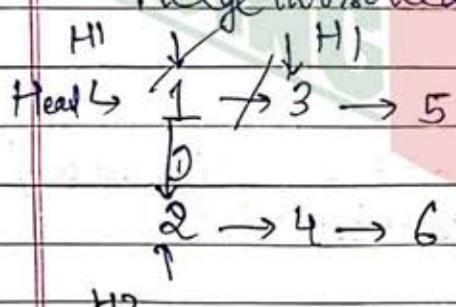


$1 \rightarrow 3 \rightarrow 5$

$2 \rightarrow 4 \rightarrow 6$

ans: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

Merge two sorted lists:



Ans $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

Node getmidnode (Node lo , Node hi)

{

slow = lo ;

fast = lo ;

while (fast != high) ;

{

slow = slow . next ;

fast = fast . next . next .

}
return slow ;

}

// conditions as
get mid function

void mergesort (Node lo , Node hi)

{

Node mid = getmidnode (lo , hi) ;

mergesort (lo , mid) ;

mergesort (mid . next , hi) ;

merge sorted list (lo , mid , high) ;

}

// Inplace mergesort .

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: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$: dist

set: 0, 3, t, 4

Q. Kth node from last.

what?

Given a list, return Kth node from last.

How?

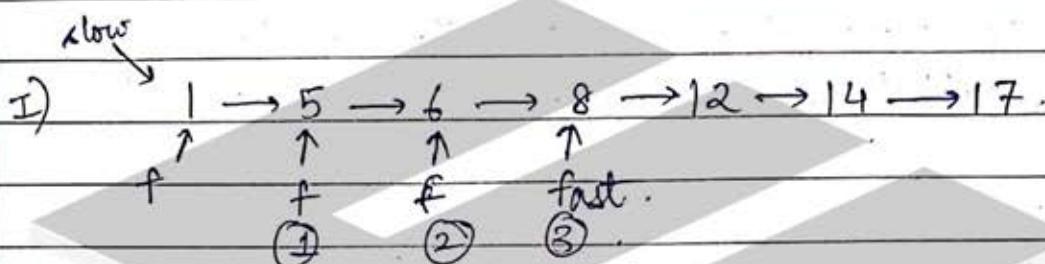
- * Keep a pointer fast and do fast = fast → next k times. Now, keep pointer slow at head to maintain

a difference of k between slow and fast.

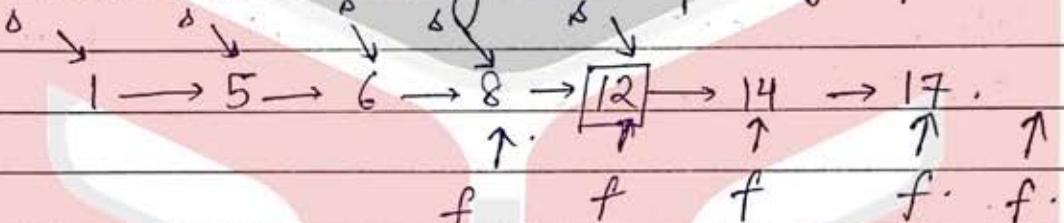
- * Now move both one by one, when fast points to NULL, slow is at k th from last node.

ex: $1 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 12 \rightarrow 14 \rightarrow 17$.

$k = 3$.



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



when $\text{fast} == \text{NULL}$, $\text{slow} = 12$ which is k th node from last ($k=3$).

ans = 12

\rightarrow Node* slow = head, *fast = head;

for (int i = 0; i < k; i++)

 fast = fast \rightarrow next;

while (fast != NULL)

{ slow = slow \rightarrow next;

 fast = fast \rightarrow next;

}

return slow \rightarrow data;

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?

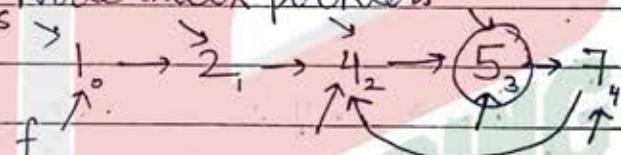
- * Keep two pointers slow and fast, both pointing to head initially.
- * Move slow by one & fast by two at each step, if they meet at a point, loop exists.

ex:



slow	fast
0	0
1	2
2	4
3	3

Node index pointers



\Rightarrow detected loop at 5, return true.

while ($fast \rightarrow next \neq NULL \&& fast \rightarrow next \rightarrow next \neq NULL$)

$slow = slow \rightarrow next;$

$fast = fast \rightarrow next \rightarrow next;$

 if ($slow == fast$)
 { return true; }

Q Cycle starting Node.

what?

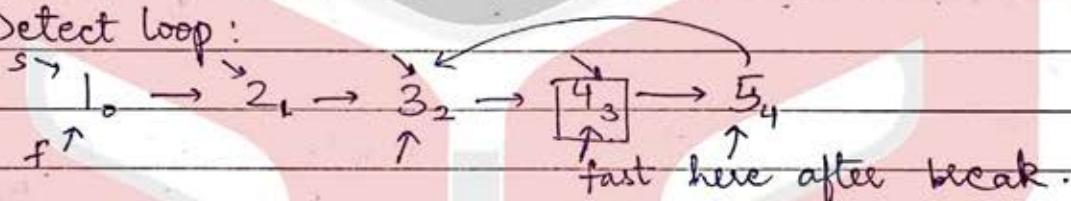
You are given a linked list which has N nodes. Return the cycle starting node.

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.

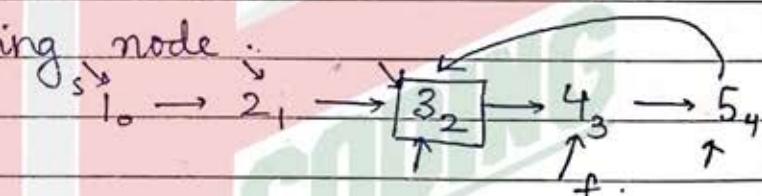
ex: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

I) Detect loop:



II) Find cycle starting node:

Slow	Fast
0	3
1	4
2	2



2. \Rightarrow return 3₂ 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;
```

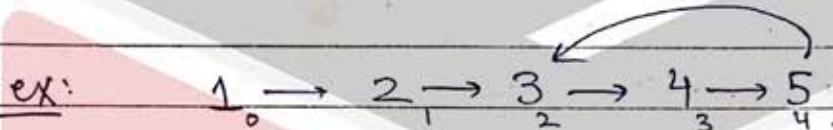
Q. Detecting length of loop:

What?

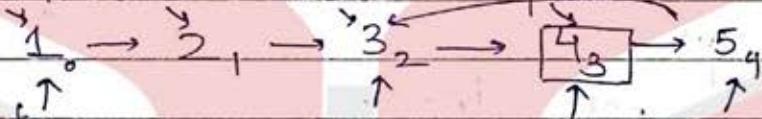
You are given a linked list with N nodes, return the number of nodes in the loop in list.

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 $\text{slow} \rightarrow \text{next} \neq \text{fast}$.

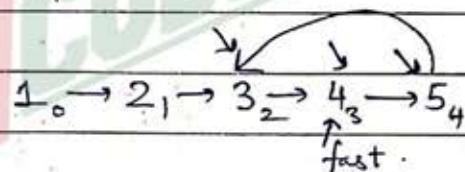


I) Loop exists at $\text{slow} == \text{fast}$ at 4.



II) $\text{while } (\text{slow} \rightarrow \text{next} \neq \text{fast})$ # Counter is initially 1
 $\text{count}++;$
 $\text{slow} = \text{slow} \rightarrow \text{next}$

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



\therefore length of loop = 3.

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.

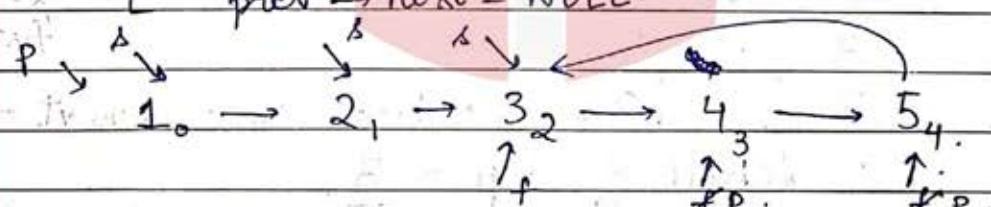
ex: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$



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. \Rightarrow slow == fast, $\therefore p \rightarrow \text{next} = \text{NULL}$.

results in:-

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