Date .....

```
public class LL{
    private Node head;
    private Node tail;
    private int size;
    public LL(){
     this.size=0;
    }
  //Insert at First
   public void insertFist(int val){
    Node node = new Node(val);
    node.next = head;
     head =node;
      if(tail == null){
       tail= head;
       }
      size+=1;
   }
  //Insert at Last
     public void insertLast(int val){
      Node node = new Node(val);
       tail.next =node;
        tail= node;
         size++;
       if(tail==null){
         insertFirst(val);
            return;
          }
       }
```

Spiral

Date .....

```
//Display
public void display(){
   Node temp = head;
    while (temp != null){
     System.out.print (temp.val + " → ");
       temp = temp.next;
     }
  Sysout("END");
}

//Insert at any specific index
   public void insert (int val, int index){
      if (index == 0){
        insertFirst(val);
          return;
        }
      if (index == size){
        insertLast(val);
          return;
        }
      //otherwise check from head to our index
      Node temp = head; //0 itself the loop from 1 to index
      // add element at index 3 means iterate from
          1 to 2
      for (int i=1; i< index; i++){
          temp = temp.next;
          }
      Node node = new Node (int val, int index);
                                          tempnext
        temp.next = node; //created node
        size++;
        }
```

Spiral

Date .....

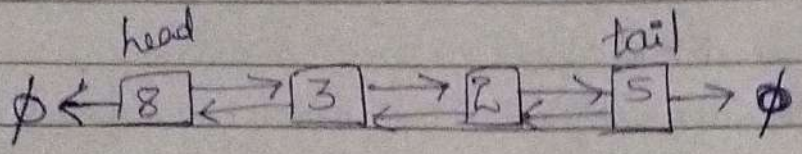
delete first de from Linked list.

```
public int deleteFirst(){
    int val = head.value;
    head = head.next;          //i.e., if list has only 1 element
    if(head==null){            //then it deleted after head is
        tail = null;           //null because no value is there
    }                          at next.
    size--;                    list: {3} → delete first
    return val;                      {3} → next = null
}                              /* Hey head = tail = null */
```

Spiral

Date 29/4/25

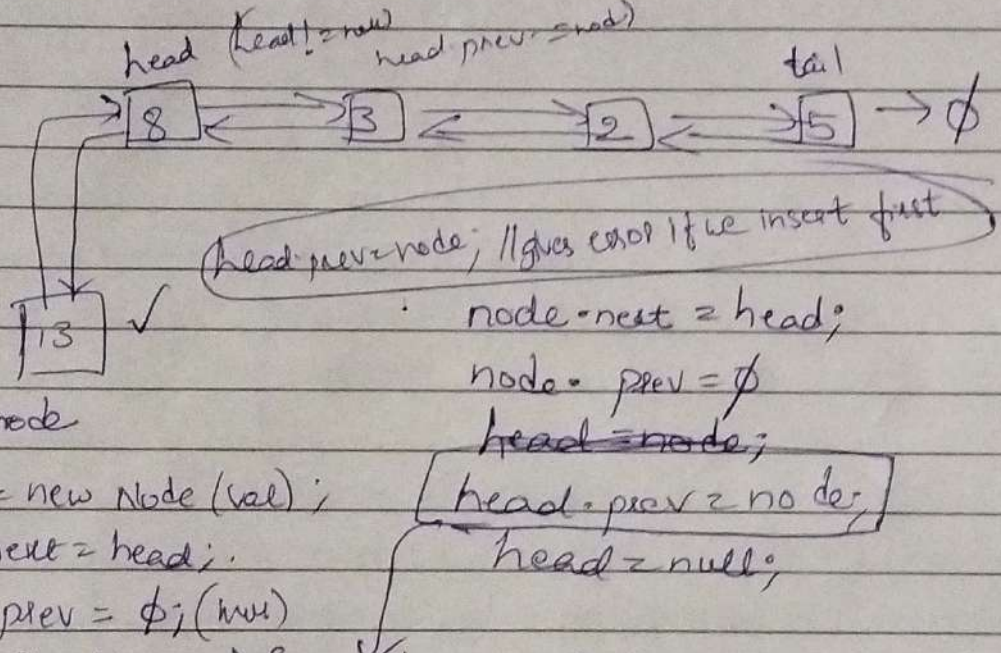
Doubly linked list :- Duties we traverse back it is useful than S.L.L.



→ Every single box having the structure :-

```
class Node {
    int val;
    Node next;
}
```

① insert 13 at first then

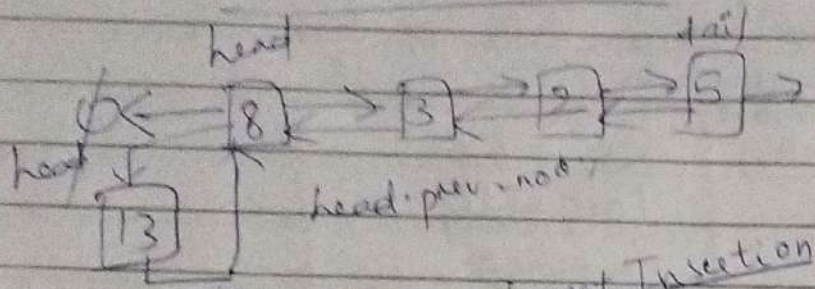



```
Node node = new Node(val);
node.next = head;
node.prev = ϕ; (null)
if (head != null) {
    head.prev = node;
}
head = node;
}
```

here we check for null pointer Exception

Date .....

Doubly Linked list (Backwards traversal)



First Element Insertion of List

```
Node node = new Node(val);
// node.next = head;
head = node;
node.prev = null;
head.next =
head.prev = node;
```

```
node.next = node;
node.prev = ∅;
head = node;
head.prev = node;
```

Causes 3 are satisfied

interview level question

→ Causes Null Pointer Exception then checks it.

Micro) Mic 101 is hiring

Date .....

// display() Method

Node node = head; ~~last~~ Node last = null;

while (node != null) {

print(node.val + " ->");

node = node.next;

last = node.

}

Sout("END");

}

last = null

last = node;

also display in Reverse

→ firstly we need tail node here;

public display() {

Node node = head;

Node last = null;

while (node != null) {

print (node.val + " ->");

node = node.next;

last = node; node = node.next;

}

Sout("END");

sout("In Reverse:");

while (last != null) {

Sout(last.val + " ->");

last = last.prev;

}

} last = null

1 → 2 — 3 here

last = node; head;

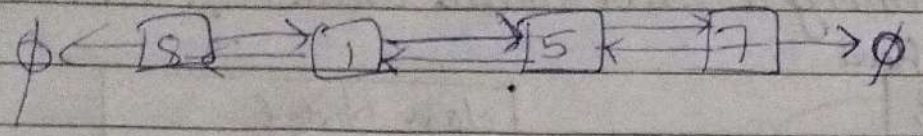
node = node.next

last = node;

Spiral

Date .....

Insert element at last




```
node = last.next;
node.next = ϕ;
node.prev = last;
if (head == ϕ){
    head.prev = ϕ;
    head = node;
}
```

* for given order in the list :-
traverse the list is;

7 -> 5 -> 1 -> 8 -> START

* Insertion at middle of the element is also just like as same as we did in the Single linked list.

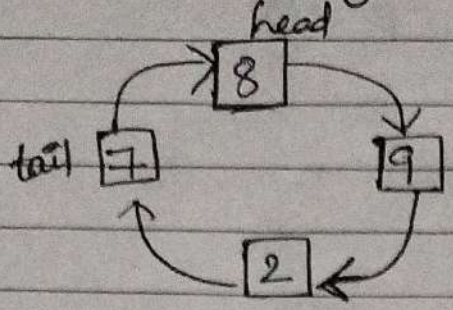
// Method to find out the value & particularly return that node then

```
public Node find(){
    Node node = head;
    while(node != Null){
        if (node.val == value){
            return node;
        }
        return null; node = node.next;
    }
    return null;
}
```

Spiral

Date .....

* Circular linked list :- (only 2 operations :- deletion & Insertion)
A linked list in a chain.
// here creating a box is called Node. i.e., class btw,

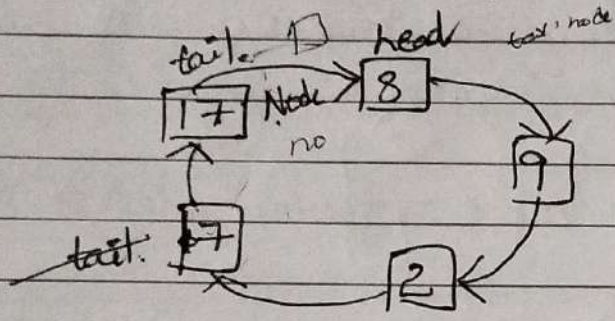



```
class Node{
    int val;
    Node next;
}
```

// Just like in the Single linked list.
// Here no things are null. no one is pointing to the null until the list is empty

Insertion :-

1) After the tail :-
insert 17 after tail.



Assumptions :-

// Create a new node first.
tail
Node.next = node
tail = node
node.next = head;
if (head == null){
    tail = node;
    head = node;
}

Date .....

```
CLL {
 private class Node {
      int val;
     Node next;
     Node (int val) {
      this val = val;
      }
  private Node head;
  private Node tail;
      CLL () {
    this.head = null;
    this.tail = null;
    }
  public void insest (int val) {
        Node node = new Node (val);
          if (head == null) {
              head = node;
              tail = node;
              return;
             }
            tail.next = node;
            node.next = head;
             tail = node;
           }
          public void display () {
            Node node = head;
              while (node != head) {
                  node = node.next;
                }
          }
```

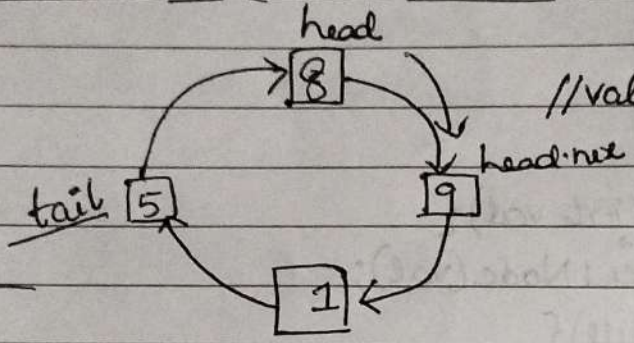
Spiral

Date .....

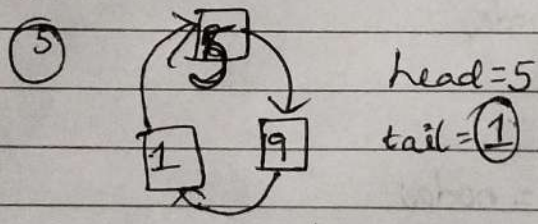
also we can do it by do-while

```
if (head != null) {
    do {
    System.out.println(node.val + "->");
    node = node.next;
    } while (node != head);
  } System.out.println("HEAD");
}
```

* Deletion in circular linked list



//here you can't reach it to 5 because it is the de to deleted



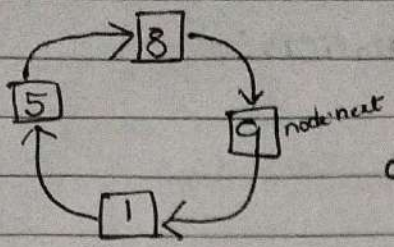
head = head.~~to~~ next

→ * 8 = node; if we want to delete 9 then then node starts from 1 to the value we want to delete.

for example node.next == val (we want to delete)
node.next = n.next.

Spiral

Date .....



delete 1 then

(1) Consider node = 8:

node.next = 9

(node.next == val) X then go 2nd pass

(2) node = node.next ⇒ 9

then node.next = (1)

So, here node.next == val

1 == 1

then update the criteria to the CL: List.

i.e.,

node.next = node.next.next;

(3) Here we are working with 2 pointers.

```
public void delete (int val.) {
    Node node = head;
    if (node == null) {
        return;
    }
    if (node.val == val) {
        head = head.next;
        tail.next = head;
    } return;
    }

//otherwise I need to check the Item.
```

Spiral

Date .....

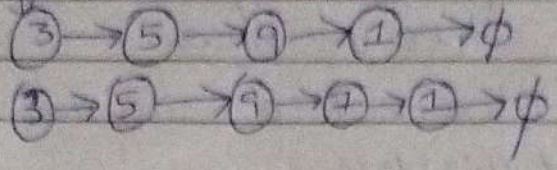
// By using Do-while loop consideration :-

```
do{
Node n = node.next;
    if (n.val == val){
        node.next = n.next;
        break;
        }
    node  = node.next;
    } while (node != head);
//we need to do this until node ≠ head.
```

Date .....

* Recursive Insertion in linked list

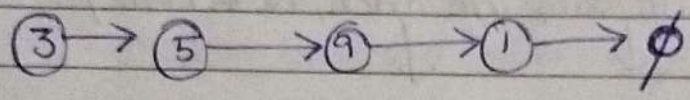
Ex: head



2 things to do in recursive LinkedList:

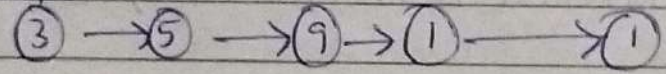
1) void return type & changes make in linked list.
2) node return type that returns the list node to change the structure.

head to index = 3, val = 7 (3, 7) (index, val) → using this we can insert it.



pass-1:

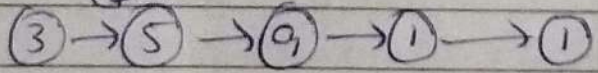
(3, 3) X



// then it create

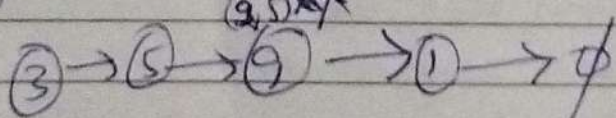
pass-2:

(3, 5) X



pass-3:

(3, 5) X X



// then here at this index add new Node & then traverse the operations here.

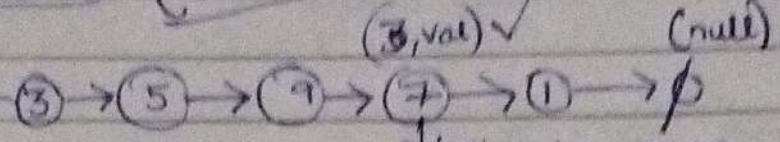
Spiral

//Stack Overflow condition :-

(sum > (Max val - least)/10) {
return 0;
}

Date .....

pass-3 :- (final result)



→ instead of 7, we need to write the value we want to insert at this place.

Things to to :-

1) Visualize what should we do :-

2) Try go to get what nodes Could the connections appropriately (like exactly without misinterpret) i.e, node, next, prev etc.

public Node insertRec(int val, int index, Node node)

↓ ↓ ↓

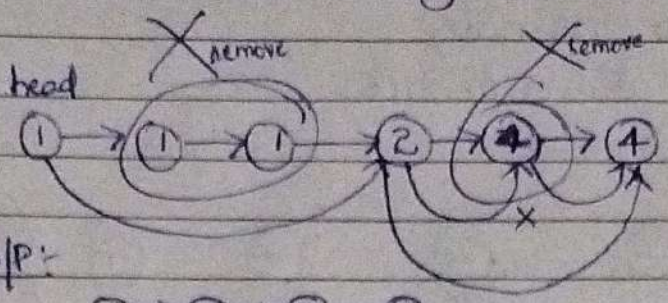
what val is to be add? at? index we need to add? reference

public Node insertRec(int val, int index, Node node) {

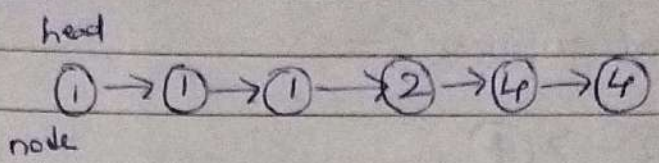
Scanned with OKEN Scanner

Date .....

Remove duplicates from single linked list:-



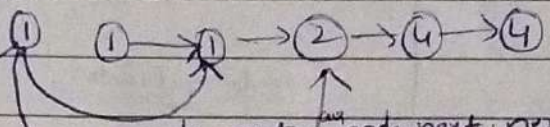
pass-1:



node = node next

skip (or) delete that element.

pass-2:



pass-3 is do check (node.next == node)

then also skip it (or) delete here the node.

pass-4:

Curnode.next = Curnod.next.next;

//then 1 == 2 X

then add / move ahead.

Cur.node.next = Cur node.; {No it's not}

//then simply node = node.next.

pass 5:- Again node.next = node, No (its not) then move foward.

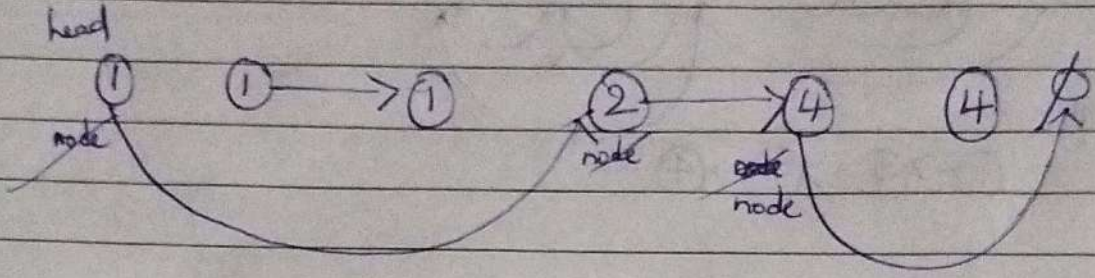
Spiral

Date .....

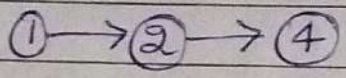
pass - 6:

node.next = curr.node;

then move forward to null.



then final answer is:-



Hese node stasts from head & ends at tail:-

functionality:

i.e., tail = node;
tail.next = ϕ

node.next=node;

node.next = node.next.next;

node.next = node;

node.next = node.next; next

node.next != node

node = node.next;

node.next=node;

node.next=node;

node.next != node

node=node.next;

Date .....

```
public void duplicates() {
    Node node = head;
    while (node.next != null) {
        if (node.val == node.next.value) {
            node.next = node.next.next;
            size--;
        } else {
            node = node.next;
        }
    }
    tail = node;
    tail.next = null;
}
```

Spiral

Date .....

* Merge two sorted lists :-

List-1: head1 → 1 → 3 → 4

List-2: head2 → 1 → 2 → 9 → 14

} Both are sorted

* final Result: [1→1→2→3→5→9→14]

//By comparing smaller elements from both 2 lists & add it to the new lists

pass-1:

h1 h1 h1 (h1.value < h2.value)

1 → 3 → 5

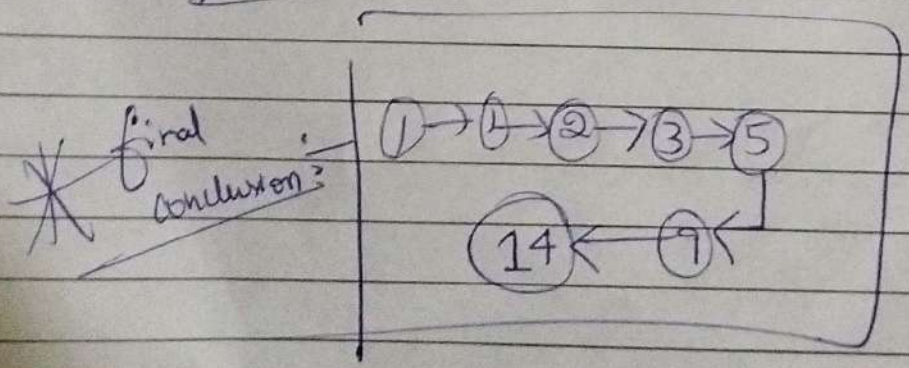
h2 h2 l2

while(node != temp) 1 → 2 → 9 → 14

h1 < h2 (any 1)

(2) h2 < h2 (h2++)

pass-2:



Spiral

Date .....

// Merge 2 sorted lists :-

```
ListNode dummy = new ListNode(0);
ListNode current = dummy;
  while (l1 != null && l2 != null) {
      if (l1.val < l2.val) {
        current.next = l1.val;
          l1 = l1.next;
        } else {
          current.next = l2.val;
            l2 = l2.next;
          }
      }
    //then current = current.next;
    }
  if (l1 != null)  current.next = l1;
  if (l2 != null)  current.next = l2;
//current.next = (l1 != null) ? l1 : l2;
  return dummy.next;
}
```

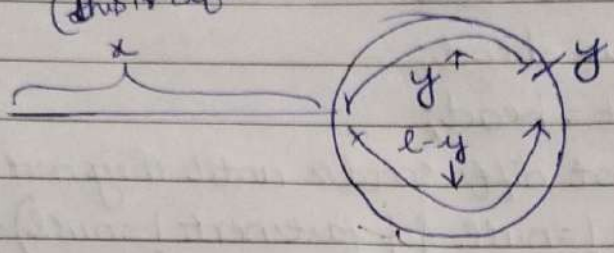
```
ListNode dummy = new ListNode(0);
ListNode current = dummy;
    while (l1 != null && l2 != null) {
      if (l1.val < l2.val) {
          current.next = l1;
          l1 = l1.next;
          } else {
          current.next = l2;
            l2 = l2.next;
          }
          current = current.next;
        }
      if (l1 != null) {
          current.next = l1;
          l1 = l1.next;
```

Spiral

Date .....

Floyd-warshall Algorithm (Hare & Tortoise) for cycle detection at which position in list:

(this is before loop begins)



l = length of the cycle.

* Pointers Distance before meeting the loop:

slow = $x + n1 \cdot l + y$

↓

n1 means no of times by slow pointer rotates the loop

fast = $x + n2 \cdot l + y$ (same as n1)

here fast = 2 * slow (according to speed)

$x + n1 \cdot l + y = 2x + 2 \cdot n2 \cdot l + 2y$

$n1 \cdot l = x + y + 2n2l$

$x + y = (n1 \cdot l - 2 \cdot n2 \cdot l)$

$x + y = K \cdot l$

if $K = 1$ then

$x + y = l$

$\boxed{x = l - y}$ Similarly index detected here. Btw.

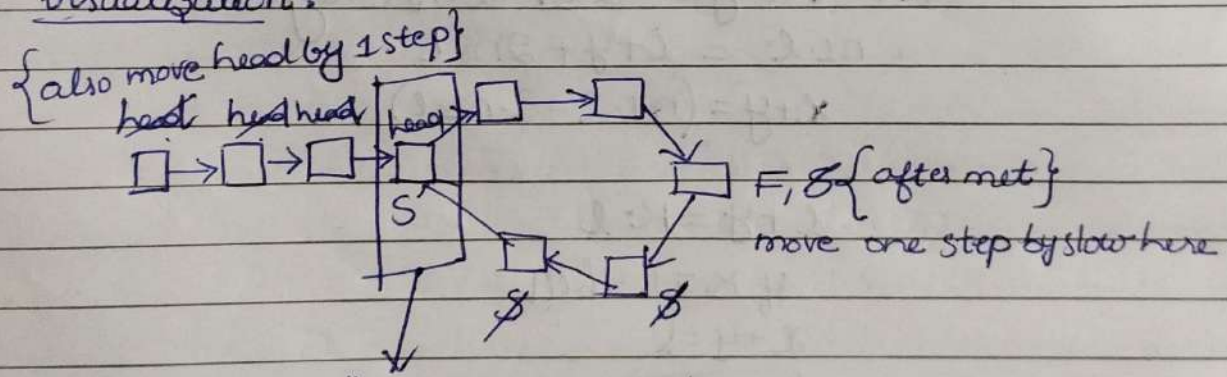
Spiral

Date .....

```
public ListNode detectCycle(ListNode head){
    //start both from head
    ListNode slow = head;
    ListNode fast = head;
    //advance both at diff speeds until they meet at once.
    while(fast != null && fast.next != null){
        fast = fast.next.next;
        slow = slow.next;
        if(slow == fast){
            while(head != slow){
                head = head.next;
                slow = slow.next;
            }
            return slow;
        }
    }
    return null; // If no ans found in the loop.
}
```

O(n/2) O(1)

visualization :-



// return this node
→ which has loop & cycle exists.

Spiral

Date .....

* Happy Number

$36 = 6^2 + 3^2 = 36 + 24$
$= 60$ } loop continues
$6^2 + 0 = 36$ }
// return false.

Efficient Approach

→ Consider a hashset of all integer values which is been squared & added to the defined variable.
→ If again it returns the value, that exactly presents in the hashset then it returns the false value, otherwise it returns true.

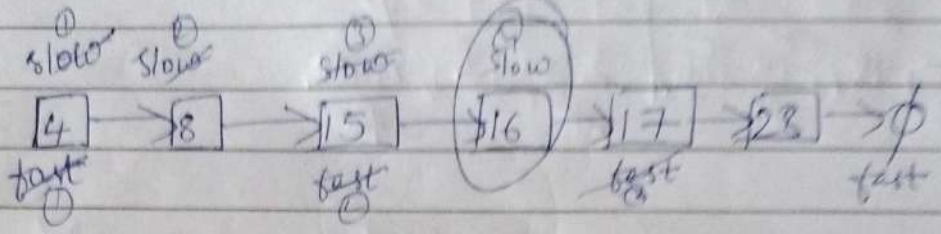
Dry Run of Code

```
boolean isHappy(int n){
Set<Integer> usedIntegers = new HashSet<>();
while(true){
//Find sum of squares
    int sum=0
    while (n!=0){
        sum += Math.pow(n%10,2.0);
        n=n/10;
        }
//If Sum is 1, then return true;
        if (sum==1) return true;
//else the current sum is the new number.
        n = sum;
//check if we have already encountered that Number.
    if(usedIntegers.contains(n))
            return false;
        usedIntegers.add(n);
    }
}
```

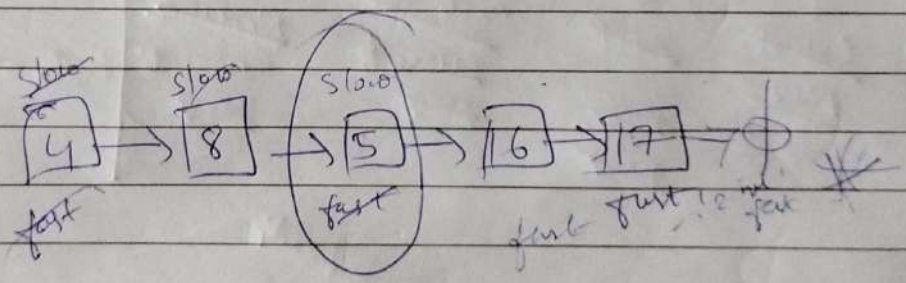
Spiral

Date .....

Middle of a linked list (876)




```
ListNode middleNode(ListNode head) {
    ListNode slowPtr = head;
    ListNode fastPtr = head;
//Travel until the fast pointer reaches until Last node or null.
    while(fast != null && fast.next != null){
        slow = slow.next;
        fast = fast.next.next;
    }
    return slowPtr;
}
```



sb: 4 8, 8, 5, 5, 6
8, 5, 6, 7, ∅

Spiral

Date .....

s s s s
4, 18, 15, 16, 23, 42, null
f f f f

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

23 42

fast = fast.next.next;
23 → null; // loop violated.

s s s
4, 18, 15, 15, 23, 27 ~~null~~
f f f

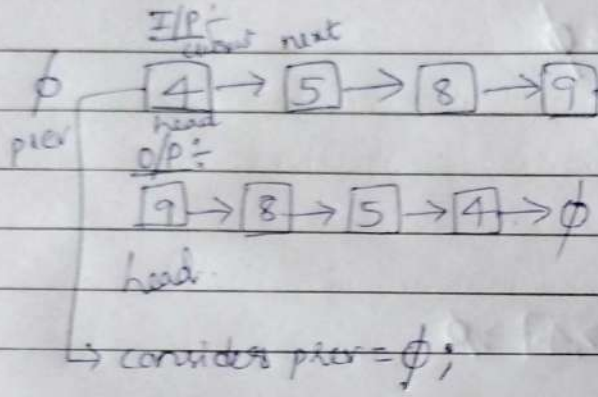
Date .....

Reverse a linked list

→ By using Stack Method but not efficient algorithm

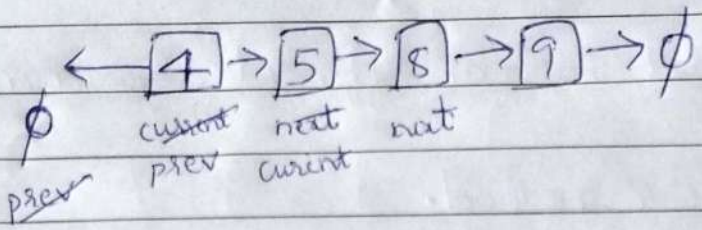
Efficient Approach:

→ using 3 pointers method.




```
current = head;
while (head != null){
    next = current.next;
current.next = prev;
    prev = current;
    current = next;
}
head = prev;
return head;
}
```

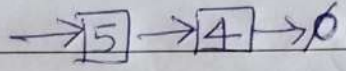
Spiral



current.next = prev = ∅

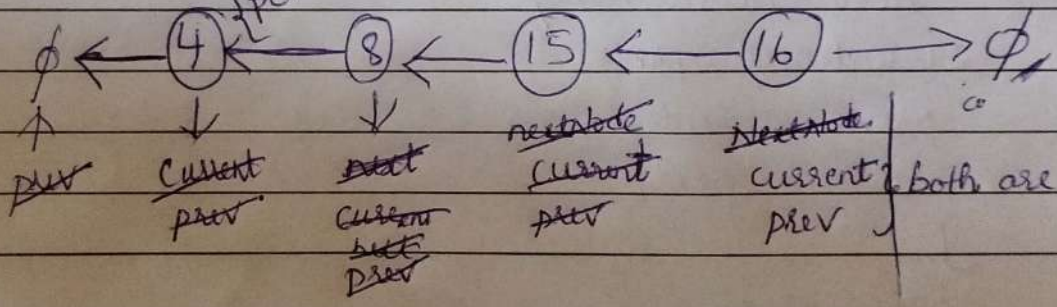
prev = current = 4

current = next = 5



① 1st pass

{pointer points to the reference of the next obj here}



1

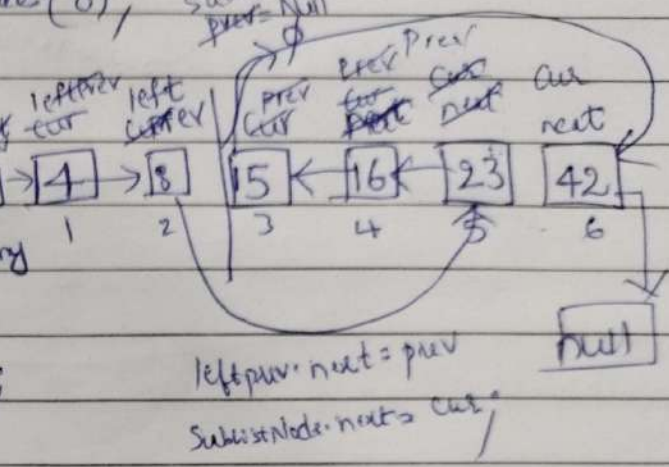
prev = head;

return head;

Date .....

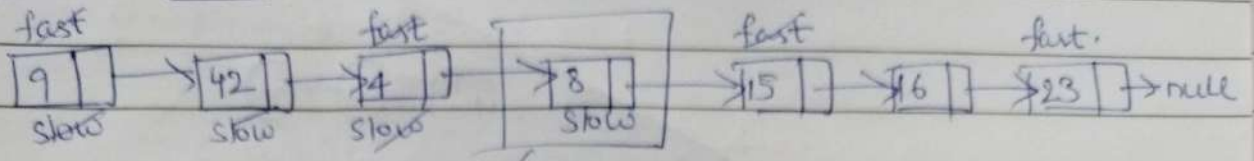
Reverse of a Linkedlist II :-

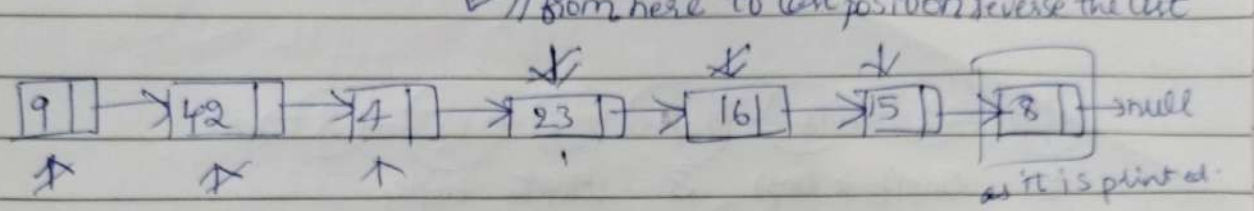
```
ListNode reverseBetween(ListNode head, int left, int right) {
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode letPre = dummy;
    ListNode curNode = head;
    for (int i=0; i<left; i++){
        letPre = letPre.next;
        curNode = curNode.next;
    }
    List SublistHead = curNode;
    ListNode prev = Null;
    for (int i=0; i<=(right-left); i++){
        ListNode NextNode = curNode.next;
        cur.next = prevNode;
        PrevNode = curNode;
        CurNode = nextNode;
    }
    //join the piecer.
    leftpre.next = preNode;
    SublistHead.next = curNode;

    return dummy.Next;
}
```



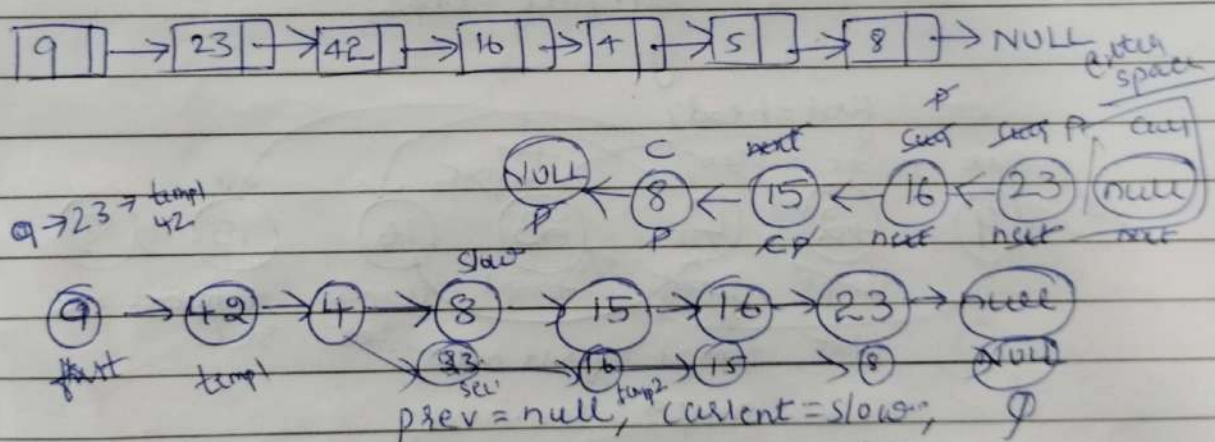
Date .....

Reorder list





→ Consider 2 pointers from here
* 1st one at head and 2nd one at from the starting position of the reversed list.
print both the values then,

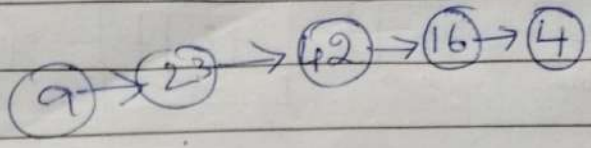


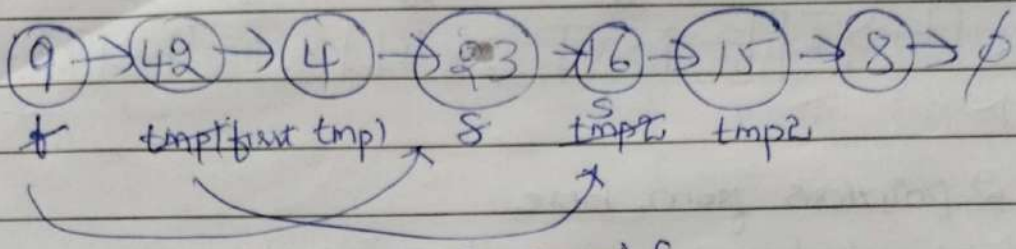
prev = null, current = slow;

```
while (curr.next != null){
    next = curr.next;
    curr.next = prev;
    prev = curr;
    curr = next;
}
```

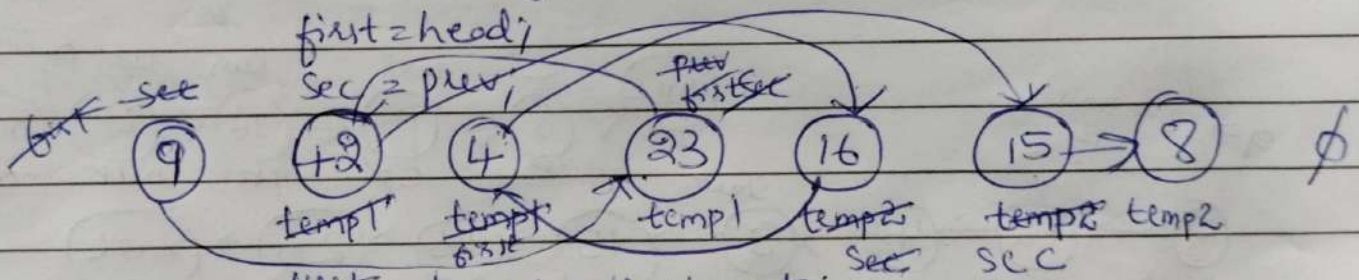
```
first temp1 = head;
sec temp2 = prev;
while (sec.next != null){
    temp1 = first.next
    temp2 = sec.next;
    first = temp1;
    sec = temp2;
}
```

Spiral






```
while(s1=null){
    temp1 = first.next;
    temp2 = sec.next;
    first.next = S;
    Sec.next = tmp1;
}
```



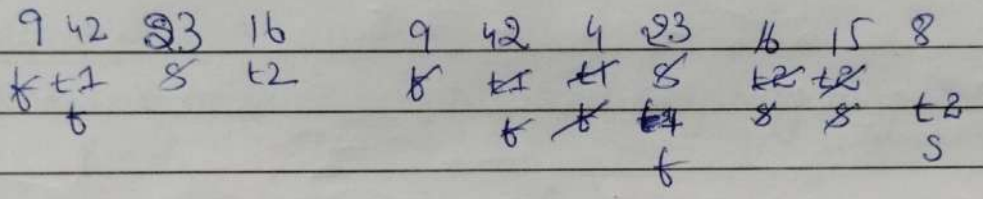
9, 23, 2, 16, 4, 15

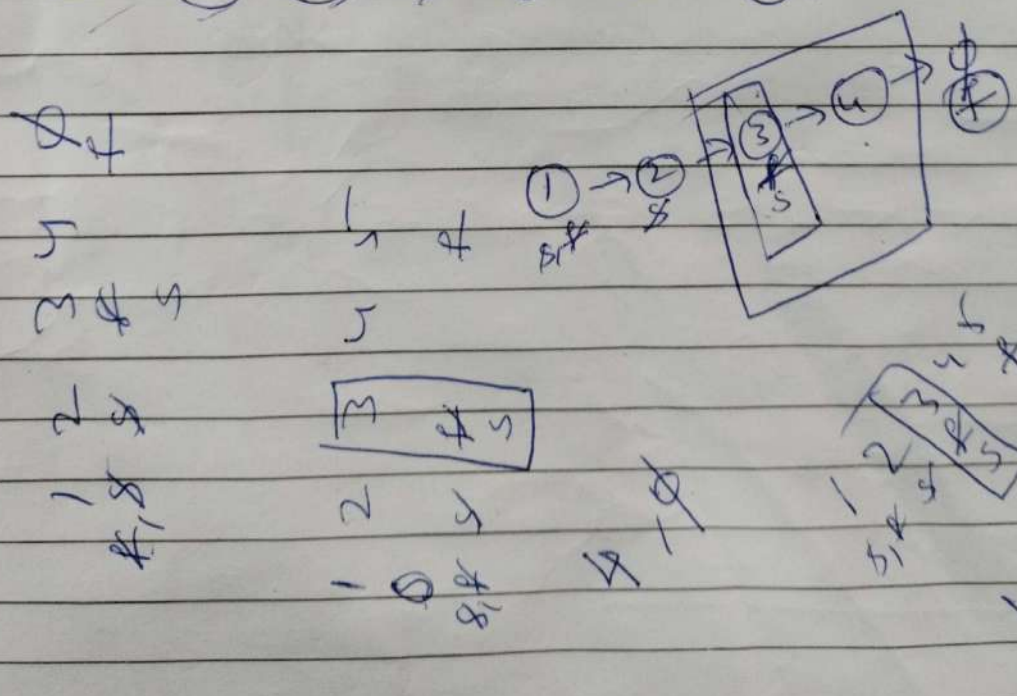
```
first.next = Sec;
sec.next = temp2;
first = temp1;
sec = temp2;
}
```

Spiral

Date .....

```
first = head;
sec = prev;
while (sec != null) {
    temp1 = first.next;
    temp2 = sec.next;
    first.next = s;
    s.next = t1;
    first = temp1;
    sec = temp2;
}
```





Date .....

CP example-1:-

* Reverse K No. of nodes in a list

I/P:- 1→2→2→4→5→6→7→8

K=4 [reverse (group) reverse]

O/P:- 4→2→2→1→8→7→6→5

I/P→2:- 1→2→3→4→5

K=3:- reverse ↓ merge ↓ reverse

O/P:- 3→2→1→5→4

Assumptions:-

* Needn't to print anything/read anything only return the head.

Time complexity :- O(N)

Space complexity :- O(1)

$1 <= N <= 10^5$ } constraints

$1 <= K <= N$ }

Spiral

Date .....

```
if (head==null){
    return;
```

ϕ 1→2→2→4→5→6→7→8

↑ ↑ ↑

prev current nxt

~~nxt~~

count = 0;

pass - 1:

ϕ 1 2 2 4 5 6 7 8

prev cur nxt

count

ϕ ← 1 2 | 3 4 5 6 7 8 ϕ

~~prev~~ ~~cur~~ nxt

~~nxt~~ prev cur

pass - 2:

count = 1;

ϕ ← 1 ← 2 3 4 5 6 7 8 ϕ

~~prev~~ ~~current~~ next

prev current

3→2→1→4→5→8→ϕ.

pass - 3:

count = 2

[ϕ ← 1 ← 2 ← 3] 4 5 6 7 8 ϕ

~~prev~~ ~~current~~ next

prev cur

At pass 4: count = 3 then we stop reversing the list.

Spiral

Date .....

1st group:- 3 → 2 → 1.

Critical check:-
head points to 4.

if (next != null) {

Code Snippet:-

~~if (head == null){~~

```
static Node reverse (Node head, int k){
    if(head == null){
        return null;
    }
    Node cur = head; count = 0;
    Node prev = null;
    while(count < k && cur.~~next~~ != null){
        Node next = cur.next;
        cur.next = prev;
        prev = cur;
        cur = next;
        count++;
    }
    if (next != null){
        head.next = reverse Node(next, k);
                                 head,
    }
    return prev; // after reversing it is the head of the
}                    node exactly.
```

Date .....

$\phi \leftarrow 1 \leftarrow 2 \leftarrow 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad \phi$

prev curr

prev=null, 4, next=null

Iteration 2:

Pass-1:

$\phi \leftarrow 4 \quad 5 \quad 6 \quad 7 \quad 8$

prev curr next

~~curr~~ prev curr

Pass-2:

$\phi \leftarrow 4 \leftarrow 5 \quad 6 \quad 7 \quad 8$

prev curr next

~~prev curr~~ curr

head

3→2→1→6→5→4

Pass-3:

$\phi \leftarrow 4 \leftarrow 5 \leftarrow 6 \;||\; 7 \quad 8$

prev curr. prev | next curr

// here Second group is reversed i.e., 4→5

6→5→4

prev

Iteration-3: 7!=null

$\phi \leftarrow 7 \quad 8 \quad \phi$

prev ~~curr~~ next

~~next~~ prev curr

3→2→1→6→5→4→8→7

Pass-2:

$0 \leftarrow 7 \leftarrow 8 \quad \phi$

prev ~~curr~~ ~~next~~

prev curr

8→7

Spiral

Date .....

① if (head == null || k == 1) {
    return head;
}

→ head is empty, nothing to reverse here so reverse ∅ (null)
→ also k = 1 (reversing 1 node at a time, no need to run a universal logic).

→ // for better efficiency & correctness.

↓

It's important because :-
→ prevents nullPointerException, unnecessary computation
saves time when k=1, need not to traversal.
// If k no of nodes do not exist then return head; otherwise reverse it and unleash the prev.

② ListNode check = head;
int count = 0;
while (check != null) {
    check = check.next;
    count++;
}
if (count < k) return head;

Date .....

Final code :-

```
if (head == null || k == 1){
    return head;
    }
//check K nodes exist(s) not
ListNode check = head; int count = 0;
  while (check != null && count < k){
      check = check.next;
       count++;
      }
   if (count < k) return head;
//reverse K no. of nodes.
ListNode prev = null;
 ListNode next = null; current = head;
 ListNode while (current != null){
      next = current.next;
       current.next = prev;
        prev = current;
        current = next;
        }
      if (next != null){
   head.next = reverseKGroup(next, k);
      }
   return prev;
}
```

Date .....

Rotated Linked List:-

head

1 → 2 → 3 → 4 → 5 → 6 → ∅

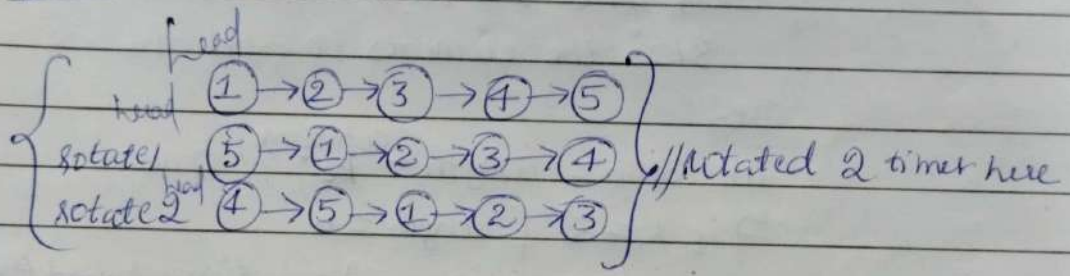
→ Rotate 1 time:-

2 → 3 → 4 → 5 → 6 → 1

→ Rotate 2nd time

3 → 4 → 5 → 6 → 1 → 2 → ∅

Ex 1:-



2 % 6

6) 2 (0

0

2

12 % 6

12) 6 (0

0

6

6 % 6

6) 6 (1

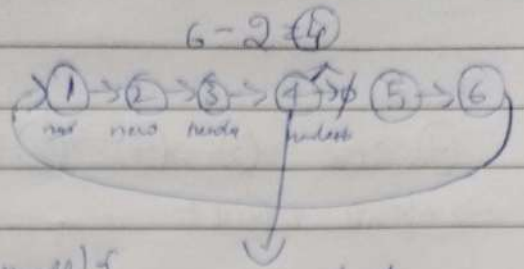
6

0

Spiral

Date .....

```
if (k<=0 || head==null || head.next==null)
    { return head;
    }
ListNode last = head;
int length = 1;
while (last.next != null) {
        last = last.next;
        length++;
    }
last.next = head;
//now need to make no of rotations
int rotations = k % length;
// then find the new node here.
    Skip the elements to find the new node
i.e.,
int skip = length - rotations; // times
⇒ if k=2 & length = 6
        then skip 6-2 = 4 times to find the
        newhead of the node.
ListNode newLast = head;
//now land at the 4th element Node here.
for (int i=0; i<skip-1; i++) {
    newLast = newLast.next;
    }
}
head = newLast.next;
newLast.next = null;
return head;
}
```



0, 1, 2, 3

Spiral