NO.

Title

Linked Lists

Enjoy the most efficient handwriting experience! Taking notes on the go, whether for inspiration, ideas, knowledge learning, business insights, or even sketches...

NOTE IN

Creative notes

By reading we enrich the mind;
by writing we polish it.

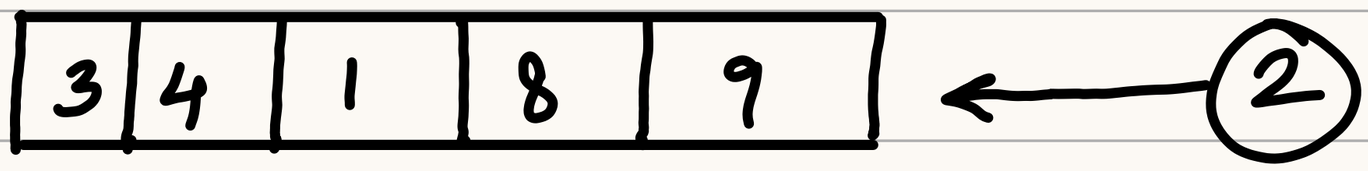
Linked Lists :

Limitation of Arrays \ Array List

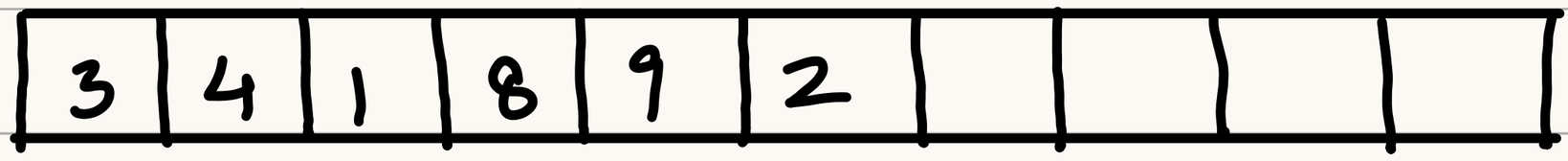
- Array is not a continous memory allocation
That means it cannot exceed its memory allocation

```
int [] arr = new int [3],
arr [0] = 1,   ✓
arr [1] = 2;   ✓
arr [2] = 3,   ✓
arr [3] = 4,   X  ⟶ Index Out of Bound
```

- The other way to add memory to array is ArrayLists



in arraylist the memory gets doubled that means

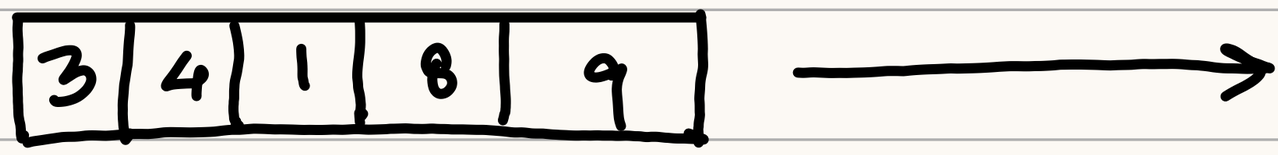


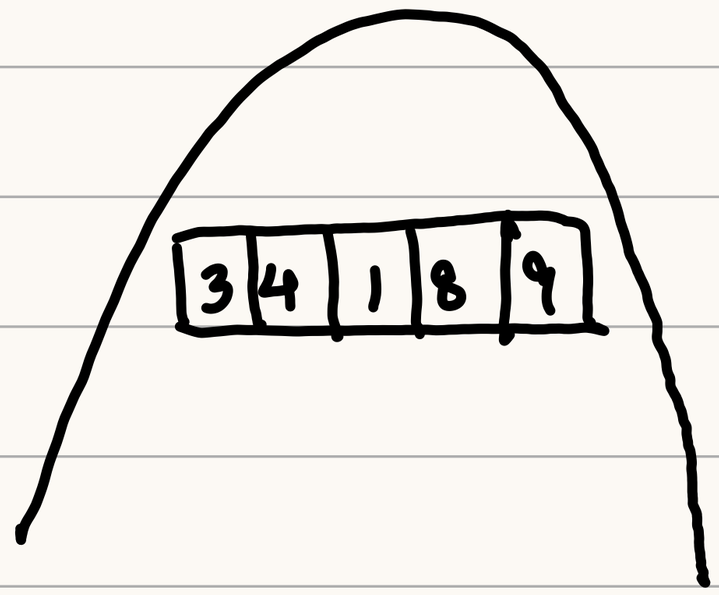
Even though memory is getting added but it is not effiient as there are extra memory allocations of the arrayList

To solve this problem we use LinkedLists.

What does LinkedList offer?

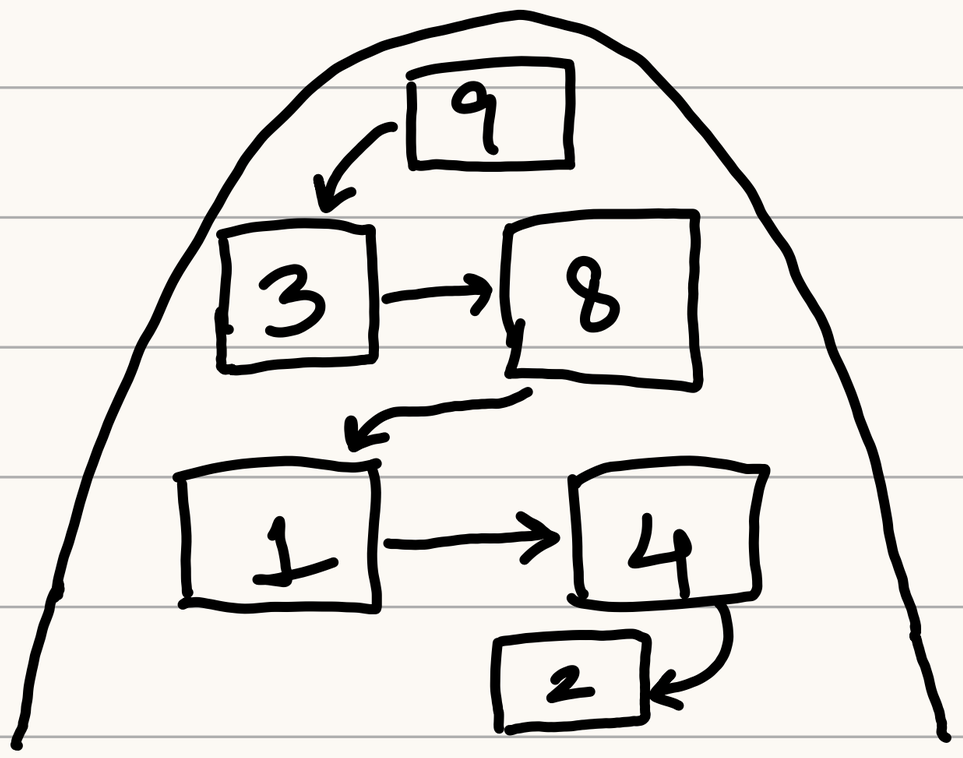
- LinkedList doesnot offer continous memory allocation (not fixed)
- Instead it tries to break these array into seperate boxes.





Heap

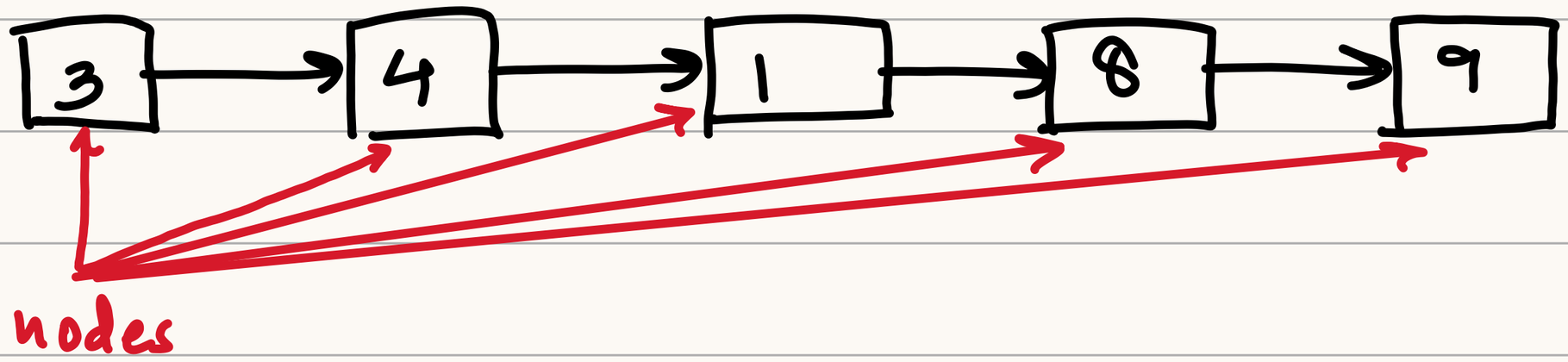
Internal Implementation of Array



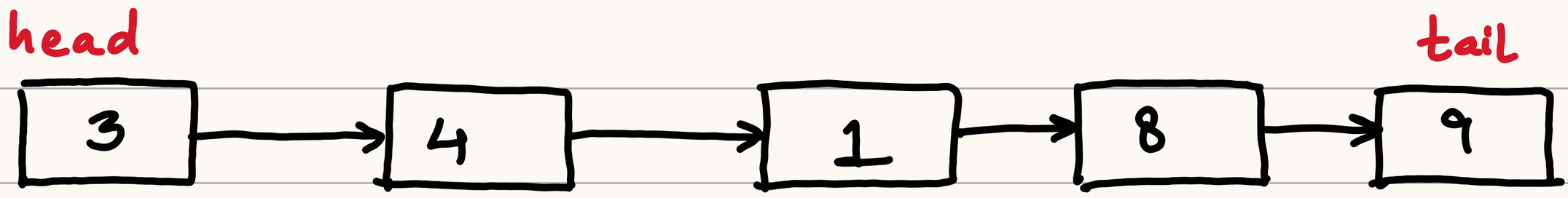
Not having continous memory for every block we put it in some random memory

Internal Implementation of Linked List

Since LinkedList are not continous they donot have indexs like arrays



Singly Linked List.

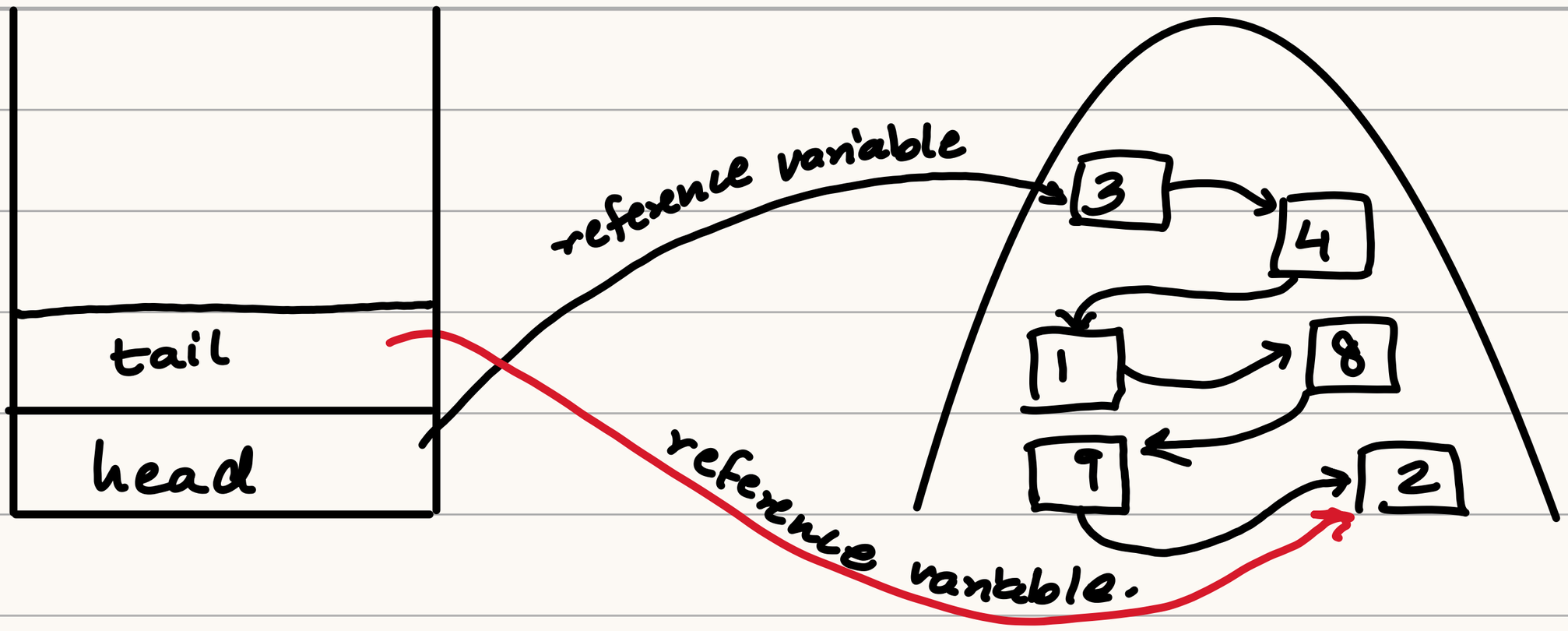


- Every single item Knows about the next item

- Every single linked List have two refesence variaboles namely Head & tail

- Head It is a reference variable that points to the very first node.

- Tail. It is a refexence variable that points to the very last node



- A node itself is a type class Node

- class node consists of some properties that applies to all nodes

```
class Node {
    int val,
}
```

- It will also point to the next node
- Means a node will point to a node of type node.

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

What is a problem over here?

You cannot directly access the index

list get [3] ✗

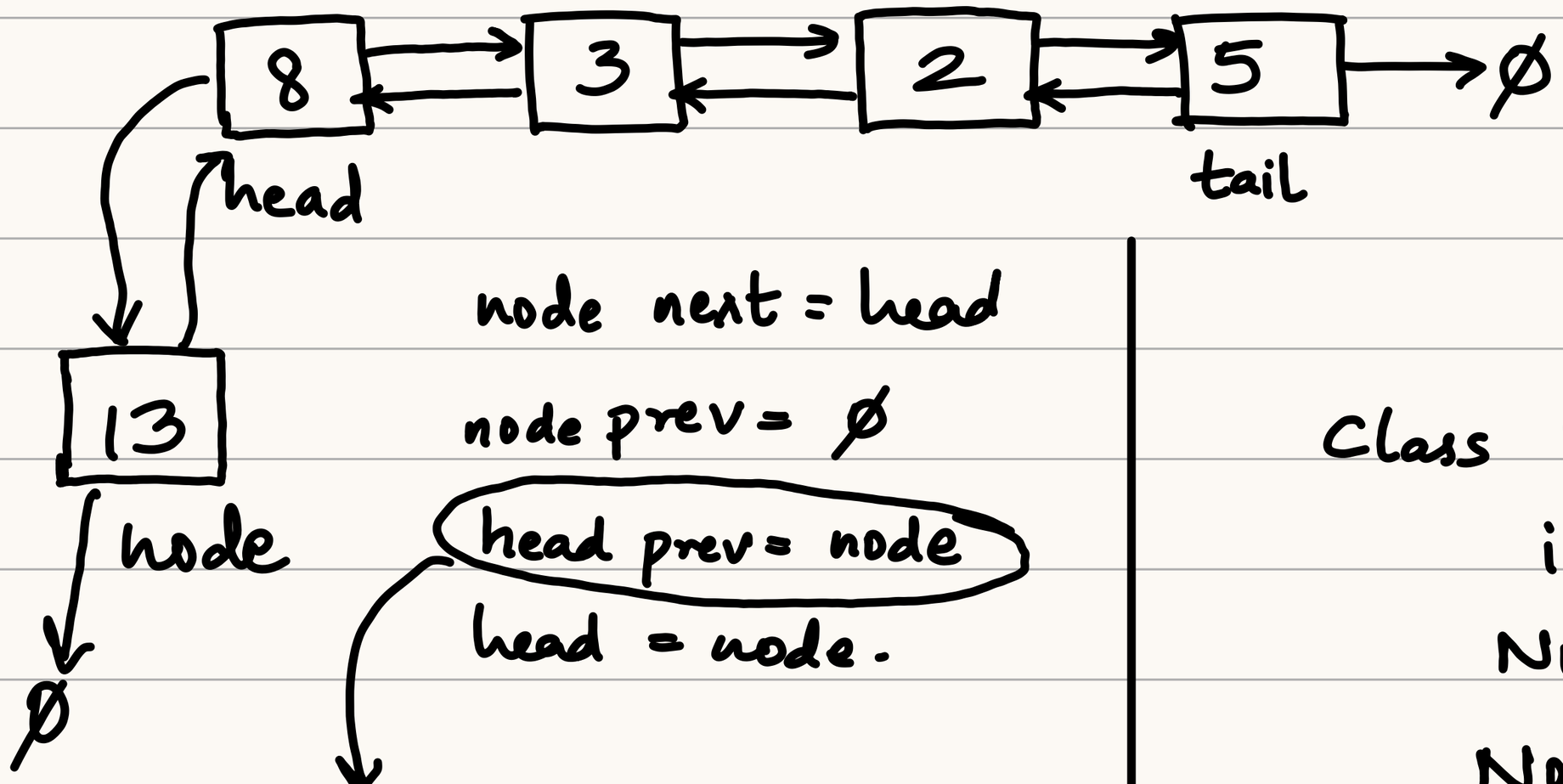
List get [4] ✗

because every node has a idea of only the next node Only Last element Knows when the list ends No other element Knows when List will end.

NO node has any idea about how many elements does the linkedList have.

Hence we need a pointer to check it

Doubly Linked List:



node next = head

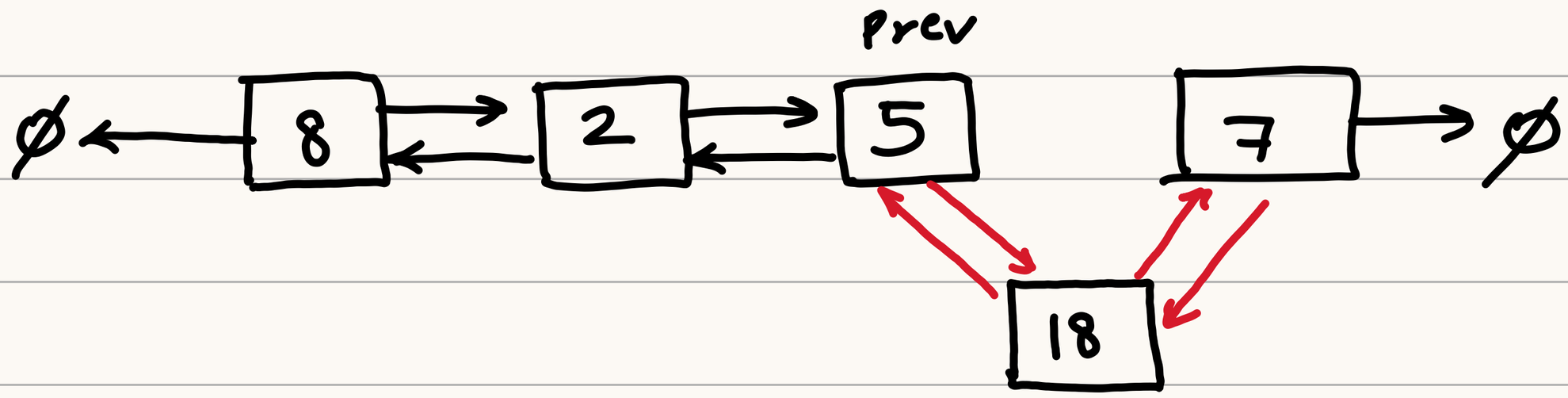
node prev = ∅

head prev = node

head = node.

check for null pointer

```
Class Node {
    int val;
    Node next;
    Node prev;
}
```



node next = prev next

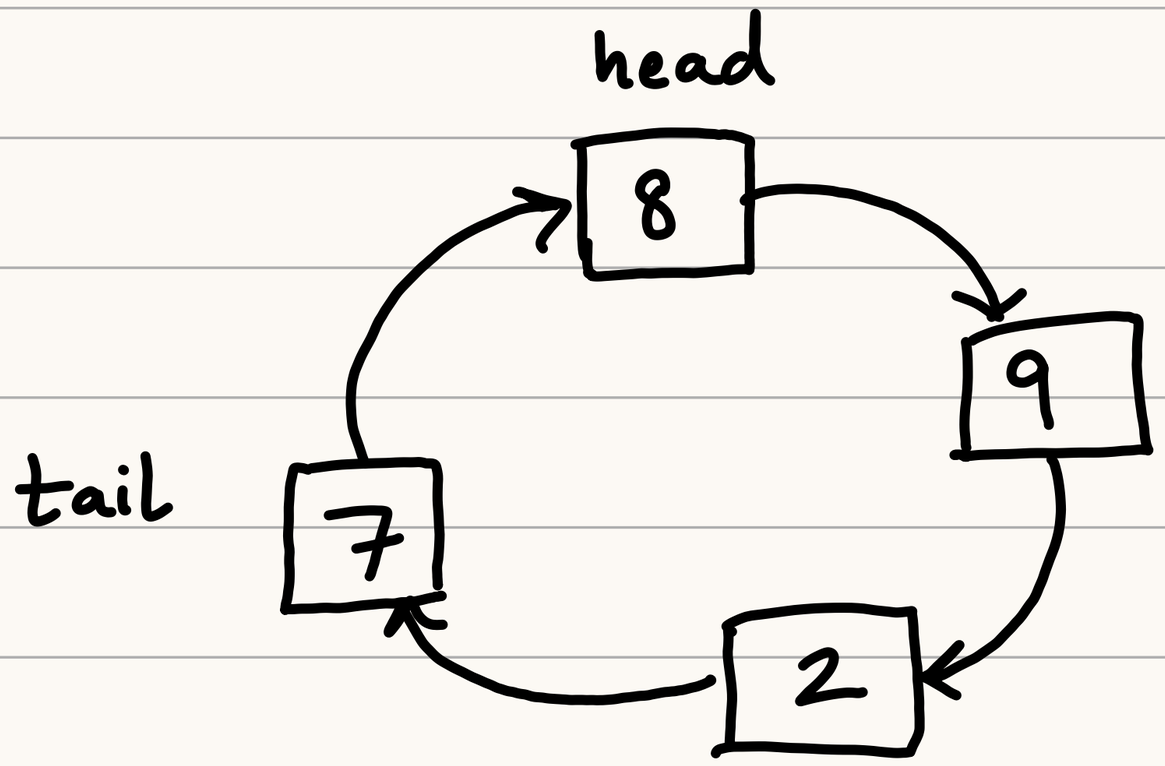
prev next = node

node prev = prev

node next prev = node

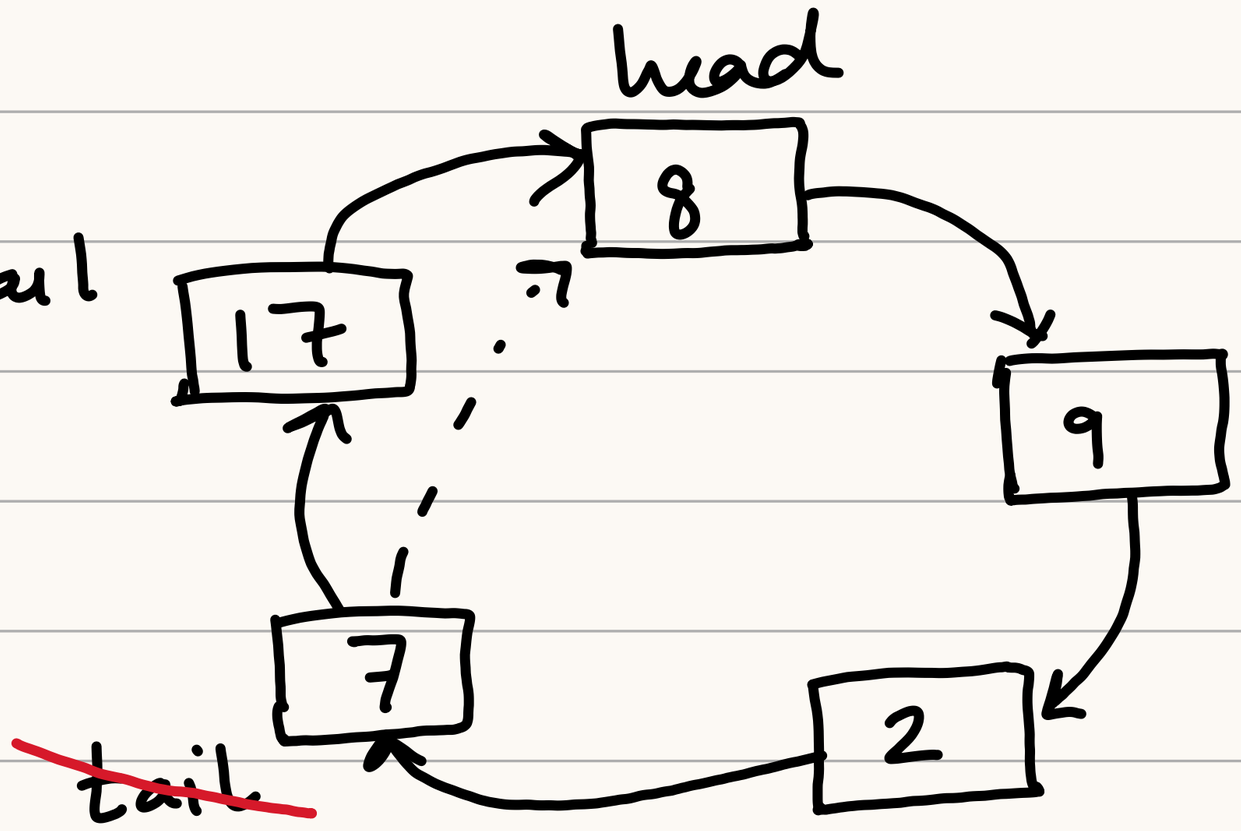
may be null

Circular LinKed List:



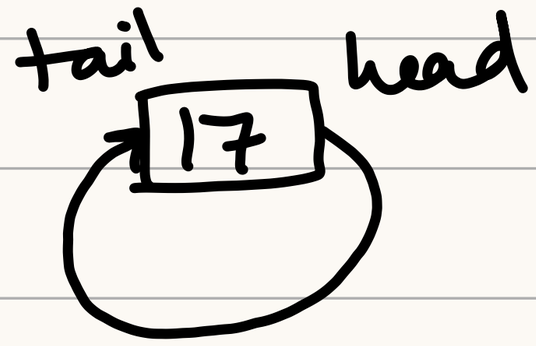

```
class Node {
        int val,
      Node next;
}
```

Insertion



```
tail next = node,
node next = head,
tail = node,
```

```
if (head == null)
        head = node,
        tail = node;
```



★ Code for Singly Linked List (Linked List)

★ Linked List.java

```
public class LL {

    private Node head, // Assigning head        head ⟶
    private Node tail, // Assigning tail        ⟶ tail
    private int size, // Assigning size         size
    public LL() {
        this.size = 0;  // size is initially zero
    }

private class Node { // Making new Nodes using class.
    private int value, // Value of the node
    private Node next, // Assigning next value
    public Node (int value) {
        this.value = value,
    }

    public Node (int value, Node next) {
        this.value = value,
        this.next = next,
    }
}
```

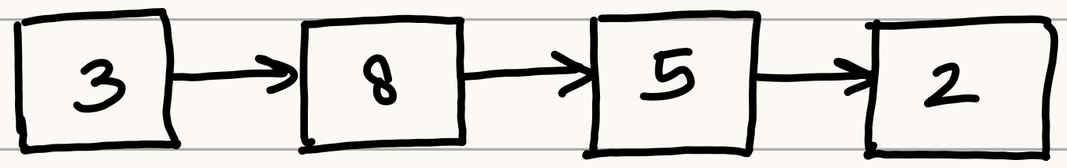
* Main java

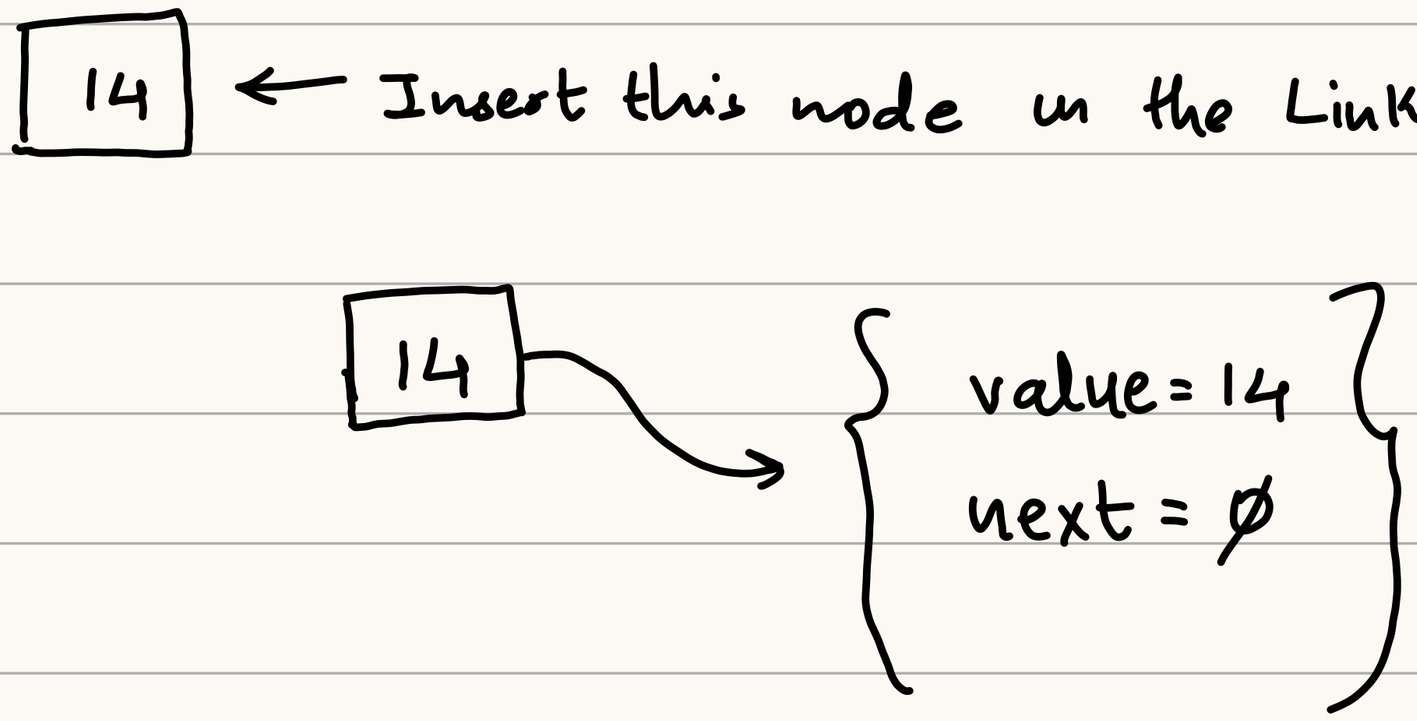
```
public class Main {
    public static void main (String[] args) {
        LL list = new LL();
    }
}
```

→ Insertion in Linked List at the start

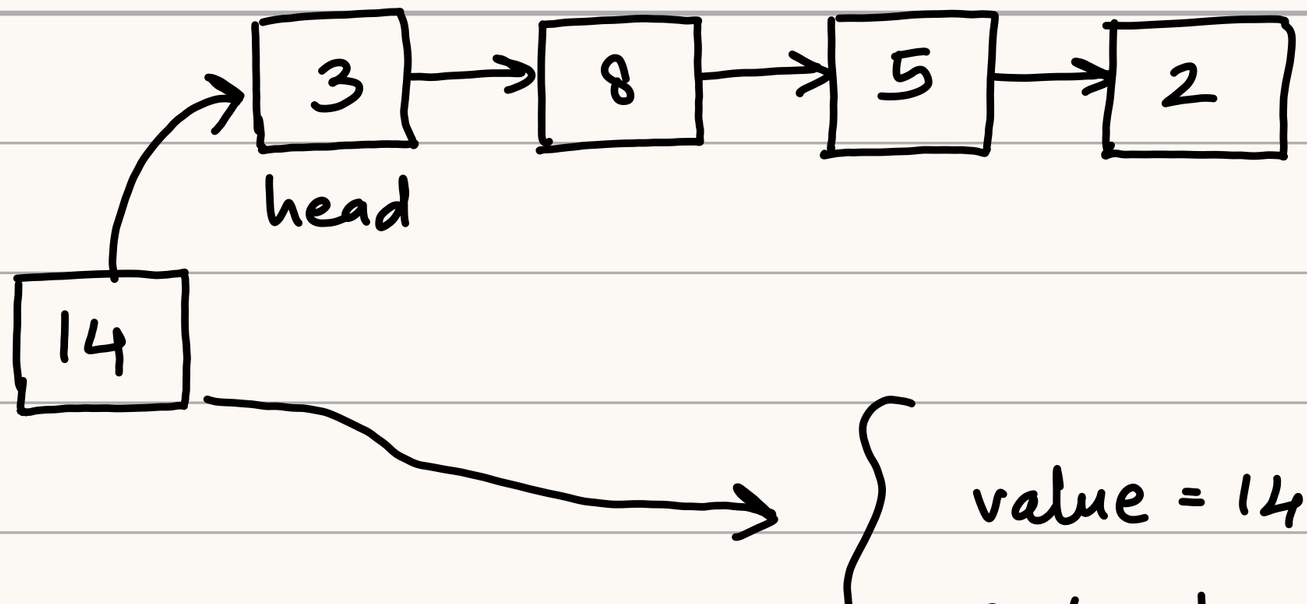
You already have a linked List like this.

①





(2)

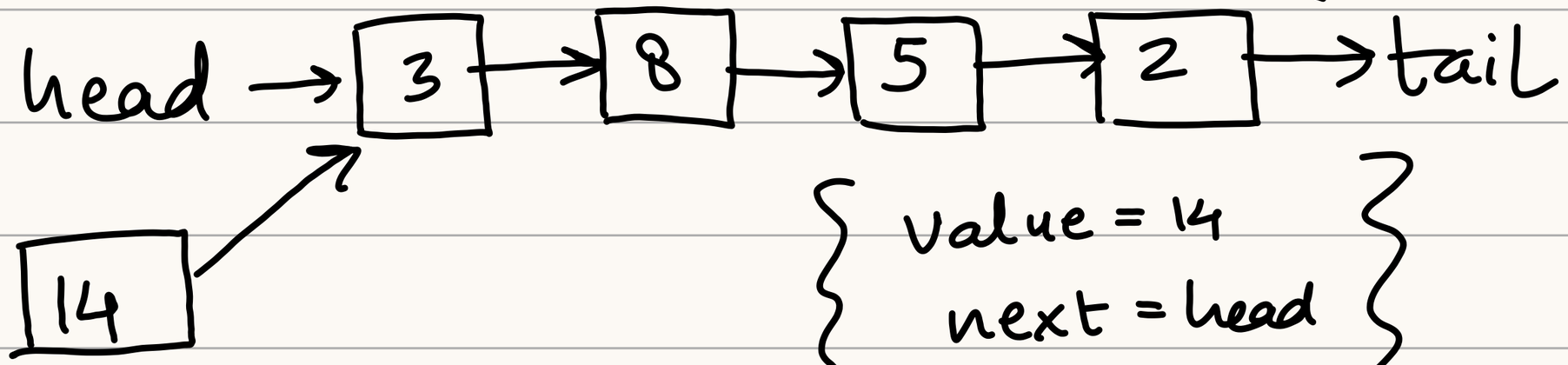


value = 14

next = head

node next =

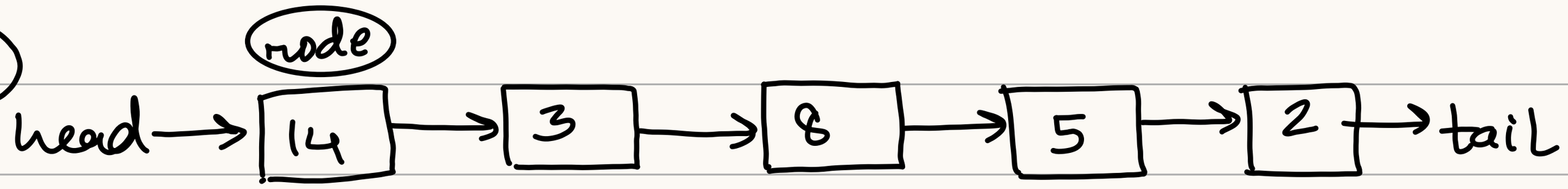
(3)



value = 14

next = head

(4)



node next = head

head = node.

(5) For the first Item added

node

18 → head and tail will be same

head, tail

If (tail == null)

tail = head

size ++,

★ Insertion of node in Linked List: (at first)

LL.java

```
public class LinkedList{
    private Node head,
    private Node tail,
    private int size,
    public LL() {
      this.size = 0;
} // Insertion at first starts here
public void insertFirst(int val) {
    Node node = new Node(val),
    node.next = head,
    head = node,
    if(tail == null) {
        tail = head,
    }
    size += 1,
} // Insertion at first ends here
private class Node {
    private int value,
    private Node next,
    public Node(int value) {
        this.value = value,
    }
```

```
public Node (int value, Node next) {
    this value = value,
    this next = next,
}
```

* Displaying Linked List

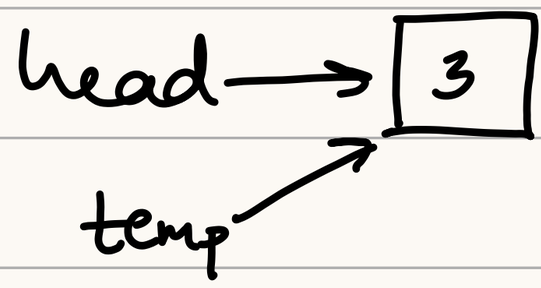

```
while (head != ∅)
    print (head val)
    head = head next
```

3, 5, 9, 8

Wrong

* This is because you are literally changing the position of head Hence when you try to print it second time the structure linked list is changed and head → ∅ poiunt towards null

To solve this issue we will create a new temporary node lik below



* Code for displaying Linked List.

LL.java

```
public class LinkedList{
    private Node head,
    private Node tail,
    private int size,
    public LL() {
        this.size = 0;
    } // Insertion at first starts here
public void insertFirst(int val) {
        Node node = new Node(val),
        node.next = head,
        head = node,
        if(tail == null) {
            tail = head,
        }
        size += 1,
    } // Insertion at first ends here
public void display() { // Displaying LL starts here
    Node temp = head,
    while (temp != null) {
        System.out.print (temp.value + "→");
        temp = temp.next,
    }
    System.out.println("END"),
} // Displaying LL ends here
```

```
private class  Node {
    private int value,
    private Node next,
    public Node (int value) {
        this value = value,
    }
    public Node (int value, Node next) {
        this value = value,
        this next = next,
    }
}
```

Main java

```
public class Main {
    public static void main (String [] args) {
        LL list = new LL(),
        list insertFirst ( 3);
        list insertFirst ( 2),
        list insertFirst (8),
        list insert Fist (17),

        list display ();
    }
}
```

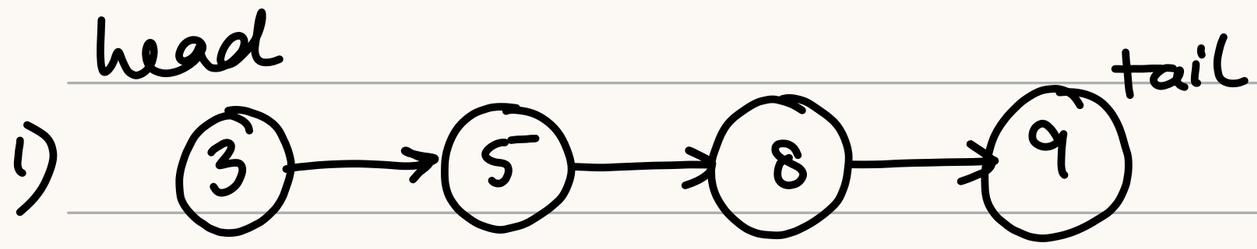
* Output:

17 → 8 → 2 → 3 → END

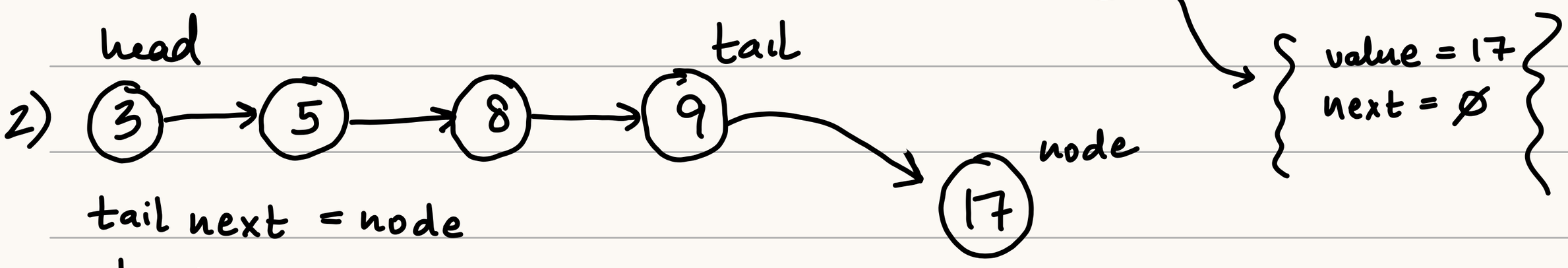
Time complexity O(n) → n is number of nodes

* Insertion of node in Linked List (at Last)

1)



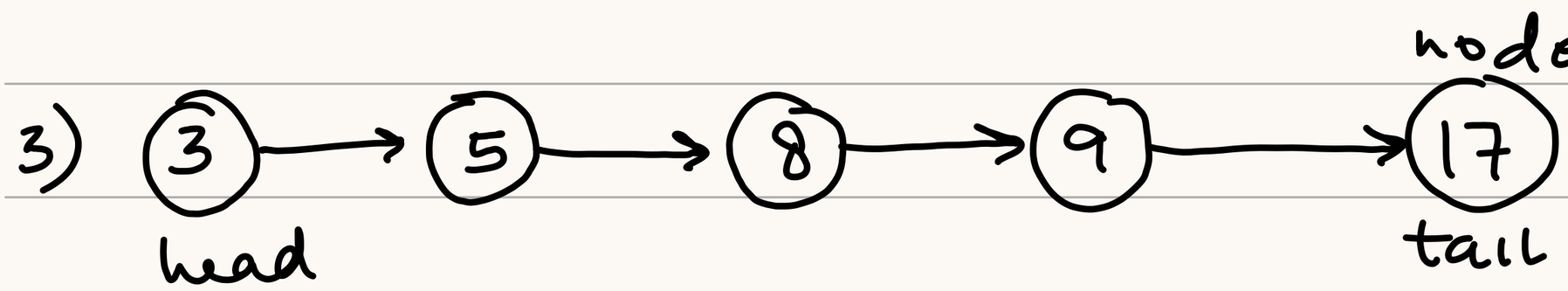
2)



tail next = node

tail = node

3)




```
if (tail = null) { insertfirst (val) },
  tail next = node,
  tail = node,
  size ++,
```

★ Code for Insertion of node (at Last)

LL.java.

```
public class LinkedList {
    private Node head,
    private Node tail,
    private int size,
    public LL() {
      this.size = 0;
}
```

// Insertion at first starts here

```
public void insertFirst(int val) {
      Node node = new Node(val),
      node.next = head,
      head = node,
      if(tail == null) {
          tail = head,
    }
      size += 1,
} // Insertion at first ends here
```

// Insertion at last starts here

```
public void insertLast(int val) {
  if (tail == null) {
     insertFirst(val),
     return,
  }
  Node node = new Node(val),
  tail next = node,
   tail = node;
   size++,
} // Insertion at last end here
```

```
public void display() { // Displaying LL Starts here
    Node temp = head,
    while (temp != null) {
        System out print (temp value + "→");
        temp = temp next,
    }
     System.out println("END"),
} // Displaying LL ends here
```

Main java

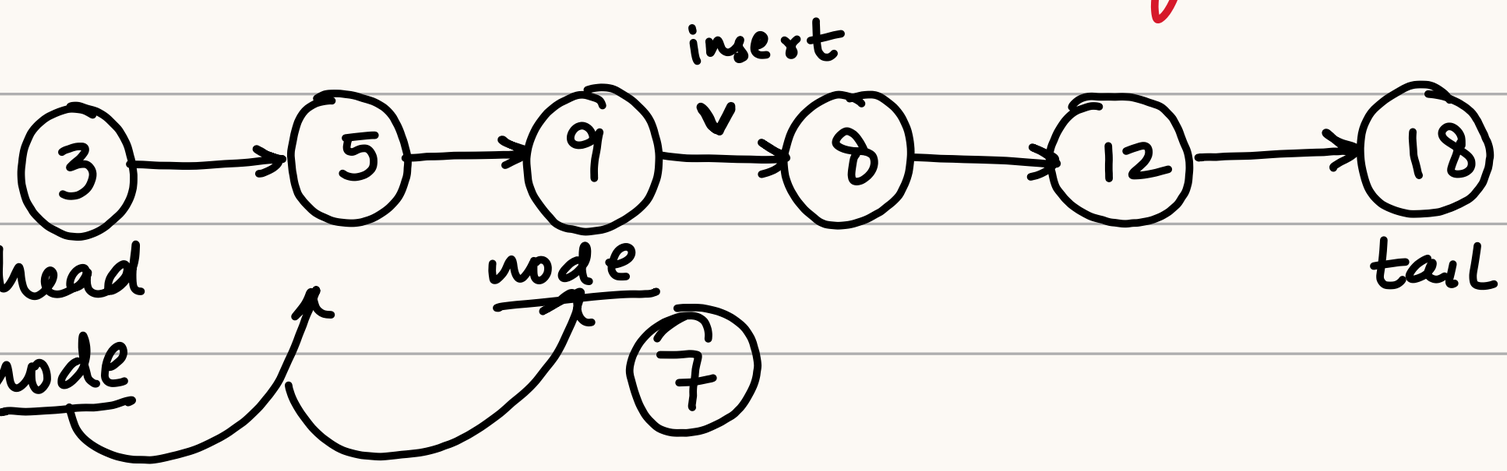
```
public class Main {
    public static void main (String [] args) {
        LL list = new LL(),
        list insertFirst (3);
        list insertFirst (2),
        list insertFirst (8),
        list insertFirst (17),
        list insertLast (99),
        list display ();
    }
}
```

* **Output:**

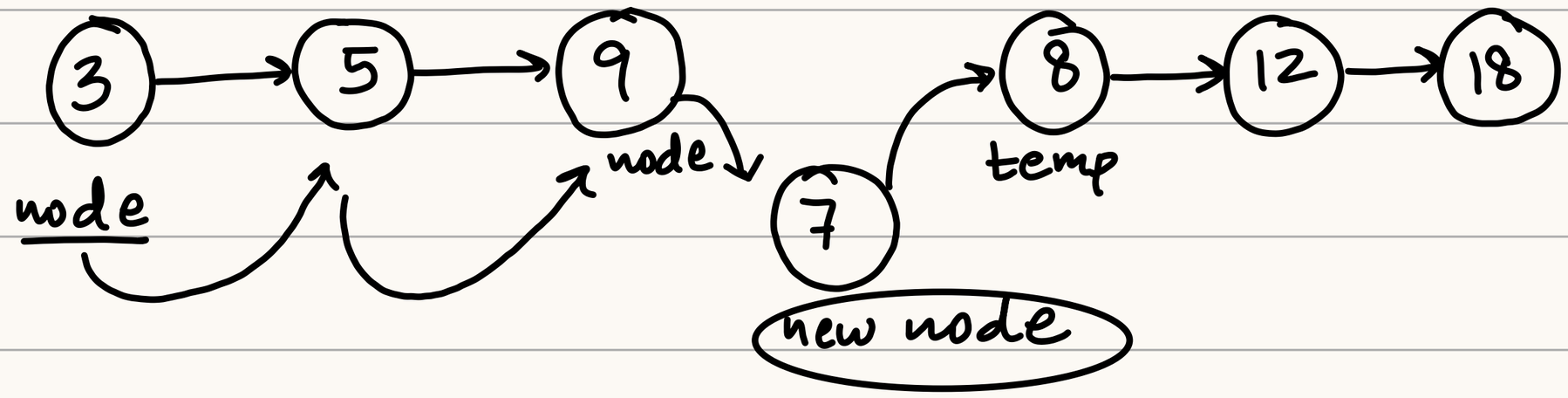
17 → 8 → 2 → 3 → 99 → END

* Insertion in a middle Index of Linked list

1)



2)



* Code for Insertion of node (at Last)

LL.java.

```
public class LinkedList{
    private Node head,
    private Node tail,
    private int size,
    public LL(){
        this.size = 0;
    }
```

// Insertion at first starts here

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

    size += 1;
} // Insertion at first ends here
```

// Insertion at last starts here

```
public void insertLast(int val) {
    if (tail == null) {
        insertFirst(val);
        return;
    }
    Node node = new Node(val);
    tail.next = node;
    tail = node;
    size++;
} // Insertion at last end here
```

// Insertion in the middle of index· starts here

```
public void insert (int val, iut index) {
        if (index == 0) {
            insertFirst(val),
            return;
        }
        if (index == size) {
            insert Last (val),
            return,
        }
    Node temp = head,
        for (int i =1; i< index ; i++) {
            temp = temp next,
        }
        Node node = new Node (val, temp next),
        temp next = node,
        size++,
}
```

// Insertion in the middle of index· ends here

```
public void display() { // Displaying LL Starts here
    Node temp = head,
    while (temp != null) {
        System out print (temp value + "→");
        temp = temp next,
    }
    System.out println("END"),
} // Displaying LL ends here
 private class Node {
     private int value,
     private Node next,
     public Node (int value) {
         this value = value,
     }
 public Node (int value, Node next) {
     this value = value,
     this next = next,
}
}
```

Main java

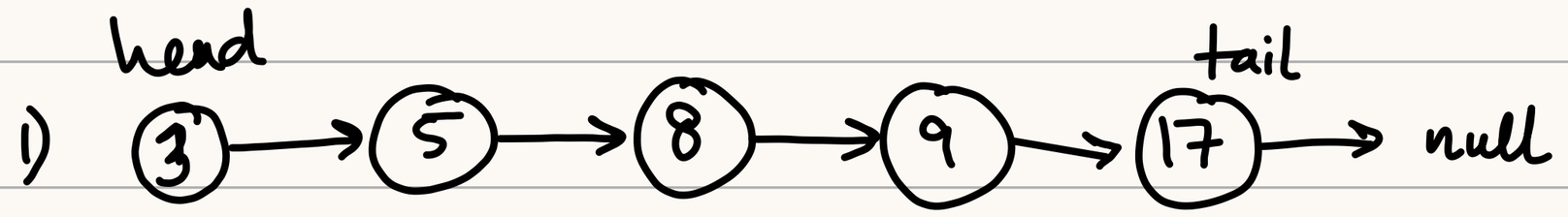
```
public class Main {
    public static void main (String [] args) {
     LL list = new LL(),
     list insertFirst (3);
     list insertFirst (2),
     list insertFirst (8),
    list insertFirst (17),
   list insertLast (99),
   list insert (100, 3),
        list display ();
    }
  }
```

* Output:

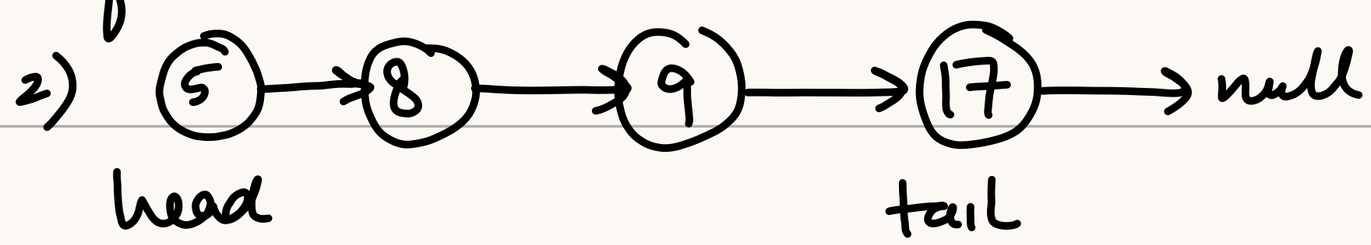
17 → 8 → 2 → 100 → 3 → 99 → END

* Deletion of Node in Linked List

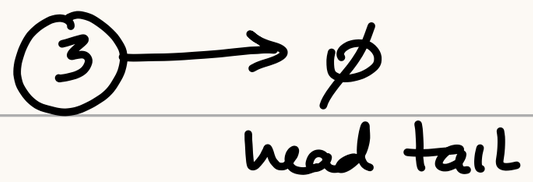
→ Delete first



after delete first.



head = head.next



LL.java.

```
public class LinkedList {
  private Node head,
  private Node tail,
  private int size,
  public LL() {
    this size = 0;
}
```

// Insertion at first starts here

```
public void insertFirst(int val) {
        Node node = new Node(val);
        node.next = head;
        head = node;
        if(tail == null) {
            tail = head;
        }
        size += 1;
} // Insertion at first ends here
```

// Insertion at last starts here

```
public void insertLast(int val) {
    if (tail == null) {
        insertFirst(val);
        return;
    }
    Node node = new Node(val);
    tail.next = node;
    tail = node;
    size++;
} // Insertion at last end here
```

// Insertion in the middle of index· starts here

```
public void insert (int val, int index) {
        if (index == 0) {
            insertFirst(val),
            return;
        }
        if (index == size) {
            insertLast(val),
            return,
        }
    Node temp = head,
        for (int i =1; i< index ; i++) {
            temp = temp next,
        }
        Node node = new Node (val, temp next),
        temp next = node,
        size++,
}
```

// Insertion in the middle of index· ends here

// Delete First starts here:

```
public int deleteFirst() {
    int val = head.value;
    head = head.next;
    if (head == null) {
        tail = null;
    }
    size--;
    return val;
}
```

```
public void display() { // Displaying LL starts here
    Node temp = head;
    while (temp != null) {
        System.out.print (temp.value + "→");
        temp = temp.next;
    }
    System.out.println("END");
}
// Displaying LL ends here
```

```
private class  Node {
    private int value,
    private Node next,
    public Node  (int value) {
        this value = value,
    }
public Node (int value, Node next) {
    this value = value,
    this next = next,
}
}
```

Main java

```
public class Main {
   public static void main (String [] args) {
       LL list = new LL(),
       list insertFirst (3);
      list insertFirst (2),
       list insertFirst (8),
       list insertFirst (17),
      list insertLast (99),
       list insert (100, 3),
      System.out println (list deleteFirst()),
    list display ();
   }
}
```

→ Delete Last

Lets say we want to delete the last item

can we do tail = tail-1, ? no we can't

Every node has one particular reference point that is next

Do we have previous ? NO

Then what can we do?

- We can reach at the second last item and just make it tail.

and make the original tail as null

- The real Question is how do we get to size-2 item

LL.java.

```
public class LinkedList{
    private Node head,
    private Node tail,
    private int size,
    public LL(){
        this size = 0;
    }
```

```
// Insertion at first starts here
public void insertFirst(int val) {
        Node node = new Node(val);
        node next = head;
        head = node;
        if(tail == null) {
            tail = head;
        }
        size += 1;
} // Insertion at first ends here
```

```
// Insertion at last starts here
public void insertLast(int val) {
    if (tail == null) {
        insertFirst(val);
        return;
    }
    Node node = new Node(val);
    tail next = node;
    tail = node;
    size++;
} // Insertion at last end here
```

```
// Insertion in the middle of index· starts here
public void insert (int val, int index) {
        if (index == 0) {
            insertFirst(val),
            return;
        }
        if (index == size) {
            insertLast(val),
            return,
        }
        Node temp = head,
        for (int i =1; i< index ; i++) {
            temp = temp next,
        }
        Node node = new Node (val, temp next),
        temp next = node,
        size++,
}
// Insertion in the middle of index· ends here
```

// Delete First starts here:

```
public int deleteFirst() {
    int val = head.value;
    head = head.next;
    if (head == null) {
        tail = null;
    }
    size--;
    return val;
} // Delete first ends here
```

// Delete last starts here

```
public int deleteLast() {
        if (size <= 1) {
            return deleteFirst();
        }
        Node secondLast = get(size-2);
        int value = tail.value;
        tail = secondLast;
        tail.next = null;
        return val;
    }
// Delete last ends here
```

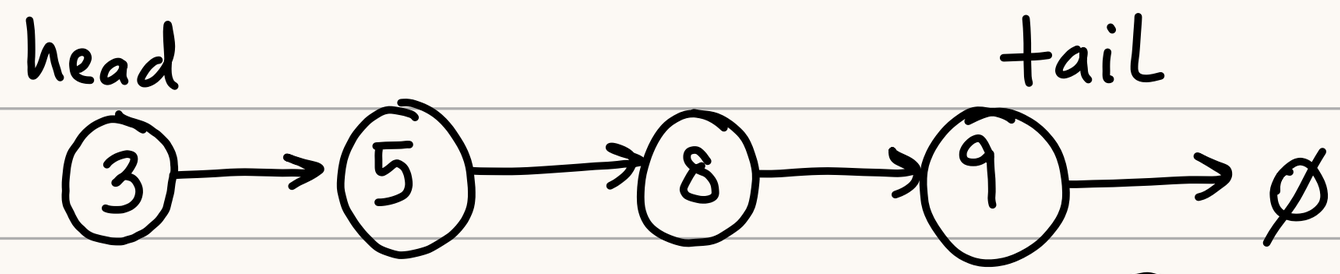
```
public Node get (int index) {
    Node node = head,
    for (int i=0, i< index ; i++) {
        node = node next,
    }
    return node.
}
```

```
public void display ( ) { // Displaying LL Starts here
    Node temp = head,
    while (temp != null) {
        System out print (temp value + "→");
        temp = temp next,
    }
    System.out println("END"),
}
// Displaying LL ends here
```

Main java

```
public class Main {
    public static void main (String [] args) {
        LL list = new LL(),
        list insertFirst (3);
        list insertFirst (2),
        list insertFirst (8),
        list insertFint (17),
        list insert Last (99),
        list insert (100, 3),
        System.out println (list deleteLast()),
        list display ();
    }
}
```

→ Delete Middle



Lets assume we need to remove (8)

— we need to reach a node behind (8) ie (5)

LL.java:

```
public class LinkedList {
    private Node head,
    private Node tail,
    private int size,
    public LL() {
      this.size = 0;
  }
// Insertion at first starts here
public void insertFirst(int val) {
        Node node = new Node(val),
        node.next = head,
        head = node,
        if(tail == null) {
            tail = head,
      }
    size += 1,
} // Insertion at first ends here
```

// Insertion at last starts here

```
public void insertLast(int val) {
  if (tail == null) {
     insertFirst(val),
     return,
  }
  Node node = new Node(val),
  tail next = node,
  tail = node;
  size++,
} // Insertion at last end here
```

// Insertion in the middle of index starts here

```
public void insert(int val, int index) {
        if (index == 0) {
            insertFirst(val),
            return;
        }
        if (index == size) {
            insertLast(val),
            return,
        }
```

```
Node temp = head,
for (int i =1; i< index ; i++) {
    temp = temp next,
}
    Node node = new Node (val, temp next),
    temp next = node,
    size++,
}
```

// Insertion in the middle of index ends here

// Delete First starts here:

```
public int deleteFirst () {
    int val = head value,
    head = head next,
    if (head == null) {
        tail = null,
    }
    size --,
    return val,
} // Delete first ends here
```

// Delete last starts here

```
public int deletelast() {
        if (size <= 1) {
            return deleteFirst(),
        }
        Node secondLast = get (size-2),
        int value = tail value;
        tail = secondLast,
       tail next = null,
      return val;
}
```

// Delete last ends here

// Delete Mid starts here

```
public int delete (int index ) {
     if (index == 0) {
        return deletefirst();
    }
    if (index == size-1) {
        return deleteLast(),
    }
  Node prev= get (index-1);
  int val = prev next value;
 prev next = prev next next,
```

```
    return val;
}

//Delete mid ends here

public Node get (int index) {
    Node node = head,
    for (int i=0, i< index ; i++) {
        node = node next,
    }
    return node.
}

public void display ( ) { // Displaying LL Starts here
    Node temp = head,
    while (temp != null) {
        System out print (temp value + "→");
        temp = temp next,
    }

    System.out println("END"),

}

// Displaying LL ends here
```

Main.java

```
public class Main {
    public static void main (String [] args) {
        LL list = new LL(),
         list.insertFirst (3);
         list.insertFirst (2),
         list.insertFirst (8),
        list.insertFirst (17),
       list.insertLast (99),
       list.insert (100, 3),
         System.out.println (list.deleteLast()),
        list.display ();
        System.out.println( list.delete (2)),
        list.display ();
    }
}
```

- Find a value in linked list:

LL.java:

```
public class LinkedList {
    private Node head,
    private Node tail,
    private int size,
    public LL() {
      this.size = 0;
  }

// Insertion at first starts here
public void insertFirst(int val) {
      Node node = new Node(val),
      node.next = head,
      head = node,
      if (tail == null) {
          tail = head,
      }

    size += 1,
} // Insertion at first ends here
```

// Insertion at last starts here

```
public void insertLast(int val) {
  if (tail == null) {
     insertFirst(val),
     return,
  }
  Node node = new Node(val),
  tail next = node,
  tail = node;
  size++,
} // Insertion at last end here
```

// Insertion in the middle of index starts here

```
public void insert(int val, int index) {
        if (index == 0) {
            insertFirst(val),
            return;
        }
        if (index == size) {
            insertLast(val),
            return,
        }
```

```
        Node temp = head,
        for (int i =1; i< index ; i++) {
            temp = temp next,
    }
            Node node = new Node (val, temp next),
            temp next = node,
            size++,
    }
```

// Insertion in the middle of index. ends here

// Delete First starts here:

```
public int deleteFirst () {
    int val = head value,
    head = head next,
    if (head == null) {
        tail = null,
    }
    size --,
    return val,
} // Delete first ends here
```

// Delete last starts here

```
public int deletelast() {
        if (size <= 1) {
            return deleteFirst(),
    }
    Node secondLast = get (size-2),
    int value = tail value;
    tail = secondLast,
   tail next = null,
  return val;
    }
```

// Delete last ends here

// Delete Mid starts here

```
public int delete (int index ) {
      if (index == 0) {
          return deletefirst();
      }
    if (index == size-1) {
          return deleteLast(),
    }
  Node prev= get (index-1);
  int val = prev next value;
  prev next = prev next next,
```

```
    return val;
}

//Delete mid ends here

// Finding node starts here
public Node find (int value) {
  Node node = head,
  while (node != null) {
     if (node value == value) {
          return node;
     }
     node = node.next;
  }

  return node;
}
public Node get (int index) {
    Node node = head,
  for (int i=0, i< index; i++) {
       node = node next,
  }
  return node.
} // Finding a node ends here
```

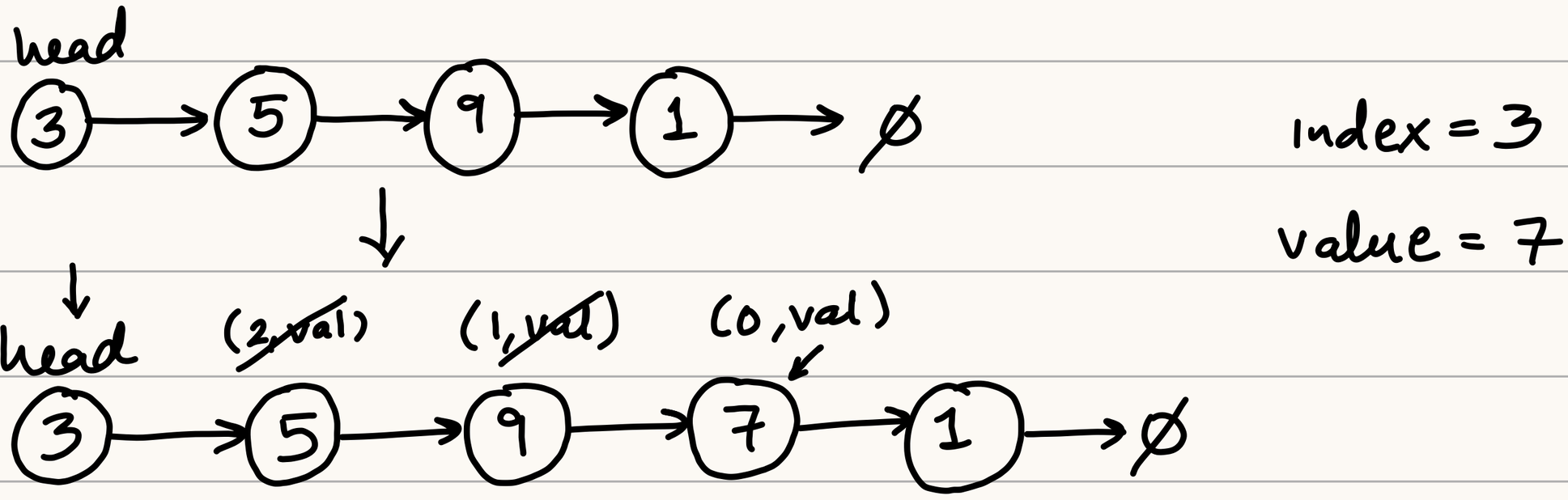
```
public void display() { // Displaying LL Starts here
    Node temp = head;
    while (temp != null) {
        System.out.print (temp.value + "→");
        temp = temp.next;
    }

    System.out.println("END");

}
// Displaying LL ends here
```

Main java

```
public class Main {
    public static void main (String [] args) {
        LL list = new LL(),
         list insertFirst ( 3);
         list insert Fist ( 2),
         list insert Fist (8),
        list insert Fist (17),
       list insert Last (99),
       list insert (100, 3),
         System.out println (list deleteLast()),
        list display ();

         System.out println( list delete (2)),
        List display ();
         list Find (3).
    }
}
```

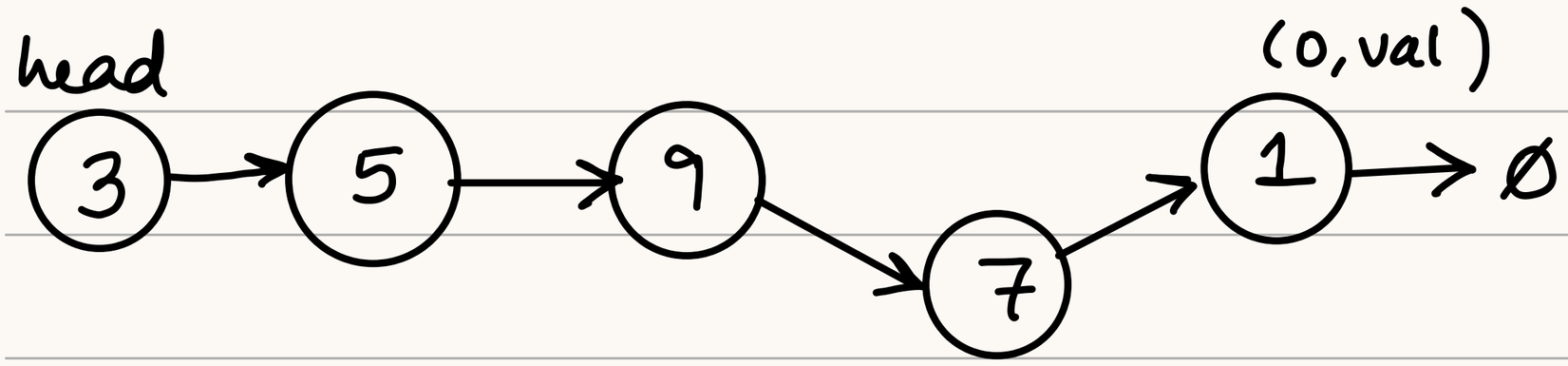
* FAANG questions

- Recursive Insertion in LinkedList:



① Void return type & make changes in Linked List

② Have a node return type that returns the list node to change the structure



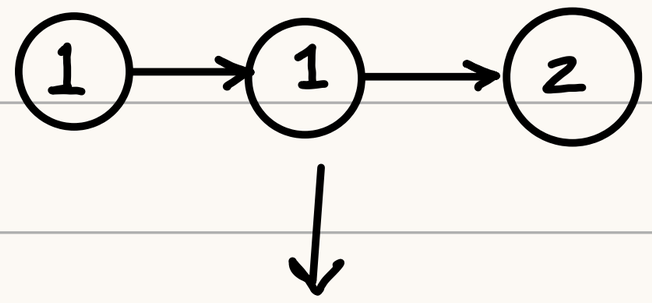
LL java

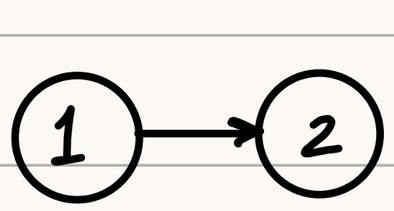
```
public void insertRec( int val, int index) {
        head = insertRec (val, index, head)
 }

 private Node insertRec (int val, int index, Node node) {
    if (index == 0) {
       Node temp = new Node(val, node),
       size++;
       return temp,
    }
    node next = insertRec (val, index--, node next),
    return node;
    }
```

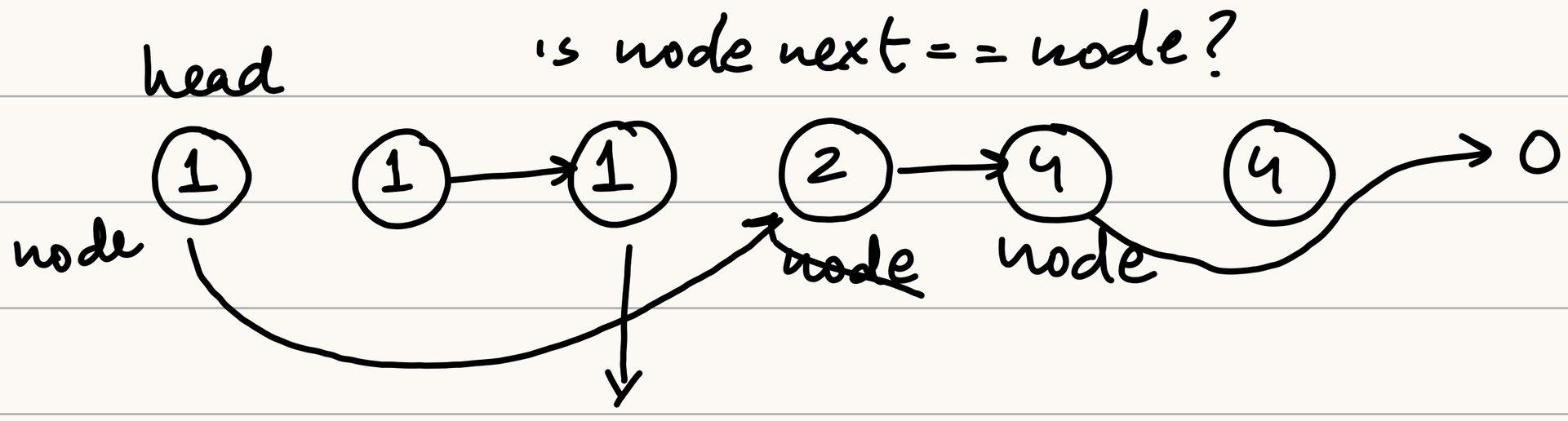
Main.java.

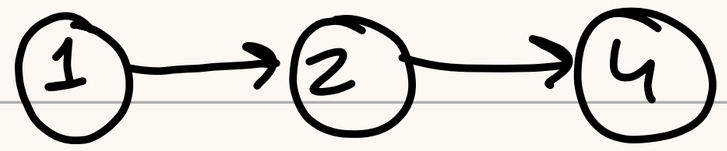
```
list insertRec(88, 2),
list display(),
```

★ Leet-Code 83 Remove Duplicates from Sorted List










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

```
        tail next = null,
}
```

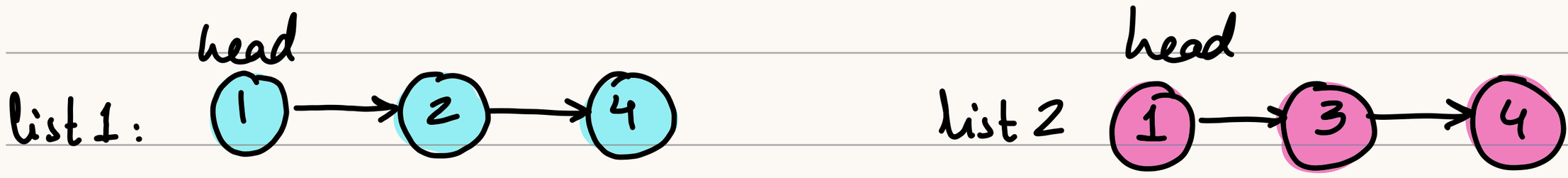
```
public static void main (String[] args) {
        LL list = new LL(),
        list insertLast (1),
       list insertLast (1),
        list insertlast (2),
        list insert last (3),
        list imert last (3);
        list imertlast (3);
        list insertlast (3);

list display (),
list. duplicates (),
    list duplay();
  }
}
```

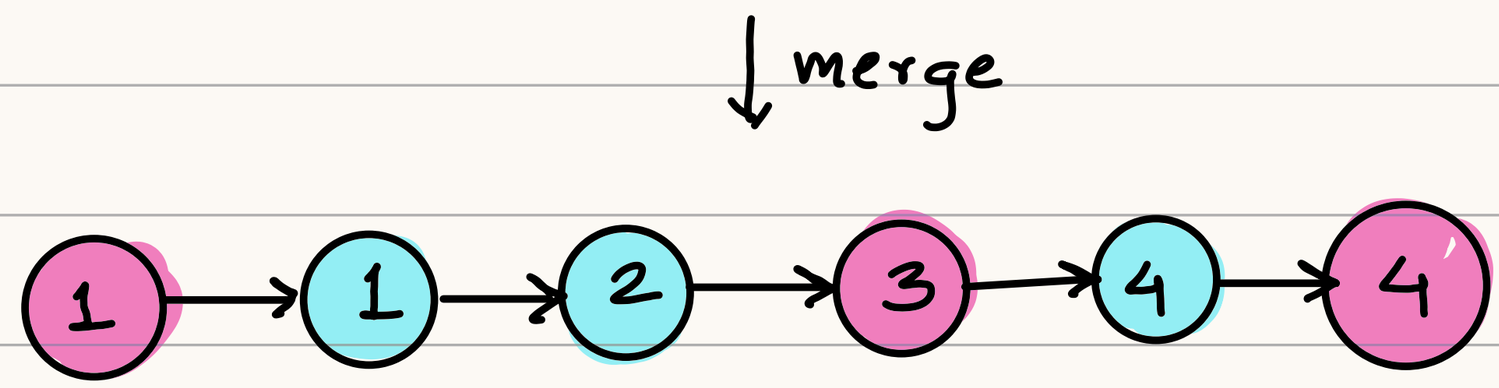
```
class Solution {
   public ListNode deleteDupliates (ListNode node) {
       if (node ==null) {
          return node,
    }
  ListNode head= node,
while (node.next != null) {
     if (node val == node next val) {
         node next = node next next,
```

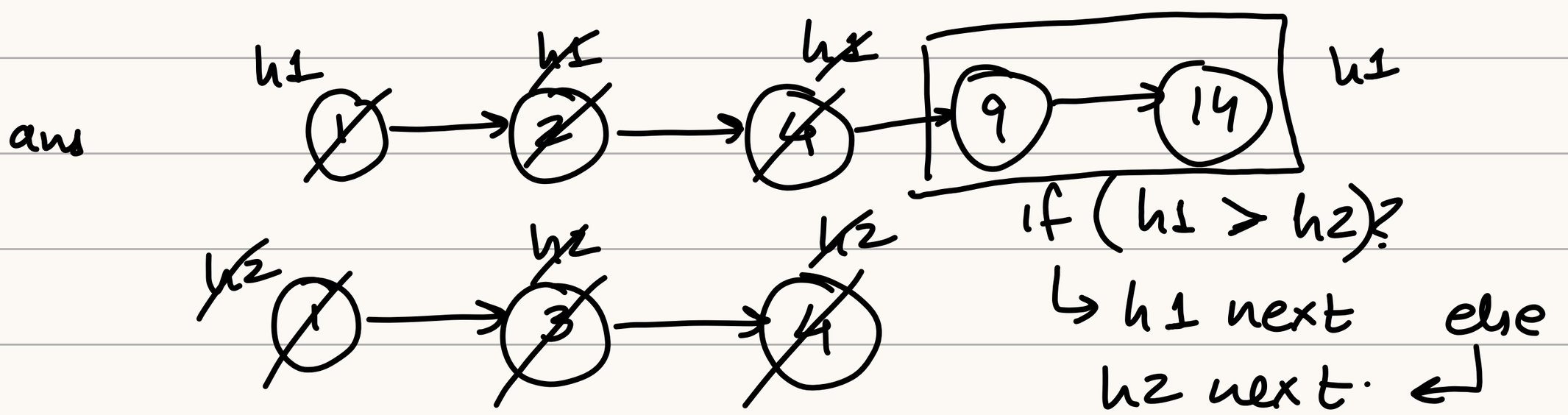
```
    } else {
        node = node.next;
    }
}
    return head;
    }
}
```

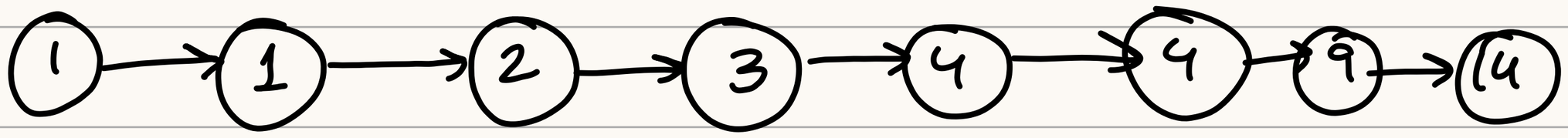
Leet code: 21 Merge two Sorted Lists:

Given heads of two sorted linked lists list1 & list2
Merge the two in one sorted list










```
public static LL merge (LL first, LL second) {
    Node f = first head,
    Node s = second head,
    LL ans = new LL(),
    while (f != null && s != null) {
        if (f value < s.value) {
            ans insertLast(f value),
            f = f next;
    }
```

```
        else {
            ans.insertLast(s.value);
            s = s.next;
        }
    }
    while (f != null) {
            ans.insertLast(f.value);
            f = f.next;
    }
    while (s != null) {
            ans.insertLast(s.value);
            s = s.next;
    }
    return ans;
}
public static void main(String[] args) {
        LL first = new LL();
        LL second = new LL();
        first.insertLast(1);
        first.insertLast(3);
        first.insertLast(5);
```

```
        second insertLast(1),
        second insertLast(2);
        second insertLast(9),
        second insertLast(14);

            LL ans = LL merge(first, second),
            ans display()
        }
    }
```

```
Class Solution {
    public ListNode mergeTwolists(ListNode list1, ListNode list2)
     ListNode dummyHead = new ListNode(),
     ListNode tail = dummyHead,
    while (list1 != null && list2 != null) {
    if (list1 val < list2 val) {
      tail next = list1;
        list1 = list1 next,
        tail= tail next,
    } else {
      tail next = list 2,
      list2 = list 2 next,
    tail = tail next,
```

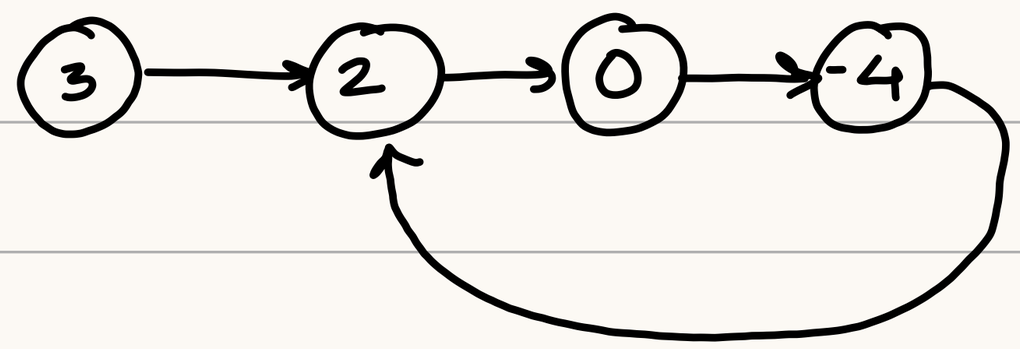
```
    }
}
```

```
tail.next = (list1 != null) ? list1 : list2;
return dummyHead.next;
    }
}
```

Leetcode 141 Linked List cycle

Given the head of LinkedList, determine if the linkedlist has a cycle in it

Example:



Input head = [3,2,0,-4], pos = 1

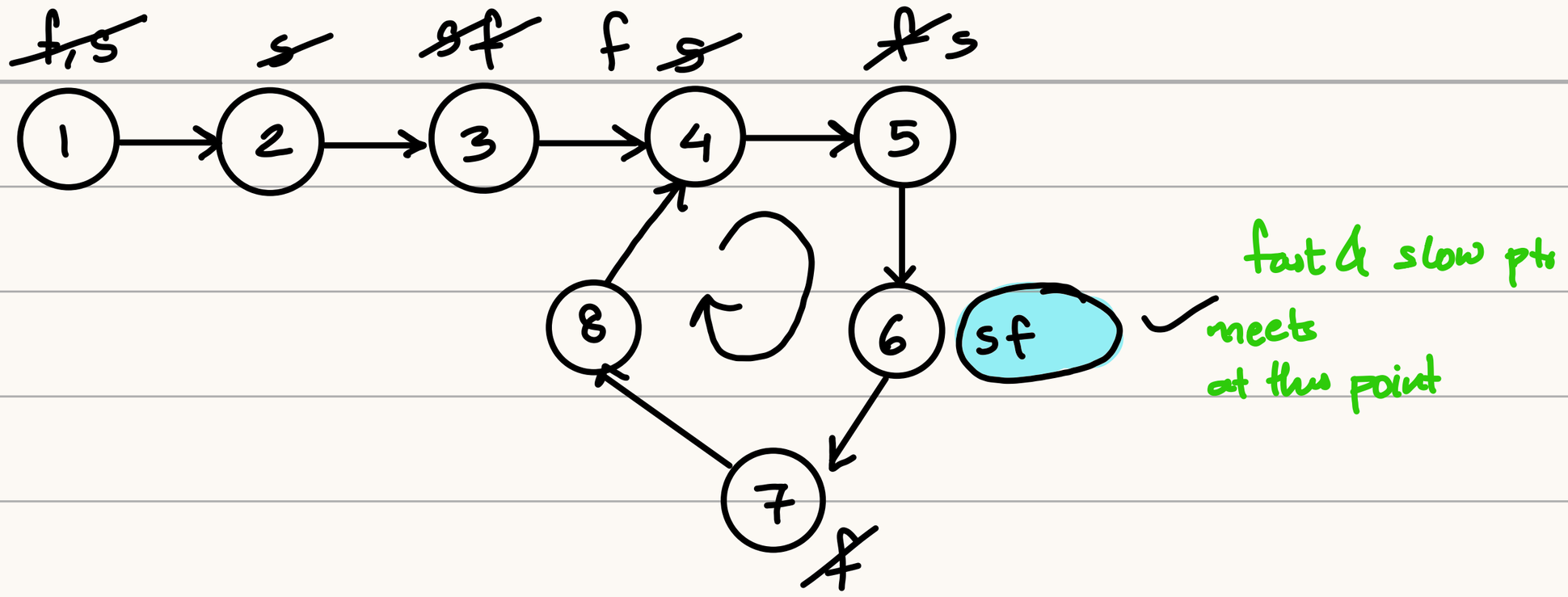
Output = true

* Cycle Detection in Linked List

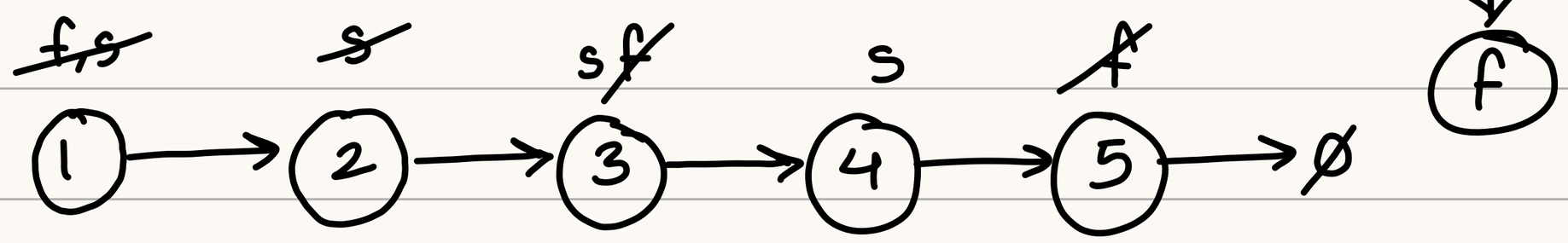
Fast & Slow pointer.

* Cycle detection

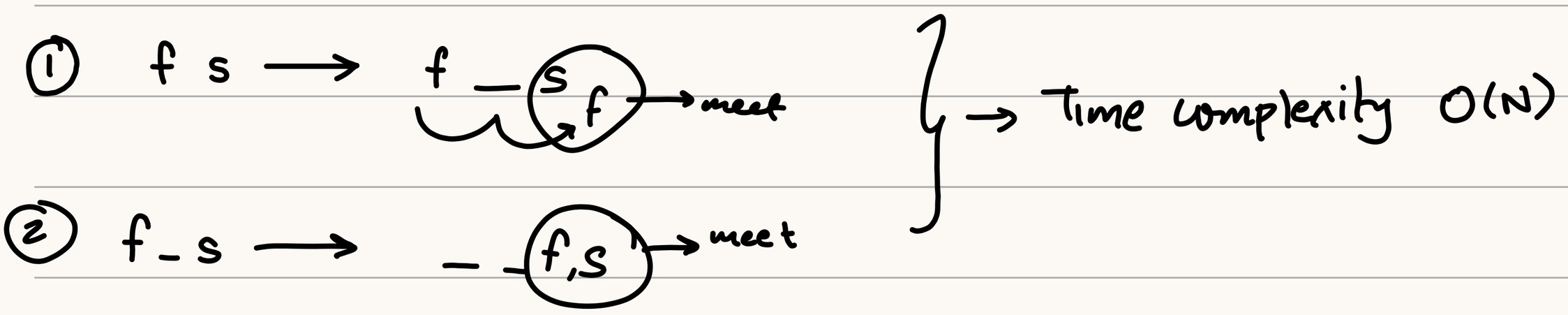
* Finding a node in cycle, etc



If cycle is not present cycle will be pointing to null,



* Two possiblities:



```
public class Solution {
   public boolean hasCycle(ListNode head) {
        ListNode fast = head,
        ListNode slow = head,
     while (fast != null && fast.next != null) {
        fast = fast.next.next,
        slow = slow.next,
      if (fast == slow) {
         return true,
    }
}
   return false;
 }
}
```

---

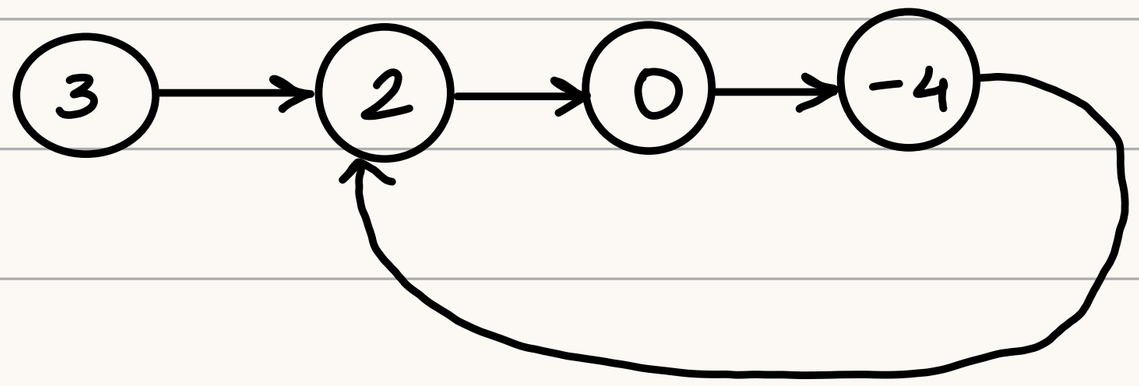
* Length of LinkedList Cycle

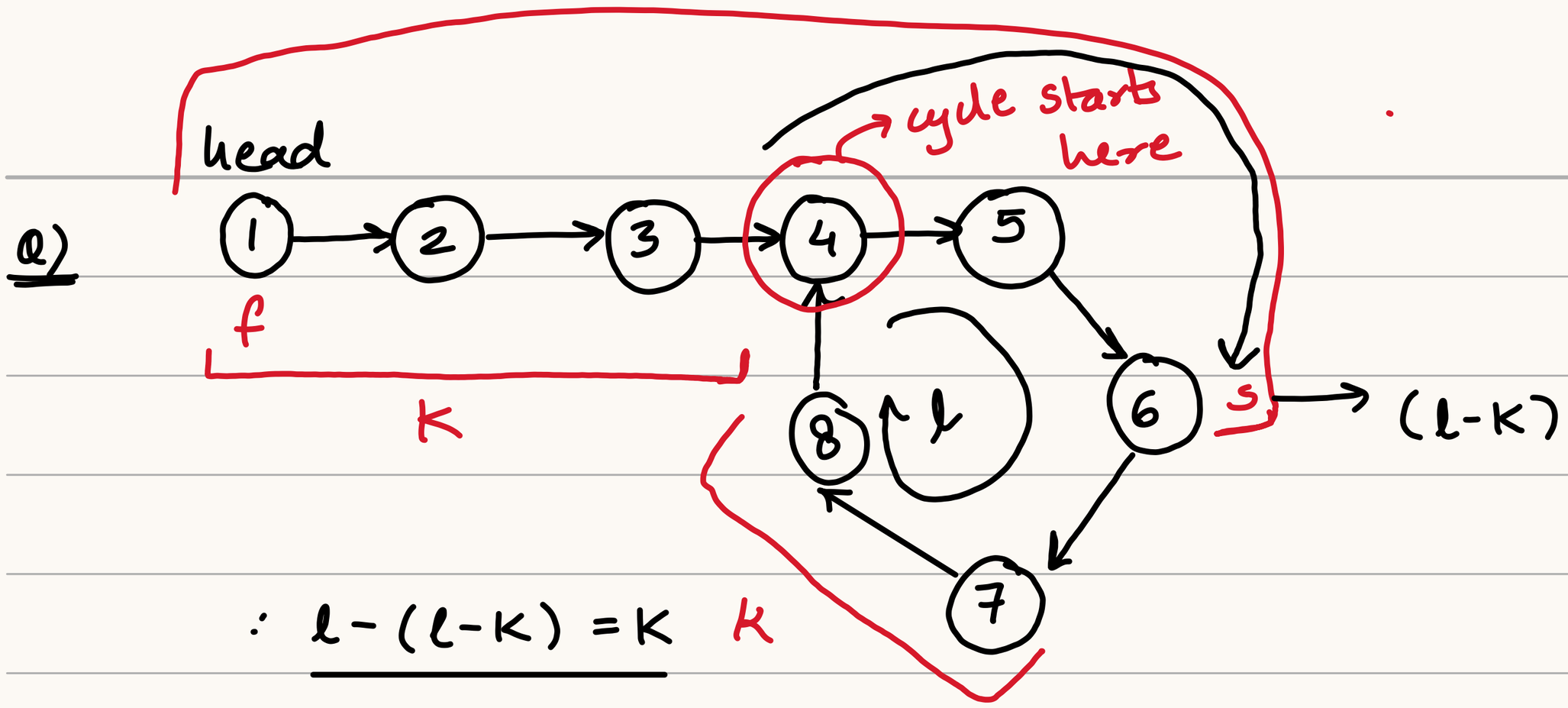
```
public int lengthCycle (ListNode head) {
  ListNode fast = head,
  ListNode slow = head;
 while (fast != null && fast.next != null) {
    fast = fast.next.next,
   Slow = slow.next,
```

```
if ( fast == slow) {
  ListNode temp = slow,
 int length = 0,
 do {
      temp = temp next,
     length ++,
   }
  while (temp != slow),
      return length;
 }
}
return 0,
}
}
```

Leetcode · 142 Linked list Cycle II (Medium).

Given the head of a LL, return the node where the cycle begins
If there is no cycle return null





Q) $\therefore l - (l-K) = K$

Return the node where the cycle starts

① Find length of cycle

② move s ahead by length of cycle times

③ Move s and f one by one, it will meet at start

```
public ListNode detectCycle (ListNode head) {
        int length = 0,
        ListNode fast = head,
        ListNode slow = head
    while (fast != null && fast.next != null) {
     fast = fast.next.next;
      slow = slow.next;
    if (fast == slow) {
       length = lengthCycle (slow),
       break,
    }
```

```
if (length == 0) {
    return null;
}
// find start node
ListNode f = head;
ListNode s = head;
while (length > 0) {
    s = s.next;
    length--;
}
// Keep moving both forward and they will meet at
cycle start
while (f != s) {
    f = f.next;
    s = s.next;
}
return s;
}
```

LeetCode · 202 Happy Number Google

Input · $n = 19$

Output true

Explanation $1^2 + 9^2 = 82$

$8^2 + 2^2 = 68$

$6^2 + 8^2 = 100$

$1^2 + 0^2 + 0^2 = 1$

Example $12 \longrightarrow 1^2 + 2^2 = 1 + 4 = 5$

$5^2 = 25$

$2^2 + 5^2 = 29$

$2^2 + 9^2 = 89$

Repeat

$8^2 + 9^2 = 145$

$1^2 + 4^2 + 5^2 = 42$

$4^2 + 2^2 = 20$

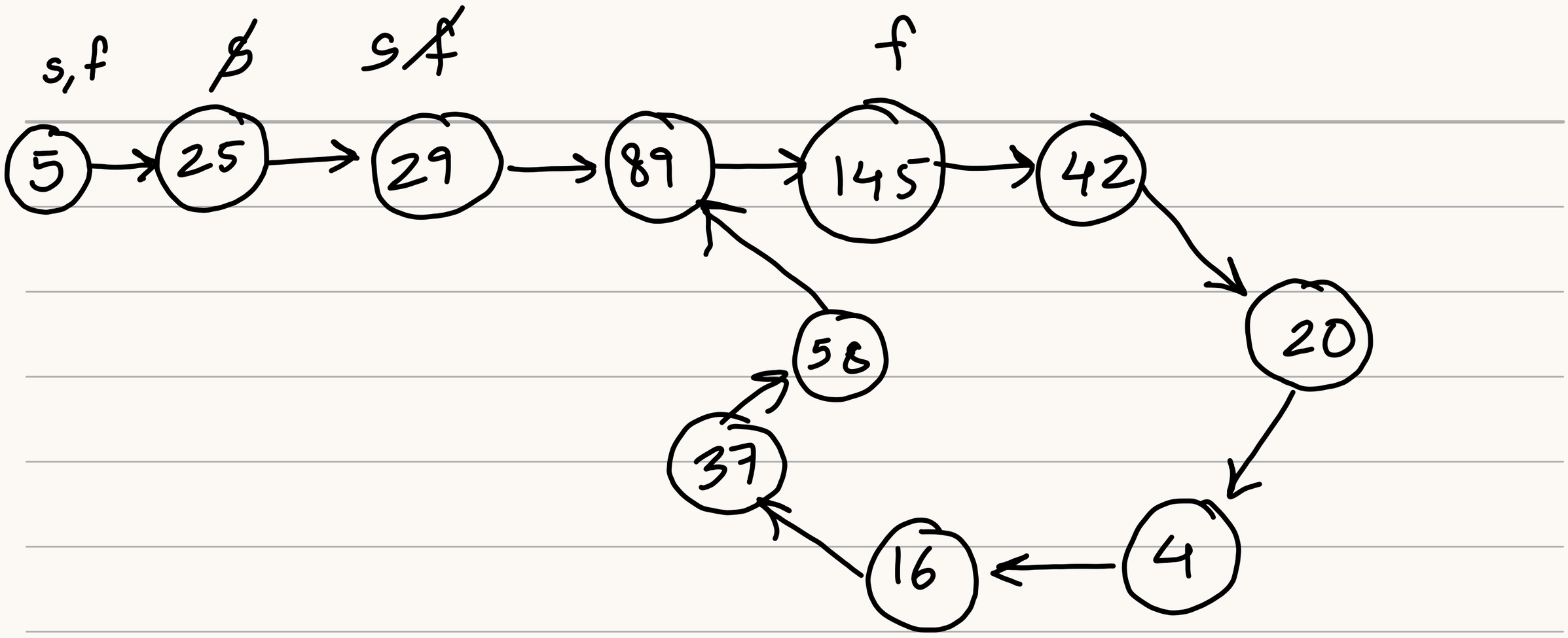
$2^2 + 0 = 4$

$4^2 = 16$

$1^2 + 6^2 = 37$

$3^2 + 7^2 = 9 + 49 = 58$

$5^2 + 8^2 = 25 + 64 = 89$



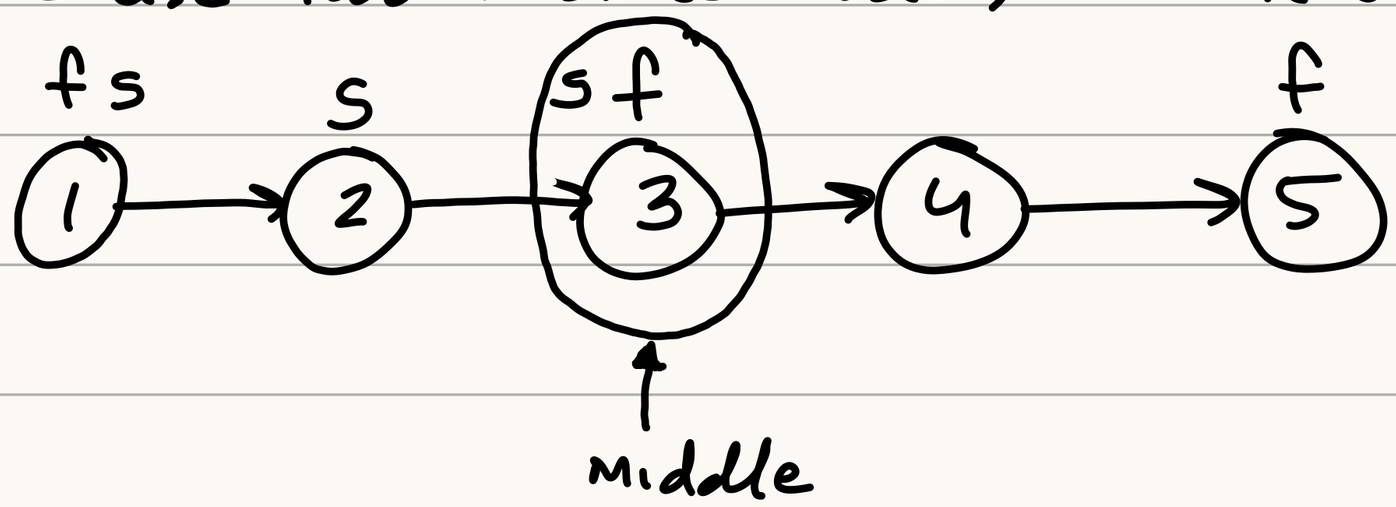

```
public boolean isHappy (int n) {
    int slow = n;
    int fast = n,
    do {
      slow = findSquare(slow);
      fast = findSquare (findSquare(fast));
    } while (slow != fast),
    if (slow == 1) {
        return true,
    } return false.
    }

private int findSquare (int number) {
      int ans = 0,
      while (number > 0) {
        int rem = number % 10,
        ans = ans + rem * rem,
        number = number/10,
      }
```

```
        return ans,
    }
}
```

Leet Code 876 Middle of the Linked List.

Given the head of a singly LinkedList, return the middle node of the LinkedList

If there are two middle nodes, return second middle node




```
class solution {
  public ListNode middleNode (ListNode head) {
        ListNode s = head,
        ListNode f = head,
        while (f != null && f.next != null) {
              s= s next,
              f = f next next.
        }
        return s,
    }
}
```

LeetCode 148 Sort List

Given the head of linked list, return the list after sorting it in ascending order

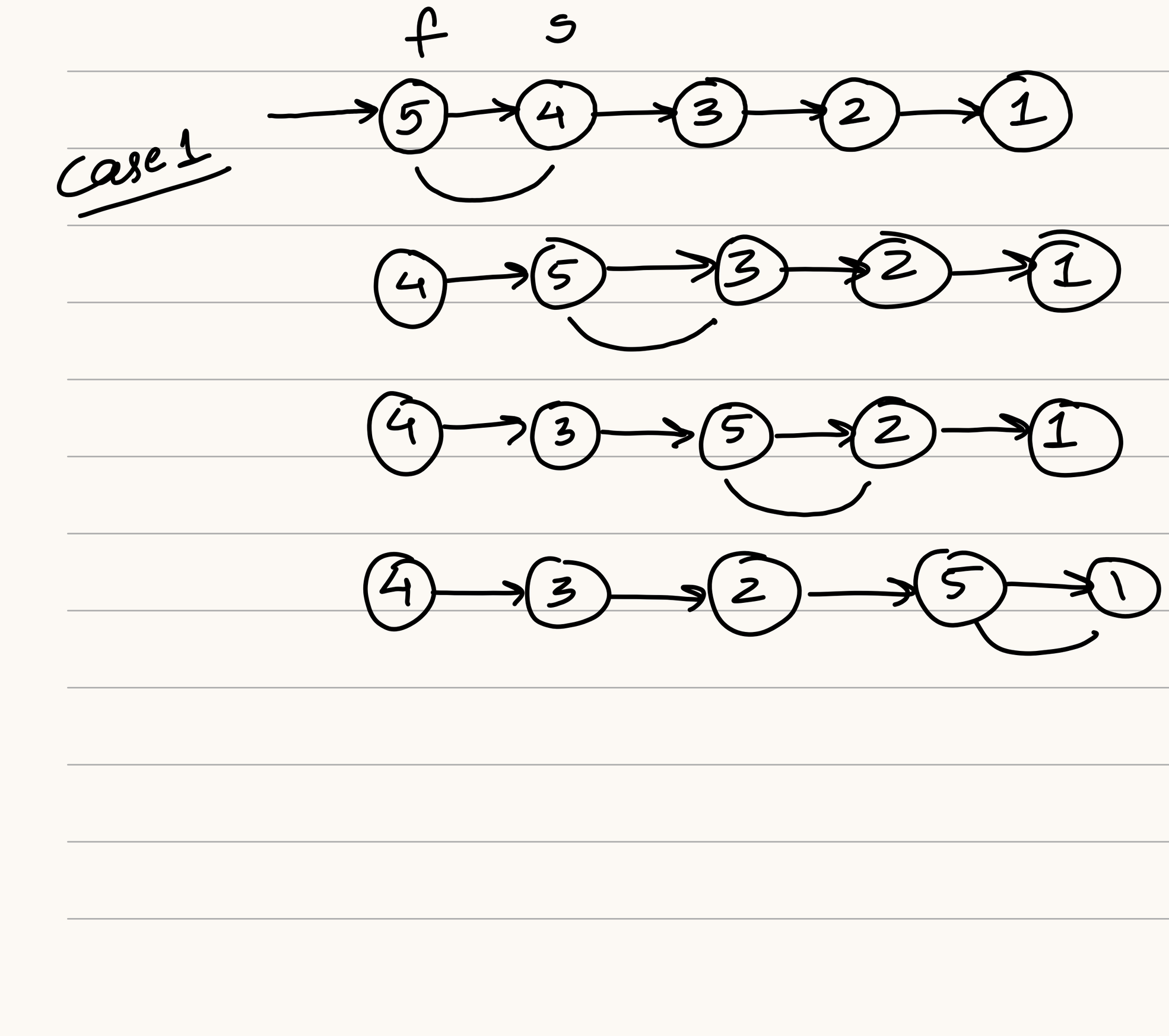
```
public class MergeSort {
  ListNode merge (ListNode list1, ListNode list2) {
    ListNode dummyHead = new Listnode (),
    ListNode tail = dummyHead,
   while (list1 != null && list2 != null) {
     if (list1 val < list2 val) {
          tail next = list1;
          list1 = list1 next,
          tail = tail next,
     } else {
          tail next = list2;
          list2 = list2 next,
          tail = tail next,
     }
  }
  tail next = (list1 != null) ? list1 : list2,
  return dummyHead next,
}
```

```
ListNode getMid (ListNode head) {
    ListNode midPrev = null,
    while (head != null && head.next != null) {
     midPrev = (midPrev == null) ? head : midPrev.next,
        head = head.next.next,
    }
ListNode mid = midPrev.next,
midPrev.next = null,
return mid,
}
}
```

```
class ListNode {
  int val;
  ListNode next;
  public ListNode() {
   }

  ListNode (int x) {
        val = x,
        next = null,
   }
}
```

```
public ListNode sortList (ListNode head) {
    if (head == null || head.next == null) {
        return head;
    }
    ListNode mid = getMid (head);
    ListNode left = sortList (head);
    ListNode right = sortList (mid);
    return merge (left, right)
  }
}
```

Bubble Sort.

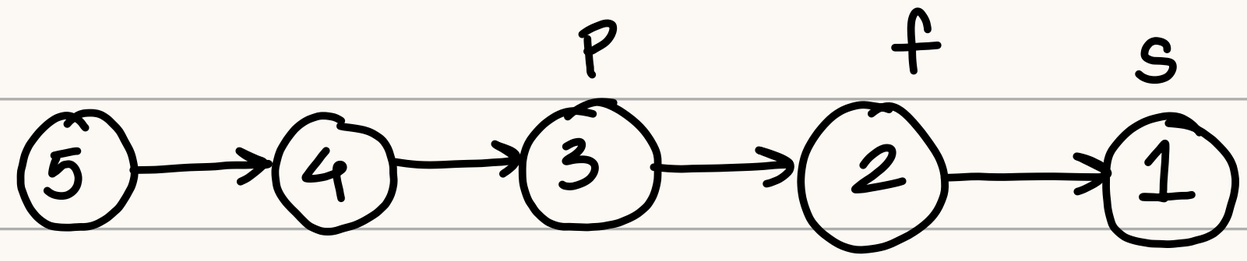


head = s,

f next = s next,

s next = f,

Case 2.

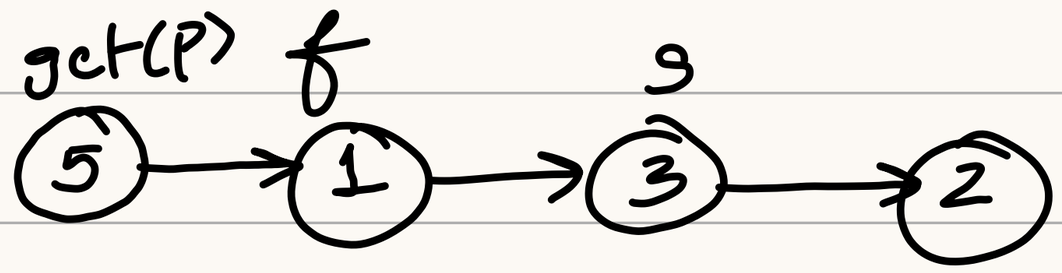


p next = s

tail = f

f next = ∅

s next = tail



p next = s

f.next = s next

s next = f

```
public void  bubbleSort() {
      bubbleSort(size-1, 0),
}

private void bubbleSort (int row, int col) {
   if (row == 0) {
       return,
   }
   if (col< row) {
       Node first = get(col);
       Node second = get(col+ 1),
      if ( first value > second value) {
        // swap
     if ( first == head) {
        head = second;
        first.next = second next;
        second next= first,
     } elseif ( second = = tail) {
       Node  prev= get(col-1);
       prev.next  = second,
       tail = first.
       first.next = null;
       second next = tail;
   }
```

```
else {
        Node prev = get(col-1),
        prev.next = second,
        first.next = second.next,
        second.next = first;
    }
  }
  bubbleSort(row, col+1),
}
 else {
   bubbleSort(row-1, 0),
}
```

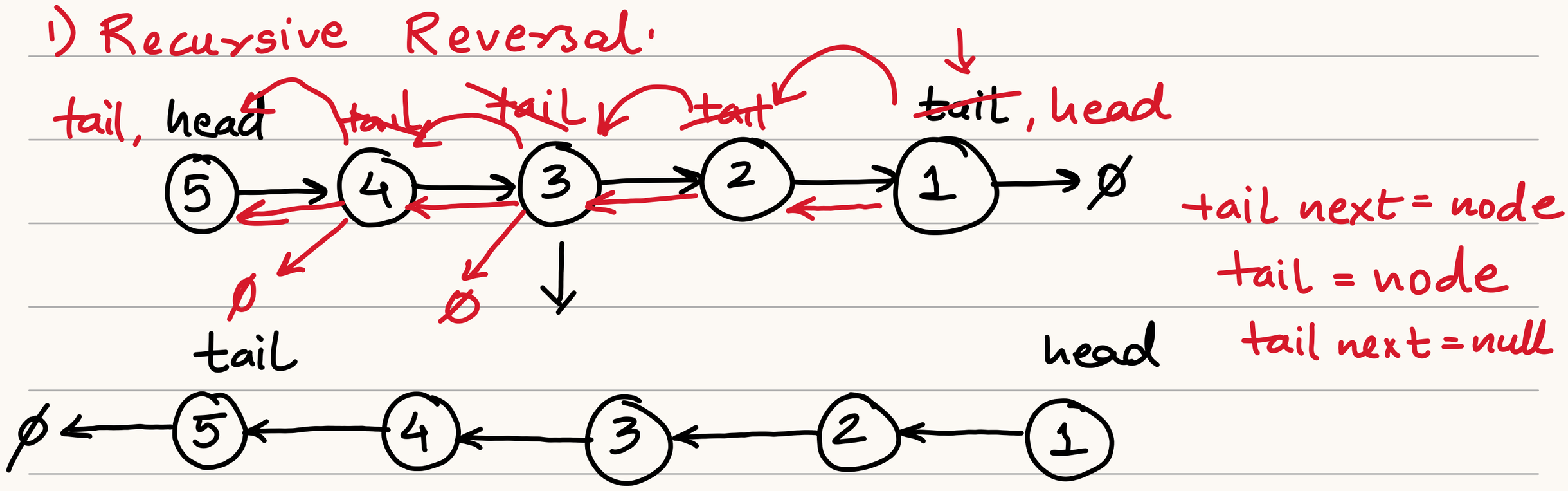
★ Reversing a Linked List.

Two ways.

1) Recursive

2) Iterative

1) Recursive Reversal.



Recursion means dividing a problem into sub problems

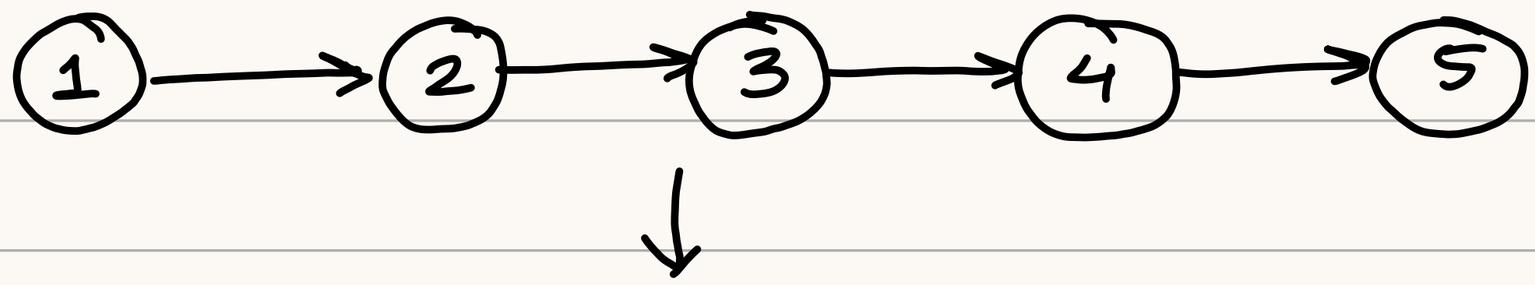
Code for Recursive Reversal

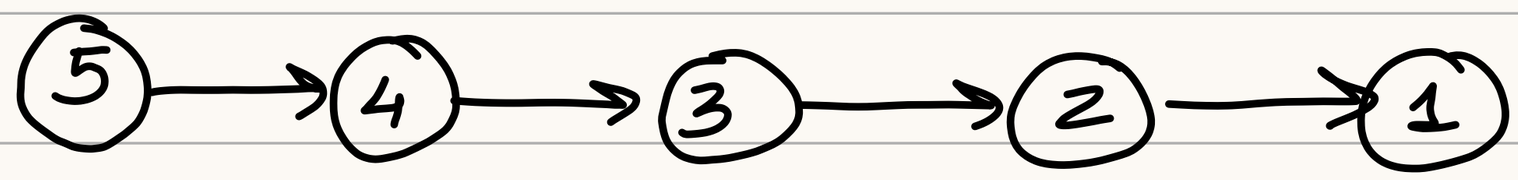
```
private void reverse (Node node ) {
    if (node == tail) {
        head = tail,
        return,
    }
reverse (node next),
tail next = node,
tail = node,
```

```
  tail next = null,
}
```

LeetCode · 206 Reverse Linked List · Google, Microsoft, Amazon, Apple

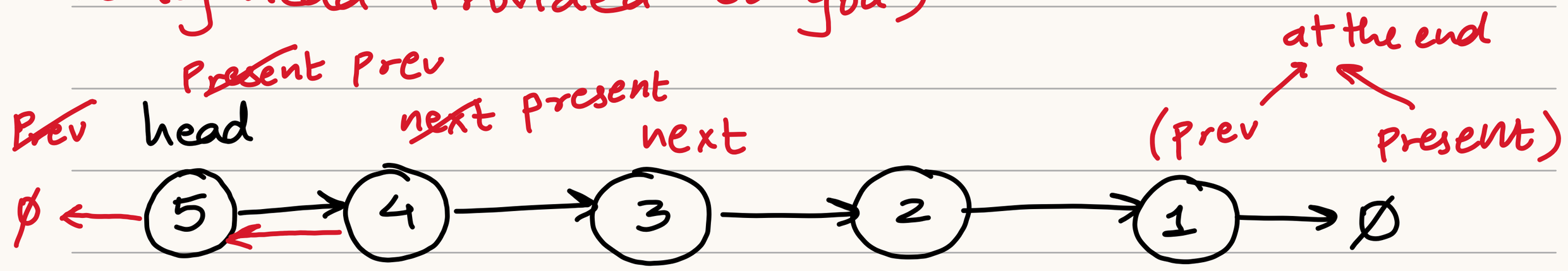
Given the head of a singly linkedList, reverse the list and return the reversed list,



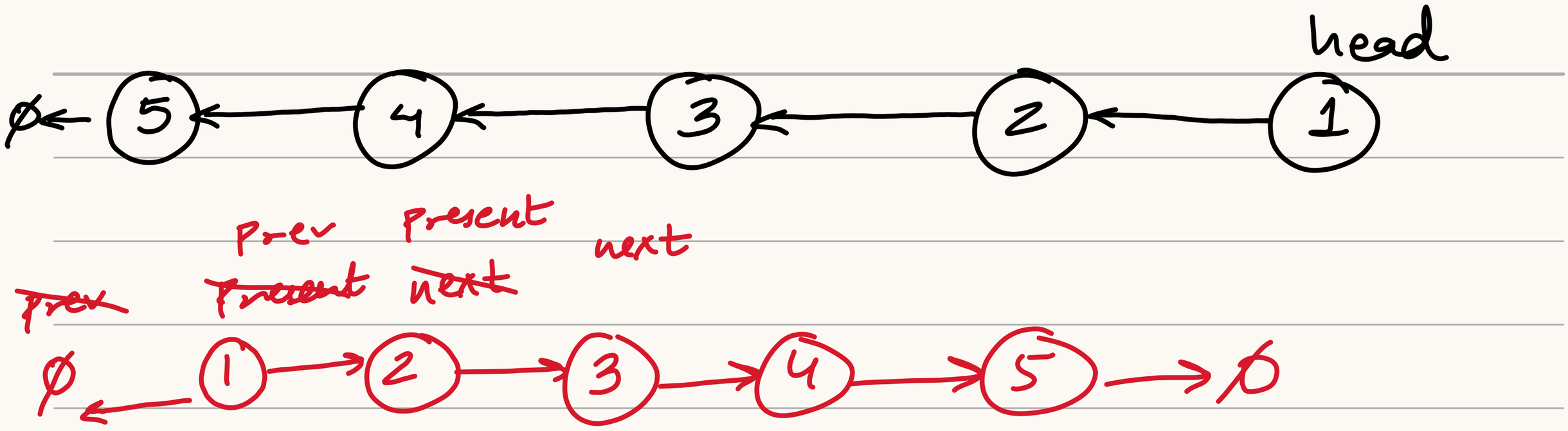


2) Iterative Reversal

(Only head Provided to you)




```
while (present != null)

    present next = prev

    prev = present
    present = next
     next = next next  // note. null pointer exception check
```

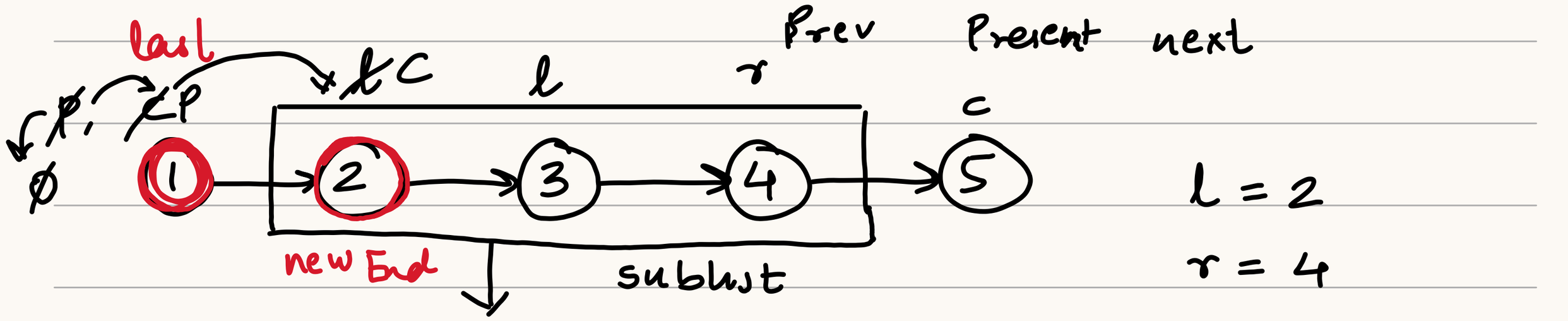


LeetCode · 206 Reverse Linked List · Google, Microsoft, Amazon, Apple

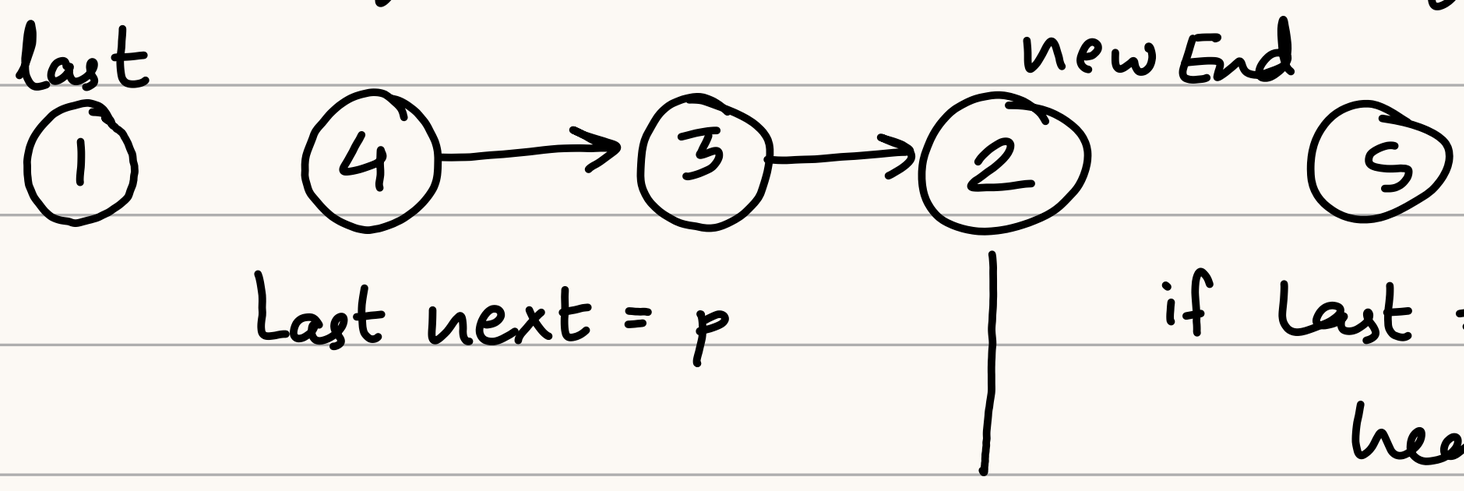
```
class solution {
    public ListNode reverselist (ListNode head) {
        if (head == null) {
            return head,
        }
        ListNode prev = null,
        ListNode present = head,
        ListNode next = present.next,
        while (present != null) {
            present next = prev,
            prev = present,
            present = next,
            if (next != null) {
                next = next next,
            }
        } return prev;
}
```

LeetCode 92 Reversed Linked List II (Google, MS, Meta)

Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right. and return the reversed list



What we require Previous element of sublist



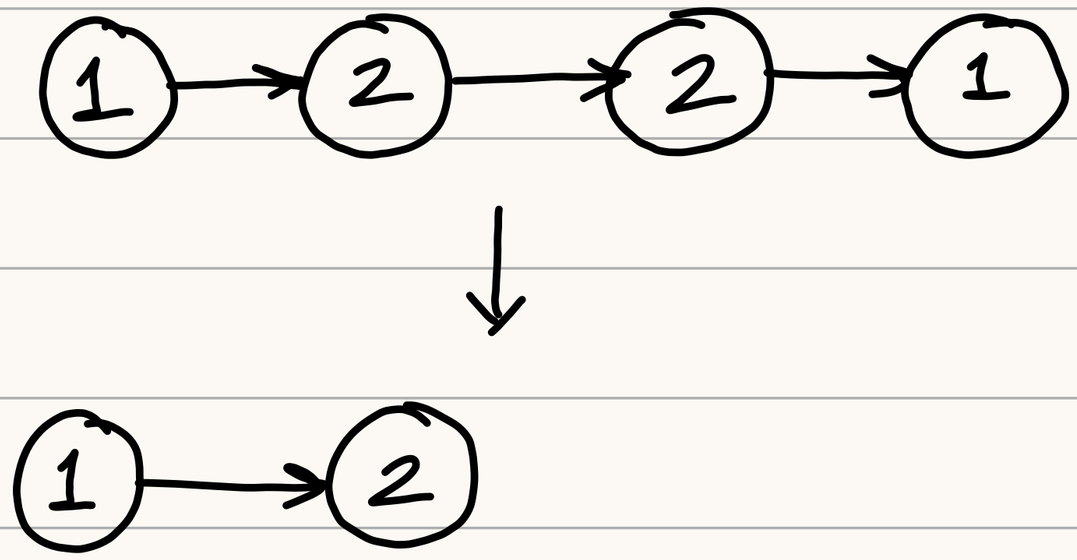
```
public ListNode reverseBetween (ListNode head, int left,
                                int right){
    if (left == right) {
        return head,
    }
 // Skip the first left-1 nodes
 ListNode present = head,
 ListNode prev = null,
 for (int i=0, present != null && i< left-1, i++) {
     prev = present;
     present = present next,
 }
 ListNode Last = prev,
 ListNode newEnd = present,
 // reverse between left & right
  ListNode next = present next,
   for (i=0, present!=null && i< right - left +1, i++) {
      present next = prev,
      prev = present,
      present = next,
      if (next != null) {
          next = next next,
      }
}
```

```
if (last != null) {
    Last next = prev,
} else {
    head = prev,
}
newEnd next = present,
return head
}
```

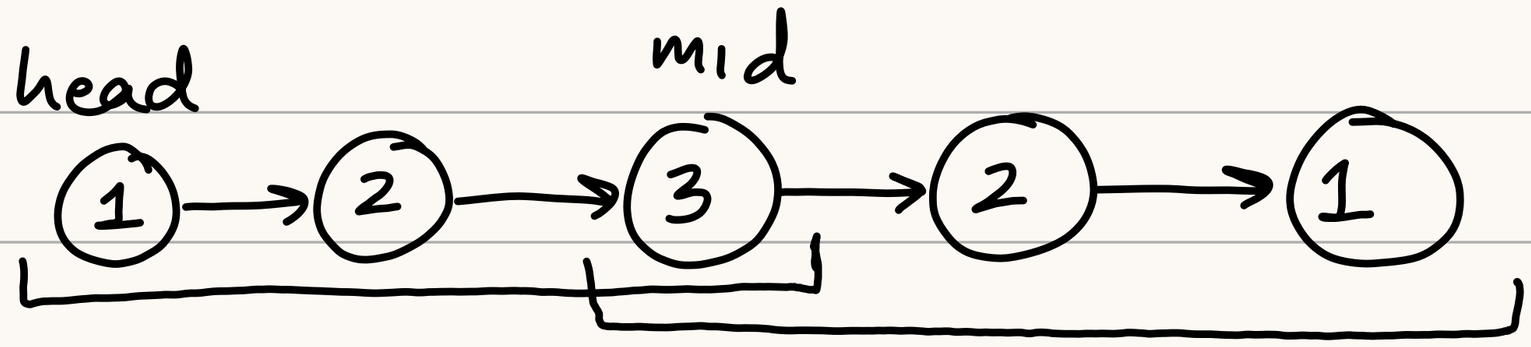
Google, Microsoft, facebook, LinkedIn, Amazon, Apple

★ LeetCode 234 Palindrome LinkedList.

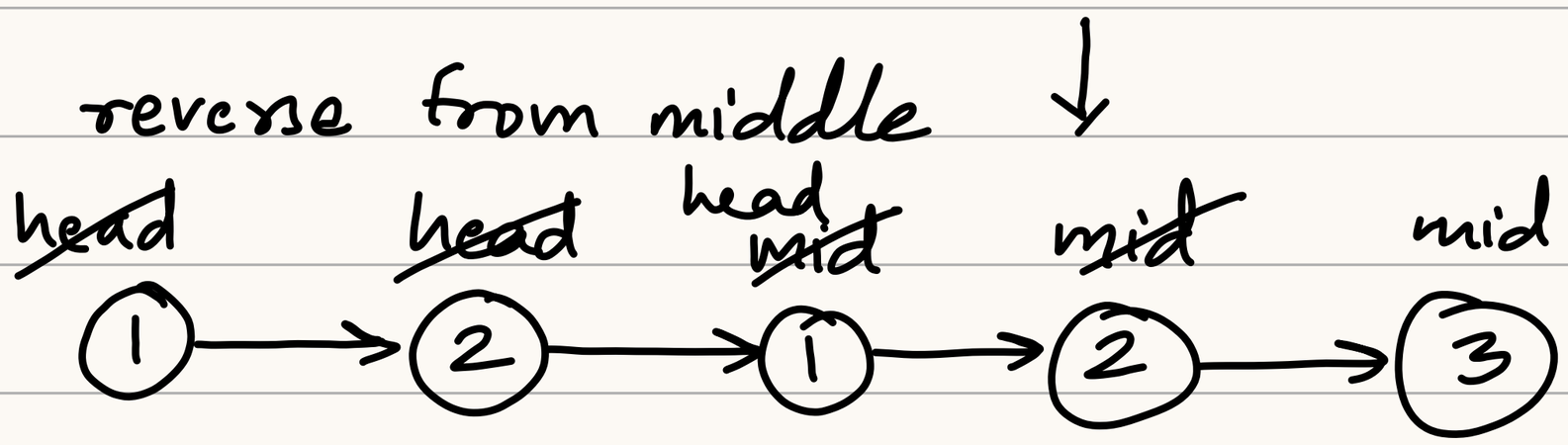
Given the head of a singly linked list, return true if it is a palindrome



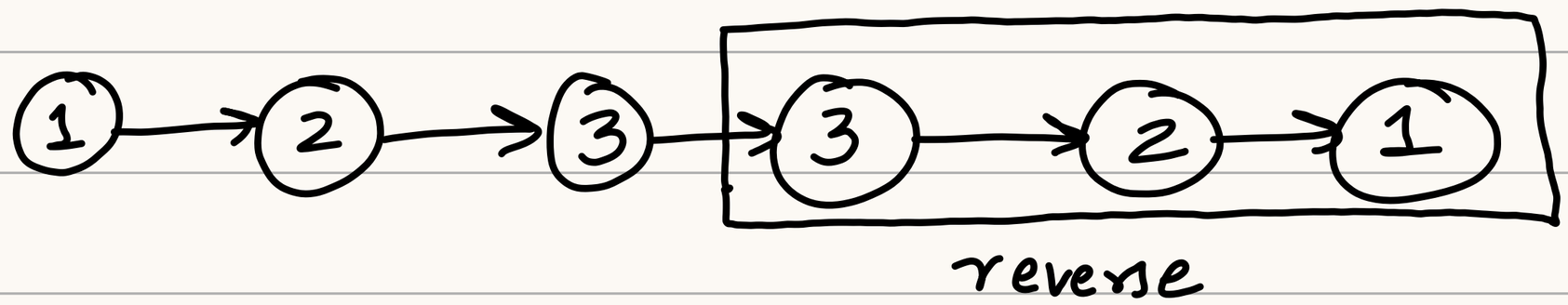
★ Palindromic LinkedList.

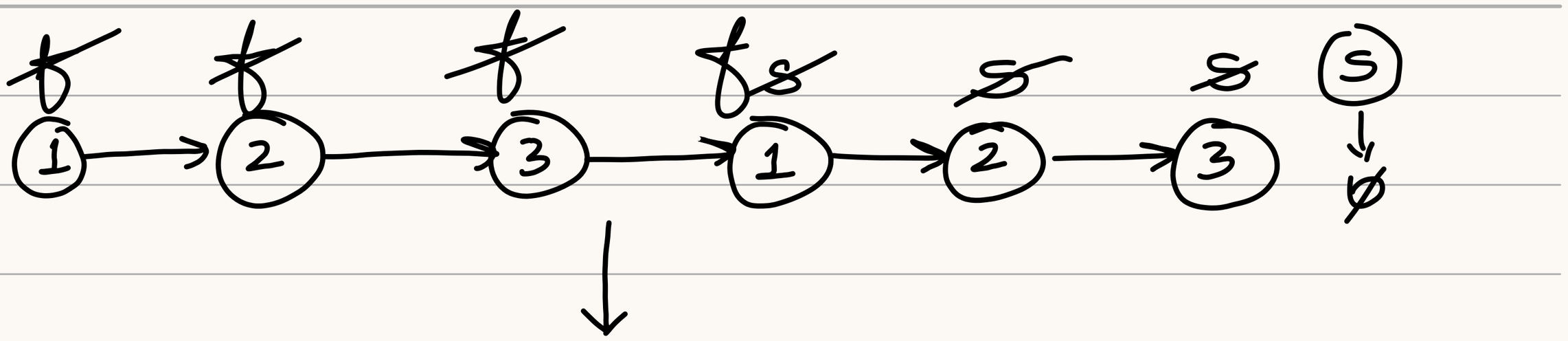


reverse from middle ↓



For even number.





reverse the second half

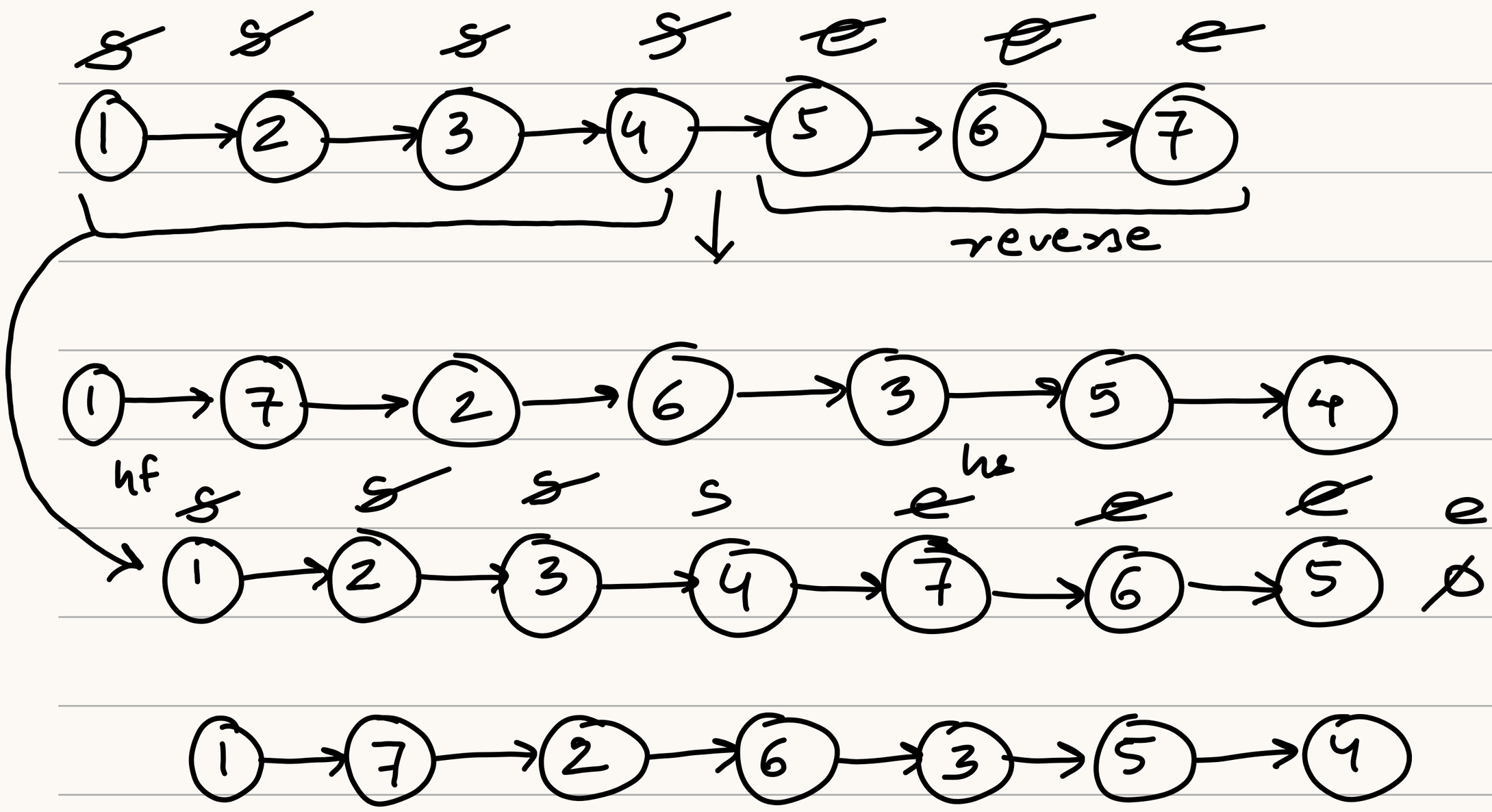
```
Public boolean isPalindrome (ListNode head) {
   ListNode mid = middleNode (head),
   ListNode headSecond = reverseList (mid),
  ListNode rereverseHead = headSecond, // storing
  // compare both the halves
   while (head != null && headSecond != null) {
   if (head val != headSecond val) {
            break;
   }
   head = head.next,
   headSecond = headSecond next;
}
   reverseList (rereverseHead),
   if (head == null || headSecond == null) {
      return true,
   }
   return false,
}
```

```
public ListNode middleNode (ListNode head) {
    ListNode s = head,
    ListNode f = head,
    while (f != null && f.next != null) {
            s = s next,
            f = f next next,
    }
   return s,
}
```

```
public ListNode reverseList (ListNode head) {
    if (head == null) {
        return head,
    }
 ListNode prev = null,
ListNode present = head,
ListNode next = present next;
while (present != null) {
      present.next = prev,
      prev = present,
      Present = next;
     if ( next != null) {
        next = next next,
   }
}
```

```
        return prev;
    }
}
```

★ Leetcode 143 Reorder List Google & Facebook



★ Important to know:

Cycle detection ✓

Fast and slow pointers ✓

Reversal of LinkedList ✓

temp = hf next

hf next = hs

hf = temp

temp = hs next

hs next = hf

hs = temp

```
public void reorderList (ListNode head) {
        if (head == null || head next == null) {
                return,
        }
    ListNode mid = middleNode (head),
    ListNode hs = reverseList (mid),
    ListNode hf = head,

// Rearrange
while (hf != null && hs != null) {
    ListNode temp = hf next,
     hf.next = hs;
     hf = temp,
     temp = hs next,
     hs next = hf,
     hs = temp,
}
```

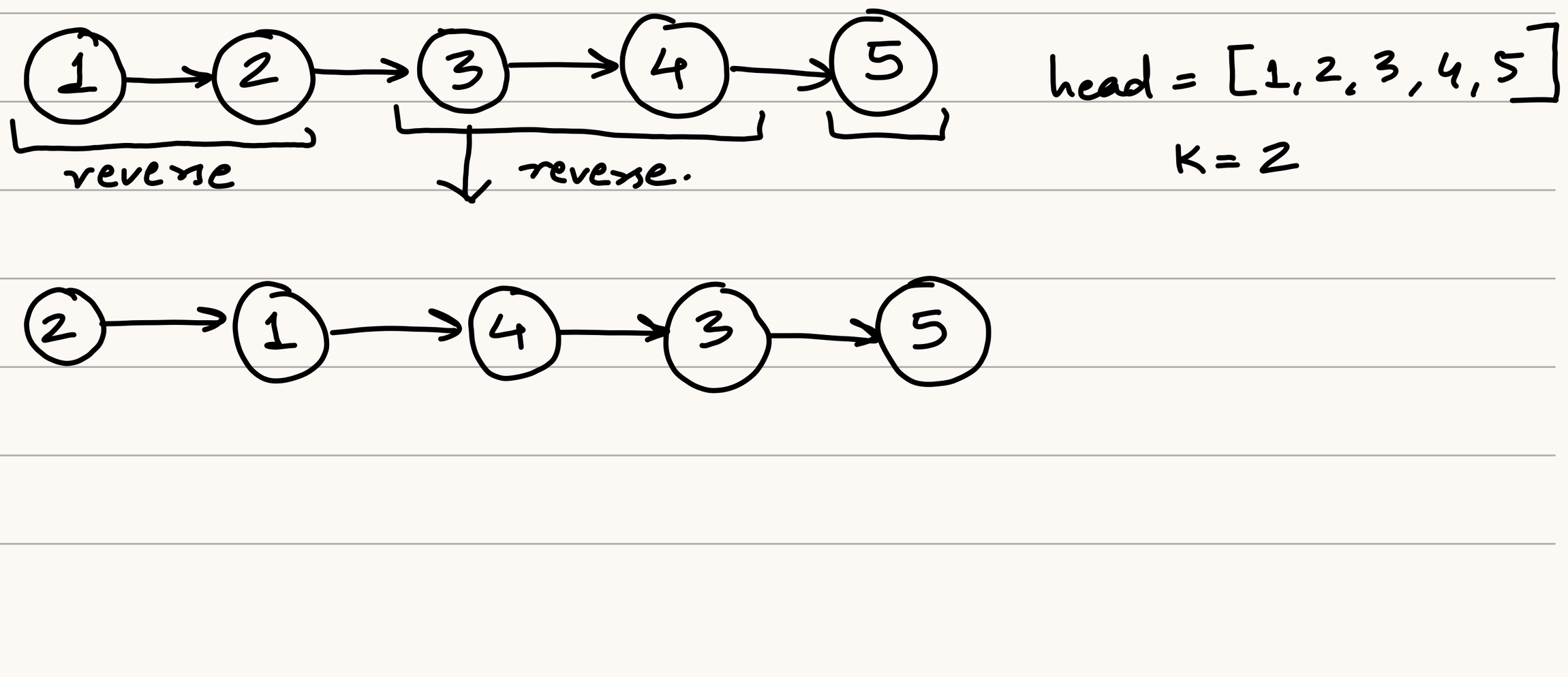
```
// next of tail to null
if (hf != null) {
    hf.next = null,
}

public ListNode middleNode(ListNode head) {
    ListNode s = head,
    ListNode f = head,
    while (f != null && f.next != null) {
        s = s.next,
        f = f.next.next,
    }

    return s,
}

public ListNode reverseList(ListNode head) {
    if (head == null) {
        return head,
    }
    ListNode prev = null,
    ListNode present = head;
    ListNode next = present.next,
    while (present != null) {
        present.next = prev,
        prev = present,
        present = next,
```

```
    if ( next ! = null) {
        next =  next next,
    }
}
```

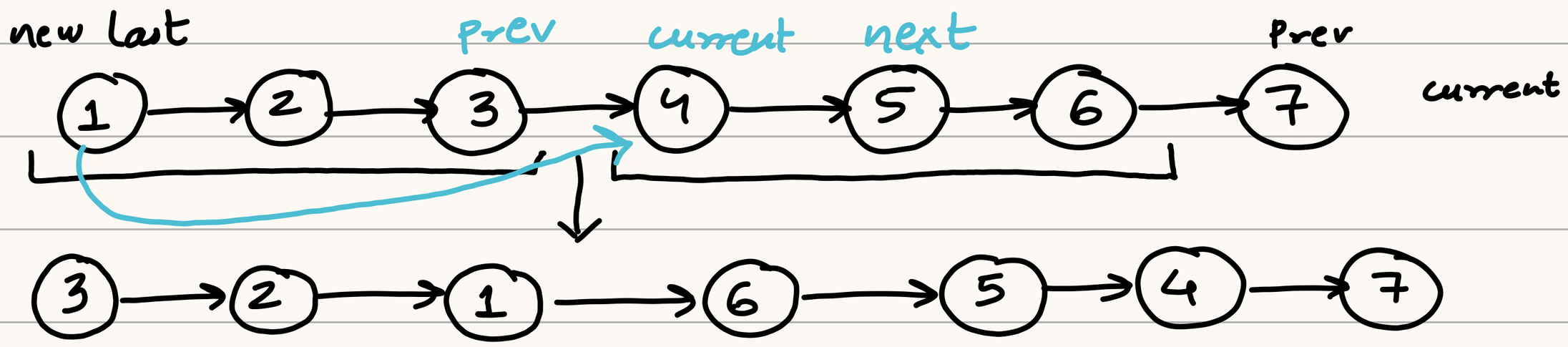
```
  return prev,
}
```

★ Leetcode 25. Reverse Nodes in K-Group

Given a linked list, reverse the node of a linked list
k at a time and return its modified list
k is a positive integer and is less than or equal to
the length of linked list If the number of nodes
is not a multiple of k then left-out nodes,
in the end, should remain as it is
You may not alter the values in the list's nodes,
only nodes themselves may be changed



Q · Reverse every K-node K = 3



run a for loop K-times till current is not equals to null

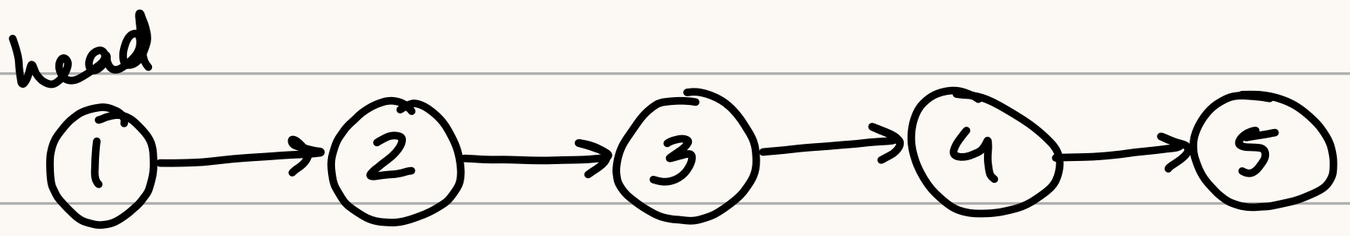
newlast next = current

```
public ListNode reverseKGroup (ListNode head, int k) {
    if (k<=1 || head == null) {
        return head,
    }
ListNode current = head,
ListNode prev= null,
 while (true) {
ListNode last = prev,
ListNode newEnd = current,
```

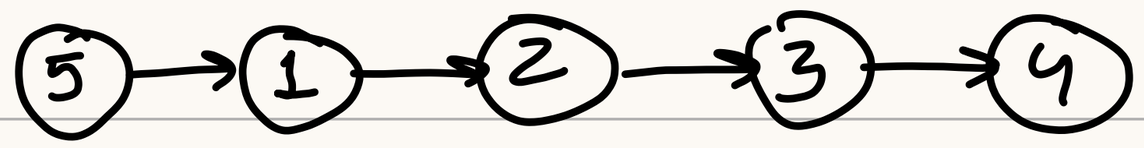
```
ListNode next = current.next,
for (int i = 0 , current != null && i < k , i++){
   current next = prev;
   prev = current,
   current = next,
   if (next != null) {
         next = next next,
   }
}
if (last != null) {
    last next = prev;
}
else {
    head = prev;
}
newEnd next = current,
if (current == null) {
   break;
}
   prev = newEnd,
}
return head,
}
```

★ LeetCode · 61 Rotate List Facebook, Google, Twitter

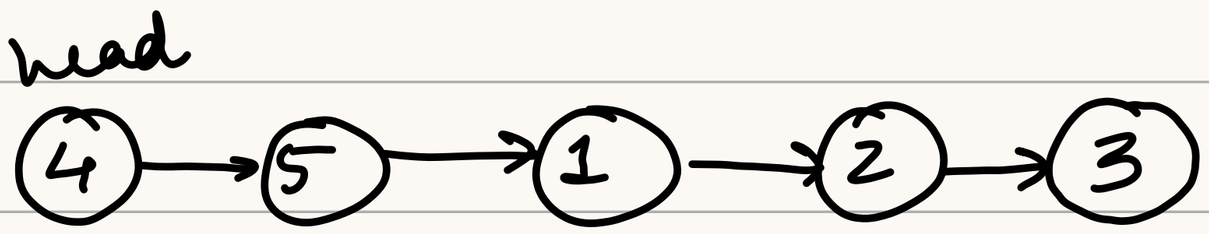
Given the head of a linked list, rotate the linkedList to k places



rotate 1



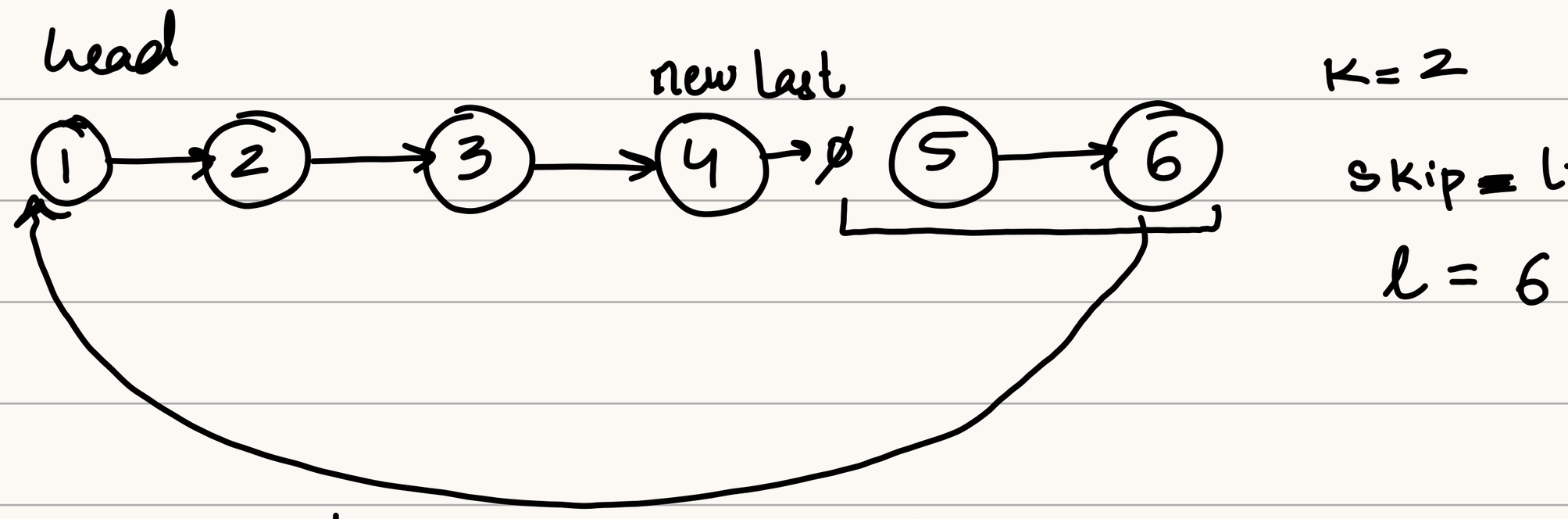
rotate 2



Original Last node next = original head

new head = start of sublist

new tail = node before start of sublist



head = newlast next

new Last next = ∅

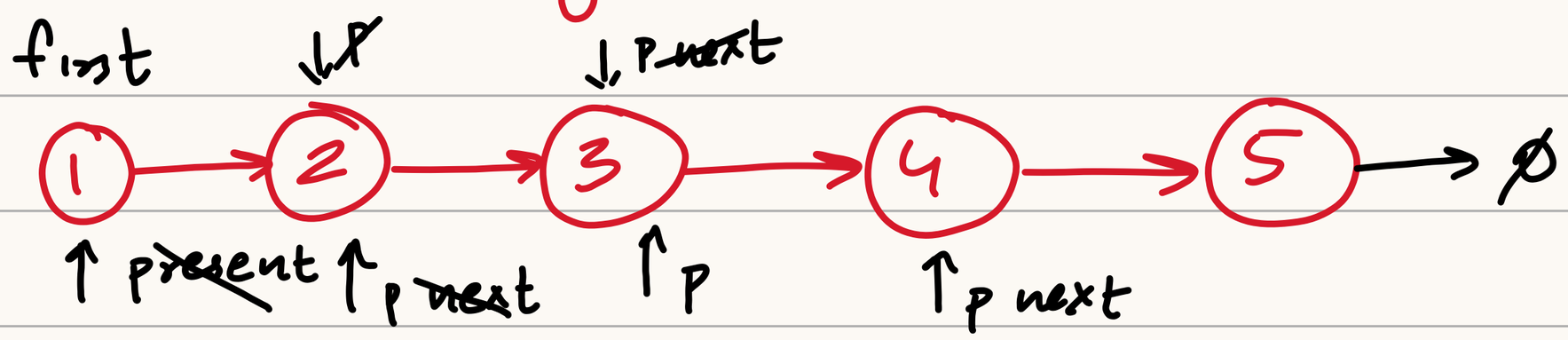
rotateon = k / l

```
public ListNode rotateRight (ListNode head, int K) {
        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,
    int rotations = K %. length;
    int skip = length - rotations,
ListNode newlast = head;
    for( int i=0, i< skip-1; i++) {
newlast = newlast.next;
}
    head = newlast.next.
        newLast.next = null,
        return head;
    }
}
```

* Inplace Reversal of LL
* Fast & slow pointer
* Recursion

```
public class MyLinkedO{
    Static class Node {
     public Node (double item, Node next) {
    this item= item,
    thus next = next,
}
Node first,
```

Q1) Return Size of Linked List

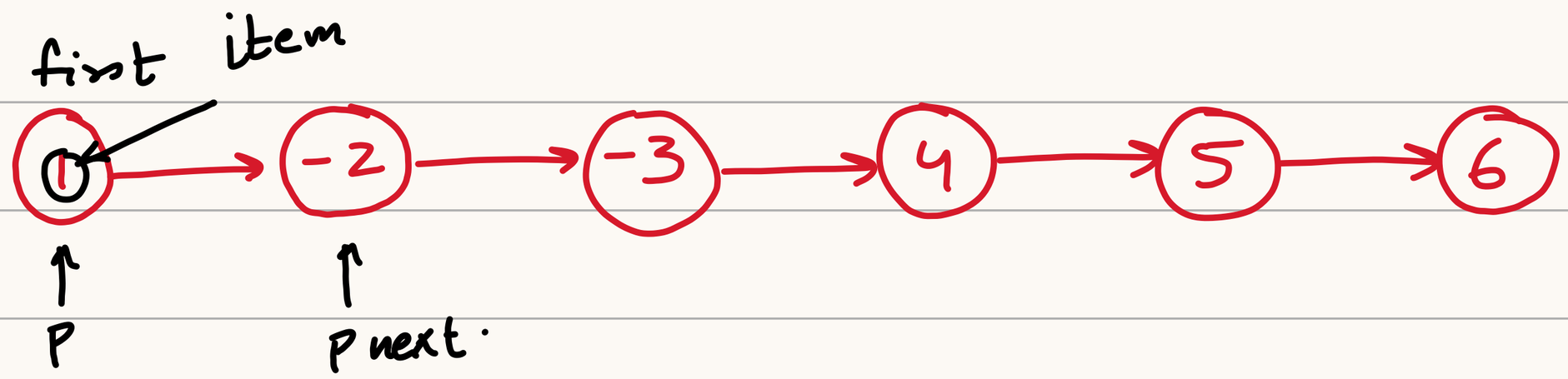



```
public int size () {
  int count = 0,
  Node present = list first ,
  While ( present ! = null ) {
      count ++,
      present = present next ,
  }
```

return count;

}

Q) Return Sum of Positive elements in LL

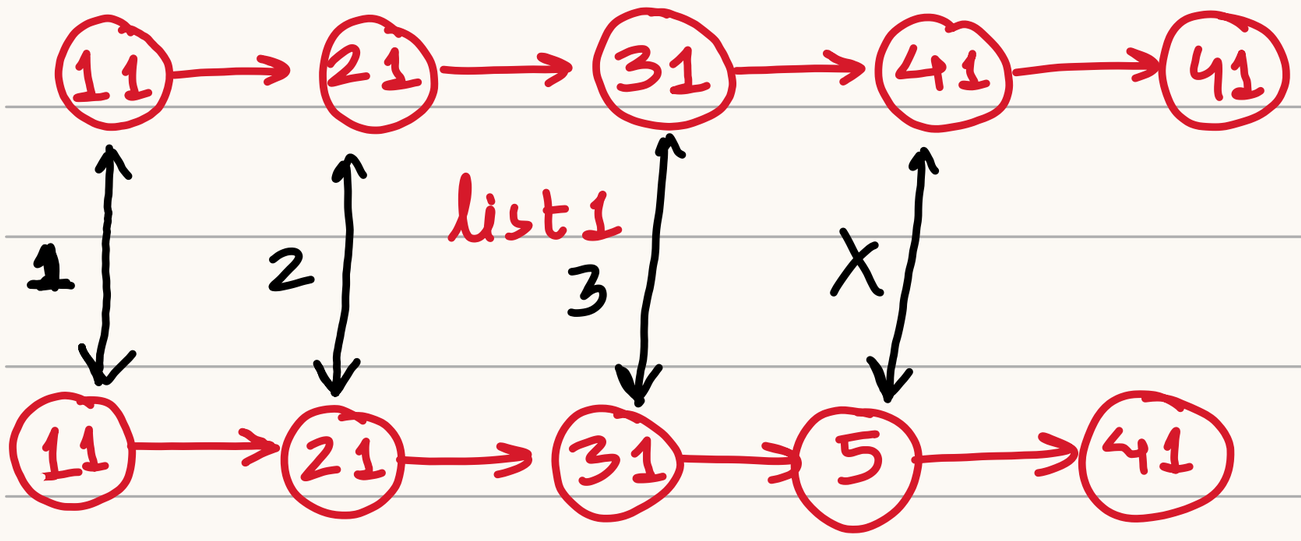



```
public double sumPositiveElement( ) {
double sum=0,
  Node present = first,
  while (present != null) {
      if ( present item > 0) {
        sum = sum + present item;
      }

      present= present next;
  }

  return sum;
}
```

Q) Return sum of negative elements in LL

```
public double sumNegativeElement() {
  double sum = 0;
  Node present = first;
  While (present != null) {
        if (present item < 0) {
            Sum = sum + present item;
          }
      present= present next;
  }
  return sum;
}
```

Q 4) Find the length of common Prefix for this & that

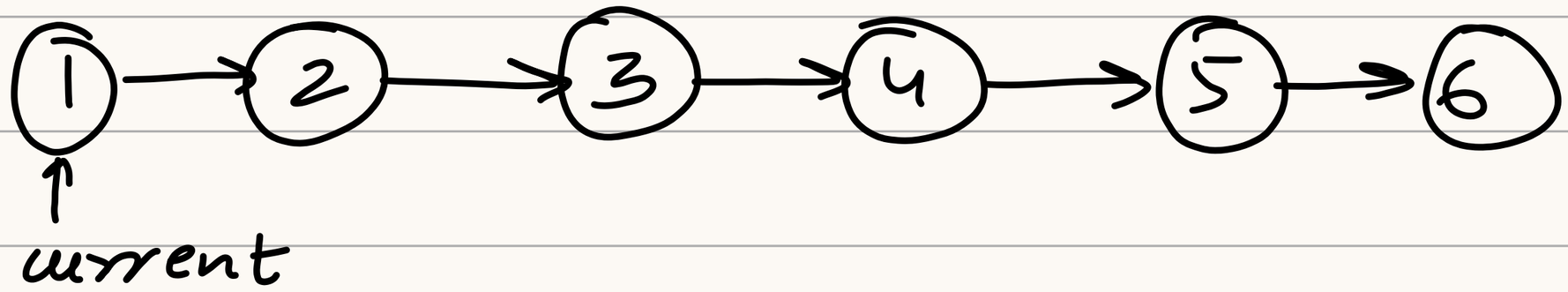


length of common prefix = 3

list 2

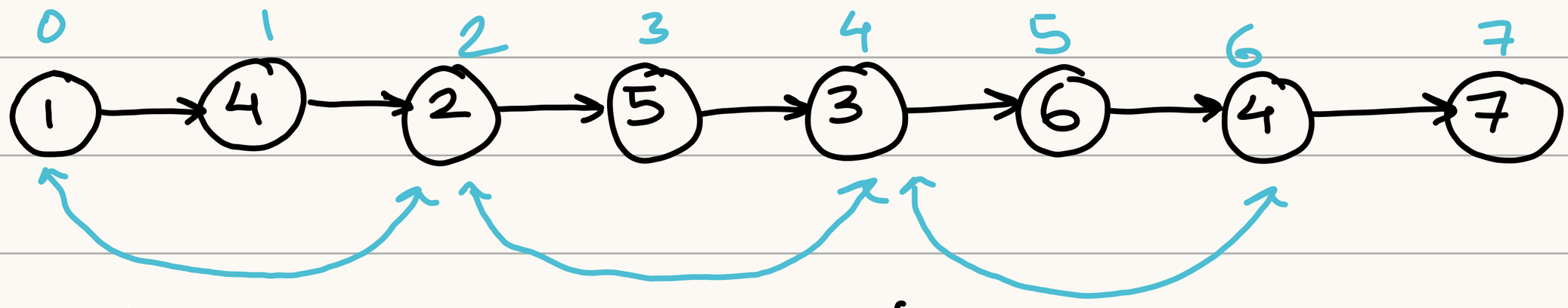
```
public int LengthofCommonPrefix (MyLinkedO that) {
  Node plist1      = this first;
  Node plist2      = that first,
  int Length = 0
  while (plist11 = null && plist21 = null && plist1 item ==
                                          plist2 item){

      length = length+1,
      plist1 = plist1 next;
      plist2 = plist2 next,
}
return length,
}
```

Q) Return true if the elements are strictly increasing




```
public boolean isIncreasing() {
  Node current = first,
  while (current != null && current.next != null) {
     if (current.item >= current.next.item) {
          return false,
        }
        current = current.next,
     }

  return true,
}
```

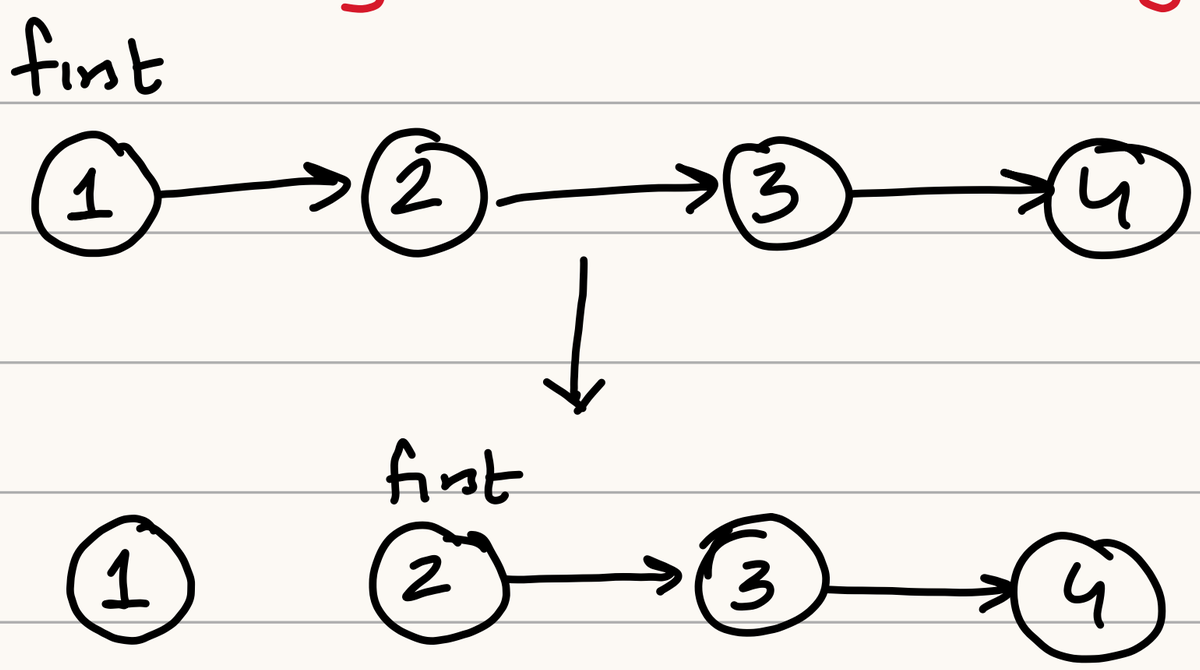
Q) Return true if even Indices are strictly increasing




```
public boolean evenIncrease (){
Node current= first
while ( current != null && current next != null &&
                                    current next next != null)
  {
   if ( current item >= current next next item) {
        return false,
    }
      current = current next.next;
}
return true,
```

Q) Delete the first element from the list, if it exists

Do nothing if list is empty

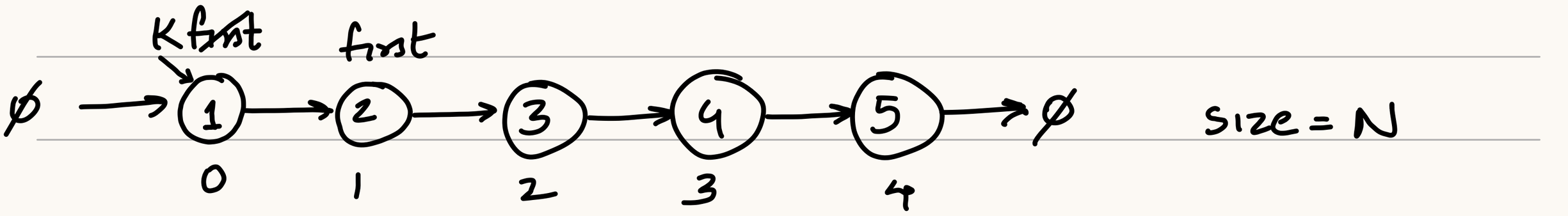



```
public void deletefirst() {
    if (first == null) {
      return,
    }
    first = first next
```

*MyLinked2

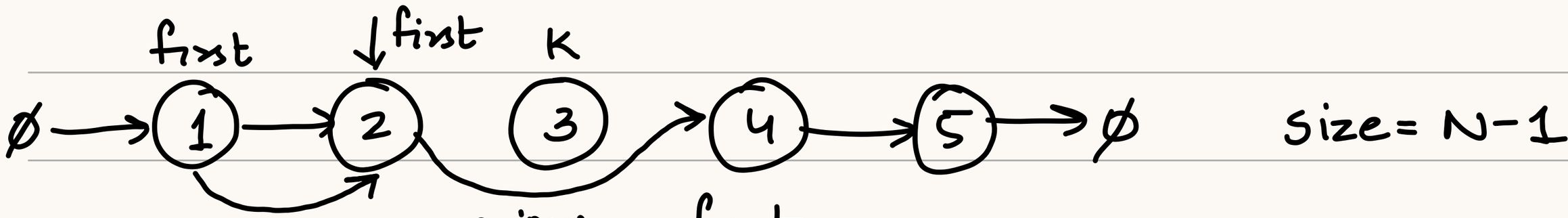
```
public class MyLinked2 {
    static class Node {
        public Node (double item, Node next) {
          this.item = item,
          this.next = next,
        }

        public double item,
        public Node next
}

int N;
Node first,
```

Q) Delete the K^{th} element (where K is between 0 and N-1 inclusive).



if K=0

first = first next



previous = first

previous next = previous.next next,

```
public void delete (int k) {
  if ( k<0 || k >= N) throw new IllelegalArgumentException
                                  ("Index out of
                                      bound");
  if (k==0){
     first = first next,
  }
else {
     Node previous = first,
     for (int i=0 , i < k-1, i++) {
        previous = previous next,
    }
    previous·next = previous next next,
    }
    N--;
}
```