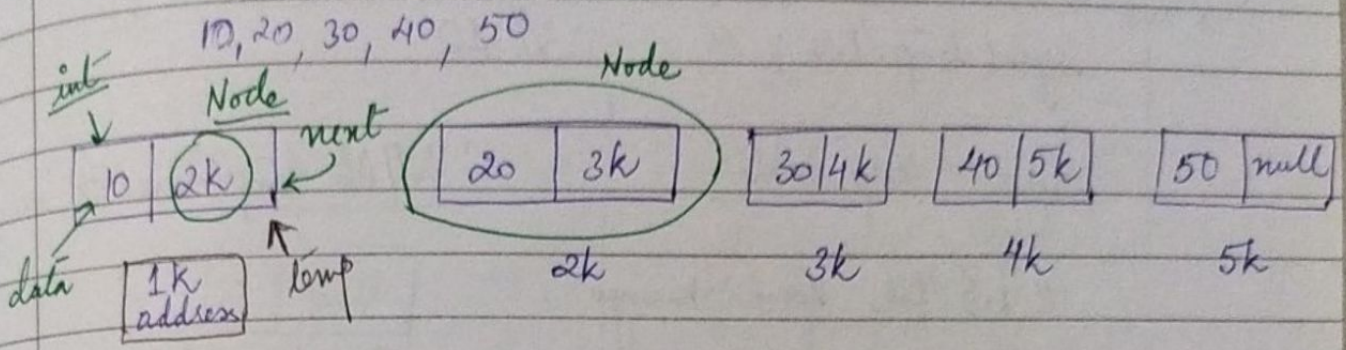
Linked List

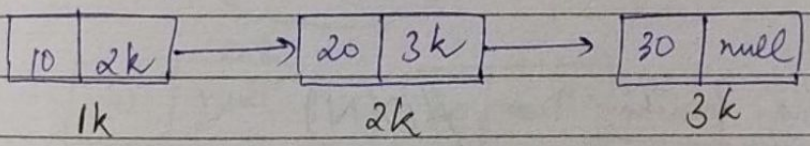
PAGE NO. icon
DATE : / /




```
Node {
    int data
    Node next;
}
```

temp.data = 10
temp.next = 2k
temp = 1k

First node jahan hai, uski value toh store karwani padegi



Node head = 1k

public class LinkedList {

```
    private class Node {
        int data;
        Node next;
    }
```

// inner class banana zaroori tha

} data members of Node class not LinkedList class

```
    private Node head;
    private Node tail;
    private int size;
}
```

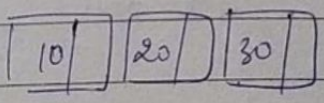
PAGE NO. icon
DATE : / /

# Node temp = head
syso(temp.data) // 10

temp = temp.next // Ab temp me 2k aa gya

// Kab tak kaam karenge

```
while (temp != null) {
    syso(temp.data);
    temp = temp.next;
}
```



Dry Run

10	20	30	while loop se bahar
temp = 2k	temp = 3k	temp = null	

// Display function // O(N)

```
public void display () {
    Node temp = this.head;
    while (temp != null) {
        syso(temp.data);
        temp = temp.next;
    }
}
```

Add last // O(1)

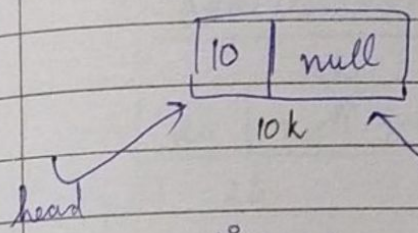
head = null
tail = null
size = 0
} summary object

PAGE NO. icon
DATE : / /

| 10 | null |

Node nn = new Node(); // new node creation
nn.data = 10;

First time node addⁿ



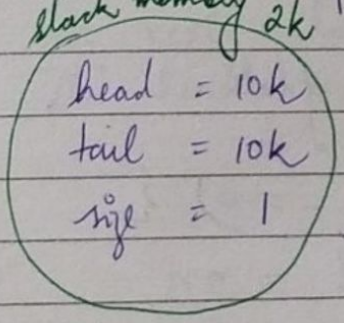
size++;

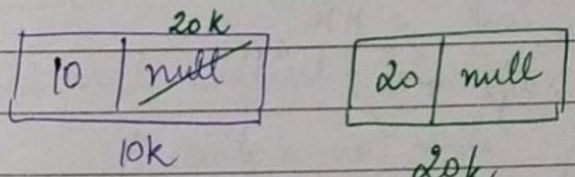
Update summary object

```
if (size == 0) {
    head = nn;
    tail = nn;
    size++;
}
```

Now,

Configⁿ





↳ addLast

if(size>=1){ tail.next = nn; → Now tail = 20k
tail = nn;
size++;

Add last fnⁿ

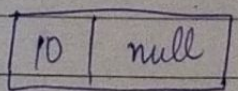
Code :-

```
public void addLast (int item) {

    // create a new node
    Node nn = new Node();
    nn.data = item;
    nn.next = null;

    // attach
    if (this.size >= 1) {
        this.tail.next = nn;

    // summary of update
    if (this.size == 0) {
        this.head = nn;
        this.tail = nn;
        this.size++;

    } else {
        this.tail = nn;
        this.size++;
    }

}
```

AddFirst // O(1)

head = null
tail = null
size = 0

1. Create a new node

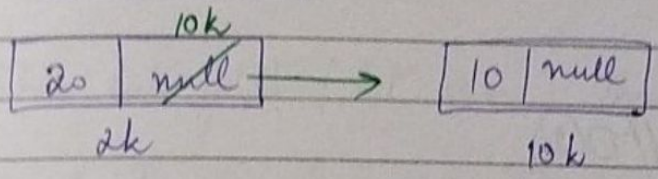


10k

2. attach (when size >= 1)

head = 10k
tail = 10k
size = 1

addFirst (20)




```
if (size >= 1) {
    nn.next = head;
    this.head = nn;
}
```

Code :

```
public void addFirst(int item) {
    // create a new node
    Node nn = new Node();
    nn.data = item;
    nn.next = null;

    // attach
    if (this.size >= 1) {
        nn.next = head;
    }
    // summary obj update
    if (this.size == 0) {
        this.head = nn;
        this.tail = nn;
    }
    this.size++;
}
```

```
        else {
            this.head = nn;
            this.size++;
        }
    }
```

GetFirst // O(1)

```
public int getFirst() throws Exception {
    if(this.size == 0) {
        throw new Exception("dl is empty");
    }
    return this.head.data;
}
```

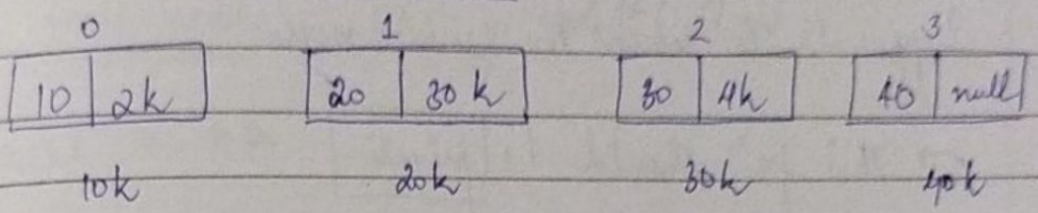
GetLast // O(1)

```
public int getLast() throws Exception {
    if(this.size == 0) {
        throw new Exception("dl is empty");
    }
    return this.tail.data;
}
```

PAGE NO. icon
DATE : / /

// Get At // O(N)

Given : index




```
public int getAt (int idx) throws Exception {
    if (this.size == 0) {
        throw new Exception ("LL is empty");
    }
    if (idx < 0 || idx >= this.size) {
        throw new Exception ("Invalid Index");
    }
    Node temp = this.head;
    for (int i=1; i<=idx; i++) {
        temp = temp.next;
    }
    return temp.data;
}
```

PAGE NO. icon
DATE : / /

GetNodeAt // O(N)

Supposed to return the entire node

→ this fn should be private

⇒ kyunki iss fn ye return kya kr rha hai → Node type

⇒ Toh client ke paas GetNodeAt krne ke baad pura node aa jayegi

⇒ Ab client class ke paas, Node ka address aa gya Toh woh usko manipulate kar sakta hai

⇒ Ye authority client class ke paas na ho, toh isliye iss fn ko private bana rhe hain

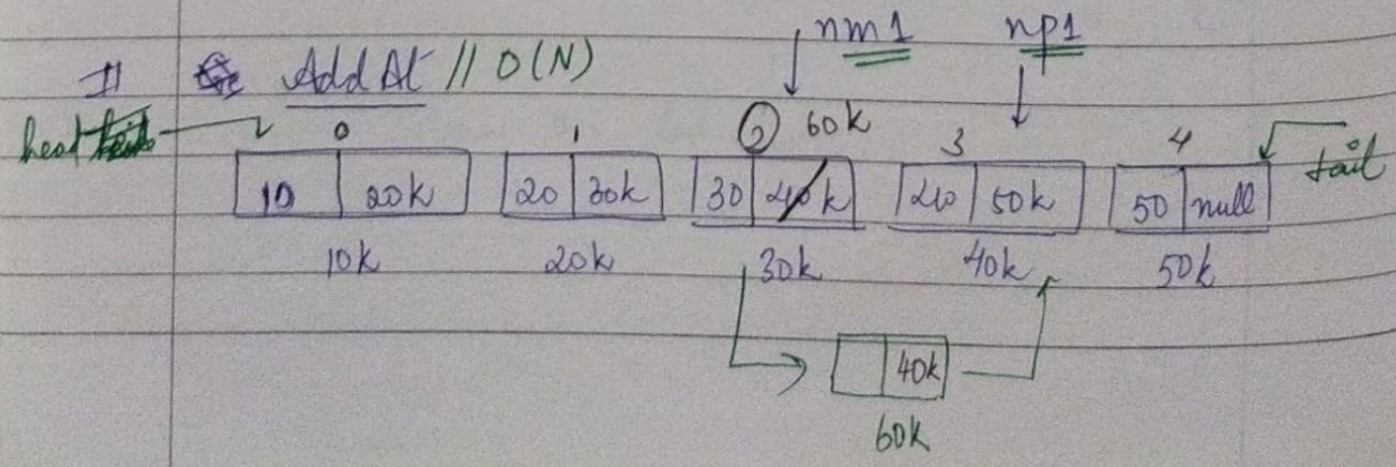
⇒ Last code me bas private Node GetNodeAt(int idx){

// Code

return temp;

}

// Add At // O(N)



PAGE NO. icon
DATE : / /

nm1 = getNodeAt(idx - 1); // 2 node

np1 = nm1.next OR getNodeAt(idx);

nm1.next = nn

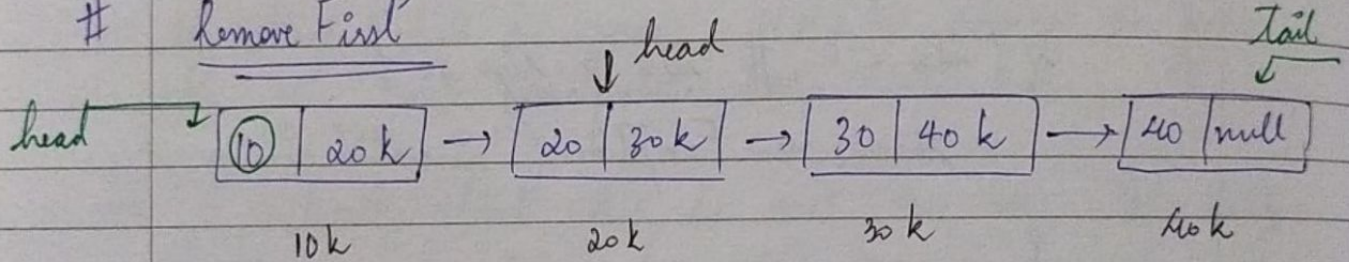
nn.next = np1

Code :-

```
public void addAt(int item, int idx)
    if(idx < 0 || idx > size) {
        throw new Exception("Invalid Index");
    }

    if(idx == 0) {
        addFirst(item);
    } else if(idx == this.size) {
        addLast(item);
    } else {

        // create a new node
        Node nn = new Node();
        nn.data = item;
        nn.next = null;
```

```
// attach
Node nm1 = getNodeAt (idx - 1);
Node np1 = nm1.next;

nm1.next = nn;
nn.next = np1;

// summary update
this.size++;
```

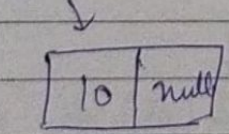
Remove First




```
int rv = head.data;
head = head.next;
```

→ head = 20k

```
size--;
```



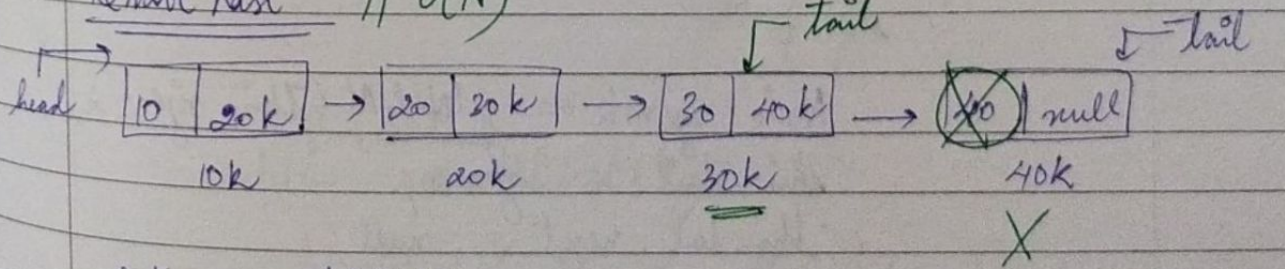
h = 10k
t = 10k
s = 1

```
rv = 10;
if(size == 1) {
    head = null;
    tail = null;
    size = 0
}
```

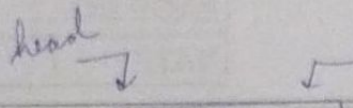
// O(1)

```
public int removeFirst() throws Exception {
    if (this.size == 0) {
        throw new Exception("LL is empty");
    }
    int rv = this.head.data;
    if (this.size == 1) {
        this.head = null;
        this.head = null;
        this.size = 0;
    } else {
        this.head = this.head.next
        this.size--;
    }
    return rv;
```

Remove last // O(N)




```
int rv = tail.data;
Node lm1 = getNodeAt(size-2)
tail = lm1;
tail.next = null;
size--;
```



head = 10k
tail = 10k
size = 1

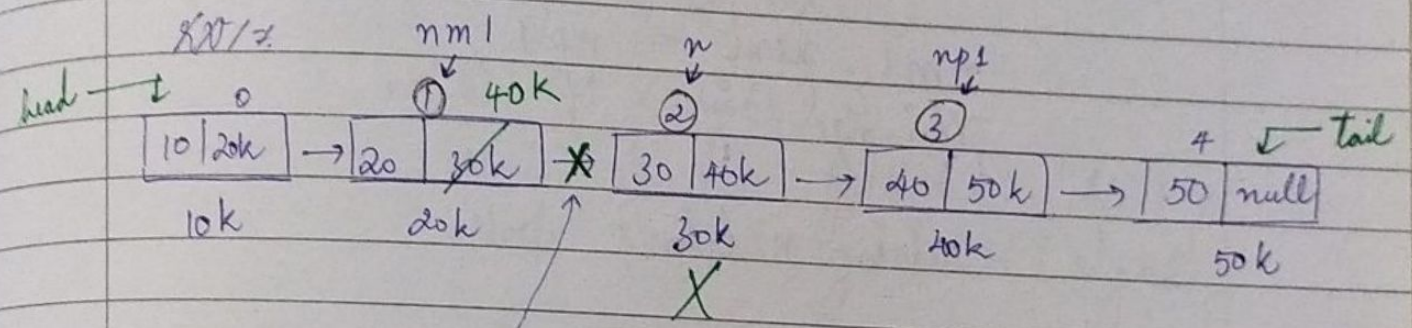
```
if(size == 1){
    head.next = null;
    tail.next = null;
    size = 0;
}
```

Code :

```
public int removeLast() throws Exception {
    if( this.size == 0 ) {
        throw new Exception ("dl is empty");
    }
    int rv = this.tail.data;
    if(this.size == 1) {
        this.head = null;
        this.tail = null;
        this.size = 0;
    } else {
        Node sizem2 = getNodeAt(this.size - 2);
        this.tail = sizem2;
        this.tail.next = null;
        size--;
    }
    return rv;
}
```

Remove At // O(N)

idx = 2




```
nm1.next = np1;
size--;
```

Code :-

```
public int removeAt(int idx) throws Exception {
    if (idx < 0 || idx >= this.size) {
        throw new Exception("Invalid Index");
    }
    if (size == 0) {
        throw new Exception("LL is empty");
    }
    if (idx == 0) {
return removeFirst();
    } else if (idx == this.size - 1) {
        return removeLast();
    } else {
```

nm1 → node minus 1
np1 → node plus 1

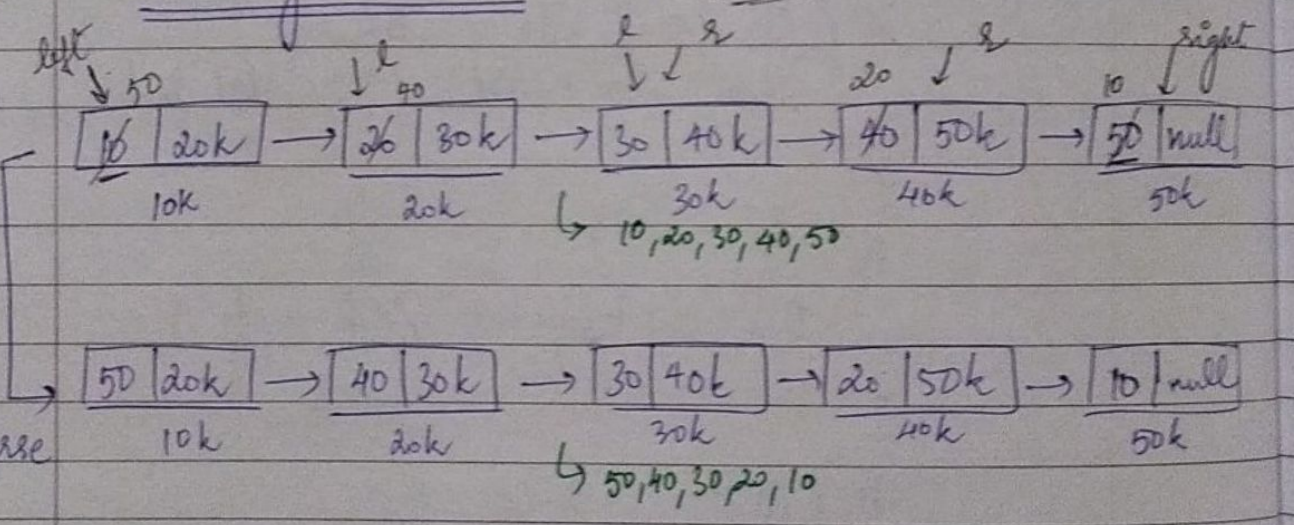
PAGE NO. icon
DATE : / /

```
Node nm1 = getNodeAt(idx - 1);
Node n = nm1.next;
Node np1 = n.next;

nm1.next = np1;
this.size--;

return n.data;

}
```

Reverse a LL

- reverse data
- reverse pointers

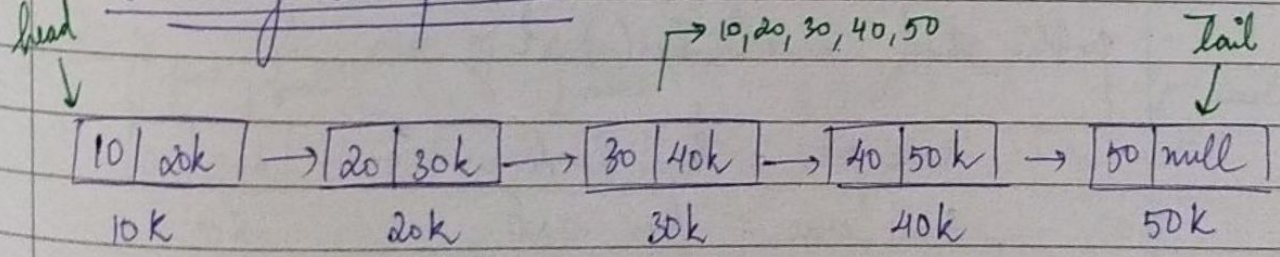
Reversing the data

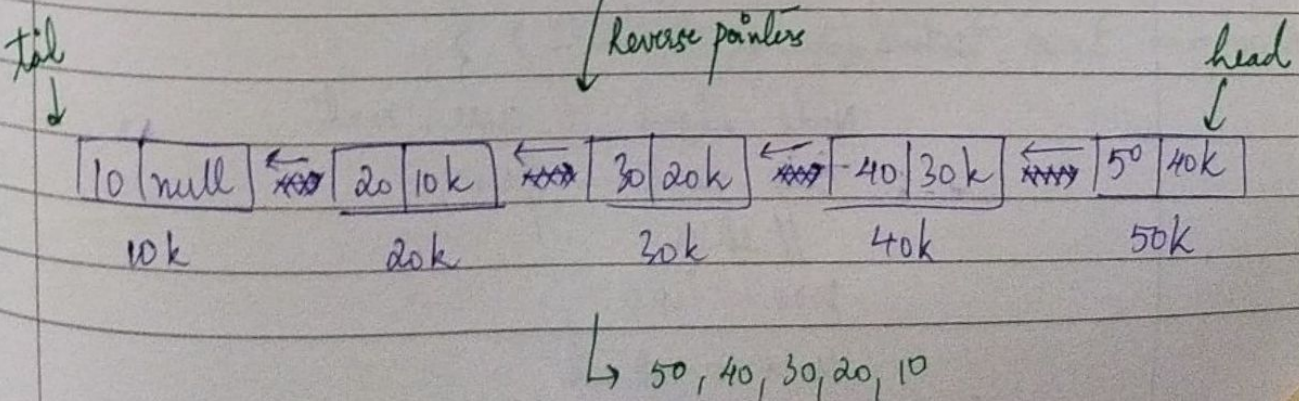


Code :

```
public void reverseData() throws Exception {
    int left = 0;
    int right = this.size - 1;
    while (left < right) {
        Node ln = getNodeAt(left);
        Node rn = getNodeAt(right);
        int temp = ln.data;
        ln.data = rn.data;
        rn.data = temp;
        left++;
        right--;
    }
}
```

Reversing the pointers



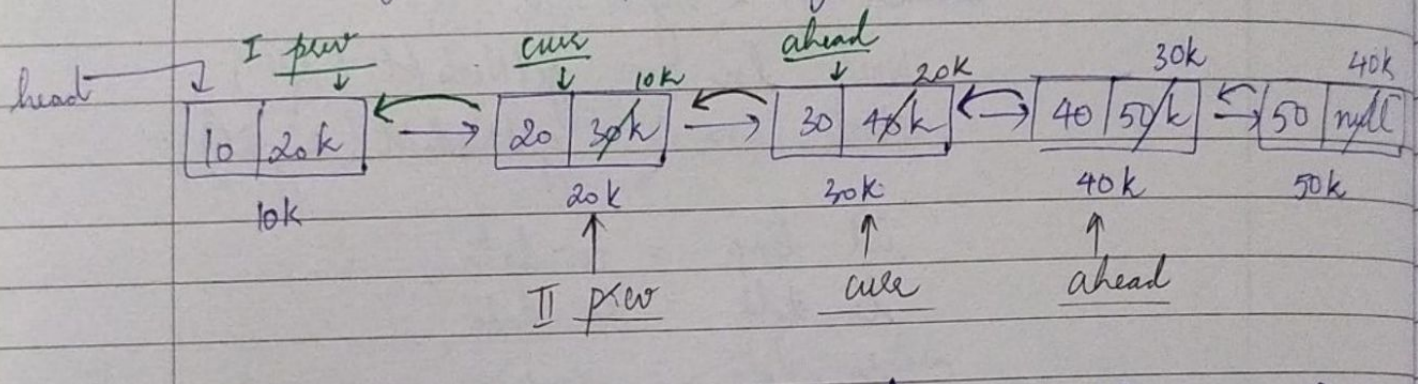


PAGE NO. icon
DATE : / /

We take 3 pointers :- prev, curr, ahead

① curr.next → prev

② Then shift all 3 pointers by 1 position



Code :-

```
public void reversePointers() {

    Node prev = this.head;
    Node curr = prev.next;

    while (curr != null) {
        Node ahead = curr.next;
        curr.next = prev;
        // shift
        prev = curr;
        curr = ahead;
    }
```

PAGE NO. icon
DATE : / /

```
//swap
Node t = this.head
this.head = this.tail;
this.tail = t;
this.tail.next = null;
```

head → 50k
tail → 10k

}

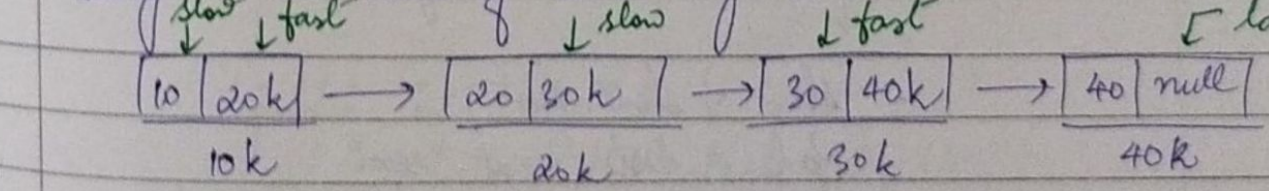
Mid point of linked list

* Constraints → can't use size variable
→ One traversal me mid point nikalna hai

Idea → 2 pointers lo
→ ek pointer ki speed x
→ dusre pointer ki speed 2x

(Take case of odd & even length LL)

For eg: In case of even length LL



Here, fast.next.next == null

Code :-

```
public int mid() {

    Node slow = this.head;
    Node fast = this.head;

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

        slow = slow.next
        fast = fast.next.next;

    }

}

// But in case, you need to return the second node
   of 2 (even case)

    Node slow = this.head;
    Node fast = this.head;

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

        slow = slow.next;
        fast = fast.next.next;

    }

}
```

PAGE NO. icon
DATE : / /

Size & IsEmpty fn^n

Size

```
public int size() {
    return this.size;
}
```

IsEmpty

```
public boolean isEmpty() {
    return this.size() == 0;
}
```

Linked List as Stack

(By writing above implementation of LL, we are now using it to create a stack)

```
public class LLAsStack {
    private LinkedList stack;

    public int size() {
        return this.stack.size();
    }

    public boolean isEmpty() {
        return this.stack.isEmpty();
    }
}
```

```
public void push (int item) {
    this.stack.addFirst(item);
}
public int pop() throws Exception {
    return this.stack.removeFirst();
}
public int top() throws Exception {
    return this.stack.getFirst();
}
public void display() {
    this.stack.display();
}
}

public class DLAsStackClient {
    main {
        DLAsStack stack = new DLAsStack();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();
    sysout(stack.pop());
        stack.top();
}}
```

PAGE NO. icon
DATE : / /

dll as Queue

Similar implementation as we did in Stack

fxn

```
① public int size() {
      return this.queue.size();
  }

② public boolean isEmpty() {
      return this.queue.isEmpty();
  }

③ public void enqueue(int item) {
      this.queue.addLast(item);
  }

④ public int dequeue() throws Exception {
      return this.queue.removeFirst();
  }

⑤ public int front() throws Exception {
      return this.queue.getFirst();
  }

⑥ public void display() {
      this.queue.display();
  }
```