

LINKED LIST

Linked list

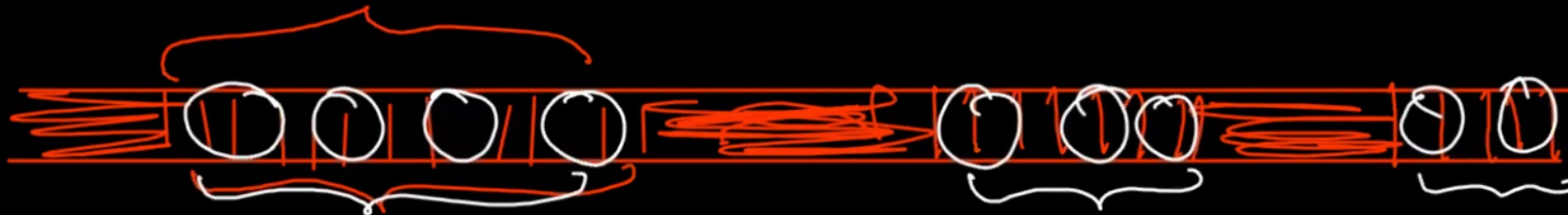
Data Structure

Time and Space

- ① Array + Array List
- ② Stack
- ③ Queue

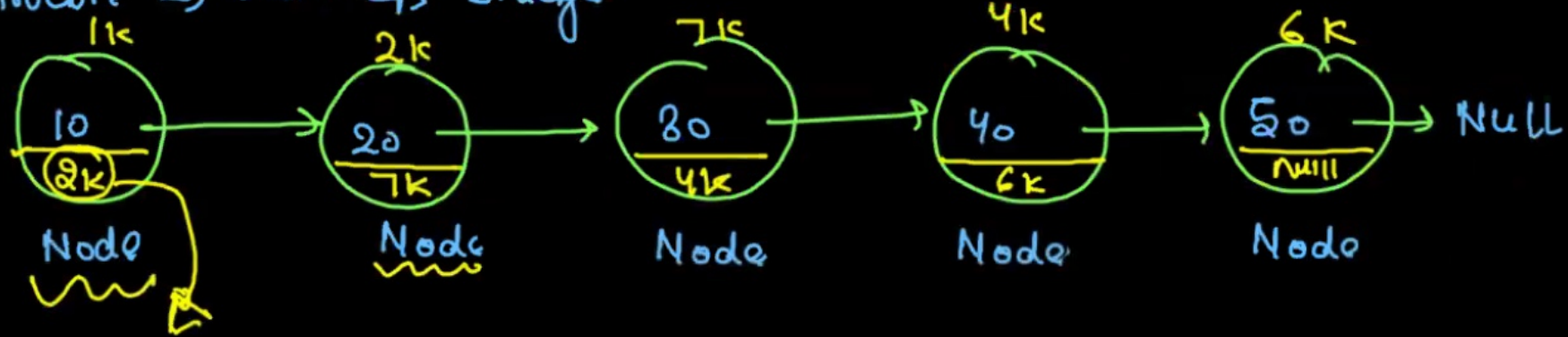


Continuous Memory.



Make use of available small memory rather than continuous memory

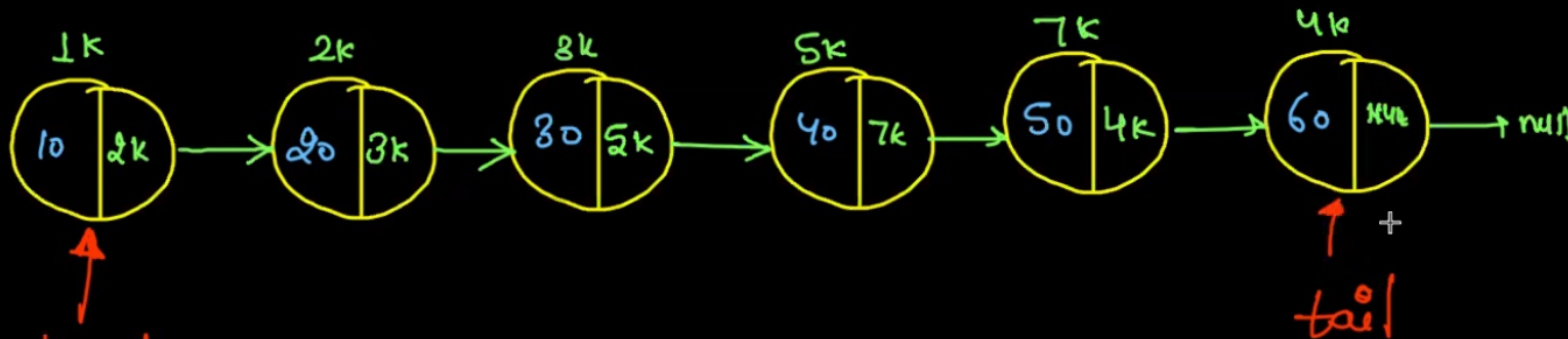
data-structure \rightarrow data \Rightarrow Integer



Reference (Address)
 \downarrow
 type of
 address
 \downarrow
 Node

Node \rightarrow { int data
 { Node next
 \uparrow
 user defined
 data type

Student (87) = new Student



Starting of Linked List

size of node?

expected from user

✓ add First
✓ add Last
✓ add At
✓ remove First
✓ remove Last
✓ remove At
✓ get First
✓ get Last
✓ get At

size
display

capacity constraint
Memory available

collection in java -> linkedlist is doubly linkedlist

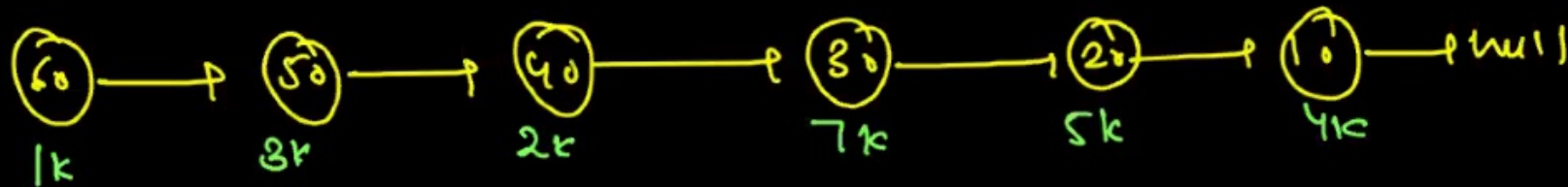
```
class linkedlist {  
  
    private Node head;  
    private Node tail;  
    private int size;  
  
    public linkedlist() {  
        this.head = this.tail = null;  
        this.size = 0;  
    }  
  
    private class Node {  
        private int data;  
        private Node next;  
  
        public Node() {  
            this.data = 0;  
            this.next = null;  
        }  
  
        public Node(int data) {  
            this.data = data;  
            this.next = null;  
        }  
  
        public Node(int data, Node next) {  
  
            this.data = data;  
            this.next = next;  
        }  
    }  
}
```

Reverse Linked List (Data Iterative)

× Recursion ✓
× Array ✓
× Hashmap ✗

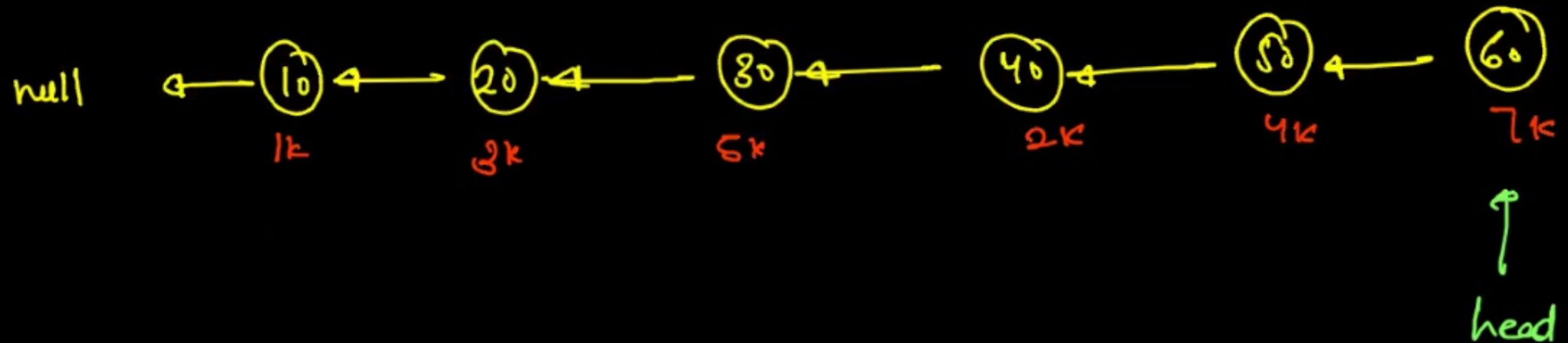
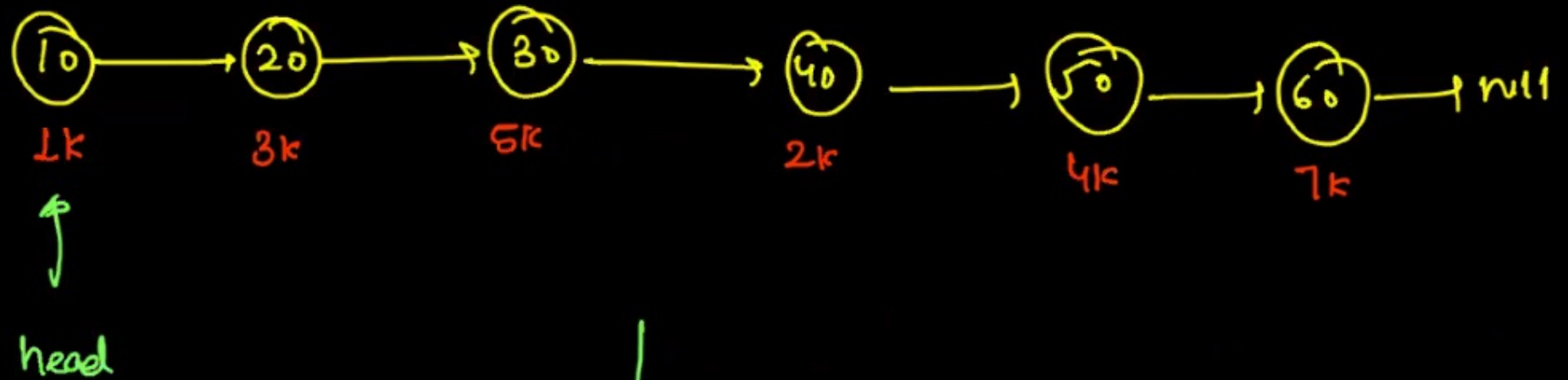


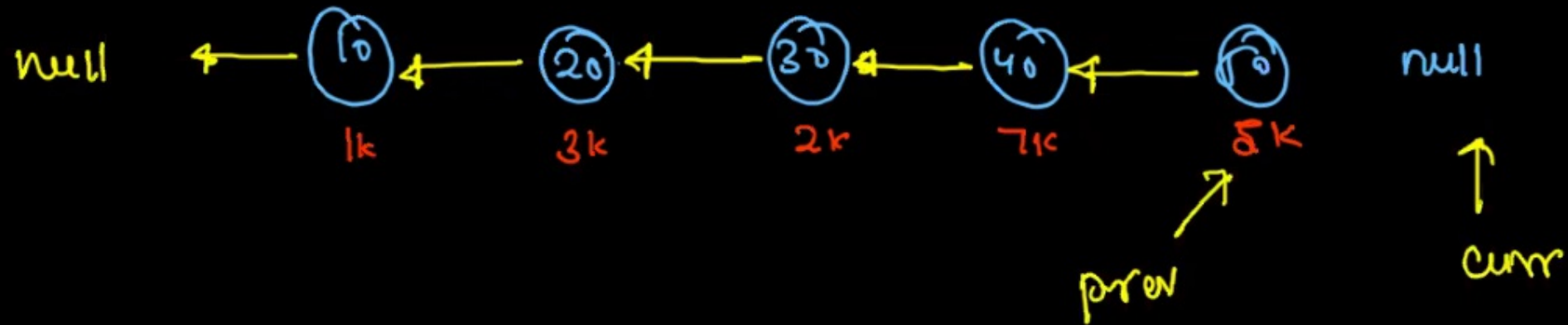
↓
* NOTE: pointer of linked list is not change.



notice : only data reverse and not reference reverse

Reverse Linked List (Pointer Iterative)





Node prev = null
Node curr = head

Loop: {
Node next = curr.next;
curr.next = prev;
prev = curr;
curr = next;
}

our custom list

Linked List →

add First → $O(1)$
✓ add last → $O(1)$
add At → $O(n)$
remove First → $O(1)$
remove Last → $O(n)$ → $O(1)$ FinCollection
remove At → $O(n)$
get First → $O(1)$
get Last → $O(1)$
get At → $O(n)$
size → $O(1)$

Stack (LIFO)

push

pop

peek

size

Queue (FIFO)

add

remove

peek

size

✓ Stack (LIFO) creation ✓ collection

push	→	add First	$O(1)$	add last
pop	→	remove First	$O(1)$	remove last
peek	→	get First	$O(1)$	get last
size	→	size		size

$$\text{Speed} = \frac{\text{distance}}{\text{time}}$$

$$\text{distance} = \text{speed} \times \text{time}$$

ptr 1

$\frac{x}{2}$

t

$$\underline{x t} = D/2$$

↓

Mid

ptr 2

$\underline{2x}$

t

$$\underline{2xt} = D$$

↓

End

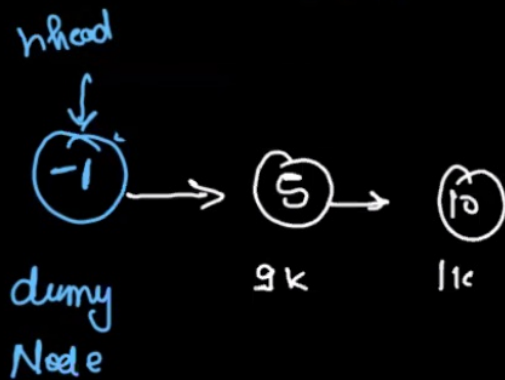
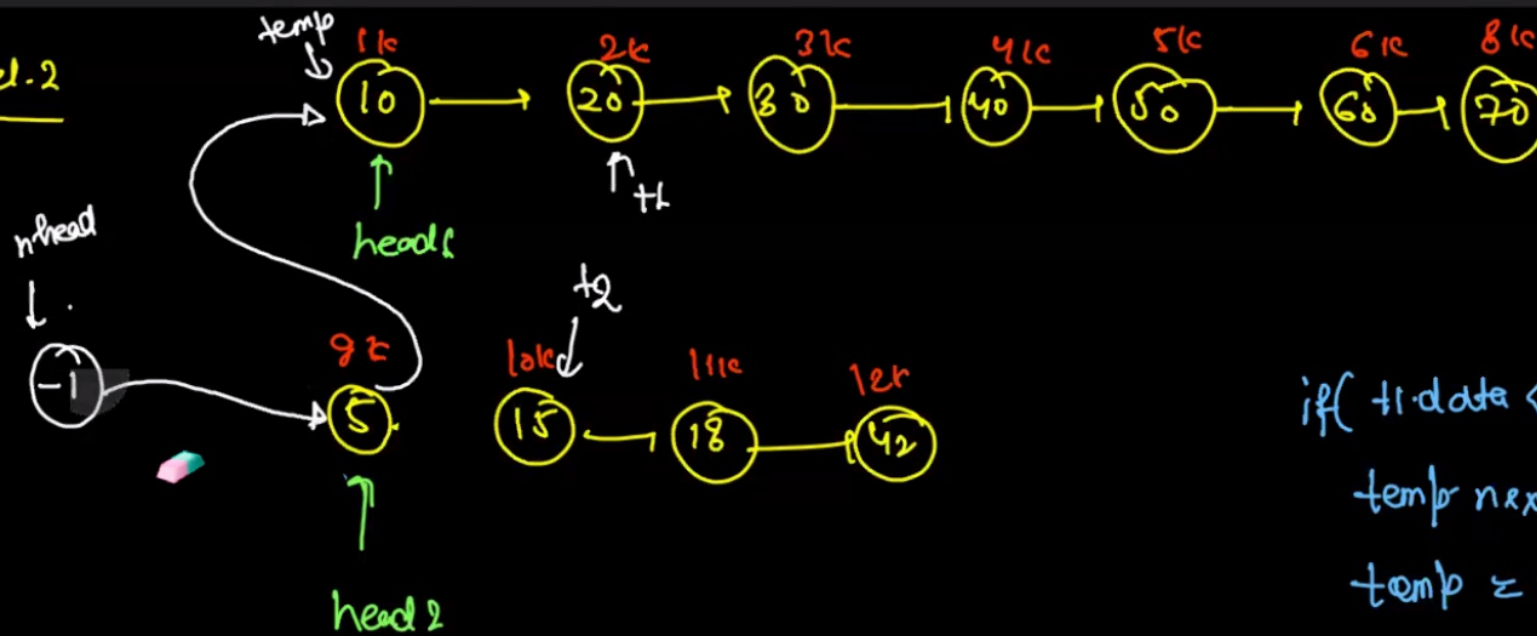
merge two link list with normal
approach

need to make addlast

```
public linkedlist mergeTwoSortedList1(linkedlist l1, linkedlist l2) {  
    // changing in given linkedlist is allowed.  
    linkedlist res = new linkedlist();  
    while(l1.size() > 0 && l2.size() > 0) {  
        if(l1.getFirst() < l2.getFirst()) {  
            res.addLast(l1.removeFirst());  
        } else {  
            res.addLast(l2.removeFirst());  
        }  
    }  
  
    // l1 left over  
    while(l1.size() > 0) {  
        res.addLast(l1.removeFirst());  
    }  
  
    // l2 left over  
    while(l2.size() > 0) {  
        res.addLast(l2.removeFirst());  
    }  
    return res;  
}
```

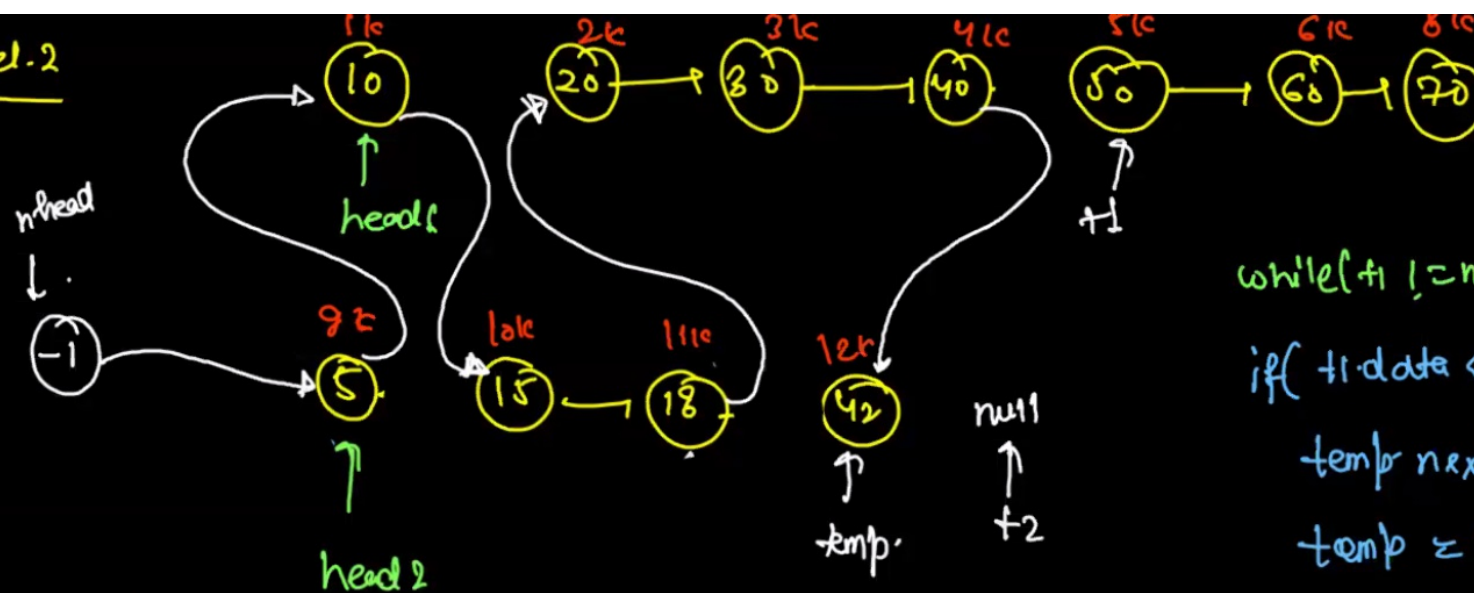
Merge Two Sorted Lists - LeetCode

Level.2



```
if (t1->data < t2->data) {  
    temp->next = t1;  
    temp = temp->next;  
    t1 = t1->next;  
} else {  
    temp->next = t2;  
    temp = temp->next;  
    t2 = t2->next;  
}
```

Level 2



```
while(t1 != null && t2 != null){
```

```
if(t1->data < t2->data){
```

```
temp->next = t1;
```

```
temp = temp->next;
```

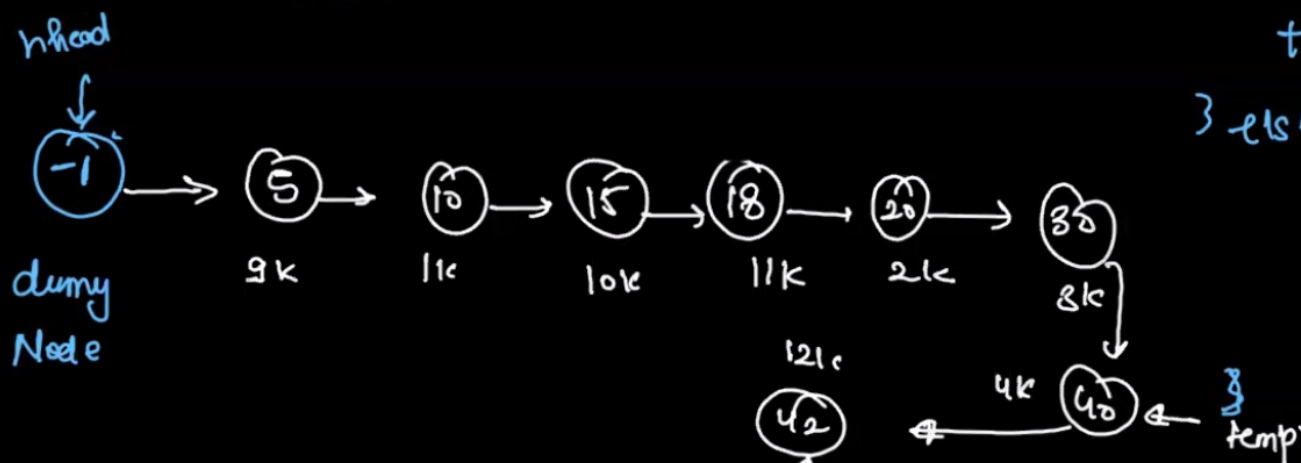
```
t1 = t1->next;
```

```
} else {
```

```
temp->next = t2;
```

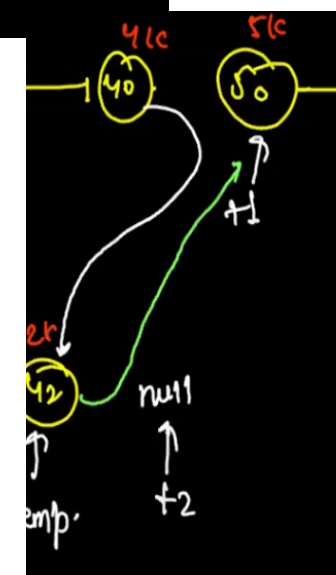
```
temp = temp->next;
```

```
t2 = t2->next;
```



when t2 null ho gaya he toh
temp.next = t1

when t1 == null
then temp.next = t2



```
public linkedlist mergeTwoSortedList2(linkedlist l1, linkedlist l2) {  
    // inplace change in original linkedlist  
    Node head1 = l1.head;  
    Node head2 = l2.head;  
  
    Node t1 = head1;  
    Node t2 = head2;  
  
    Node dummy = new Node(-1);  
    Node temp = dummy;  
  
    while(t1 != null && t2 != null) {  
        if(t1.data < t2.data) {  
            temp.next = t1;  
            temp = temp.next;  
            t1 = t1.next;  
        } else {  
            temp.next = t2;  
            temp = temp.next;  
            t2 = t2.next;  
        }  
    }  
}
```

```
if(t1 == null) {  
    temp.next = t2;  
} else {  
    temp.next = t1;  
}
```

Description

Solution

Discuss (999+)

Submissions

Java

Autocomplete

21. Merge Two Sorted Lists

Easy

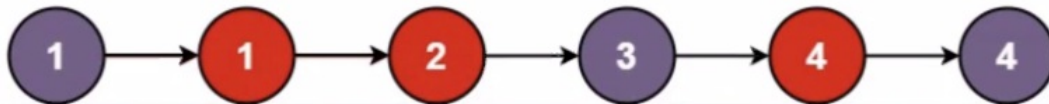
6786

777

Add to List

Share

Merge two sorted linked lists and return it as a **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Example 1:**Input:** l1 = [1,2,4], l2 = [1,3,4]**Output:** [1,1,2,3,4,4]**Example 2:**

l1 = [], l2 = []

[]

```
1  /**
2   * Definition for singly-linked list.
3   * public class ListNode {
4   *     int val;
5   *     ListNode next;
6   *     ListNode() {}
7   *     ListNode(int val) { this.val = val; }
8   *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
9   * }
10  */
11 class Solution {
12     public ListNode mergeTwoLists(ListNode head1, ListNode head2) {
13         ListNode t1 = head1;
14         ListNode t2 = head2;
15
16         ListNode dummy = new ListNode(-1);
17         ListNode temp = dummy;
18
19         while(t1 != null && t2 != null) {
20             if(t1.data < t2.data) {
21                 temp.next = t1;
22                 temp = temp.next;
23                 t1 = t1.next;
24             } else {
25                 temp.next = t2;
26                 temp = temp.next;
27                 t2 = t2.next;
28             }
29         }
30
31         if(t1 == null) {
32             temp.next = t2;
33         } else {
34             temp.next = t1;
35         }
36
37         return dummy.next;
38     }
39 }
```



```

1  /**
2   * Definition for singly-linked list.
3   * public class ListNode {
4   *     int val;
5   *     ListNode next;
6   *     ListNode() {}
7   *     ListNode(int val) { this.val = val; }
8   *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
9   * }
10 */
11 class Solution {
12     public ListNode mergeTwoLists(ListNode head1, ListNode head2) {
13         ListNode t1 = head1;
14         ListNode t2 = head2;
15
16         ListNode dummy = new ListNode(-1);
17         ListNode temp = dummy;
18
19         while(t1 != null && t2 != null) {
20             if(t1.val < t2.val) {
21                 temp.next = t1;
22                 temp = temp.next;
23                 t1 = t1.next;
24             } else {
25                 temp.next = t2;
26                 temp = temp.next;
27                 t2 = t2.next;
28             }
29         }
30
31         if(t1 == null) {
32             temp.next = t2;
33         } else {
34             temp.next = t1;
35         }
36
37         return dummy.next;
38     }
39 }

```