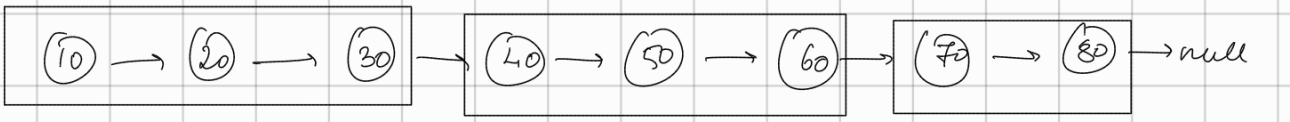
① K reverse linked list.



Add k groups in another list in reverse order by removing from front & adding at last in new list.
Then append these new lists & if size < k, add it directly

② Middle of LL

```
Node fast = head, slow = head;
while (fast != null && fast.next != null){
    slow = slow.next;
    fast = fast.next.next;
}
return slow
```

③ get Nth node from end of linked list

- In 2 traversals:

```
int size=0;
Node curr= head;
while (curr != null){
    curr= curr.next;
    size++;
}
if(n<0 || n>size) return -1;
curr= head;
while (pos-- >0) curr= curr.next;
return curr.data;
```

- In single traversal.

```
Node slow = head, fast = head;
while(n-- > 0) {
    if(fast == null) return -1;
    fast = fast.next;
}
while (fast != null){
    slow = slow.next;
    fast = fast.next;
}
return slow.data;
}
```

④ Merge 2 Sorted lists

```
if( head1 == null) return head2;
if(head2 == null) return head1;
Node dummy = new Node(-1), tail = dummy;
while(head1 != null && head2 != null) {
    if( head1.data < head2.data) {
        tail.next = head1;
        head1 = head1.next;
    } else {
        tail.next = head2;
        head2 = head2.next;
    }
    tail = tail.next;
}
if( head1 != null) tail.next = head1;
if (head2 != null) tail.next = head2;
return dummy.next;
```

⑤ Merge K sorted linked lists.

ⓐ Brute Force

```
public int minNode (ListNode[] lists) {
    int min = Integer.MAX_VALUE , index = -1;

    for (int i=0; i< lists.length; i++) {
        if lists[i] != null && lists[i].val < min {
            min = list[i].val;
            index = i;
        }
    }
    return index;
}

public ListNode mergeKlists (ListNode[] lists) {
    if ( lists.length == 0) return null;
    ListNode dummy = new Node (-1), tail = dummy;

    while (true) {
        int min = minNode (lists);
        if (min == -1) break;

        tail.next = lists[min];
        list[min] = lists[min].next;
        tail = tail.next;
    }
    return dummy.next;
}
```

O(nk^2) <= Tc

⑥ Optimized:

merge using divide & conquer approach recursively
Similar to merge sort

⇒ TC = $O(n * \log k)$

⑥ Sorting LL.

Break LL from middle & use merge 2 sorted lists recursively – Similar to merge sort.

Steps: ① find middle & break in 2 Lists
② merge sort (left), merge sort (right).
③ merge left & right and return

✱ Why merge sort is preferred for linked list and quick sort for arrays?

✱ ∵ merge sort on LL → T(N) → $O(N \log N)$
→ Space → In place (✱)
Only recursion call stack logN

✱ quick sort on array → T(N) → $O(N \log N)$ ⇐ avg case
→ Space → $O(1)$

⑦ Remove duplicates.

Approach: for each group of elements, have a start & end pointer and modify them

1 1 1 2 2 2 3 4 5

start ↑ end of 1 group ↑

→ start.next = end.

CODE:

```
ListNode temp = head;
while( temp != null){
    ListNode start= temp, end= temp.next;
    while (end != null && end.val == start.val)
        end= end.next;
    start.next = end;
    temp= end;
}
return head
```

⑧ Remove all Duplicates, similar approach to above but only add in answer list if freq == 1

CODE:

```
ListNode dummy = new ListNode(-1), temp= head, tail = dummy;
while (temp != null){
    int freq=1;
    while (temp.next != null && temp.next.val == temp.val){
        temp= temp.next;
        freq++;
    }
    if freq==1) {
        tail.next= new ListNode (temp.val);
        tail = tail.next;
    }
    temp= temp.next;
}
return dummy.next;
}
```

⑨ Big Integer

(a) Add Linked List

Same as addition done in arrays only changes in links

CODE:

```
int carry = 0;
ListNode dummy = new ListNode(-1), tail = dummy;
while (l1 != null && l2 != null) {
    int currAns = l1.val + l2.val + carry;
    ListNode curr = new ListNode(currAns % 10);
    carry = currAns / 10;
    tail.next = curr;
    tail = curr;
    l1 = l1.next;
    l2 = l2.next;
}
while (l1 != null) {
    int currAns = l1.val + carry;
    ListNode curr = new ListNode(currAns % 10);
    carry = currAns / 10;
    tail.next = curr;
    tail = curr;
    l1 = l1.next;
}
while (l2 != null) {
    int currAns = l2.val + carry;
    ListNode curr = new ListNode(currAns % 10);
    carry = currAns / 10;
    tail.next = curr;
    tail = curr;
    l2 = l2.next;
```

```
}
if(carry != 0)
    tail.next = new ListNode(carry);
return head;
```

(2) Subtract larger number from smaller
Similar to subtracting 2 arrays done in functions & arrays
CODE in next class