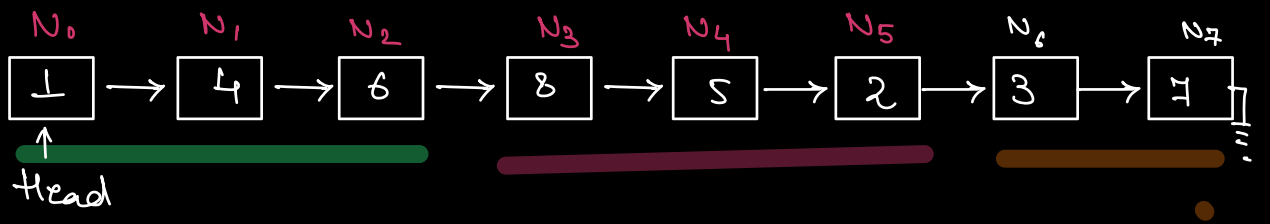
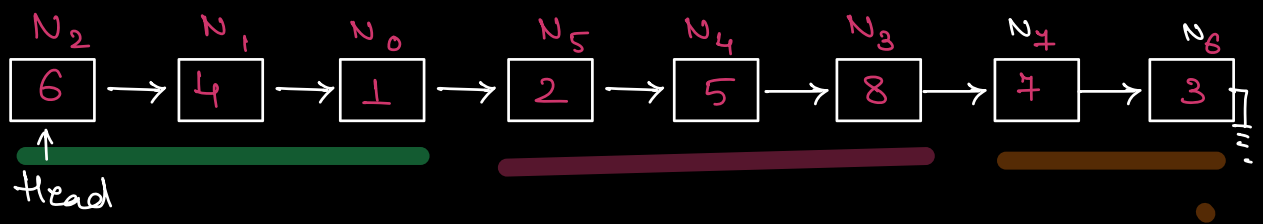
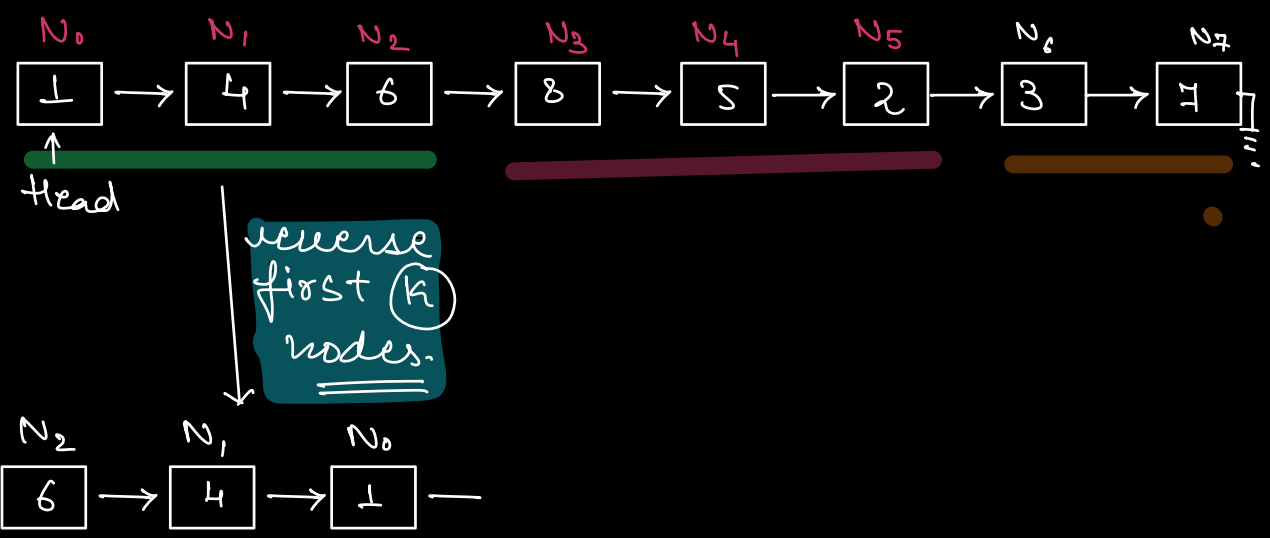
Q. Reverse L.L in group of k nodes.

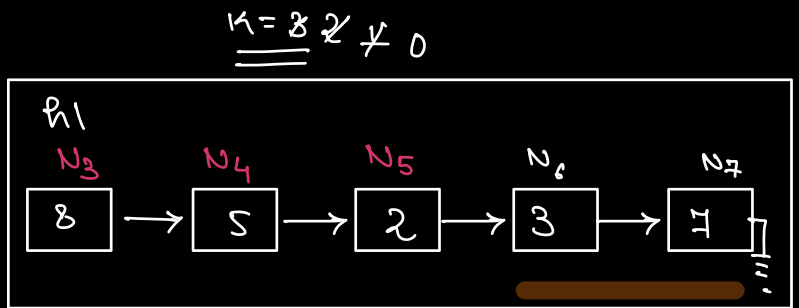
Google: Reverse k Groups.



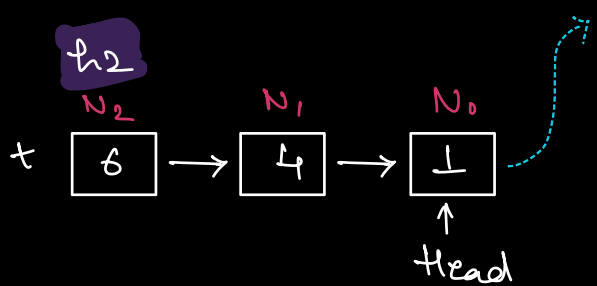




```
Node reverseInKGroups(head, k){
  // Assumption: reverseInKGroups(node) will
  // reverse the L.L with head as node in
  // group of k nodes & returns the new head;
  if(k<=1 || head == Null) return head;
  Count = k
  h1 = head, t = h1, h2 = Null;
  while(k>0 && h1 != Null){
      temp = h1;
      h1 = h1.next;
      t.next = h2;
      h2 = temp;
      k--;
  }
  head.next = reverseInKGroups(h1, Count);
  return h2;
}
```



Subproblem.

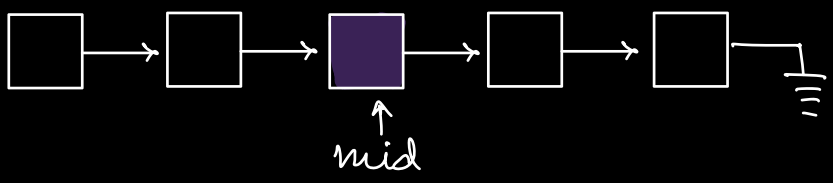


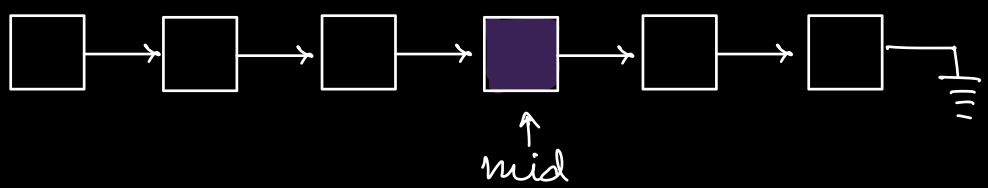
TC: O(N)

SC: O(N/K)

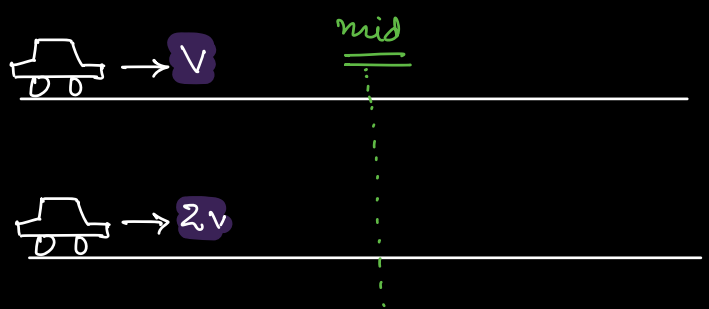
K = 2 ⇒ WC: O(N)

Q.1 Given a L.L, find the middle node.

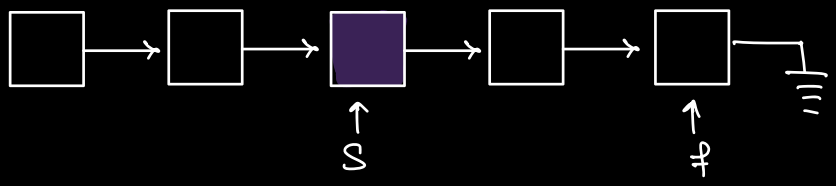


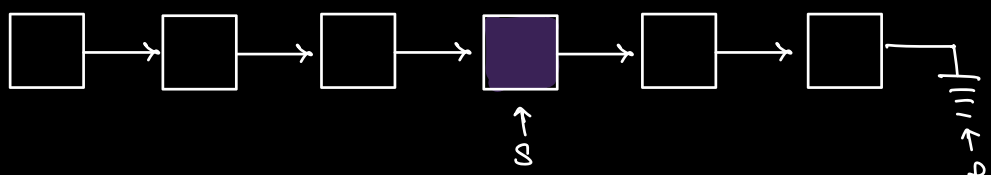


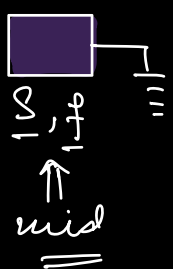
!= Null { Slow, fast, fast.next }




```
Node getMid(head){
    Slow = head;
    fast = head;
    while(fast != Null && fast.next != Null){
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}
```

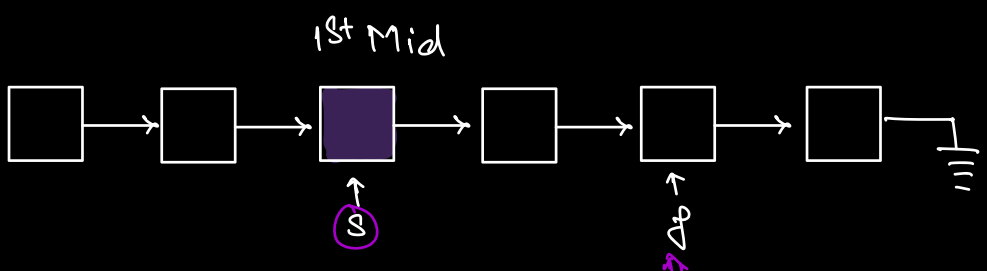


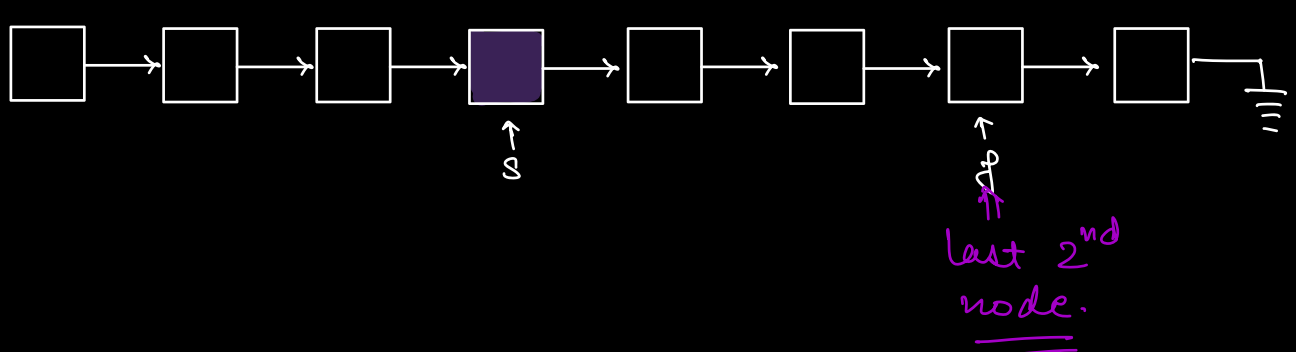




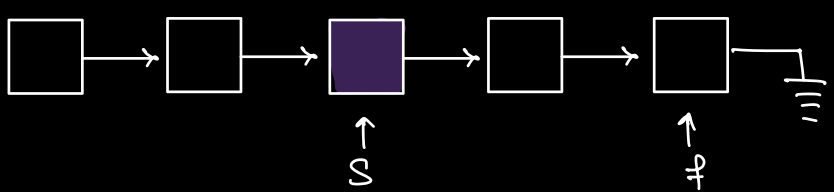
TC: O(N)

Q: If N is even, return 1st mid





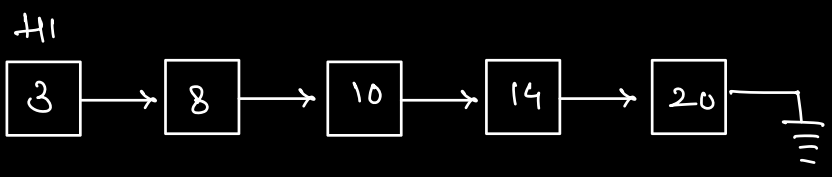
```
Node getFirstMid(head){
    if(head == Null)
        return null;
    slow = head;
    fast = head;
    while(fast != Null && fast.next != Null
          && fast.next.next != Null){
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}
```

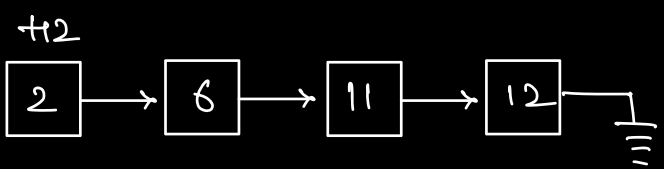


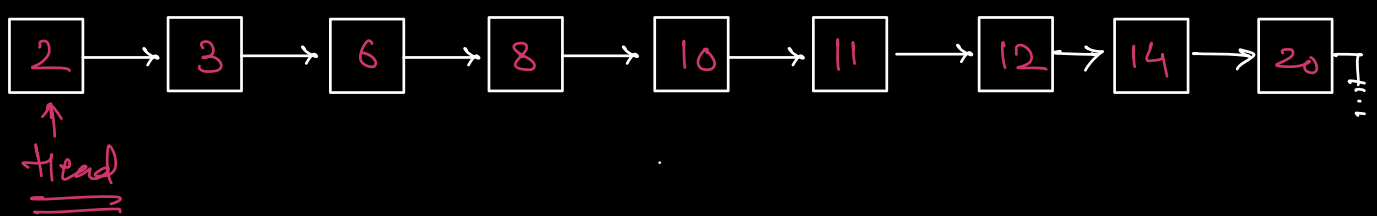
Q. Given 2 sorted L.L, Do in-place merging of these L.L to create a new sorted L.L.

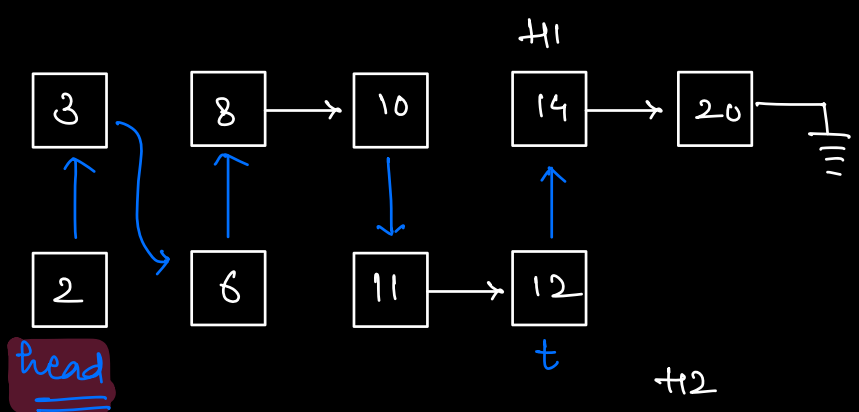
Amazon

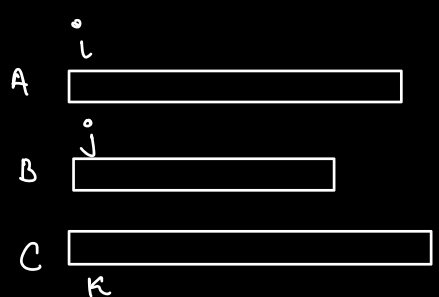
SC : O(1)



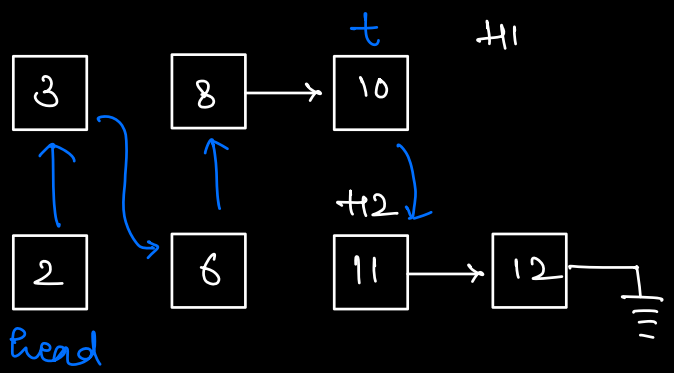








```
Node merge(h1, h2){                          → h1==Null
if(h1.data < h2.data){                         h2==Null
        head = h1;
        h1 = h1.next;
}
else{
        head = h2;
        h2 = h2.next;
}
t = head;
while(h1 != Null && h2 != Null){
        if(h1.data < h2.data){
                t.next = h1;
                h1 = h1.next;
        }
        else{
                t.next = h2;
                h2 = h2.next;
        }
        t = t.next;
}
if(h1 == Null)
        t.next = h2;
else{
        t.next = h1;
}
return head;
}
```

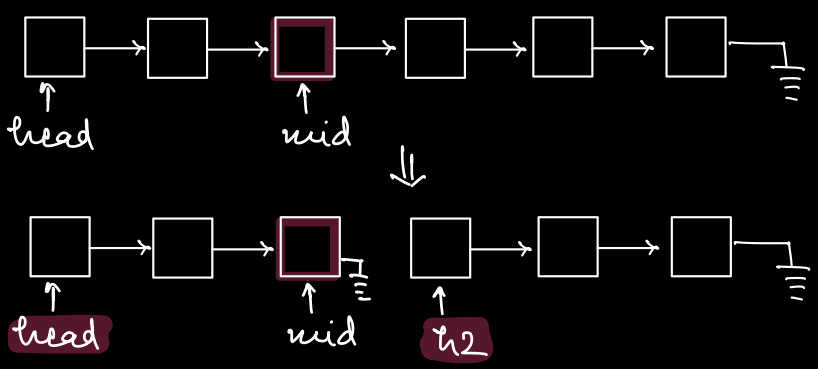


TC: $O(N+M)$ | SC: $O(1)$

N = length of 1st L.L, M = length of 2nd L.L

Q: Given a L.L, sort it using **Merge Sort**.

```
mergesort (data) {
    s1 ← mergeSort (firstHalf)
    s2 ← mergeSort (secondHalf)
    merge (s1, s2);
}
```



```
Node mergeSort(head){
    if(head == NULL || head.next == NULL)
        return head;
    Node mid = getLstMid(head);
    Node h2 = mid.next;
    mid.next = NULL;
    head = mergeSort(head);
    h2 = mergeSort(h2);
    return merge(head, h2);
}
```

getLstMid(head); → O(N): TC

merge(head, h2); → O(N): TC

$$T(N) = 2T(N/2) + O(N)$$

TC: $O(N \log N)$

SC: $O(\log N)$ → Recursion Call Stack.

New Array.

SC: $N + \log N$

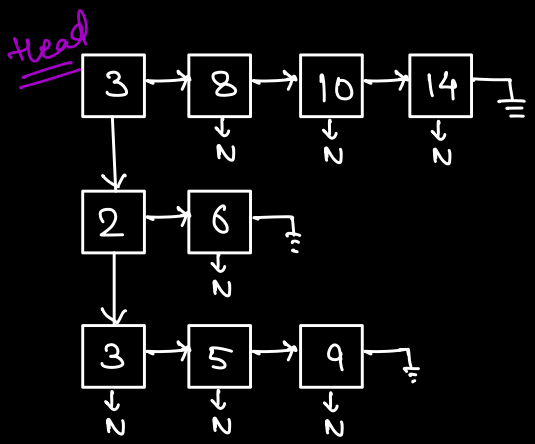
Merge sort in Arrays.

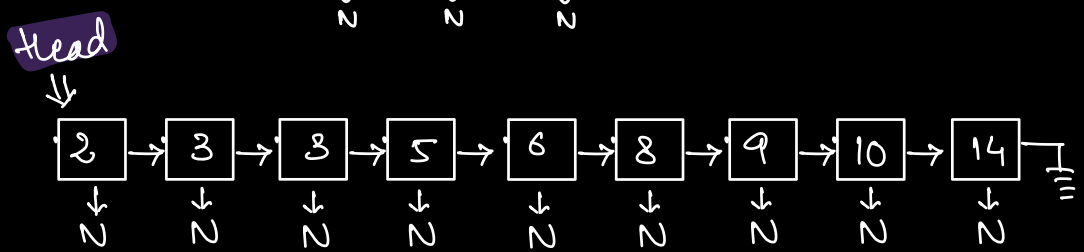
Q. Given a 2D list, flatten this L.L.

Google.

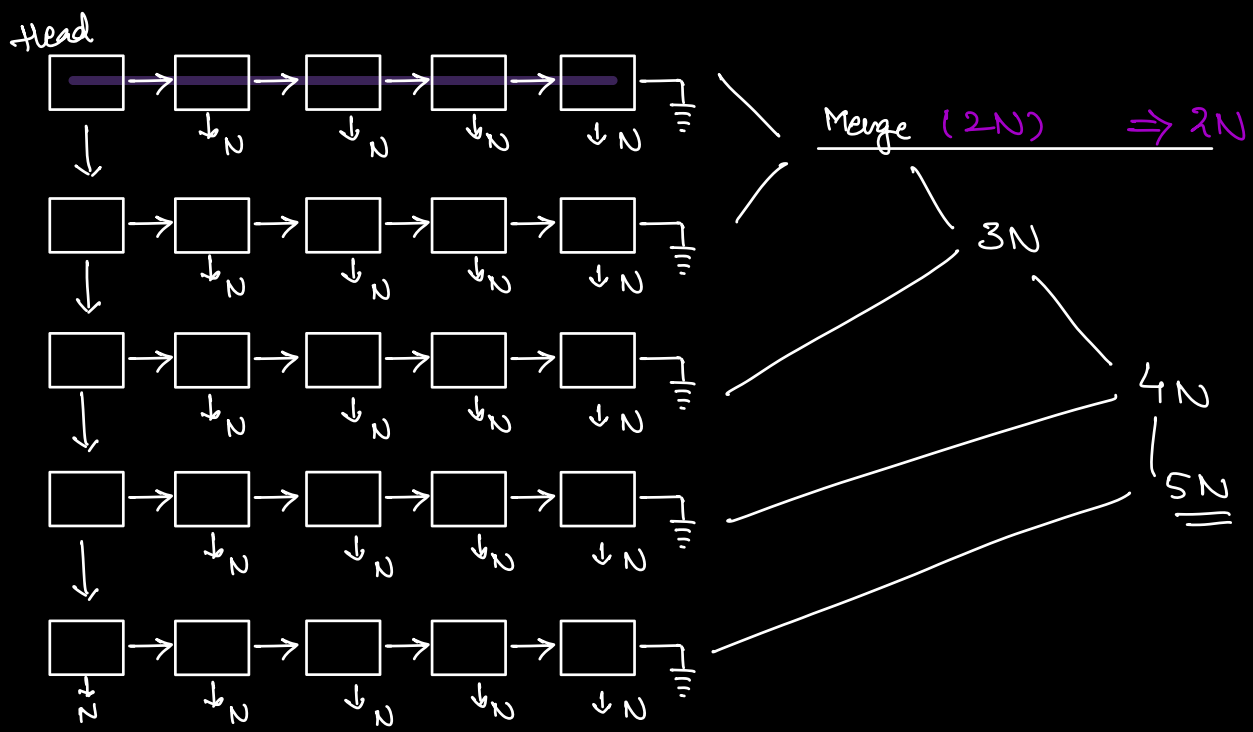
Sorted Horizontally.

N×N






```
Class Node {
    int data;
    Node next;
    Node down;
    Node(int d) {
        data = d
        next = NULL;
        down = NULL;
    }
}
```



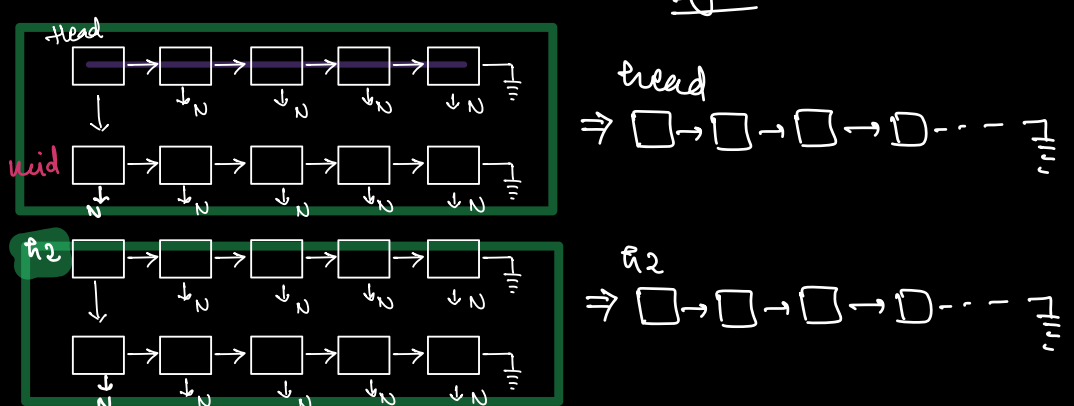
# of iterations = $2N + 3N + 4N + \cdots + N \times N$

$= N(2 + 3 + 4 + \cdots N)$

$\Rightarrow O(N^3)$

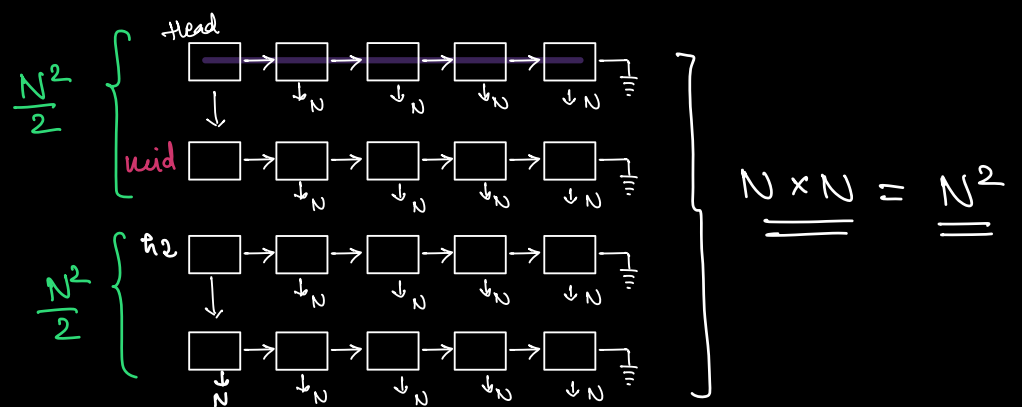
$\Rightarrow a^n = a \times a^{n-1} \Rightarrow N$

$a^n = a^{n/2} \times a^{n/2} \Rightarrow \log N$



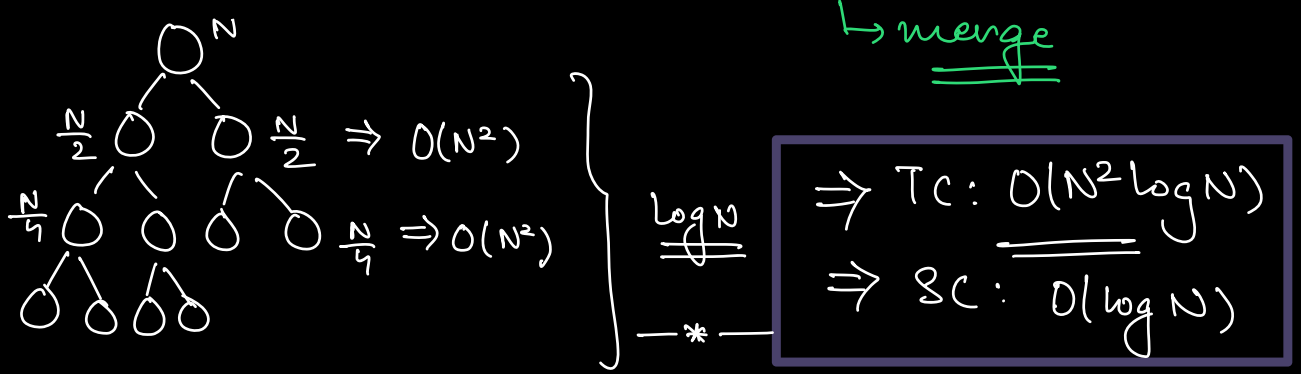
```
Node flattenLL(head) {
    if(head == NULL || head.down == NULL) {
        return head;
    }
    mid = getMid(head) // using down reference
    h2 = mid.down;
    mid.down = NULL;
    head = flattenLL(head);
    h2 = flattenLL(h2);
    return merge(head, h2);
}
```

merge → $O(N^2)$: TC



$$T(N) = 2T(N/2) + O(N^2)$$

↳ merge



#

$$T(N) = 2T(N/2) + x \rightarrow N^2$$

$$T(N) = 2\left(2T(N/4) + \frac{x}{2}\right) + x$$

$$= 4T(N/4) + 2x$$

$$= 8T(N/8) + 3x$$

$$= 4T(N/16) + 4x$$

$$= 2^k T(N/2^k) + kx$$

$K = \log N$

$$= N \cdot 1 + \log N * x$$

$$= N^2 \log N$$