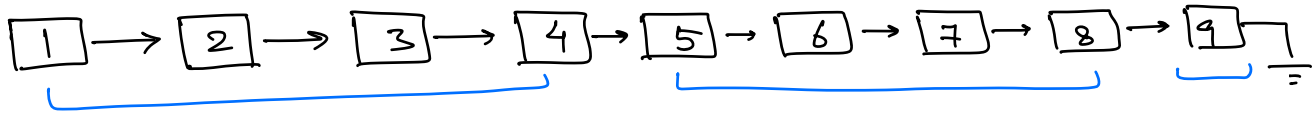
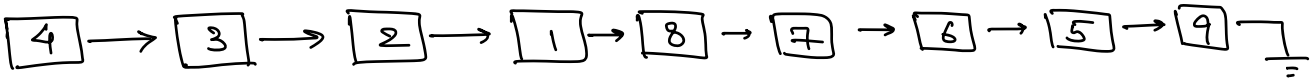
Q:1 Google

Reverse in K groups.

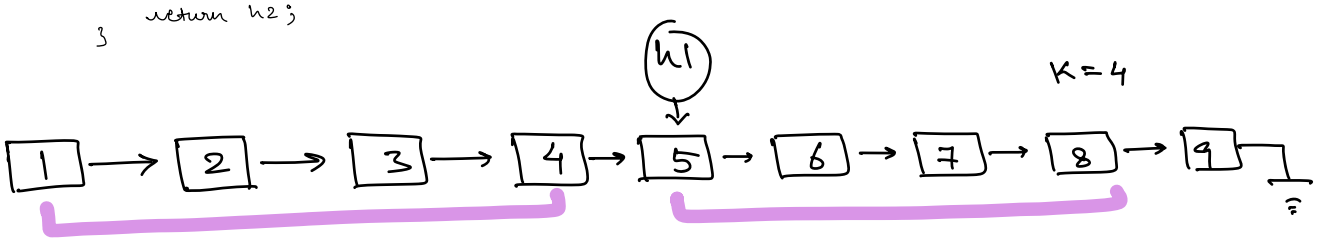
Reverse all sub-lists of size K.



K = 4




```
Node reverseFirstK ( head , K ) {
    if( K == 0 || head == Null ) return head;
    h1 = head , h2 = Null;
    while( h1 != Null && K > 0 ) {
        temp = h1;
        h1 = h1.next;
        temp.next = h2
        h2 = temp
        K--;
    }
⇒   head.next = H1;
    return h2;
}
```



* K > N ⇒ Reverse the complete list.

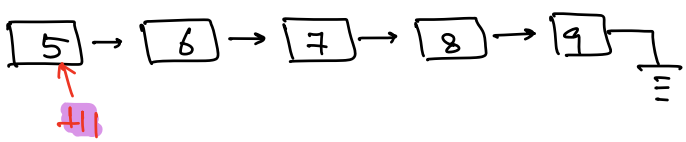
* K = 0 / K = 1 ⇒ No Change

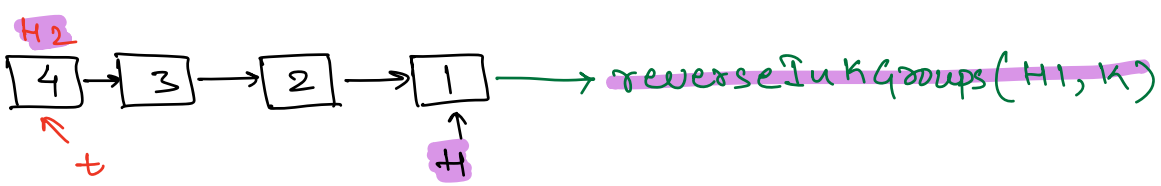
```
Node reverseInKGroups (head, K) {
    // Assumption :- reverseInKGroups(node, K)
    // will reverse all the groups of size K
    // in the list starting with node.
    if( K <= 1 || head == Null)
        return head;

    // reverse first K nodes.
    h1 = head, h2 = Null;
    count = K;
    while( h1 != Null && K > 0) {
        temp = h1;
        h1 = h1.next;
        temp.next = h2
        h2 = temp
        K--;
    }
    head.next = reverseInKGroups(h1, count);
    return h2;
}
```

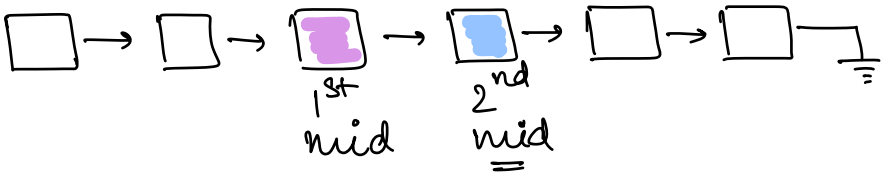
Dry run

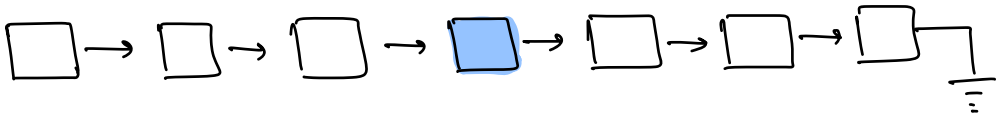
K = 4





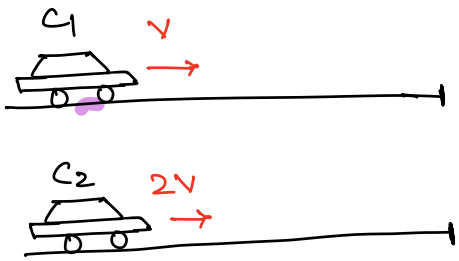
Q.2 Given a L.L, find its middle node.

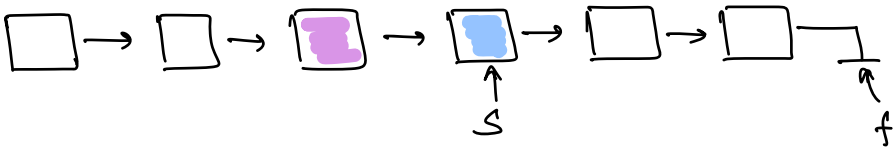


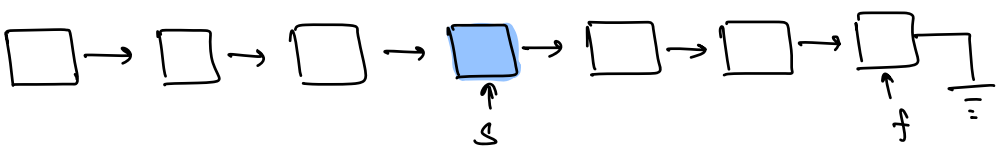


Two pointers

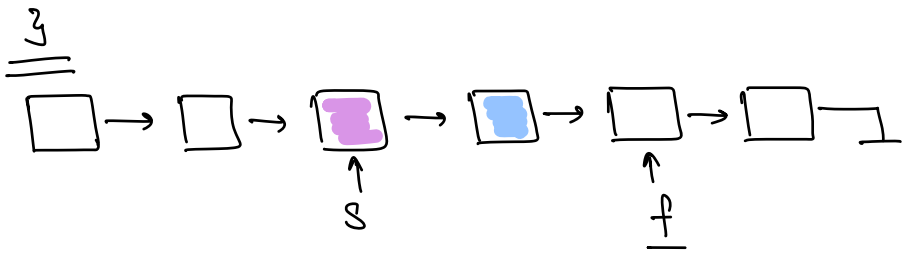
slow fast

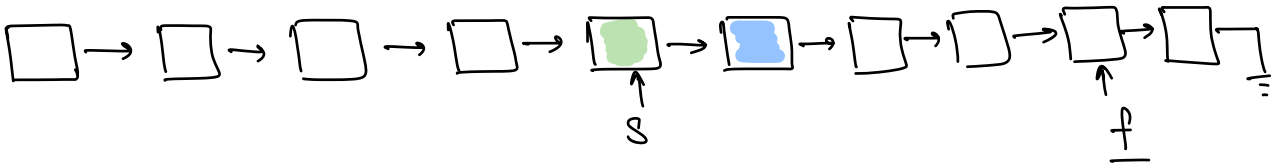




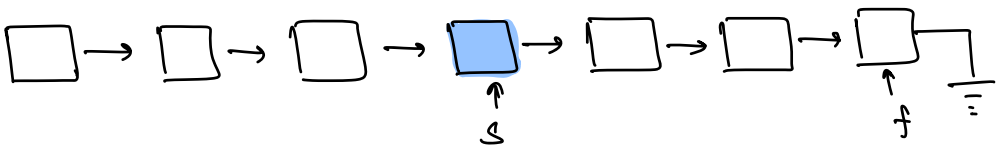


```
Node getMid(Node head){
    if(head == Null) return head;
    slow = head;
    fast = head;
    while (fast != null && fast.next != null)
    {
        slow = slow.next
        fast = fast.next.next;
    }
    return slow;
}
```





$\Rightarrow$ fast.next.next == null



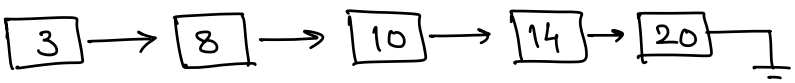
fast.next == null

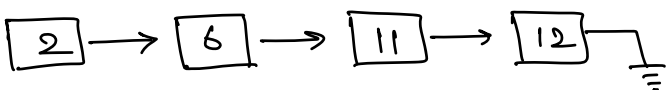
$\Rightarrow$ Return 1st mid if length of the L.L is even.

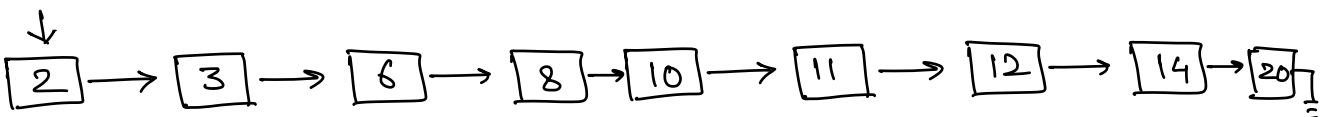
⇒ while (~~fast.next != null~~ &&
fast.next.next != null) {

}

Q: Given 2 sorted (Asc) L.L's. Do inplace merging of these L.L into a sorted L.L in ascending order.



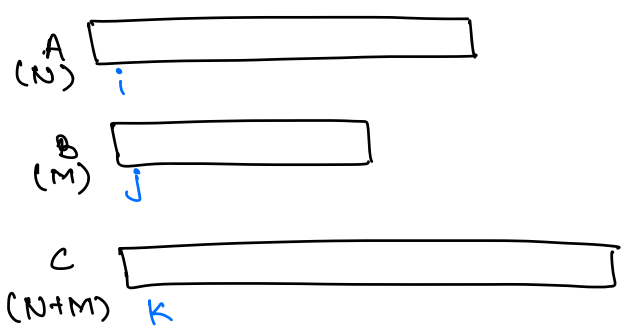




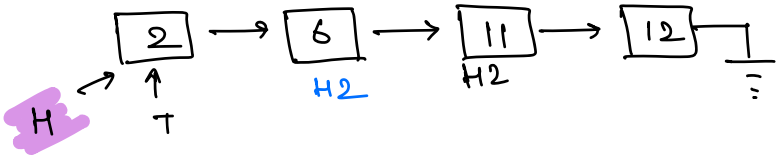
h1 : head of L.L 1

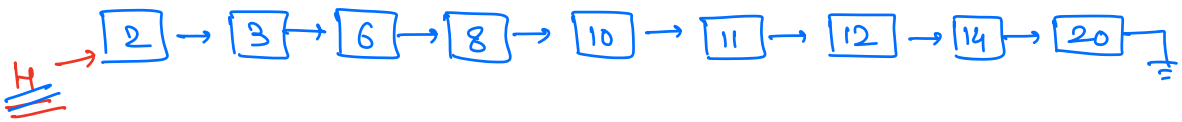
h2 : head of L.L 2

temp :



SC : O(N+M)






```
Node merge (h1, h2) {
    // fix the head;
    if(h1.val < h2.val) {
        head = h1;
        h1 = h1.next;
    }
    else {
        head = h2;
        h2 = h2.next;
    }
    temp = head;
    while (h1 != Null && h2 != Null) {
        if(h1.val < h2.val) {
            temp.next = h1;
            h1 = h1.next;
        }
        else {
            temp.next = h2;
            h2 = h2.next;
        }
        temp = temp.next;
    }
```

```
if(h1 == null) temp.next = h2;
else           temp.next = h1;
return head;
```

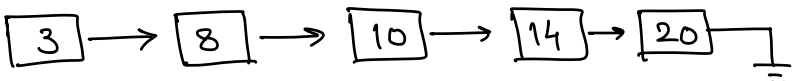
}

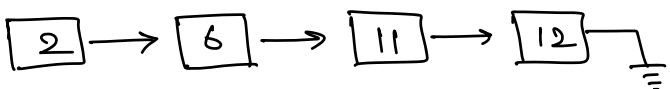
TC : O(N+M)

SC : O(1)

follow up ques :-

HW ⇒ Merge the two sorted lists to form a sorted list in descending order.





Q: Given a L.L. Sort it using Merge Sort.

Google
MS
Amazon

```
mergeSort(A, S, e) {
    if(S==e) return;
    m = (S+e)/2
    mergeSort(A, S, m);
    mergeSort(A, m+1, e);
    merge(A, S, m, e);
}
```

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

```
Node mergeSort(head) {
    // Assumption :- mergeSort(node) will sort
    // the L.L starting from node to NULL.
    if(head == NULL || head.next == NULL)
        return head;
    Node mid = getLStMid(head); ⇒ O(N)
    [ h2 = mid.next
      mid.next = Null; ]
    h1 = mergeSort(head)
    h2 = mergeSort(h2)
    return merge(h1, h2); ⇒ O(N)
}
```

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

$$\Rightarrow \boxed{TC: O(N \log_2 N)}$$

SC of M.S on Arrays : $O(N + \log_2 N)$

- N → Space req'd for merge step
- $\log_2 N$ → Call stack

SC of M.S on L.L's : $O(\log N)$

- $\log N$ → Call stack

—— * ——