**Easy**

**1) 234. Palindrome Linked List**

Following are the steps to this approach: -

* Find the middle element of the linked list.
* Reverse linked list from next element of middle element.
* Iterate through the 1st list until it reaches middle element and new list reached till end
* Check if data from the both list is same or not.

**2)** **141. Linked List Cycle**

**3) 1290. Convert Binary Number in a Linked List to Integer**

Following are the steps to this approach: -

* Traverse the list till head is not null
* Multiply result by 2 and add its head data to it.
* Return result.

**4) 19. Remove Nth Node From End of List**

**Approach :**

1. Take two dummy nodes, who’s next will be pointing to the head.
2. Take another node to store the head, initially a dummy node(dummy), and the next node will be pointing to the head. The reason why we are using this extra dummy node is that there is an edge case. If the node is equal to the length of the LinkedList, then this slow will point to slow’s next→ next. And we can say our dummy start node will be broken and will be connected to the slow next→ next.
3. Start traversing until the fast pointer reaches the nth node.
4. Now start traversing by one step both of the pointers until the fast pointers reach the end.
5. When the traversal is done, just do the deleting part. Make slow pointers next to the next of the slow pointer to ignore/disconnect the given node.
6. Last, return to the next dummy.

**5) Find Length Of Loop**

**Approach :**

1. find the loop

2) once loop is found, then take the slow pointer into temp

3) iterate the temp till slow is not met and meanwhile increment the counter

4) return the counter.

**6) 142. Linked List Cycle II**

**Approach:**

1. The following steps are required:
2. Initially take two pointers, fast and slow. The fast pointer takes two steps ahead while the slow pointer will take a single step ahead for each iteration.
3. We know that if a cycle exists, fast and slow pointers will collide.
4. If the cycle does not exist, the fast pointer will move to NULL
5. Else, when both slow and fast pointer collides, it [detects a cycle](https://takeuforward.org/data-structure/detect-a-cycle-in-a-linked-list/) exists.
6. Take another pointer, say entry. Point to the very first of the linked list.
7. Move the slow and the entry pointer ahead by single steps until they collide.
8. Once they collide, we get the starting node of the linked list.

**7) 160. Intersection of Two Linked Lists**

**Approach:**

1. Take two dummy nodes for each list. Point each to the head of the lists.
2. Iterate over them. If anyone becomes null, point them to the head of the opposite lists and continue iterating until they collide.

**8) 23. Merge k Sorted Lists**

**Approach:**

1. Add all the lists node into heap
2. Create a dummy node and iterate till heap is not empty
3. Add all the node into dummy node.
4. Return head of the dummy node.

**9) 148. Sort List**

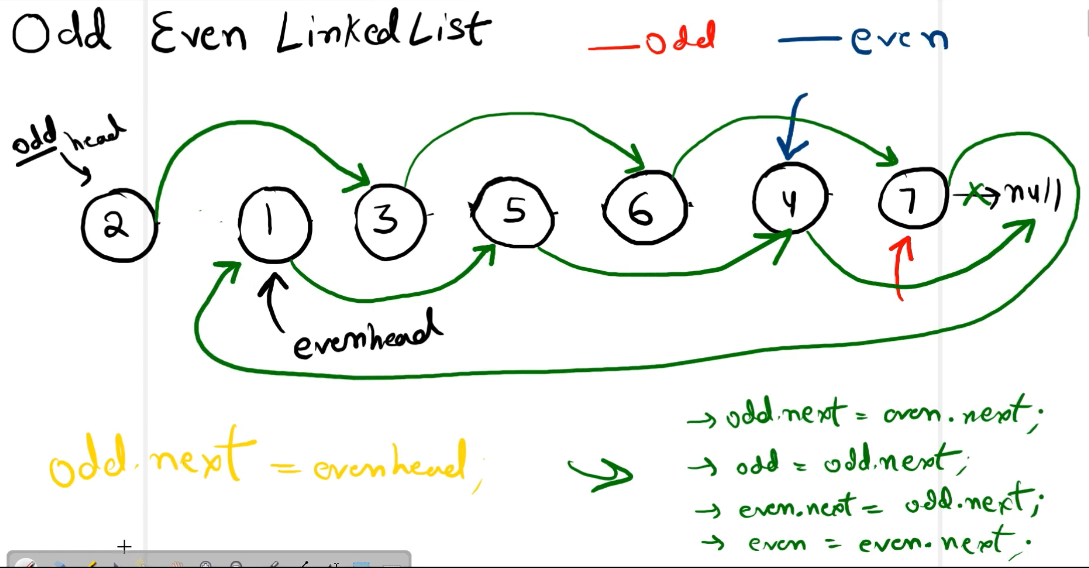
**Approach:**

1. Apply merge sort

**10)** **21. Merge Two Sorted Lists**

**Approach:**

**11) 328. Odd Even Linked List**



**12) 83. Remove Duplicates from Sorted List**

**13) 82. Remove Duplicates from Sorted List II**

**14) 203. Remove Linked List Elements**

**15) 143. Reorder List**

**16) 25. Reverse Nodes in k-Group**

**17) 61. Rotate List**