**Easy**

**206. Reverse Linked List**

**Input:** head = [1,2,3,4,5]

**Output:** [5,4,3,2,1]

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

**Input:** head = [1,0,1]

**Output:** 5

**Explanation:** (101) in base 2 = (5) in base 10

**Approach:**

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

**160. Intersection of Two Linked Lists**



**Input:** intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3

**Output:** Intersected at '8'

**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.

**141. Linked List Cycle**



**Input:** head = [3,2,0,-4], pos = 1

**Output:** true

**Explanation:** There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

**21. Merge Two Sorted Lists**



**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

**234. Palindrome Linked List**



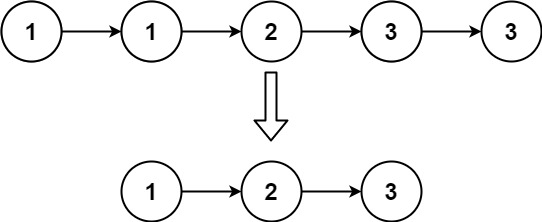
**Input:** head = [1,2,2,1]

**Output:** true

**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.

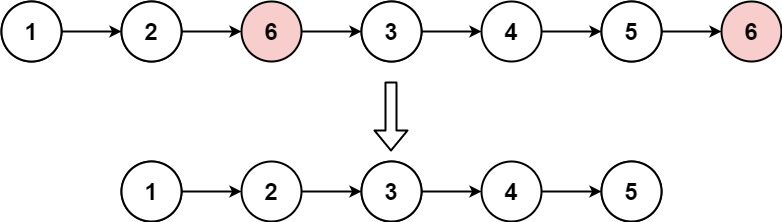
**83. Remove Duplicates from Sorted List**



**Input:** head = [1,1,2,3,3]

**Output:** [1,2,3]

**203. Remove Linked List Elements**



**Input:** head = [1,2,6,3,4,5,6], val = 6

**Output:** [1,2,3,4,5]

**Medium**

**2. Add Two Numbers**

**Example 1:**



**Input:** l1 = [2,4,3], l2 = [5,6,4]

**Output:** [7,0,8]

**Explanation:** 342 + 465 = 807.

**138. Copy List with Random Pointer**



**Input:** head = [[7,null],[13,0],[11,4],[10,2],[1,0]]

**Output:** [[7,null],[13,0],[11,4],[10,2],[1,0]]

**142. Linked List Cycle II**



**Input:** head = [3,2,0,-4], pos = 1

**Output:** tail connects to node index 1

**Explanation:** There is a cycle in the linked list, where tail connects to the second node.

**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.

**148. Sort List**



**Input:** head = [4,2,1,3]

**Output:** [1,2,3,4]

**Approach:**

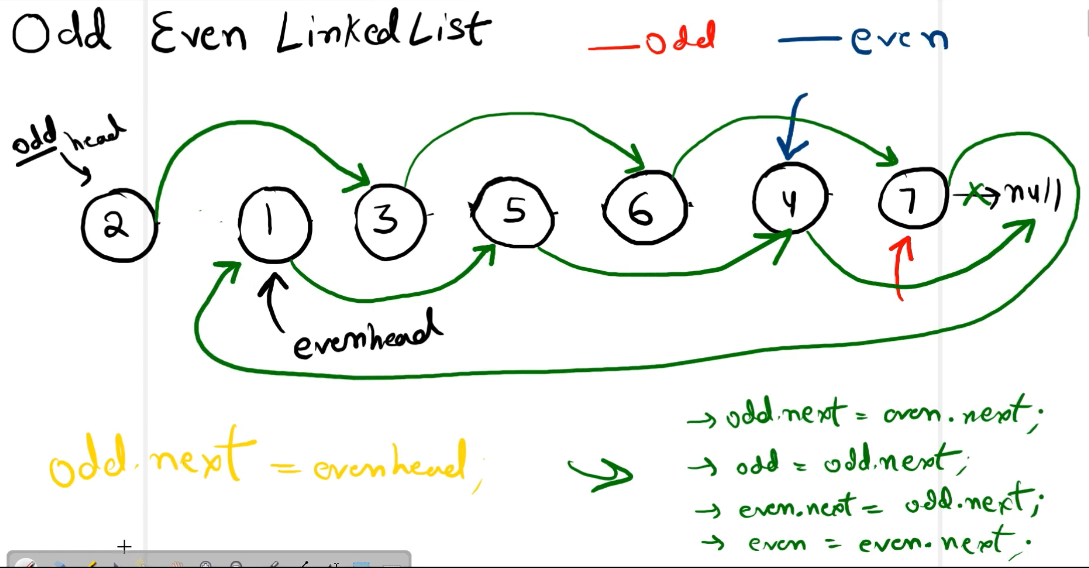
1. Apply merge sort

**328. Odd Even Linked List**



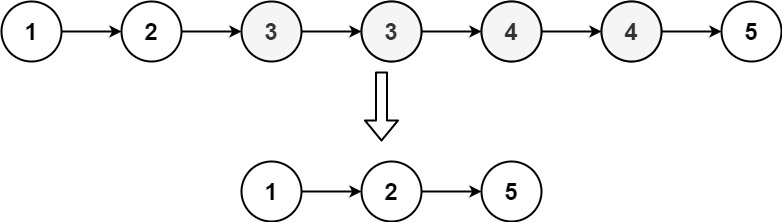
**Input:** head = [1,2,3,4,5]

**Output:** [1,3,5,2,4]



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

**Example 1:**



**Input:** head = [1,2,3,3,4,4,5]

**Output:** [1,2,5]

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



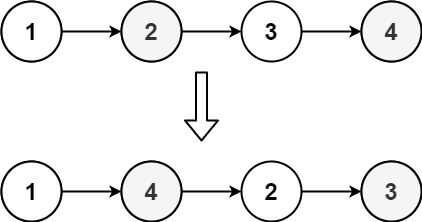
**Input:** head = [1,2,3,4,5], n = 2

**Output:** [1,2,3,5]

**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.

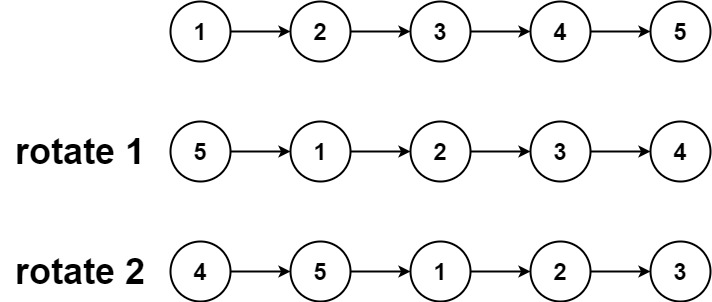
**143. Reorder List**



**Input:** head = [1,2,3,4]

**Output:** [1,4,2,3]

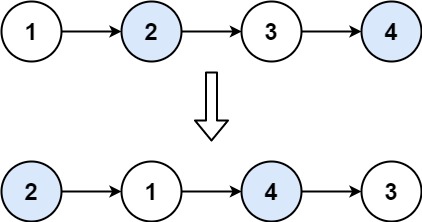
**61. Rotate List**



**Input:** head = [1,2,3,4,5], k = 2

**Output:** [4,5,1,2,3]

**24. Swap Nodes in Pairs**



**Input:** head = [1,2,3,4]

**Output:** [2,1,4,3]

**Hard**

**23. Merge k Sorted Lists**

**Input:** lists = [[1,4,5],[1,3,4],[2,6]]

**Output:** [1,1,2,3,4,4,5,6]

**Explanation:** The linked-lists are:

[

1->4->5,

1->3->4,

2->6

]

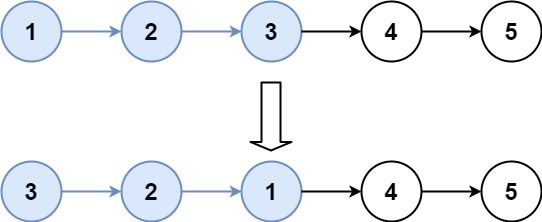
merging them into one sorted list:

1->1->2->3->4->4->5->6

**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.

**25. Reverse Nodes in k-Group**



**Input:** head = [1,2,3,4,5], k = 3

**Output:** [3,2,1,4,5]

**Other**

**1) Add 1 to the linkedlist**

**2) Find length of the loop**

**3) Flattening a linked list**

**4) Rearrange linkedlist in zig zag fashion**

**5) Remove cycle in linkedlist**