**LINKED LIST QUESTIONS:**

**Q1. 876. Middle of the Linked List**

Given the head of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle** node.

**Example 1:**

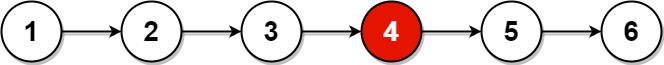


**Input:** head = [1,2,3,4,5]

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

**Explanation:** The middle node of the list is node 3.

**Example 2:**



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

**Output:** [4,5,6]

**Explanation:** Since the list has two middle nodes with values 3 and 4, we return the second one.

**Constraints:**

* The number of nodes in the list is in the range [1, 100].
* 1 <= Node.val <= 100

**Solution:**

package LinkedList;  
  
public class ListNode {  
 public int val;  
 public ListNode next;  
  
 public ListNode(int x) {  
 val = x;  
 next = null;  
 }  
}

class Solution {  
 public ListNode middleNode(ListNode head) {  
  
 ListNode slow = head;  
 ListNode fast = head;  
 if (head.next == null) {  
 return head;  
 }  
 while (fast != null && fast.next != null) {  
 slow = slow.next;  
 fast = fast.next.next;  
 }  
 return slow;  
 }  
}

Q2. **141. Linked List Cycle**

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter**.

Return true*if there is a cycle in the linked list*. Otherwise, return false.

**Example 1:**



**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).

**Example 2:**



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

**Output:** true

**Explanation:** There is a cycle in the linked list, where the tail connects to the 0th node.

**Example 3:**



**Input:** head = [1], pos = -1

**Output:** false

**Explanation:** There is no cycle in the linked list.

**Constraints:**

* The number of the nodes in the list is in the range [0, 104].
* -105 <= Node.val <= 105
* pos is -1 or a **valid index** in the linked-list.

**Follow up:** Can you solve it using O(1) (i.e. constant) memory?

Solution:

public class Solution {  
 public boolean hasCycle(ListNode head) {  
  
 ListNode slow = head;  
 ListNode fast = head;  
  
 boolean hasCycle = false;  
  
 while (fast != null && fast.next != null) {  
 slow = slow.next;  
 fast = fast.next.next;  
  
 if (slow == fast) {  
 hasCycle = true;  
 break;  
 }  
 }  
  
 return hasCycle;  
  
 }  
}

**Q3. Find the node from where cycle starts in a linkedlist.**

**Solution:**

public ListNode detectCycle(ListNode head) {  
 ListNode slow = head;  
 ListNode fast = head;  
 while (fast != null && fast.next != null) {  
 slow = slow.next;  
 fast = fast.next.next;  
 if (slow == fast) {  
 return slow;  
 }  
 }  
 return null;  
}

public ListNode findCycleStartPos(ListNode head) {  
 ListNode meetingPoint = detectCycle(head);  
 ListNode start = head;  
 while (meetingPoint != start) {  
 meetingPoint = meetingPoint.next;  
 start = start.next;  
 }  
 return start;  
}

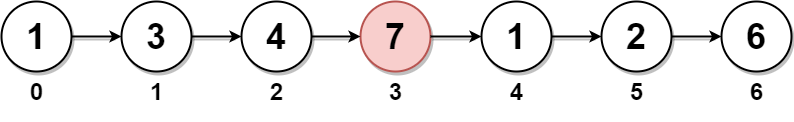
Q. **2095. Delete the Middle Node of a Linked List**

You are given the head of a linked list. **Delete** the **middle node**, and return *the* head *of the modified linked list*.

The **middle node** of a linked list of size n is the ⌊n / 2⌋th node from the **start** using **0-based indexing**, where ⌊x⌋ denotes the largest integer less than or equal to x.

* For n = 1, 2, 3, 4, and 5, the middle nodes are 0, 1, 1, 2, and 2, respectively.

**Example 1:**



**Input:** head = [1,3,4,7,1,2,6]

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

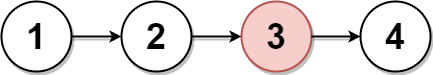
**Explanation:**

The above figure represents the given linked list. The indices of the nodes are written below.

Since n = 7, node 3 with value 7 is the middle node, which is marked in red.

We return the new list after removing this node.

**Example 2:**



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

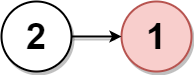
**Output:** [1,2,4]

**Explanation:**

The above figure represents the given linked list.

For n = 4, node 2 with value 3 is the middle node, which is marked in red.

**Example 3:**



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

**Output:** [2]

**Explanation:**

The above figure represents the given linked list.

For n = 2, node 1 with value 1 is the middle node, which is marked in red.

Node 0 with value 2 is the only node remaining after removing node 1.

**Constraints:**

* The number of nodes in the list is in the range [1, 105].
* 1 <= Node.val <= 105

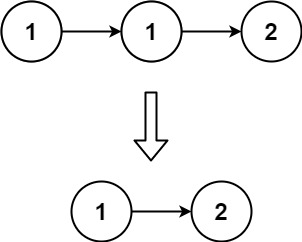
Solution:

class Solution {  
 public ListNode deleteMiddle(ListNode head) {  
 ListNode slow = head;  
 ListNode fast = head;  
 ListNode prev = head;  
  
 if (head == null || head.next == null) {  
 return null;  
 }  
  
 while (fast != null && fast.next != null) {  
 prev = slow;  
 slow = slow.next;  
 fast = fast.next.next;  
 }  
  
 prev.next = slow.next;  
  
 return head;  
 }  
}

Q. **83. Remove Duplicates from Sorted List**

Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list****sorted****as well*.

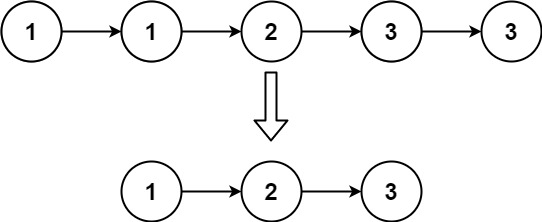
**Example 1:**



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

**Output:** [1,2]

**Example 2:**



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

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

**Constraints:**

* The number of nodes in the list is in the range [0, 300].
* -100 <= Node.val <= 100
* The list is guaranteed to be **sorted** in ascending order.

Solution:

class Solution {  
 public ListNode deleteDuplicates(ListNode head) {  
 if (head == null) return null;  
  
 ListNode cur = head;  
 while (cur.next != null) {  
 if (cur.val == cur.next.val) {  
 cur.next = cur.next.next;  
 } else {  
 cur = cur.next;  
 }  
 }  
 return head;  
  
 }  
}

Q **206. Reverse Linked List**

Given the head of a singly linked list, reverse the list, and return *the reversed list*.

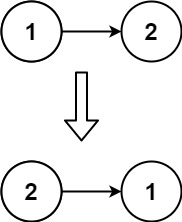
**Example 1:**



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

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

**Example 2:**



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

**Output:** [2,1]

**Example 3:**

**Input:** head = []

**Output:** []

**Constraints:**

* The number of nodes in the list is the range [0, 5000].
* -5000 <= Node.val <= 5000

Solution:

class Solution {  
 public ListNode reverseList(ListNode head) {  
 if (head == null || head.next == null) return head;  
  
 ListNode cur = head;  
 ListNode prev = null;  
 while (cur != null) {  
 ListNode temp = cur.next;  
 cur.next = prev;  
 prev = cur;  
 cur = temp;  
  
 }  
  
 return prev;  
 }  
}

**Recursive approach:**

class Solution {  
 public ListNode reverseList(ListNode head) {  
 if (head == null || head.next == null) return head;  
 ListNode newHead = reverseList(head.next);  
 ListNode nextHead = head.next;  
 nextHead.next = head;  
 head.next = null;  
 return newHead;  
  
 }  
}

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

Given the head of a linked list, remove the nth node from the end of the list and return its head.

**Example 1:**



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

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

**Example 2:**

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

**Output:** []

**Example 3:**

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

**Output:** [1]

**Constraints:**

* The number of nodes in the list is sz.
* 1 <= sz <= 30
* 0 <= Node.val <= 100
* 1 <= n <= sz

**Follow up:** Could you do this in one pass?

Solution:

class Solution {  
 public ListNode removeNthFromEnd(ListNode head, int n) {  
 if (head == null || head.next == null) {  
 return null;  
 }  
 int size = 0;  
 ListNode cur = head;  
 while (cur != null) {  
 cur = cur.next;  
 size++;  
 }  
 if (size == n) {  
 return head.next;  
 }  
  
 int index = size - n - 1;  
 ListNode temp = head;  
 int i = 0;  
 while (temp != null && i < index) {  
 temp = temp.next;  
 i++;  
 }  
 temp.next = temp.next.next;  
  
 return head;  
  
 }  
}

Q. **2. Add Two Numbers**

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Example 1:**



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

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

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

**Example 2:**

**Input:** l1 = [0], l2 = [0]

**Output:** [0]

**Example 3:**

**Input:** l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

**Output:** [8,9,9,9,0,0,0,1]

**Constraints:**

* The number of nodes in each linked list is in the range [1, 100].
* 0 <= Node.val <= 9
* It is guaranteed that the list represents a number that does not have leading zeros.

**Solution:**

class Solution {  
 public ListNode addTwoNumbers(ListNode l1, ListNode l2) {  
 ListNode head = new ListNode();  
 ListNode temp = head;  
  
 int carry = 0;  
 while (l1 != null || l2 != null || carry == 1) {  
 int sum = 0;  
  
 if (l1 != null) {  
 sum += l1.val;  
 l1 = l1.next;  
 }  
  
 if (l2 != null) {  
 sum += l2.val;  
 l2 = l2.next;  
 }  
 sum += carry;  
 int digit = sum % 10;  
 carry = sum / 10;  
  
 ListNode newNode = new ListNode(digit);  
 temp.next = newNode;  
 temp = temp.next;  
  
 }  
 return head.next;  
 }  
}

**Q23. Merge k Sorted Lists**

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

*Merge all the linked-lists into one sorted linked-list and return it.*

**Example 1:**

**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

**Example 2:**

**Input:** lists = []

**Output:** []

**Example 3:**

**Input:** lists = [[]]

**Output:** []

**Constraints:**

* k == lists.length
* 0 <= k <= 104
* 0 <= lists[i].length <= 500
* -104 <= lists[i][j] <= 104
* lists[i] is sorted in **ascending order**.
* The sum of lists[i].length will not exceed 104.

**Solution:**

class Solution {  
 public ListNode mergeKLists(ListNode[] lists) {  
 PriorityQueue<Integer> heap = new PriorityQueue<>(); // minHeap   
  
 ListNode dummy = new ListNode(-1);  
 ListNode temp = dummy;  
  
 for (ListNode head : lists) {  
 while (head != null) {  
 heap.add(head.val);  
 head = head.next;  
 }  
 }  
  
 while (!heap.isEmpty()) {  
 temp.next = new ListNode(heap.remove());  
 temp = temp.next;  
 }  
  
 return dummy.next;  
 }  
}