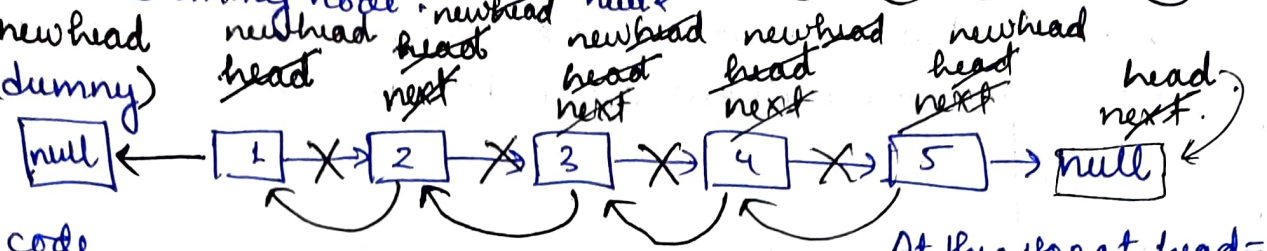
Day - 5

1> Reverse a LinkedList :- eg:- 1 → 2 → 3 → 4 → 5 → 6 → null

o/p: 6 → 5 → 4 → 3 → 2 → 1

→ make dummy node.



code

```
Listnode* reverse(Listnode* head){
    Listnode* newhead = null;
    while(head != null){
        Listnode* next = head->next;
        head->next = newhead;
        newhead = head;
        head = next;
    }
    return newhead;
}
```

At the, stop at head == null and return newhead.

TC :- O(n)
SC :- O(1)

→ used 3ptr technique.

(ii) **Middle of the linked list** :- If two middle node, return the second middle node.

eg:- 1 → 2 → 3 → 4 → 5 return 3 & hence o/p:- 3 → 4 → 5

eg:- 1 → 2 → 3 → 4 → 5 → 6 return 4 & hence o/p:- 4 → 5 → 6

① **Brute force** :- Traverse in the linkedlist and find total no. of nodes, then again traverse till $(\frac{n}{2}+1)$ and return that node.

TC :- $O(n) + O(\frac{n}{2})$
SC :- $O(1)$

② **optimal Approach** :- Doing in single traversal. (Tortoise Method).

eg:- slow slow slow

1 → 2 → 3 → 4 → 5 → null.

fast fast fast

move slow by 1 step and fast by 2 step.

Stop if (fast == null or fast → next == null)
return slow.

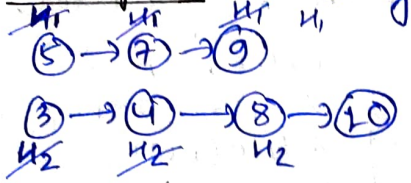
#code

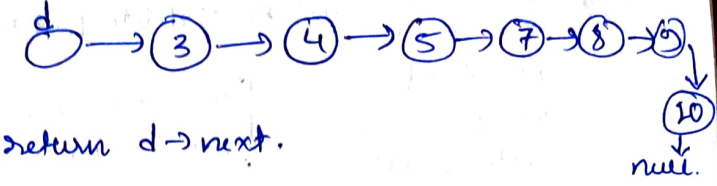
```
ListNode* middleNode(ListNode* head){
    ListNode* fast = head, * slow = head;
    while(fast != null and fast → next != null){
        fast = fast → next → next;
        slow = slow → next;
    }
    return slow;
}
```

iii) Merge two sorted lists eg:- 1→2→4 and 1→3→4 O/P:- 1→1→2→3→4→4

a) make entire new LL
b) Do it in place.

① Brute force:- Creating new linked list.

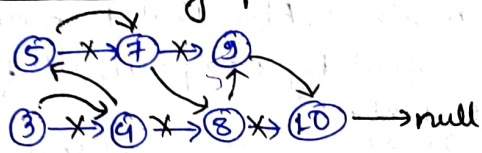




return d→next.

TC:- O(n1 + n2)
SC:- O(n1 + n2)

② Optimal (Doing Inplace)



dummy → -1
head ⇒ dummy

TC:- O(n1+n2)
SC:- O(1)

① Take 2 ptr t1 & t2 at the head of L1 and L2, ✱
② whichever is smaller attach it with a dummy node, move dummy and that ptr by one step.
③ Do this any of t1, t2 becomes null.
④ whichever becomes null attach not-null ptr to dummy.
⑤ Return head→next.

#code

```
Listnode* mergetwolist (Listnode* l1, listnode* l2) {
    if(l1==null) return l2;
    if(l2==null) return l1;
    Listnode* dummy = new listnode(-1); *t1=l1, *t2=l2
    Listnode* head = dummy;
    while(t1 and t2) {
        if(t1→val < t2→val) {
            dummy→next = t2;
            t2 = t2→next;
        }else {
            dummy→next = t1;
            t1 = t1→next;
        }
        dummy = dummy→next;
    }
    if(t1) dummy→next = t1;
    if(t2) dummy→next = t2;
    return head→next;
}
```

iv) Remove the n^{th} node from end of linked list :- (Do in-place, without using extra space)

eg:- ① → ② → ③ → ④ → ⑤ m = 2 O/P:- ① → ② → ③ → ⑤

① Brute force :- On one traversal find length of the linked list, then delete $(len - m)^{th}$ node from first in 2^{nd} traversal.

edge case :- If (len == m) ⇒ Delete 1^{st} node.

```
node = head
h → h → next;
delete (head);
```

TC :- O(n) + O(n) = O(2n)
SC :- O(1)

```
node = t → next
t → next = t → next → next;
delete (node)
```

Optimal step, But should be done to free the memory

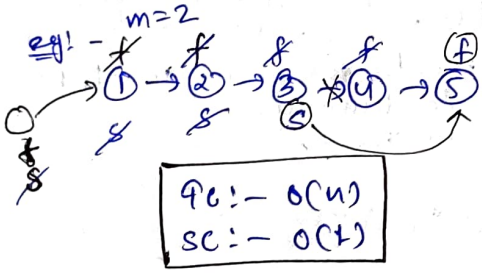
② Optimal :-

① Keep two pointer (fast, slow),
② move fast ptr m times, After that keep slow = head.
③ Now move slow & fast by 1 steps, until fast → next == null
④ then do,

```
node = s → next
s → n = s → n → n
delete (node)
```

These 2 steps are optional.

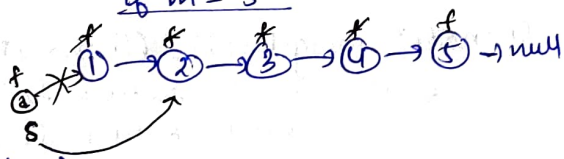
eg:- m = 2



TC :- O(n)
SC :- O(1)

⑤ at the end return dummy next. (To play safe on edge case),

if m = 5



 # code

```
listnode* remove(listnode* head, int n){
    listnode* start = new listnode();
    start → next = head;
    listnode* fast = start, * slow = start;
    for(i = 1 ——— i <= n) fast = fast → next;
    while(fast → next != null){
        fast = fast → next;
        slow = slow → next;
    }
    slow → next = slow → next → next;
    return start → next;
}
```

V) Add two number as a LL

eg:- 2 → 4 → 3 → × (342)

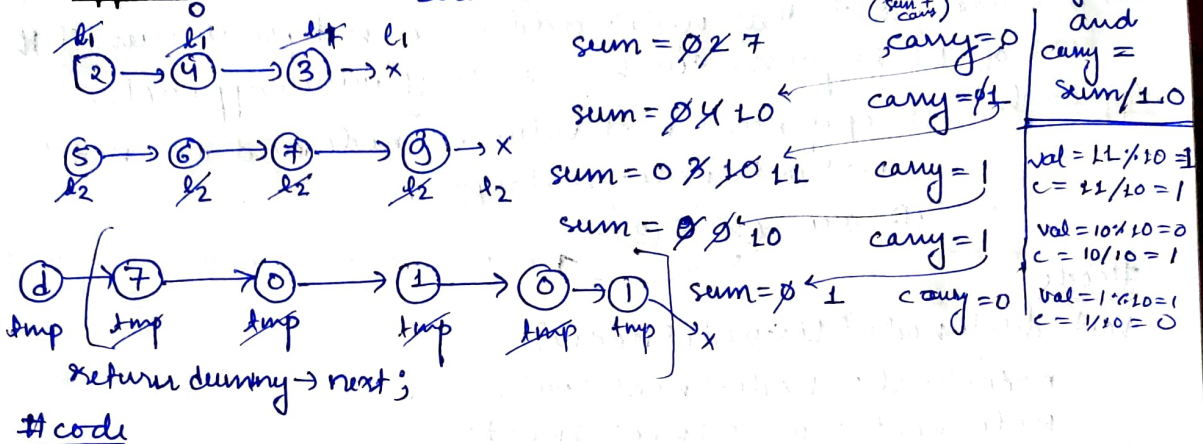
5 → 6 → 7 → 9 → × (9765)

```
  342
+ 9765
-------
 10107
```

o/p:- 7 → 10 → 1 → 0 → 1 → ×

Approach

add new node, with value = sum (sum + carry) % 10; and carry = sum/10



val = 11 % 10 = 1
c = 11/10 = 1
val = 10 % 10 = 0
c = 10/10 = 1
val = 1 % 10 = 1
c = 1/10 = 0

#code

```
listnode* add (listnode* l1, listnode* l2) {
    listnode* dummy = new listnode();
    listnode* temp = dummy;
    int carry = 0
    while (l1 != null or l2 != null or carry) {
        int sum = 0;
        if (l1 != null) {
            sum += l1 -> val;
            l1 = l1 -> next;
        }
        if (l2 != null) {
            sum += l2 -> val;
            l2 = l2 -> next;
        }
        sum += carry;
        carry = sum/10;
        listnode* node = new listnode (sum % 10);
        temp -> next = node;
        temp = temp -> next;
    }
    return dummy -> next
}
```

TC :- max(m, n) → len of l1, len of l2

SC :- O(N) → len of l1 + l2

vi) Delete a node in LL, when node is given: head of LL is not given, only node reference which is to be deleted is given.

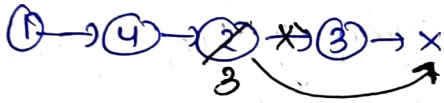
eg:-

Delete node ② , whose reference is given to you.

O/P:- ① → ④ → ③ → ×

→ To exactly delete that node's address is not possible until the head of LL is given, so here is very dumb approach.

Approach ① copy node's next value in node and on node's next put, node → next → next.



#code

```
void deleteNode(ListNode* node){
    node->val = node->next->val;
    node->next = node->next->next;
}
```