Agenda

1) Middle of LinkedList

2) Merge two sorted LL

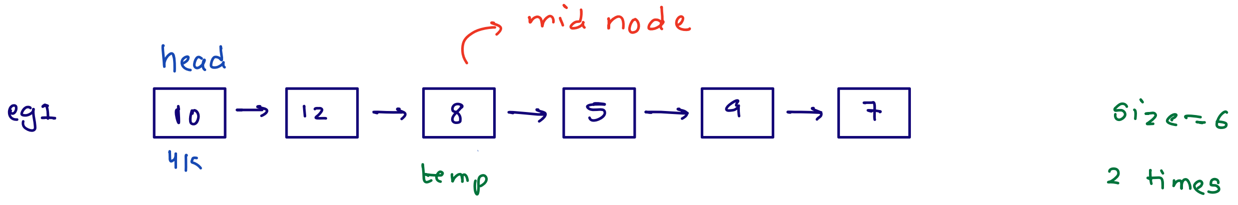
3) LL cycle detection

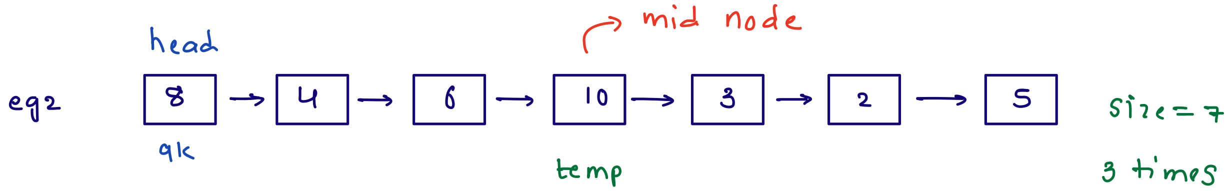
4) Start point of cycle

5) Remove cycle from LL

↳ In doubt session: check if LL is palindromic or not

Q.1 Given a LL, find and return mid node.





i) find size of LL

ii) Node temp = head

→ how many times to travel in order to find mid $= \frac{size - 1}{2}$

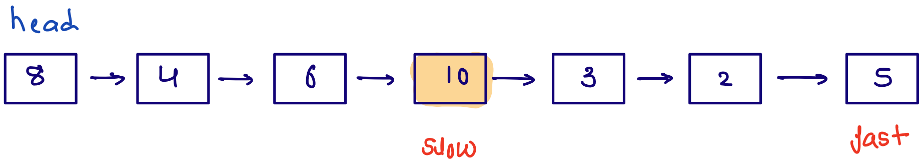
iii) return temp

Can we find mid Node in just single traversal ? Yes

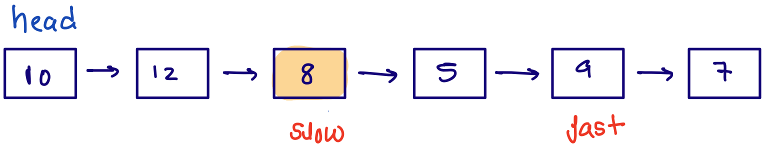
→ slow and fast pointer technique

slow → 1 at a time

fast → 2 at a time

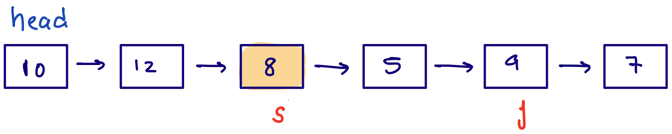


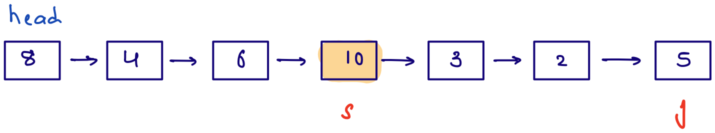
⇒ fast.next != null



⇒ fast.next.next != null

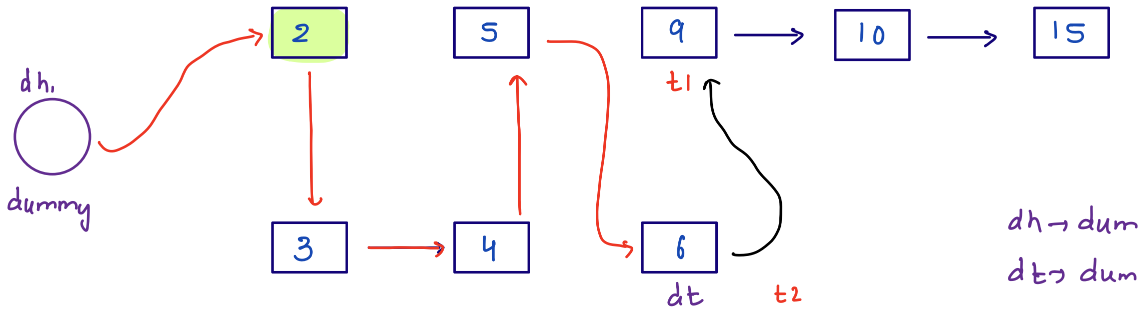
```
Node midNode (Node head) {
    Node slow = head, fast = head;

    while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
    }
    return slow;
}
```





Q-2 Given 2 sorted Linked list, merge and get final sorted List.

Note: no extra space allowed

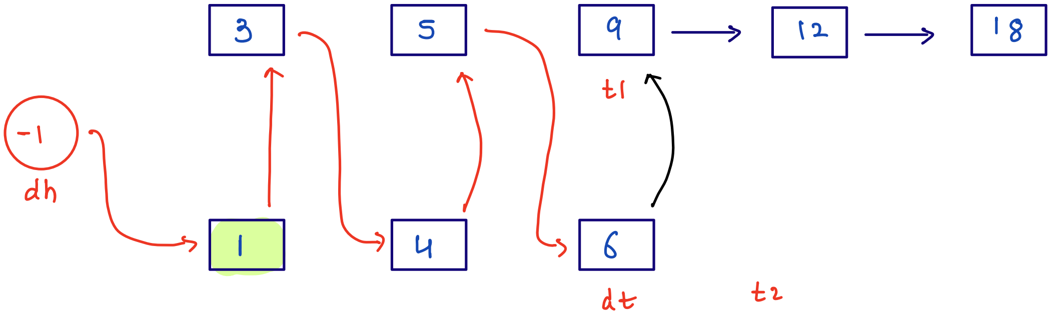


dh → dummy head
dt → dummy tail

→ dt.next = t1;

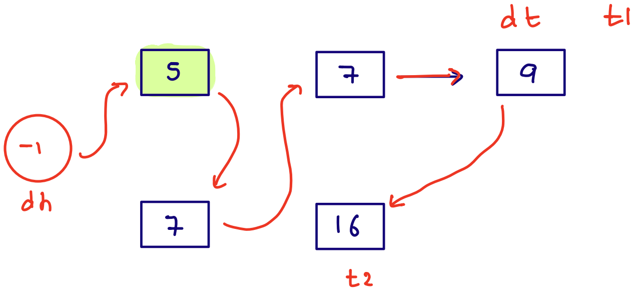
→ ans: dh.next

```
if (t1.val < t2.val) → use t1
    dt.next = t1;
    t1 = t1.next;
    dt = dt.next;

else → use t2
    dt.next = t2;
    t2 = t2.next;
    dt = dt.next;
```



ans: dh.next

```
Node dh = new Node(-1);
Node dt = dh;
Node    merge2SortedLL ( Node head1, Node head2) {
        Node dh = new Node (-1);
        Node dt = dh;
        Node t1 = head1, t2 = head2;

        while (t1 != null  && t2 != null) {
                if (t1.val < t2.val) {
                    //use t1
                    dt.next = t1;
                    t1 = t1.next;
                    dt = dt.next;
                }
                else {
                    //use t2
                    dt.next = t2;
                    t2 = t2.next;
                    dt = dt.next;
                }
        }

        if (t1 != null) {
            dt.next = t1;
        }
        if (t2 != null) {
            dt.next = t2;
        }

        return dh.next;

}
```



first LL size : n

Second LL size : m

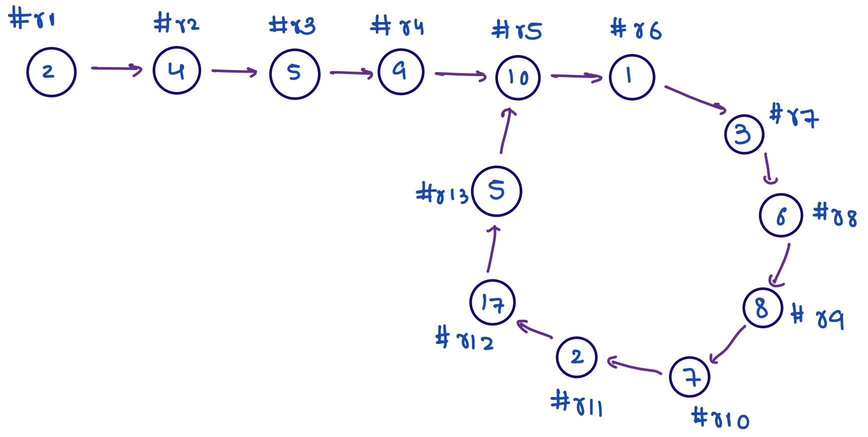
TC : O(n+m)

SC : O(1)

Q.3 Given head node of Linked list, check for cycle detection?

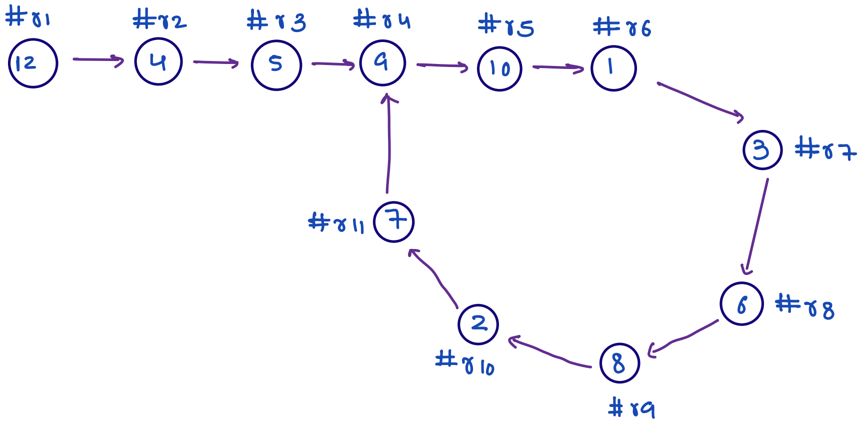
r1, r2, r3 . . .
references of
node.

eg1



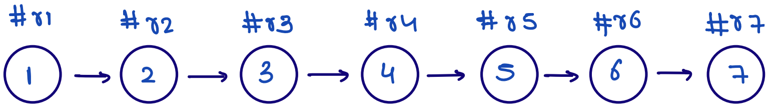
ans → True

eg2

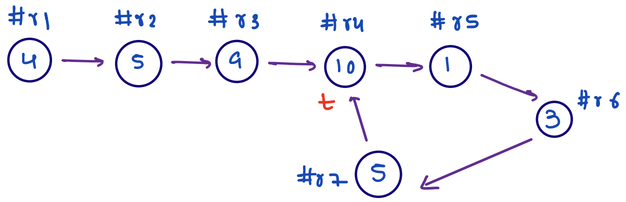


ans → True

eg3



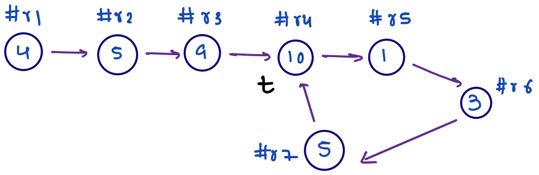
ans → false



r1 r2 r3
r4 r5 r6
r7

→ use hashset to check if we are going to a node for the second time

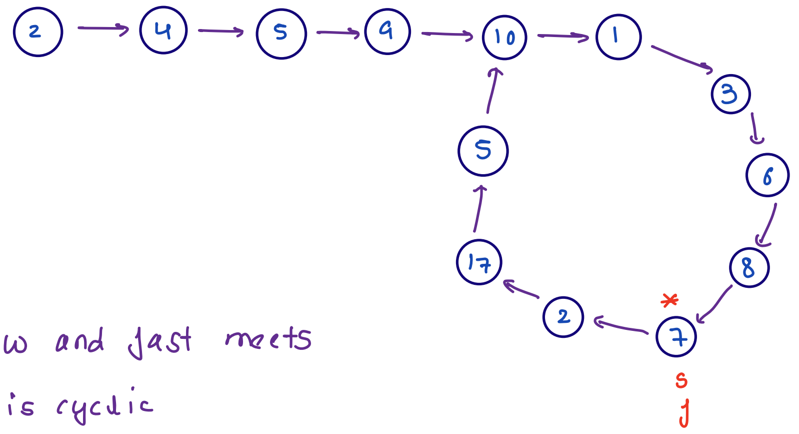
```
boolean isCyclic (Node head) {
    HashSet <Node> hs = new HashSet<>();
    Node temp = head;
    while (temp != null) {
        if (hs.contains(temp)) {
            return true;
        }
        hs.add(temp);
        temp = temp.next;
    }
    return false;
}
```



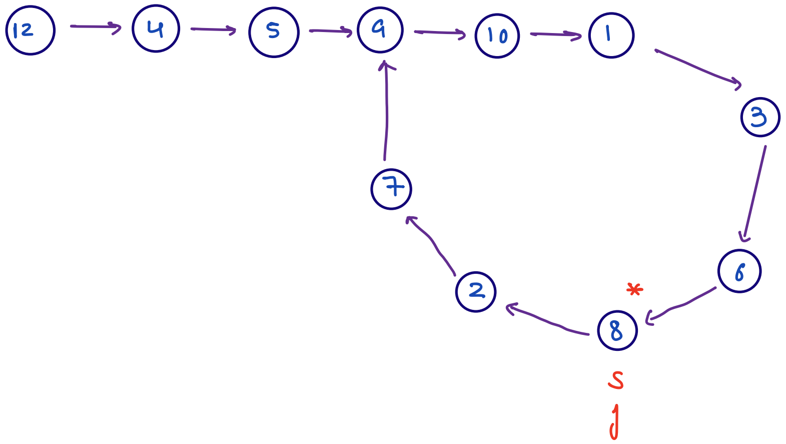
r1 r2 r3
r4 r5 r6 r7

hs

Floyd Cycle Detection → slow & Fast

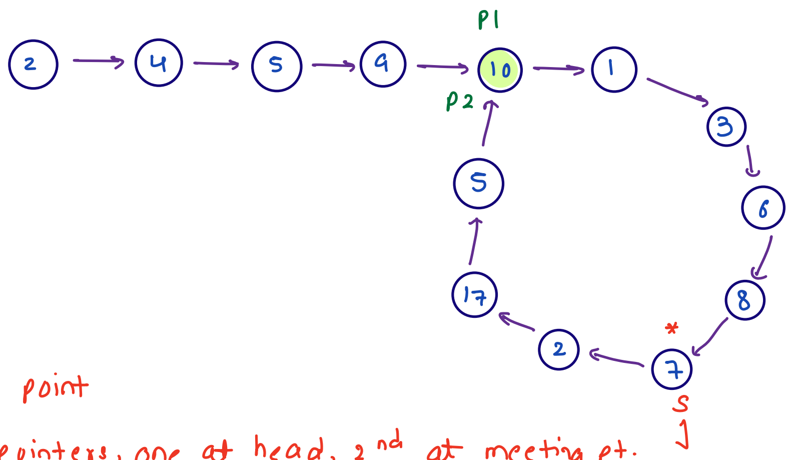


→ if slow and fast meets
LL is cyclic



```
boolean  isCyclic (Node head) {
    Node slow = head, fast = head;
    while (fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow == fast) {
            return true;
        }
    }
    return false;
}
```

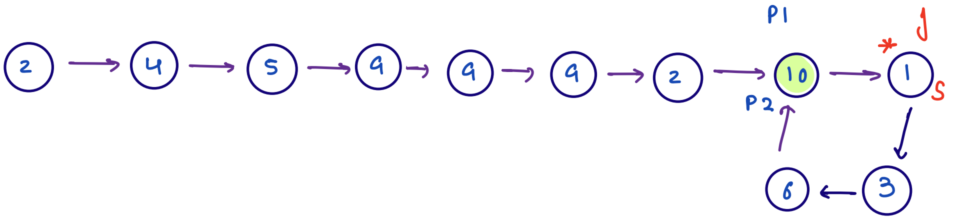
Start point of cycle → if cycle is present : return start pt. of cycle
otherwise return null



i) find meeting point

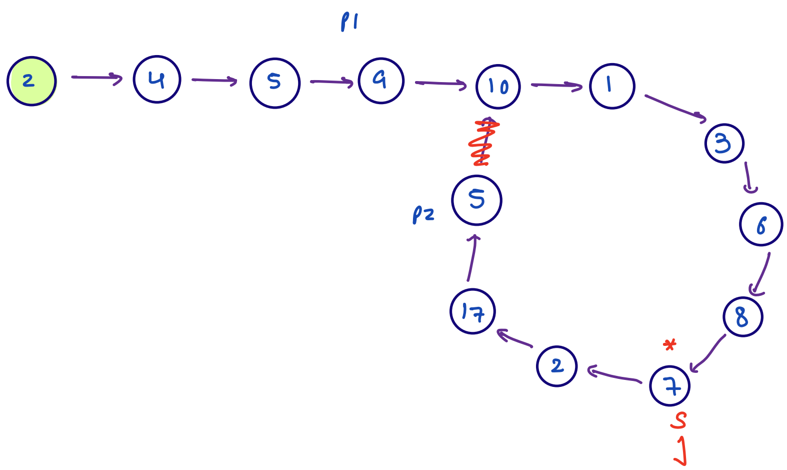
ii) create two pointers, one at head, 2nd at meeting pt.

iii) these two pointers will meet at starting point.



```
Node    StartingPointOfCycle ( Node  head) {

    Node slow = head, fast = head;

     boolean  isCycle = false;

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

              slow = slow.next;

               fast = fast.next.next;

                if ( slow == fast ) {
                        isCycle = true;
                        break;
                }
     }

      if (isCycle == false) {

              return null;

      }
                                      ↪ meeting point
       Node  p1 = head, p2 = slow;

       while (p1 != p2) {
              p1 = p1.next;
              p2 = p2.next;
       }
       return p1;   →  starting point

}
```

Remove cycle from LL and return head



```
Node   removecycle(Node head) {

    Node slow = head, fast = head;
    boolean isCycle = false;
    while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) {
                    isCycle = true;
                    break;
            }
    }

    if (isCycle == false) {
            return head;
    }
                                  ↪ meeting point
    Node p1 = head, p2 = slow;
    while (p1.next != p2.next) {
          p1 = p1.next;
          p2 = p2.next;
    }
    p2.next = null;  // p2 → last node of cycle
    return head;
}
```