Lab Session: Linked List & Sorting.

We will start at 7:07 AM

Middle element of linked list-

Given a linked list of integers, find and return the middle element of the linked list.

If there are N nodes in the linked list and N is even then return the (N/2 + 1)th element.

$1 \to 2 \to 3 \to 4 \to 5$ Ans: 3

$1 \to 2 \to 3 \to 5 \to 7 \to 10$ Ans: $\left[\frac{6}{2}+1\right]^{th} = 4^{th}$ element $= \underline{5}$

Brute force -

→ Iterate & Count all the nodes.

→ Iterate & find the middle node.

↳ T.C. → O(n)

But we are taking two iterations.

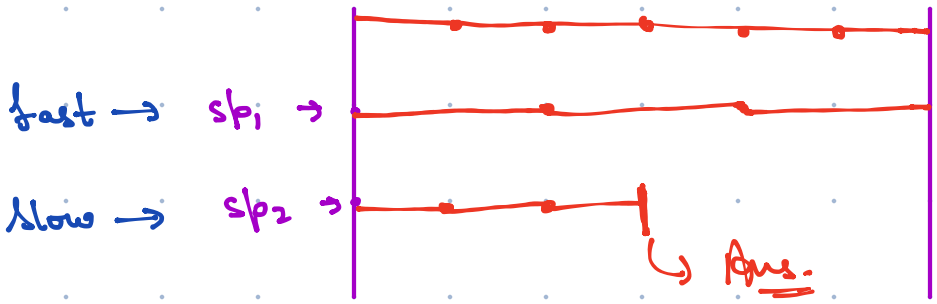
Optimized Solution-

↳ Take two pointers at head.

→ ① → s/p_1

→ ② → s/p_2

$s/p_1 = 2x\ s/p_2$



Dry Run

Eg.1

Eg.2

Code-

```
int Solve(Node A){
    Node fast=A, slow=A;
    while(fast!=null && fast.next!=null){
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow.val;
}
```

Iterations = $N/2$.

T.C → O(N)

S.C → O(1)

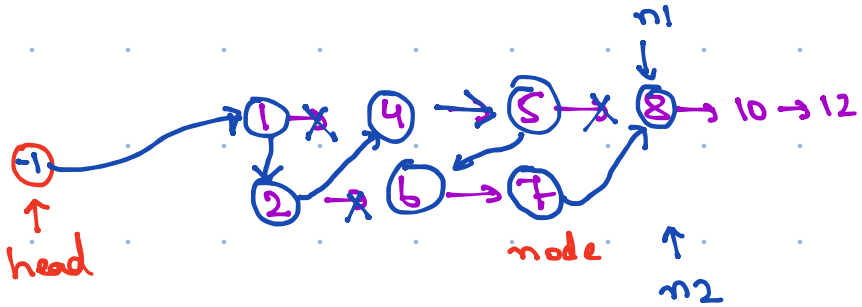
Merge two sorted linked list-

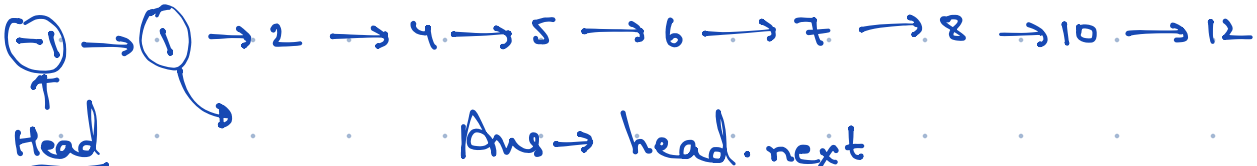
Merge two sorted linked lists, A and B, and return it as a new list.

1 → 4 → 5 → 8
2 → 6 → 7

⟩ 1 → 2 → 4 → 5 → 6 → 7 → 8

Solution-





Ans → head.next

→ Create dummy head pointer. and change the pointers of two lists to merge them.

Code-

```
Node solve(Node A, Node B) {
    if (A == null) return B;
    if (B == null) return A;
    Node head;
    Node node = new Node(-1);
    head = node;
    while (A != null && B != null) {
        if (A.val <= B.val) {
            node.next = A;
            A = A.next;
            node = node.next;
        } else {
            node.next = B;
            B = B.next;
            node = node.next;
        }
```

```
    }
    if (A == null)
        node.next = B;
    else
        node.next = A;
    head = head.next;
    return head;
}
```

T.C → O(N+M)

S.C → O(1)

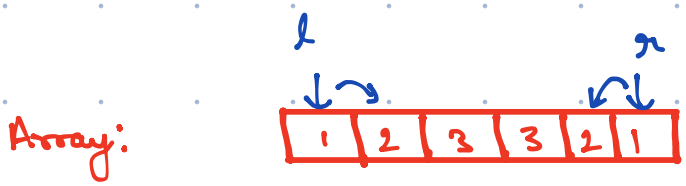
Palindrome List -

Given a singly linked list A, determine if it's a palindrome.

Return 1 or 0, denoting if it's a palindrome or not, respectively.

→ [1 2 2 1] → 1

→ [1 2 3] → 0



Challenge with L.L → Pointers can't move to backward direction.

Brute force → Copy all elements of LL to an array.

TC → O(N)

SC → O(N)

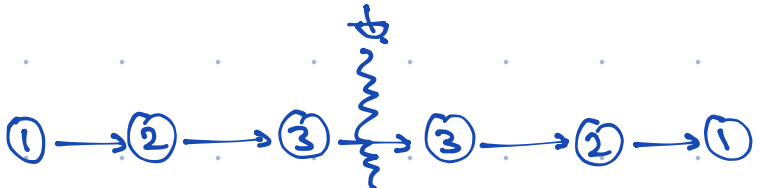
→ How to solve without using extra space ?

Sol^n-

→ Find the middle element.

→ Reverse second half of LL.

→ Compare them one-by-one to check palindrome.



Code -

```
Node slow = head, fast = head;
while (fast.next != null && fast.next.next != null) {
    slow = slow.next;
    fast = fast.next.next;
}
Node head2 = slow.next;
slow.next = null;

head2 = reverse(head2);

Node t1 = head, t2 = head2;
while (t1 != null && t2 != null) {
    if (t1.val != t2.val) return 0;
    t1 = t1.next;
    t2 = t2.next;
}
return 1;
```

```
Node reverse(Node head) {
    Node prev = null;
    Node current = head;
    Node next;
    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
```

Time Complexity → O(N)

Space Complexity → O(1)

Merge Sort a Linked list.

1 → 4 → 2 → 7 → 3]

↓

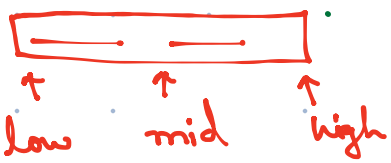
1 → 2 → 3 → 4 → 7] → ⊘

Idea → In array → ① Find middle point.

② Sort 1st half. recursively

③ Sort 2nd half. recursively.

④ Merge them.



Same idea can be used in LL.

① Find the middle node.

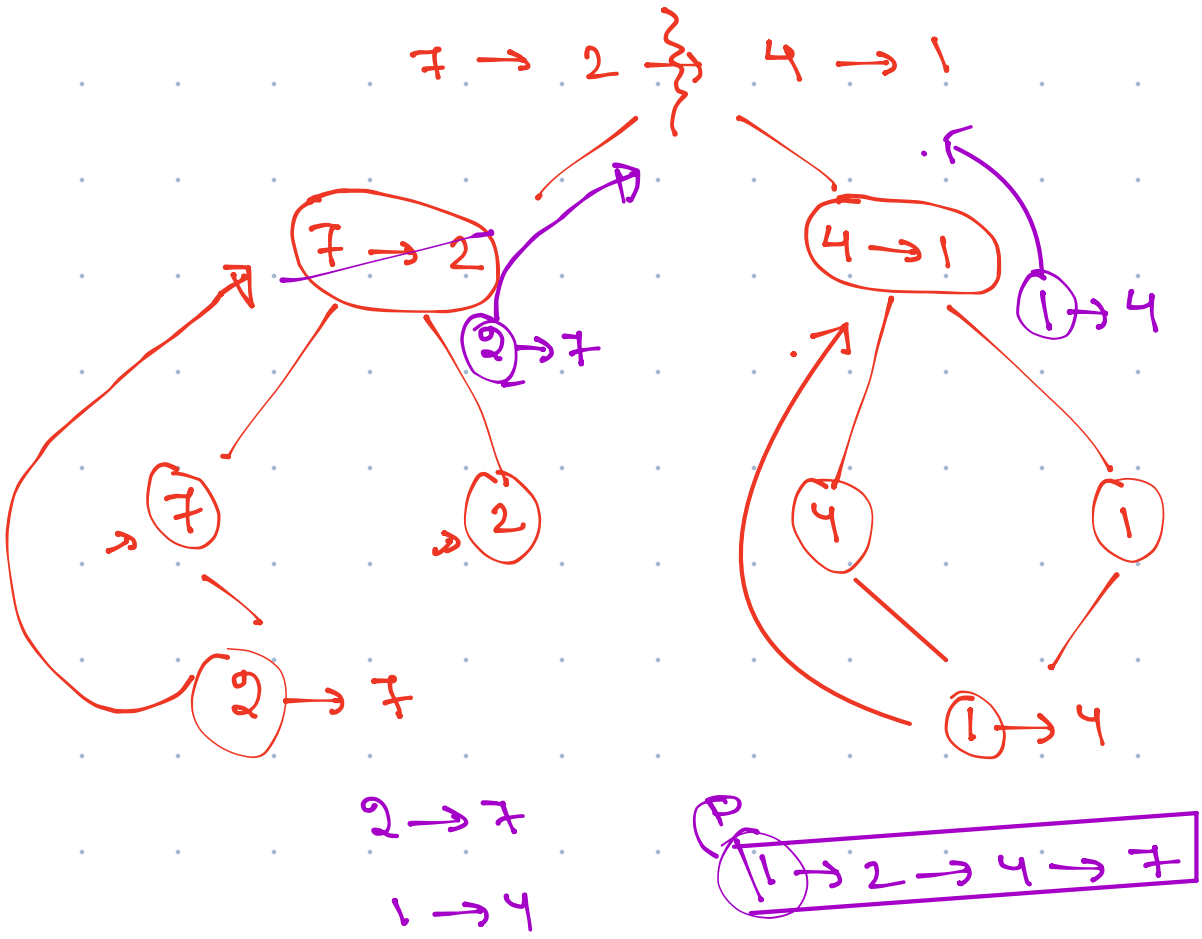
② Sort 1st half recursively.

③ Sort 2nd half recursively.

④ Merge both Sorted LL.

```
Node mergeSortLL ( Node head) {
    if (head == null || head.next == null) return head;
    Node mid = getMiddle(head);
    Node head2 = mid.next;
```

```
mid.next = null;
head = mergeSortLL(head);
head2 = mergeSortLL(head2);
return merge2SortedLL(head, head2);
}
```



T.C → O(n log n)

S.C → O(log n) → Recursive Stack.

↑

For arrays it's O(N)

↳ Need to create two arrays to merge them.

Other Sorting algorithms

Insertion Sort