Today's Content

- → Basic Linked List
- → Insert in a Sorted Linked List
- → Delete K in a sorted Linked List (Todo)
- → Reverse Linked List
- → Find the middle of a linked list.

```
class Node { // In your language of choice
    int data // variable
    Node next // Object reference → Holds address of node object
    Node (int x) { // Constructor → used to initialize data members
        data = x                                    of class
        next = NULL
    }
}
```

`Node h = new Node(30);`

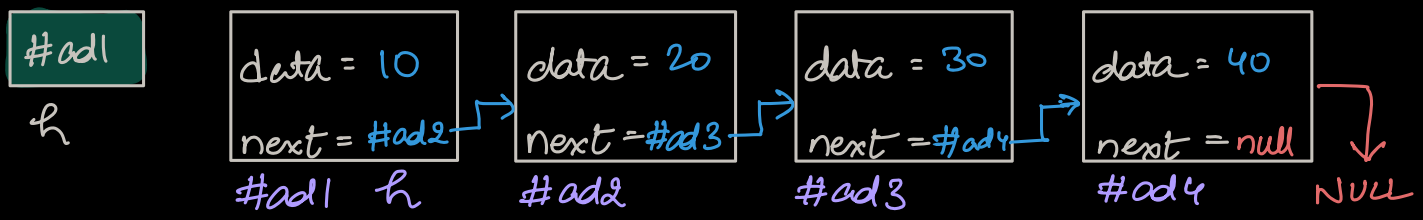
↳ an object is created & h is object reference here.

Access data & next

Note: h is just holding reference / address of an object

print(h): #ad1 print(h.data) = 30 print(h.next) = null

Linked List:



c = 0 ⇒ c = c+1 → c = c+1 → c = c+1 → c = c+1

h = h.next h = h.next h = h.next h = h.next

No. of objects in LL = 4

STOP!

c = 4

//Try this simple problem

```
int size(Node h) {
    c = 0, Node t = h
    while ( t != null ) {
        c = c+1
        t = t.next
    }
    return c
}
```

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

Note: about edge cases in linked list

→ Implementation is asked in interviews

→ to check or access h.data / h.next

```
if (h != null) {
    h.data
    h.next
}
```

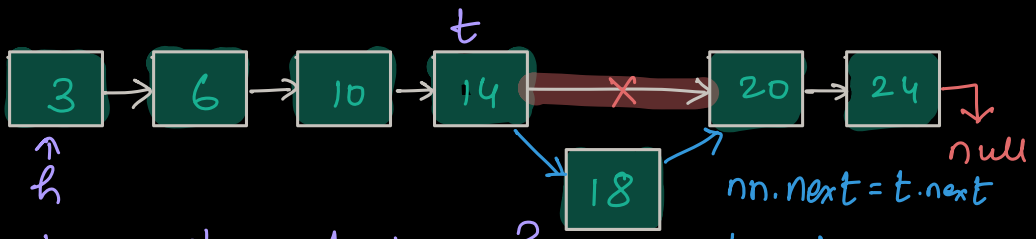
ALWAYS check if h != null.

```
if ( h != null && h.next != null) {
    h.next.data
    h.next.next
}
```

 ← Copy me

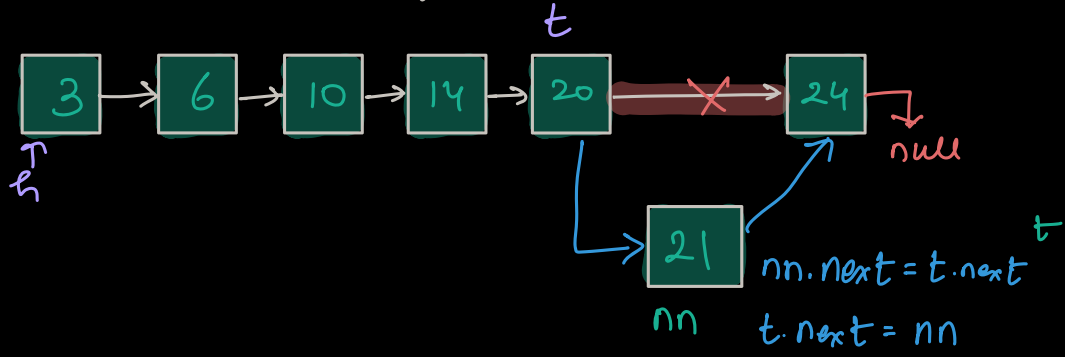
Q1: Given head of a **Sorted linked list**, create & insert new node with data = x. [List should be sorted after insertion]

Eg 1: x = 18

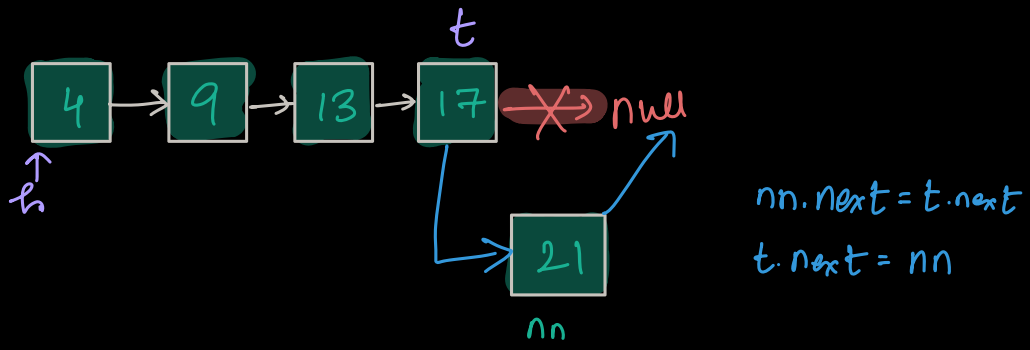


Obs: How do you know where to insert ? We iterate & stop at node if its next.data > x

Eg 2: x = 21

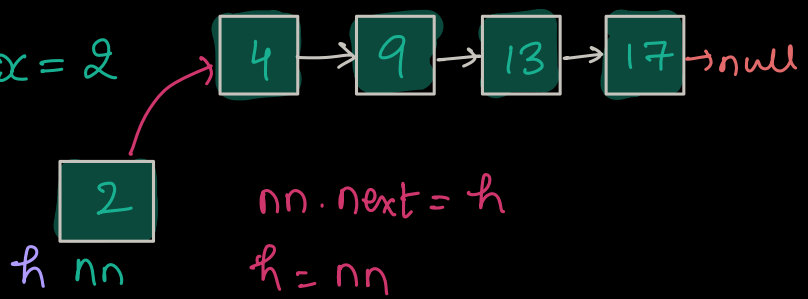


Eg-3: x = 21



Obs: if next == null, STOP. Insert new node.

Eg 4: x = 2



Obs 3: if (x < h.data) { Insert at start }

Since we might change head return head node of Linked list.

```
Node insertSorted (Node h, int x) {
    Node nn = new Node (x)
    if (h == null) { h = nn ; return h }
    if (x < h.data) { // Insert at start
        nn.next = h
        h = nn
        return h
    }
    Node t = h
    while ( t.next != null && t.next.data <= x ) {
        t = t.next
    }
    // Insert nn after t
    nn.next = t.next
    t.next = nn
    return h
}
```

TC : O(N)

SC : O(1)

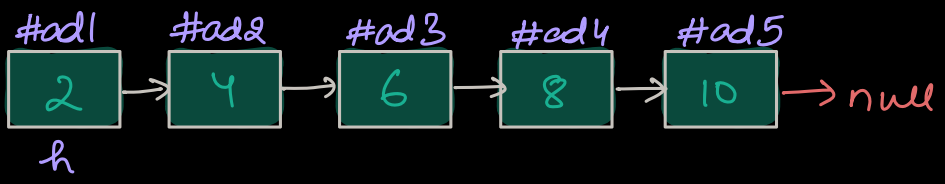
Delete K in a Sorted list ⇒ To Do (HW)

Q2: Given head of a Linked List, reverse linked list & return head.

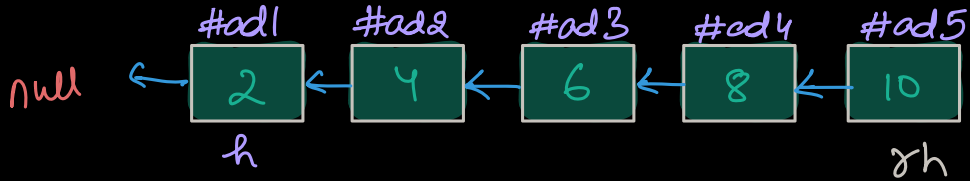
Note: No extra space & we can't change value of node

Expected TC: O(N)

Eg 1:

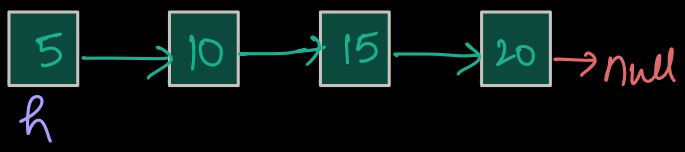


Reverse the linked list.



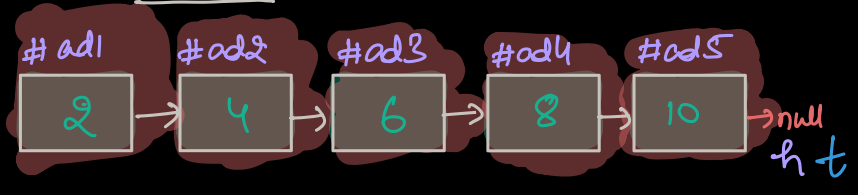
Idea: Insert nodes at the start one-by-one (20, 15, 10, 5).

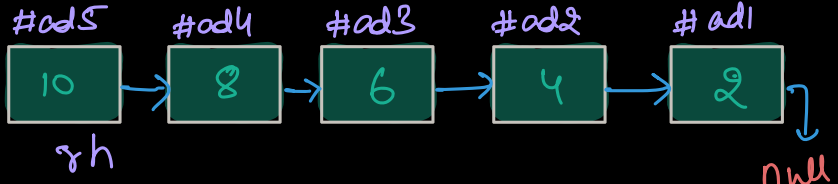
Can't create new nodes



obs:- If we insert nodes at the start, we get reversed list.

Trace






```
Node reverse(Node h) {
    Node rh = null
    Node t = h
    while(h != null) {
        h = h.next
        t.next = rh
        rh = t
        t = h
    }
    return rh
}
```

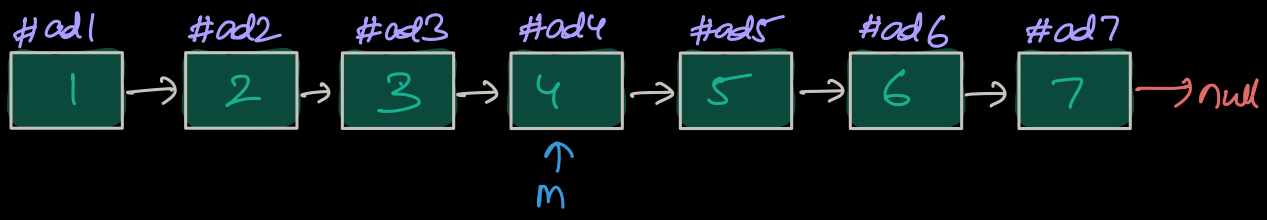
TC: O(N)

SC: O(1)

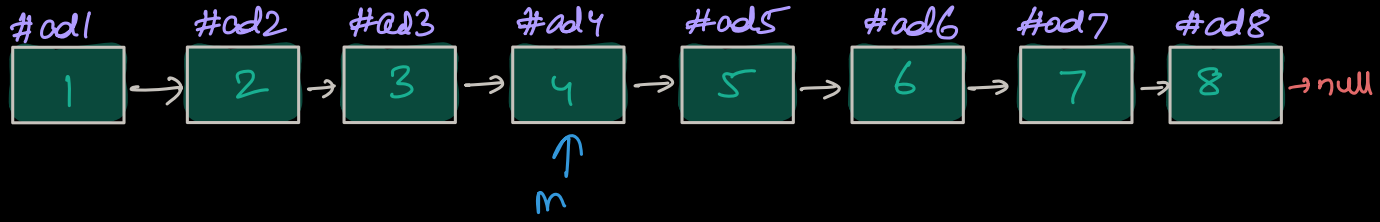
(V. Easy)

Q3: Given head of linked list, find the mid of the list.

Eg 1:



Eg 2:

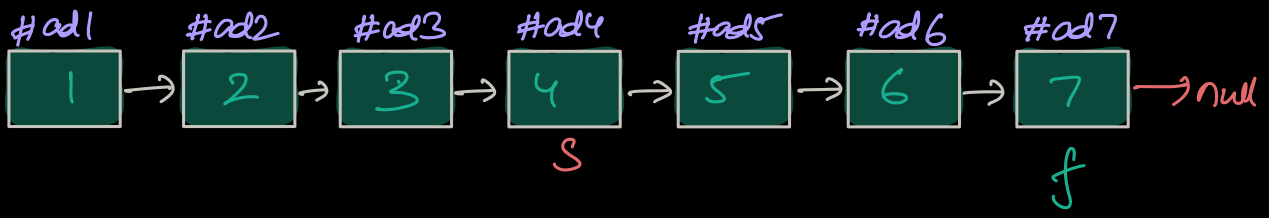


Idea:- * Iterate the list & get its size → n
* Again iterate from start & go to centre/mid

} TC: O(N)
SC: O(1)

⇒ 2 loops

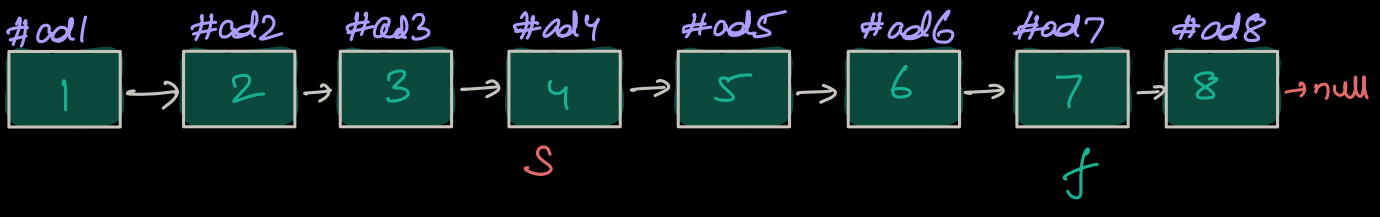
Idea 2: Find mid in a single loop.



slow moves once

fast moves twice

Obs 1: When f.next == null, STOP! return s as mid



Obs 2: When f.next.next == null, STOP! return s as mid

Hare - Tortoise Algorithm

```
Node mid(Node h) {
    if (h == null) { return h }
    Node s = h
    Node f = h
    while (f.next != null && f.next.next != null) {
        s = s.next
        f = f.next.next
    }
    return s
}
```

TC: O(n)

SC: O(1)