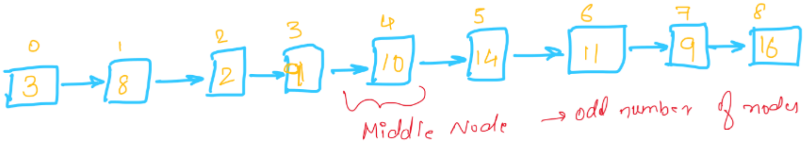
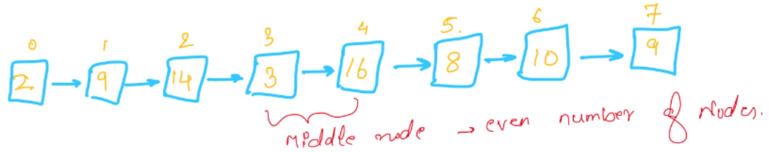
Linked List 2

Find middle of the Linked List





* If there are odd number of nodes there will be only one middle node.
But in case of even number of elements there will be 2 middle elements.

Brute force

1) Traverse & find total elements/nodes in linked list
2) Return $((N+1)/2)^{th}$ element.

$T.C = O(N)$

Optimized Approach Using Fast pointer

* Create two temp pointers slow and fast pointer.

```
slow = head;
fast = head;
```

* Move slow once & fast twice

```
slow = slow → next
fast = fast → next → next;
```

* Iterate while (slow → next != NULL && fast → next → next != NULL)
→ If (slow → next == NULL) return (slow → data);

* If head == NULL
return head

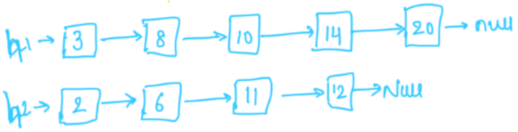
$T.C = O(N/2)$

Merge 2 sorted LL

★ Merge two Sorted linked list

→ Return a single sorted linked list

→ No extra space. Only rearrangement.



Brutefoce

→ keep creating a new node & assing values comparing both LL.

→ which so ever element is less assign to node & increment pointer.

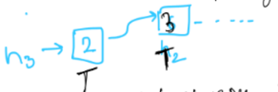
```
Temp t1 = h1 ;  Temp new-node ;
Temp t2 = h2 ;
while (h1 != NULL || h2 != NULL)
  if(h1.data < h2.data)
      new_node.data = h1.data ;  h1 = h1 → next ;
  else
      newnode.data = h2. data ;  h2 = h2 → next ;
```

T.C = O(N)

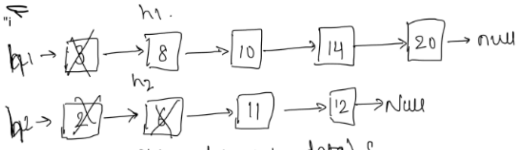
$\boxed{S.C = O(N)}$ X

Optimized Approach

- Create h_3, & assign to smallest value.



- Creat a temporary T node to keep pointing to curr index

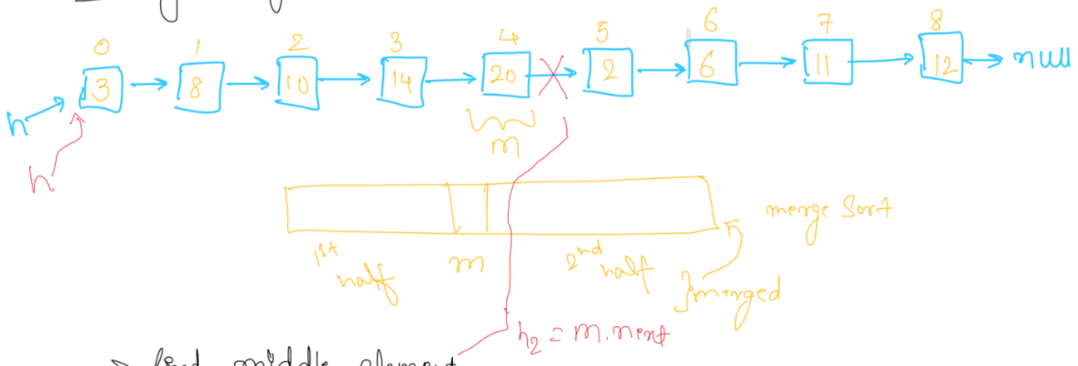


```
if(h1. data < h2.data) {
        T.next = h1
        h1 = h1. next
}       T = T. next ;
else {
        T.next = h2 ;
        h2 = h2.next ;
}       T = T.next ;
```

- Check while (h_1 != NULL && h_2 != NULL)
- if (h_1. != NULL)
    T.next = h_1;
- else (h_2 != NULL)
    T.next = h_2 ;
- return h_3

Sort LL using Merge Sort

Sort Using Merge Sort



→ find middle element

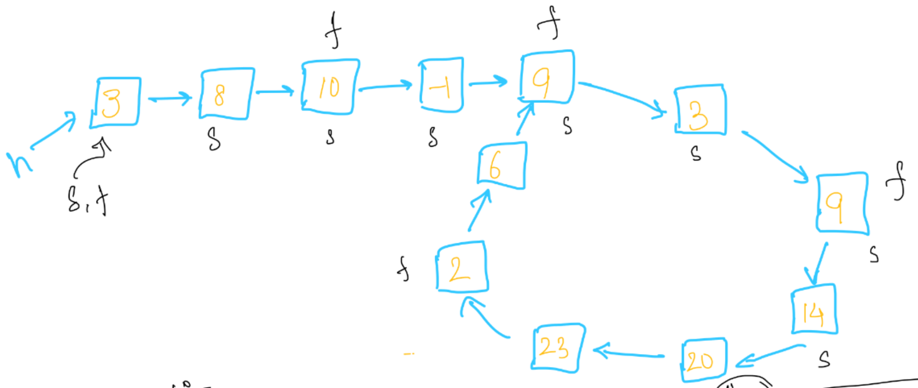
→ Assign h_1 = start ;

h_2 = m.next ;

Assignment :- Given a LL sort & return the head.

```
Node mergeSort (Node h1) {
     if (h1 == NULL) {return h1 }
     if (h1.next == NULL) {return h1}
     Node m = Center (h1); // Second last problm
     Node h2 = m.next ;
     m.next = NULL;
     h1 = mergeSort (h1) ;
     h2 = merge Sort (h2);
     return merge (h1, h2); // Last problem.
}
```

Detect Cycle in a LL



Approach 1:-

Store all references in a hashset ; f s

if we store same address twice, Cycle will be present.

if we insert null ;
cycle is not present ;

T.C = O(N)
S.C = O(N)   Hash set<Node>

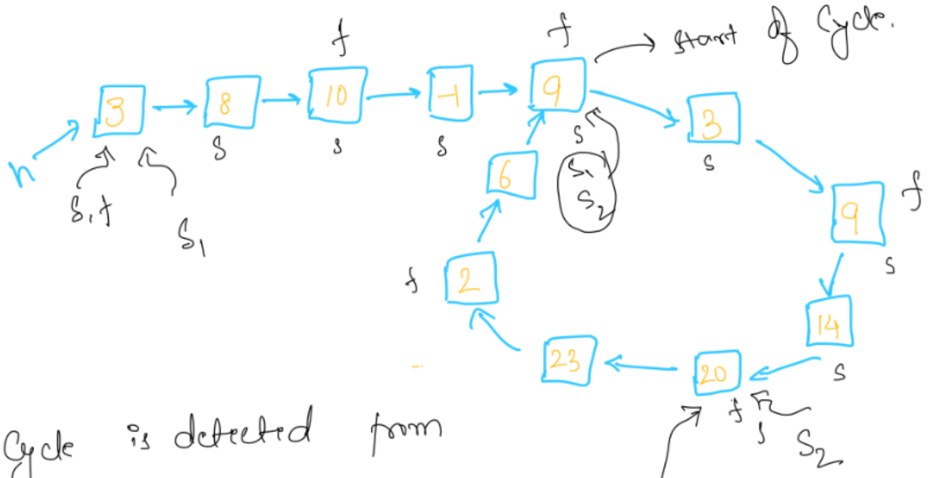
Approach 2

```
→ Create 2 pointers Slow & fast (s, f)
→ Assign both with h,
→ while ( f != NULL && f.next != NULL)
      if (s == f)
          return cycle;
      s = s → next;
  }   f = f → next;
```

T.C = O(N)
S.C = O(1)

Find start of a cycle

4 9:07 PM



→ Once Cycle is detected from above problem.

→ Create two pointers s_1 & s_2

```
    s1 = h
    s2 = intersection
```

→ keep moving one by one

```
    s1 = s1.next
    s2 = s2.next
```

→ Until

```
    (s1 != s2)
```

→ if $(s_1 == s_2)$

```
    return (s1.data || s2.data)
```