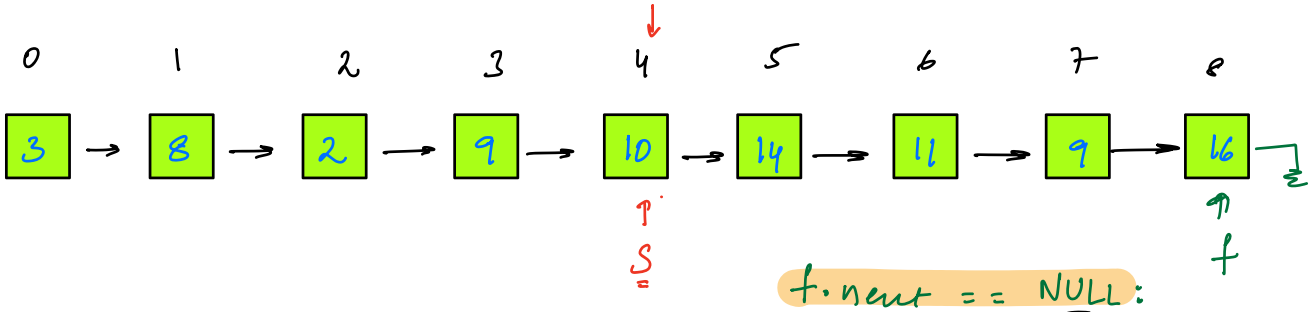
Today's Content:

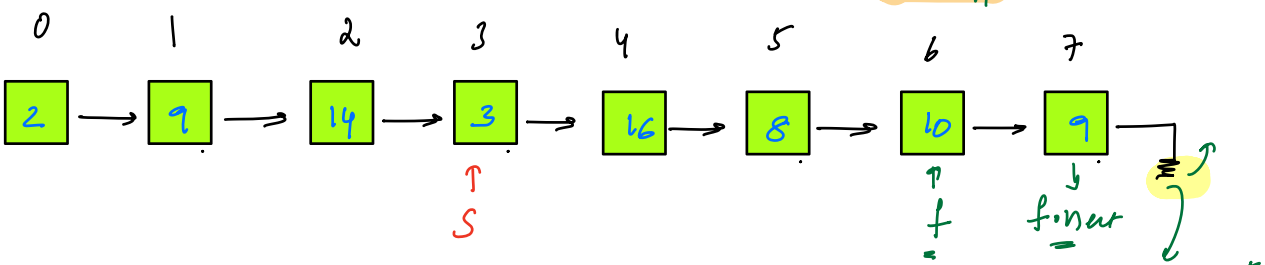
- → Find Middle ✓
- → Merge 2 sorted arrays ✓ → using dummy node
  → without dummy node
- → Merge Sort / Merge N Sorted Links → ✓
- → Re-arrange → {TODO} → Assignment
- → Cycle detection ✓
- → Loop detection → {Proof in Doubts Session}

Find Middle

fast pointer → 2 steps slow pointer → 1



f.next == NULL : break



f.next.next == NULL : break

Sol1

1) Given linked list

1 ↙ { N = size(head) }, N >= 1

2 ↙ { get k^Th Elem(head, N/2) } → C2
    { get k^Th Elem(head, (N-1)/2) } → C1

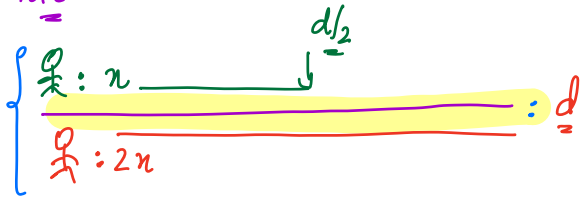
2 Iterations

Q) If odd get center

If even get C1 i.e. first center

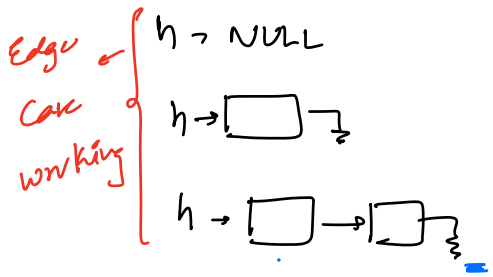
Note: In a single iteration

Idea



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

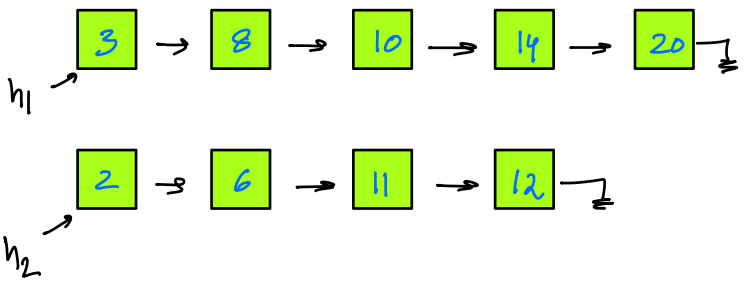
Edge case working

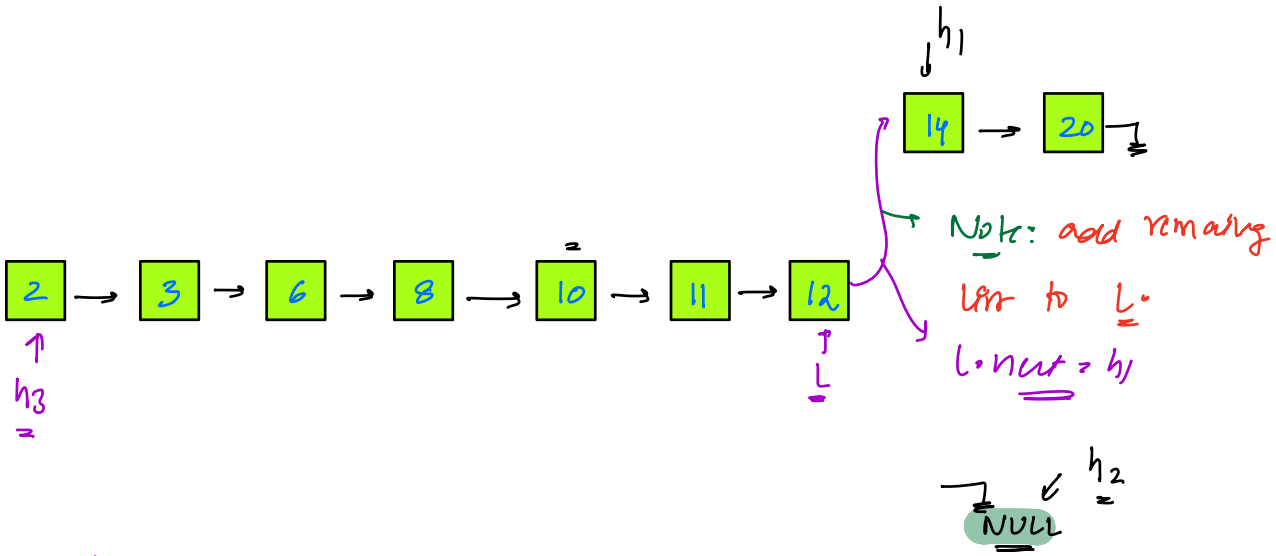


Merge 2 Sorted Linked Lists

} return a single sorted linked list
No Extra Space, only re-arrangement

Ex:

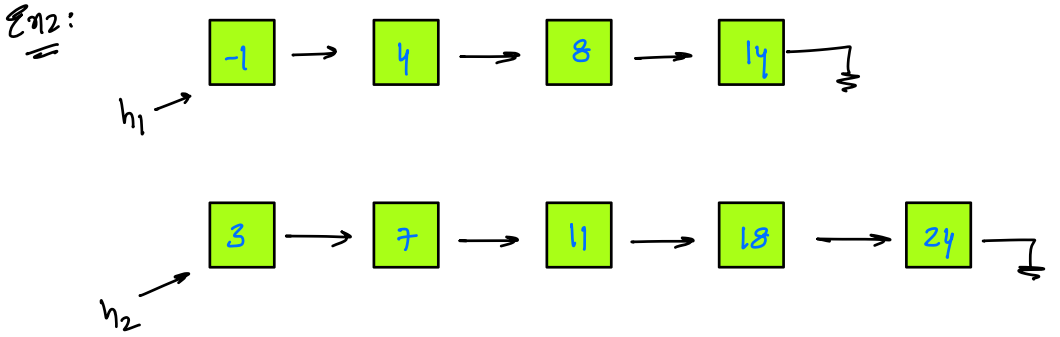


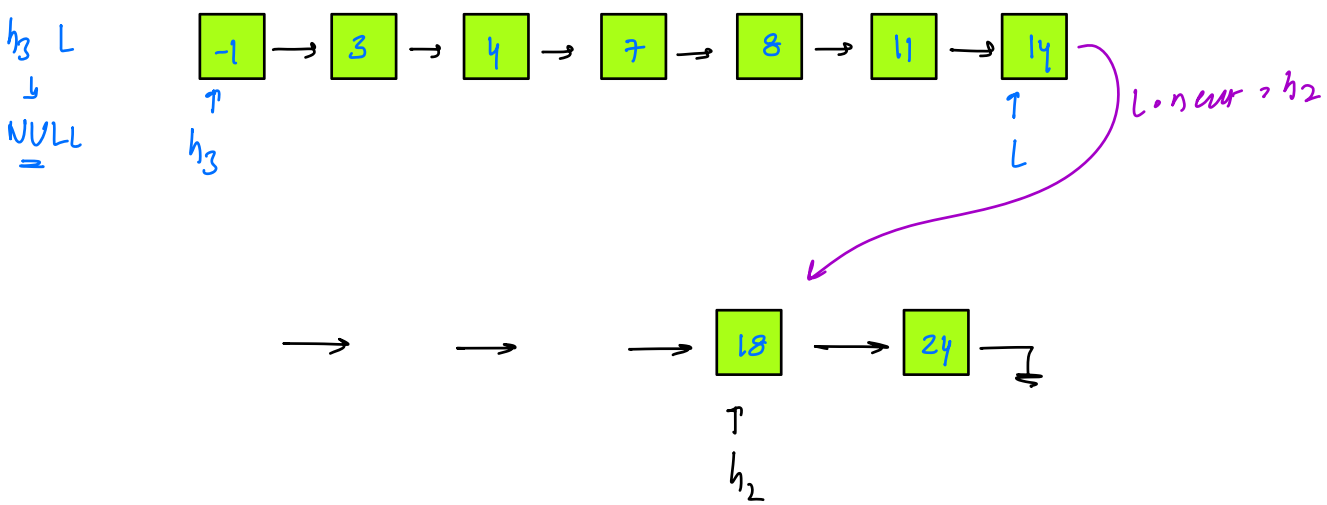



```
if( h1.data < h2.data){
    Add h1 at back
    L.next = h1
    h1 = h1.next
    L = L.next
}
```

```
else {
    Add h2 at back
    L.next = h2
    h2 = h2.next
    L = L.next
}
```

Eg2:

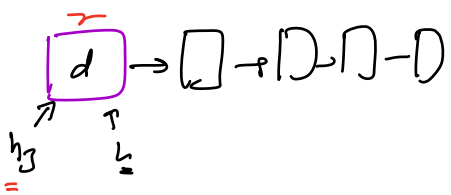




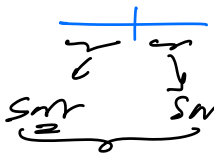
Note: Avoid usage of dummy node.

{ Even though it's a single node, if we run same function N times it will create N dummy nodes }

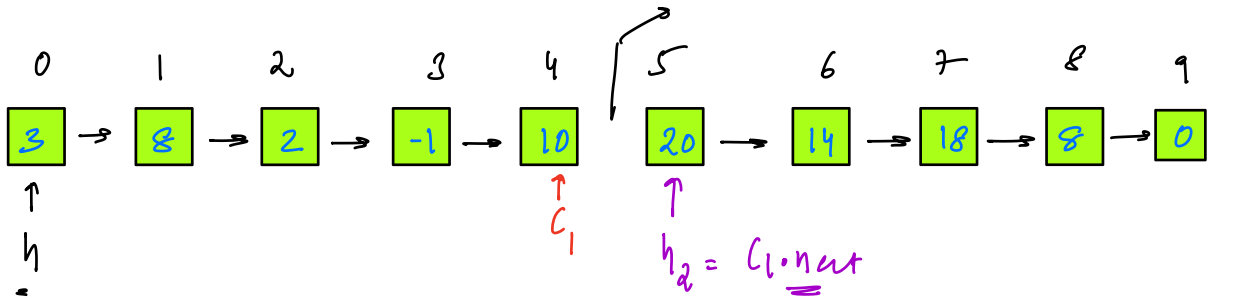
```
Merge (Node h1, Node h2){
    if (h1 == NULL) {return h2}
    if (h2 == NULL) {return h1}        // Edge Cases

    Node h3 = NULL, L = NULL
    if (h1.data < h2.data) { h3 = h1, L = h1, h1 = h1.next }
    else { h3 = h2, L = h2, h2 = h2.next }
    or
    Node h3 = new Node(-1);
    L = h3

    while (h1 != NULL && h2 != NULL) {
        if (h1.data < h2.data) { Add h1 at back }
        else { Add h2 at back }
    }

    // Adding remaing part
    if (h2 != NULL) { L.next = h2 }
    else { L.next = h1 }

    // return head of sorted Linked List
    return h3   // if we are using dummy node
                return h3.next
}
```

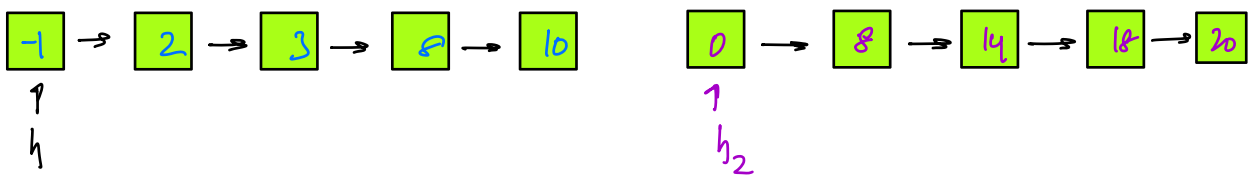


Merge Sort →



sort sort & merge break $c_1.next = NULL$






```
Node mergeSort(Node h){
    if(h == NULL){return h}
    if(h.next == NULL){return h}
    Node c1 = center(h);
    Node h2 = c1.next
    c1.next = NULL
    h = mergeSort(h)
    h2 = mergeSort(h2)
    return merge(h, h2)
}
```

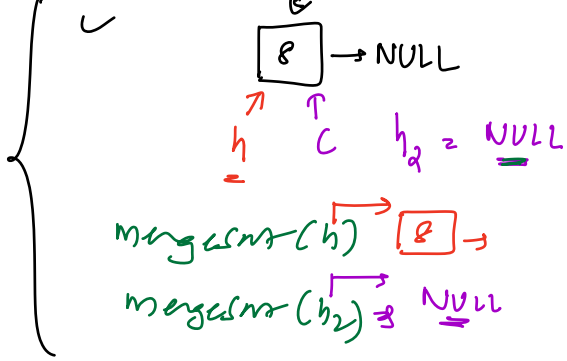
Leave Recursive stack

SC = ↳ O(1)

∞ loop → stack overflow

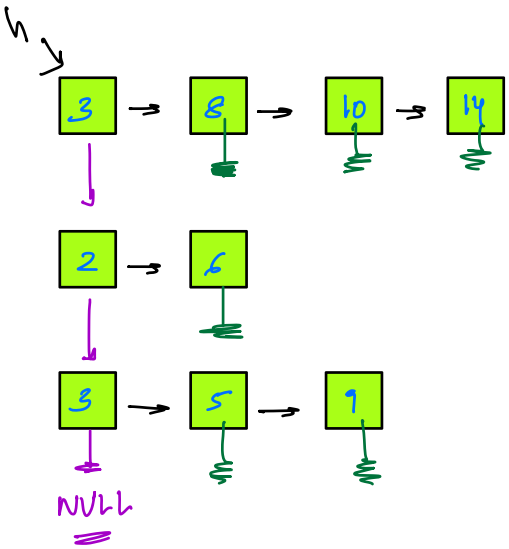
Linked List Edge Cases

1) No Element, 1 Element, 2 Elements

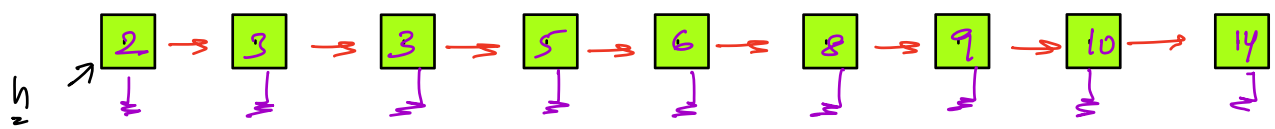


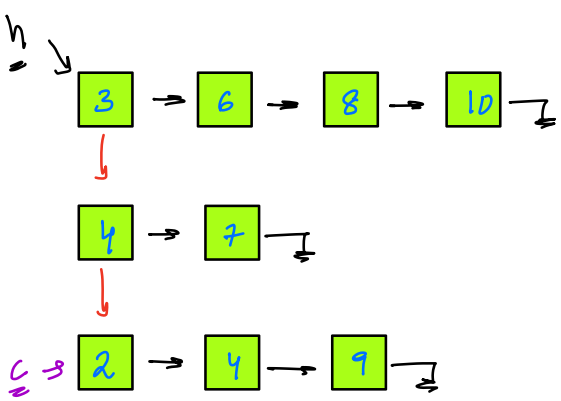
308 Given N Sorted Linked Lists merge all of them.

```
class Node {
    int data;
    Node right
    Node down;
    Node(int x) {
        data = x
        right = down = NULL
    }
}
```

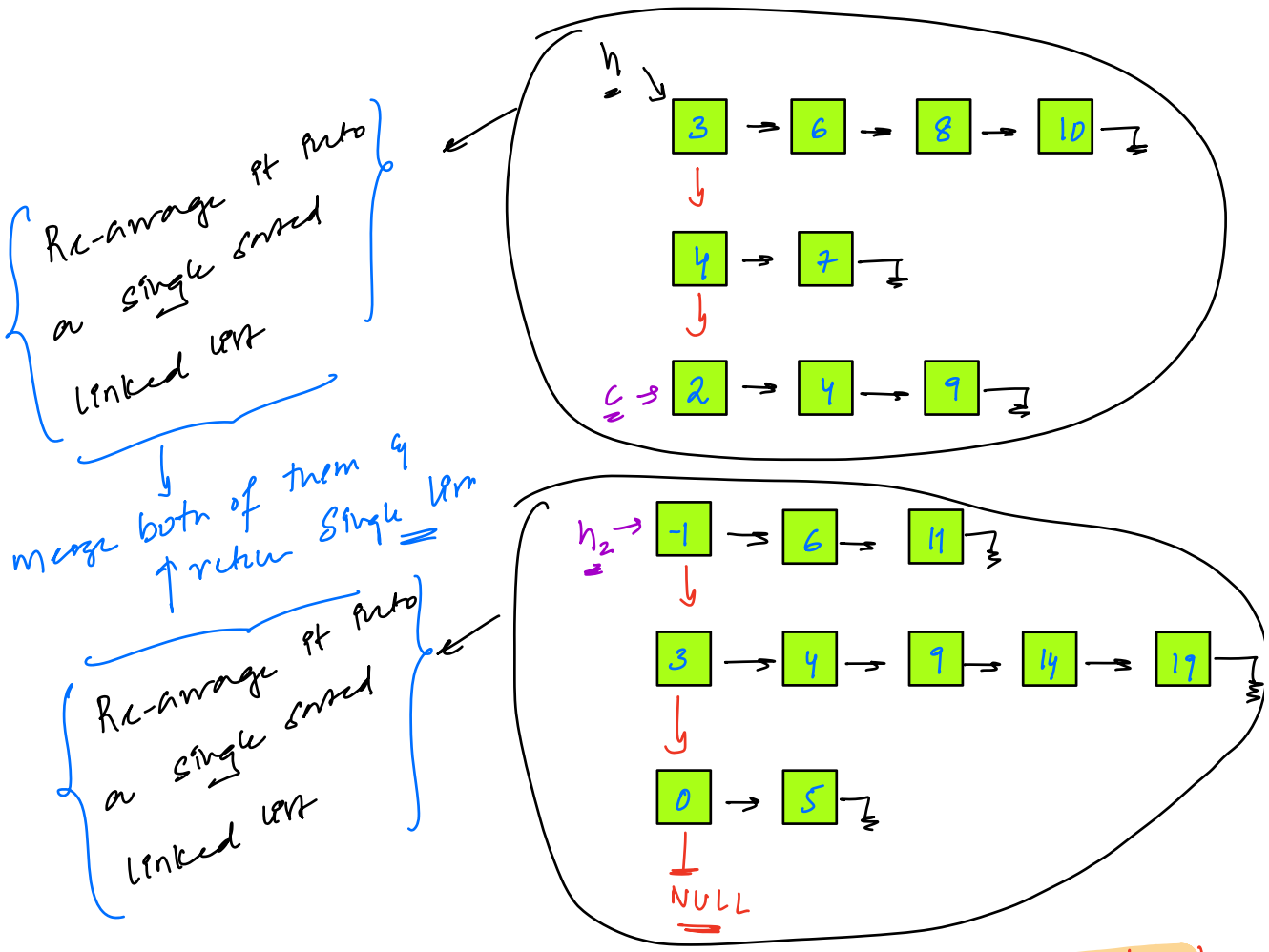


Sol: → Flatten Sorted Linked List





→ merge a return a single sorted LinkedList



↦ {mergeSort in N Sorted Lists}

// ——— mergeNSort(Node h){

if (h == NULL || h.down == NULL) {return h}

Node c = center(h) ↦ using down reference

Node h2 = c.down

c.down = NULL

h = mergeNSort(h)

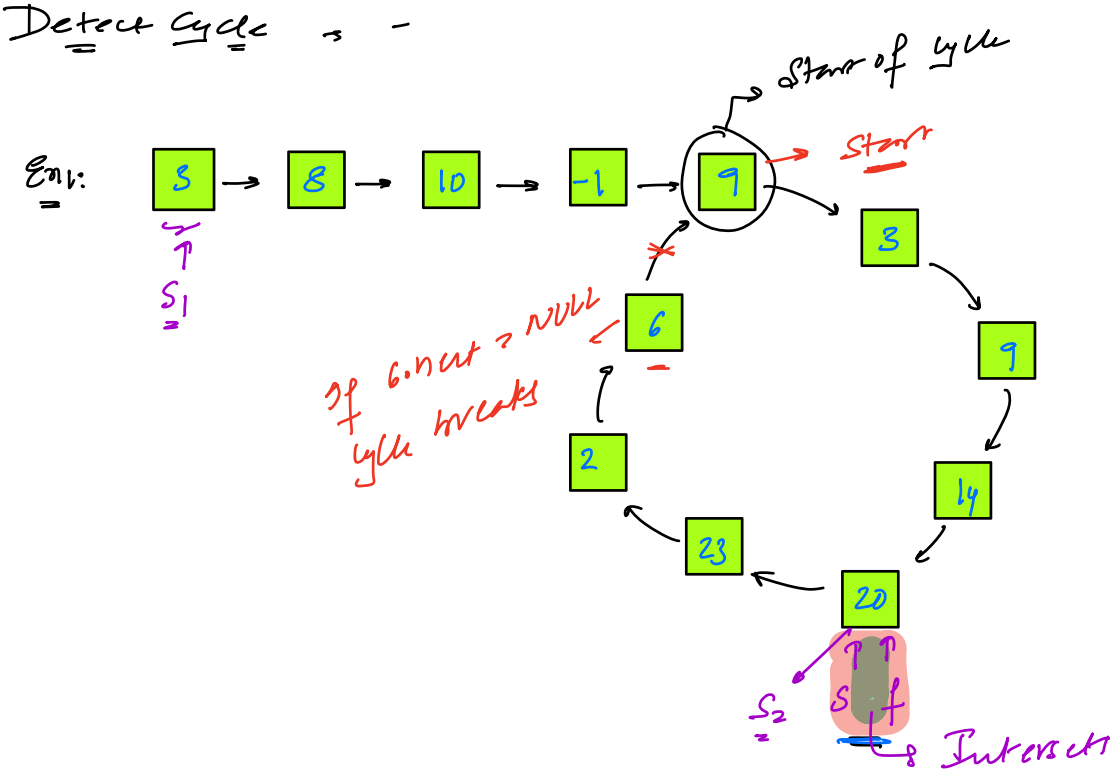
h2 = mergeNSort(h2)

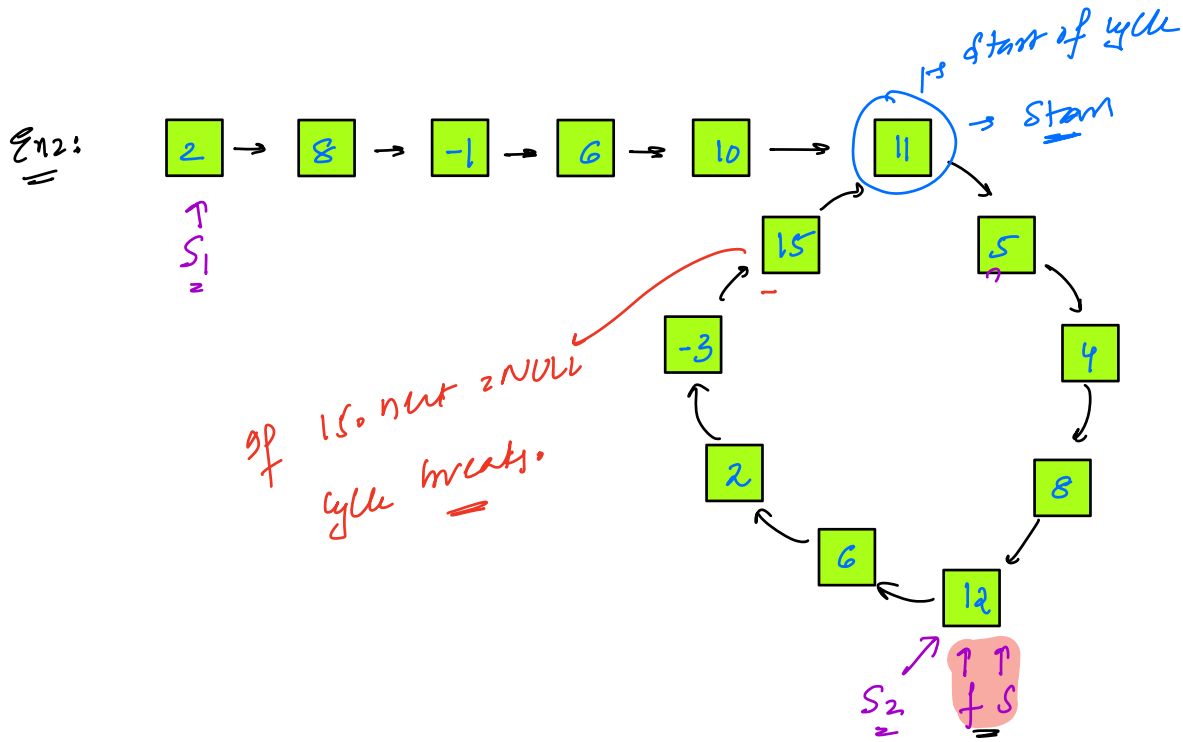
return merge(h, h2) ↦ using right reference

TC as TODO ?

}

Detect Cycle :-





Approach 1 : Store all references HashSet ?

TC: O(N)
SC: O(N)

→ If we insert same address 2 times?
→ cycle is present

→ If we insert temp = NULL
cycle is not present

HashSet < Node >

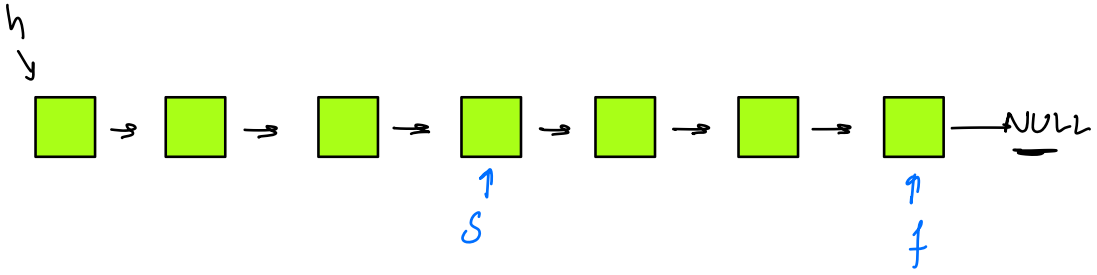
How to store Node reference in your language of choice?

Approach 2 : s = h, f = h

If they is a cycle s == f

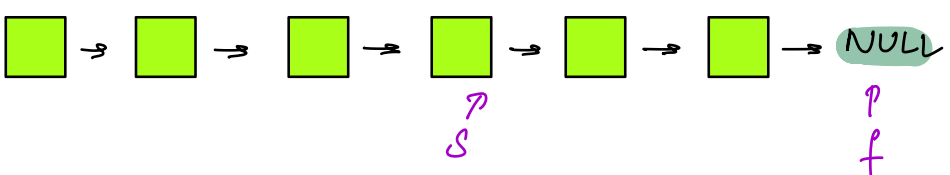
If f == NULL ∧ f.next == NULL : no cycle

Eg 1:



if (f.next == NULL)
no cycle

Eg 2:



if (f == NULL)
no cycle

— Detect Cycle (Node h) {

// TODO

→ Check cycle there or not? ✓

→ If cycle is there ✓

→ You have to get start of cycle?

→ $S_1 = h$, $S_2 =$ (keep at intersection)

update both slow pointer till they are same

They will meet at start of cycle

→ You need to remove the cycle?

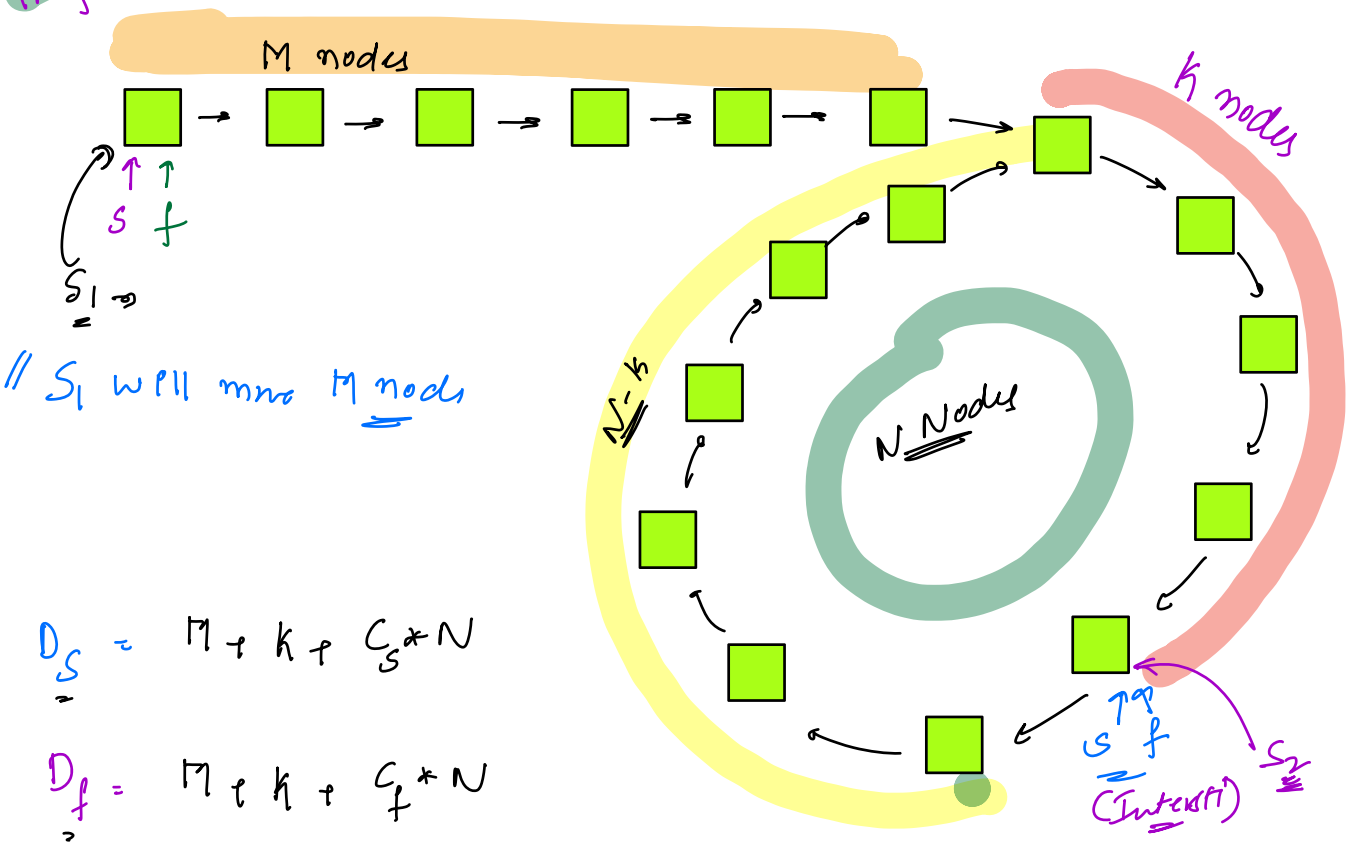
Private & get node when

node.next == Start of Cycle

{ node.next = NULL to
remove cycle

}

Proof of concept → { Explain from Start }



// S_1 will move M nodes

$D_s = M + K + C_s * N$

$D_f = M + K + C_f * N$

$D_f = 2 D_s$

$(M + K + C_f * N) = 2 (M + K + C_s * N)$

$M + K + C_f N = 2M + 2K + 2 C_s N$

$C_f N - 2 C_s N - K = M$

$(N)(C_f - 2C_s) - K = M$ → C

$M = (N)(C) - K$

$M = (C-1)(N) + (N-K)$

S_2 is at intersection + $(N-K)$

// finally Both S_1 & S_2 are meeting at start