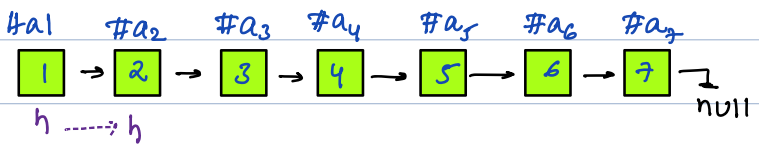
Todays Content:

a) Merge
b) Re-arrange Linked list
c) Cycle detection
    i) Detect cycle
    ii) Find start of cycle
    iii) Remove cycle

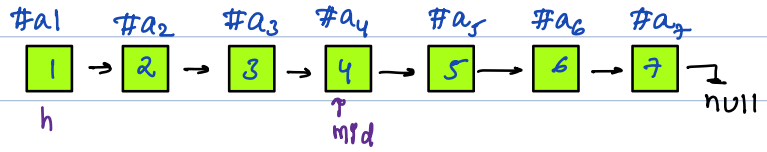
Diagrams:



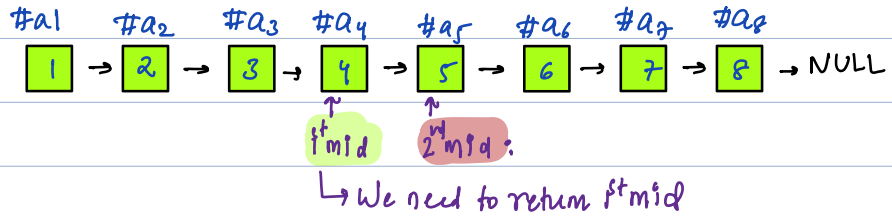
print(h) = a1
print(h.data) = 1
print(h.nent) = a2
h = h.nent

18) Given head node find mid of linked list

Ex1:



Ex2:



↳ We need to return 1st mid

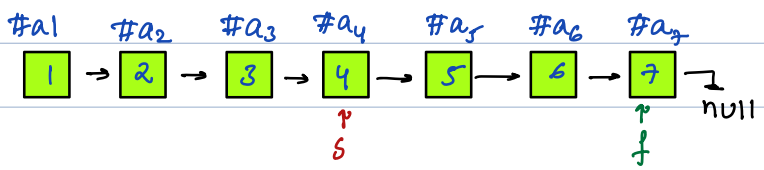
Idea1: Iterate on linked list, get length say = N } Using 2 independent loops.
Iterate till ≈ N/2 & get mid

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

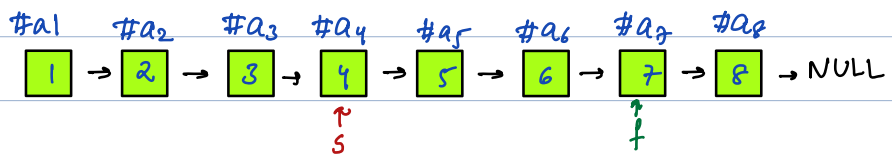
Idea2: Usage of slow and fast pointer

slow = slow.next // 1 step

fast = fast.next.next // 2 steps



Odd length: if(f.next == null) { s is on mid}



Even length: if(f.next.next == null) { s is on mid}

Note: Use both conditions, once loop stops, s is on mid

```
Node mid(Node h){ TC: O(N) SC: O(1)
    Node s=h, f=h;
    if(h==NULL){return NULL}
    while(f.next!=NULL && f.next.next!=NULL){
        s=s.next;
        f=f.next.next
    }
    return s;
}
```

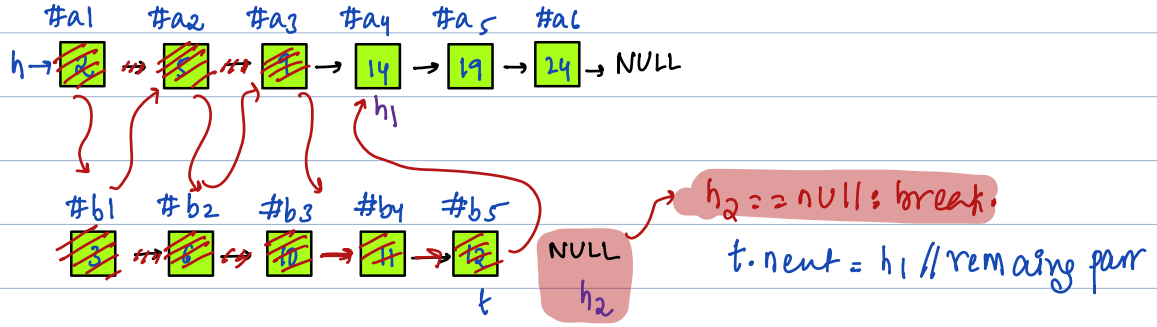
Note: Even, if one of the condition fails we stop

fail

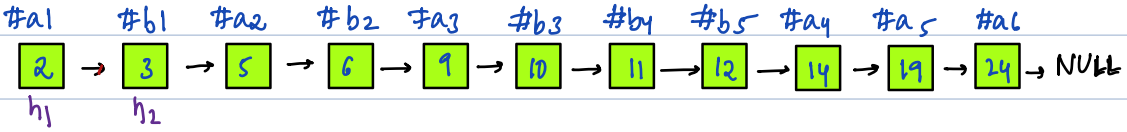
20) Given 2 sorted linked lists, Merge & get final sorted list

TC: SC:

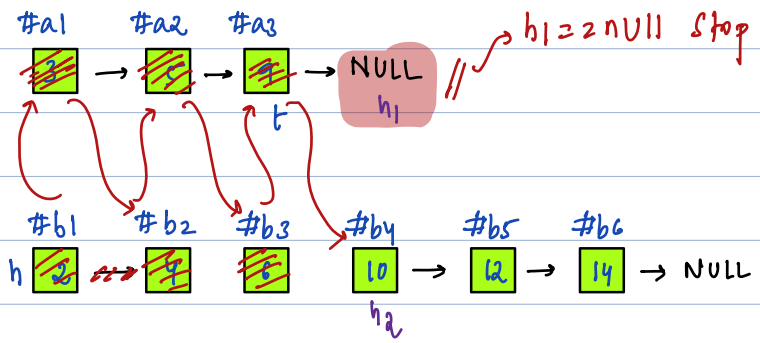
En1:



final:



En2:



Pseudo Code:

```
Node merge(Node h1, Node h2){ TC: O(N+M)  SC: O(1)
    Node h = null, t = null;
    if(h1 == null){return h2}
    if(h2 == null){return h1}

    if(h1.data < h2.data){ h = h1, t = h1; h1 = h1.next}
    else{ h = h2, t = h2, h2 = h2.next}
    while(h1 != NULL && h2 != NULL){
        if(h1.data < h2.data){
            t.next = h1;
            h1 = h1.next
        }   t = t.next
        else{ //>=
            t.next = h2;
            h2 = h2.next
        }   t = t.next
    }
    // Add leftout part?
    if(h1 != NULL){ t.next = h1}
    if(h2 != NULL){ t.next = h2}
    return h;
}
```

10:20 break

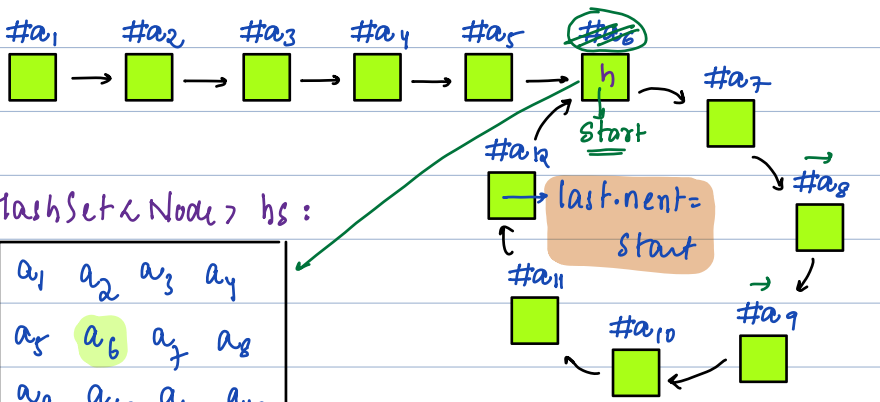
30) Given a head node of LinkedList, check for cycle detection?

Note: When last node points to any prev node, thats when cycle formed.

Note1: If no cycle return NULL → {last node ⇒ NULL} & }

Note2: If cycle is there, remove cycle & return start node of cycle

Ex1:



Idea: HashSet<Node> hs :

a_1	a_2	a_3	a_4
a_5	a_6	a_7	a_8
a_9	a_{10}	a_{11}	a_{12}

Obs: 1. If address is already present in hashset

a) Cycle present

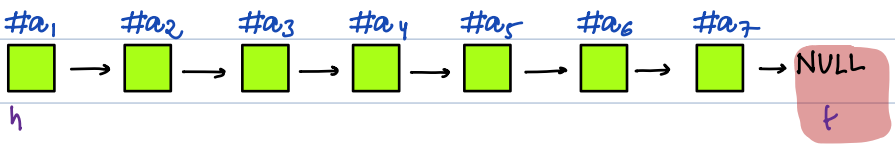
b) 1st repeating node = Start of cycle

c) Iterate & Identify last node = last.next = start of cycle

last.next = null // Break cycle

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

Ex2:



Idea: HashSet<Node> hs :

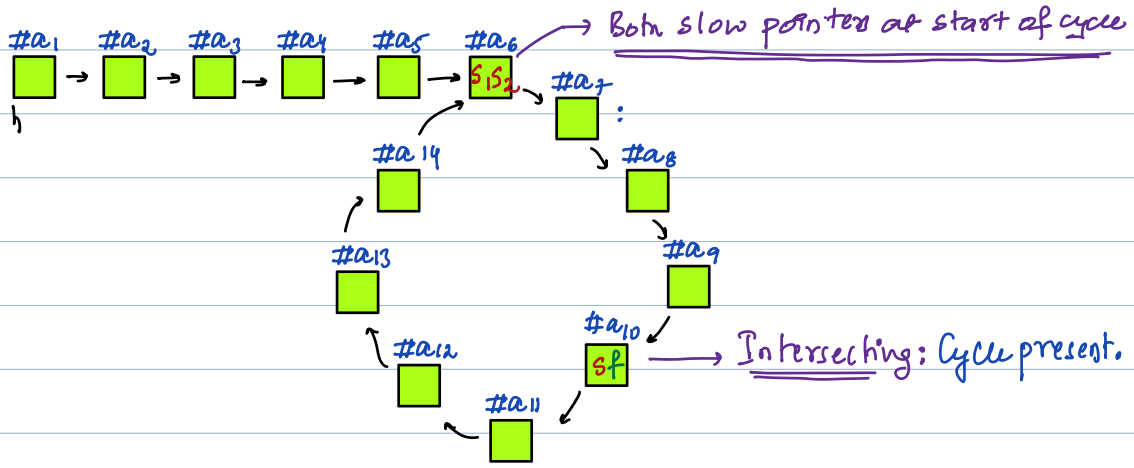
a_1	a_2	a_3	a_4
a_5	a_6	a_7	

Note: In above approach, if we reach NULL, In that case no cycle

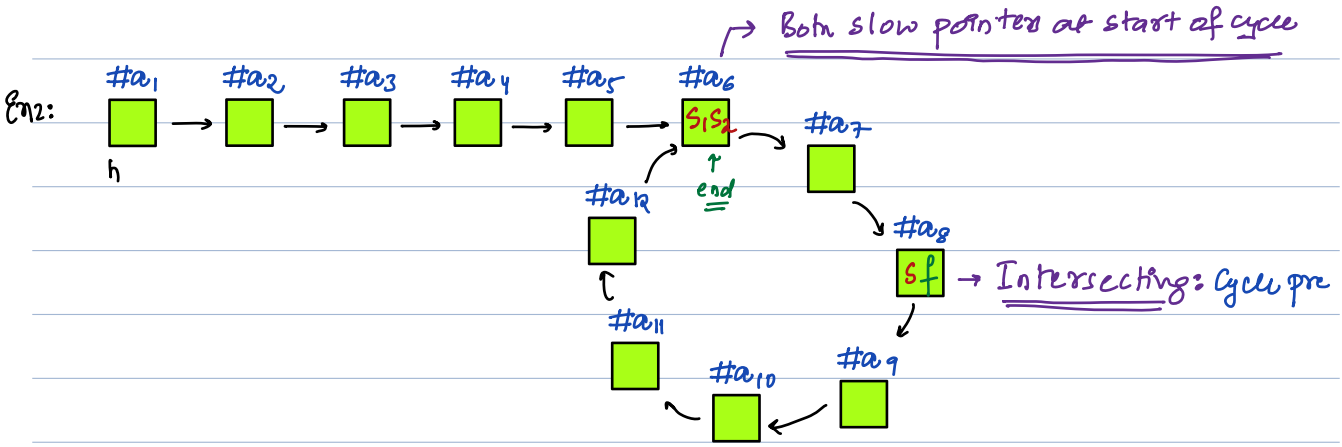
Idea2: In a circle, if 🚶: 1 step 🚶: 2 steps, irrespective of their initial distance, they will meet because, distance between decreases by 1 step for each iteration in Circle.

Idea 2:

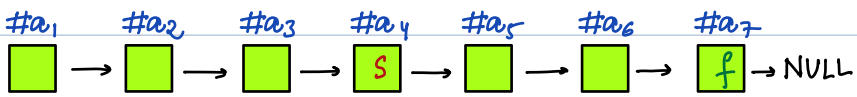
Eg1:



Eg2:

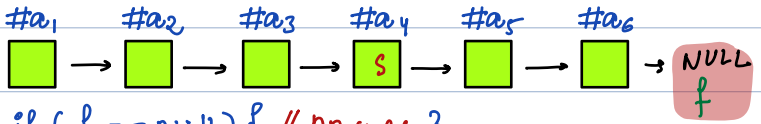


Eg3:




```
if (f.next==null) { // no cycle }
```

Eg4




```
if (f==null) { // no cycle }
```

Steps:

1. s = h && f = h, keep updating s & f till they Intersect or end
2. S_1 = h && S_2 = Intersection, keep updating till they are same = Start
3. Using start get end of cycle & break cycle & return start

Q): Given linked list if cycle present.

: remove cycle : last node next = NULL

: return start of cycle :

If cycle not present : return null

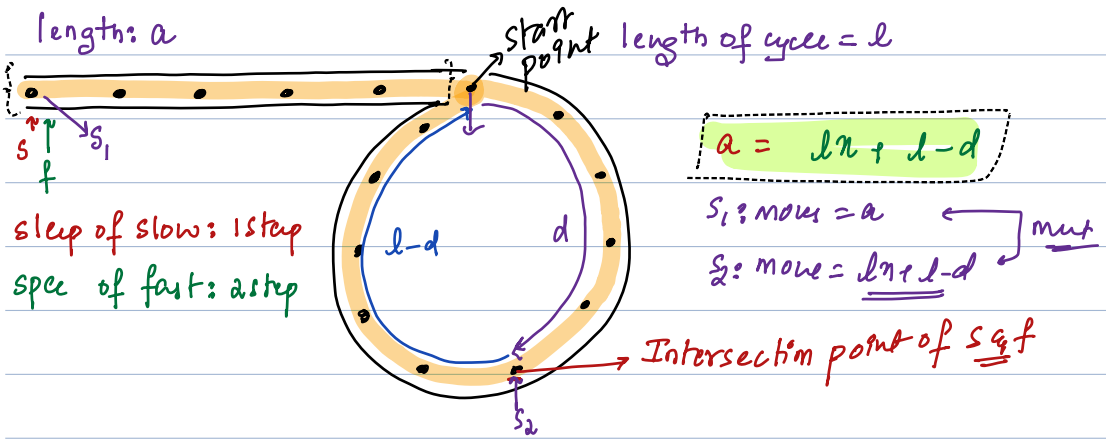
```
Node detectCycle(Node h){ TC: O(N) SC: O(1)
    Node s=h, f=h
    boolean isCycle = false;
    while(f!=NULL && f.next!=NULL){          } cycle detection
        s = s.next
        f = f.next.next
        if(s==f){ isCycle = true; break}
    }
    if(isCycle==false){ return null}

    Node s1=h, s2=f;                         } getting start
    while(s1!=s2){ // Why intersect at start?
        s1=s1.next; s2=s2.next
    }

    Node start=s1, last=s1;                  } getting last
    while(last.next!=start){
        last = last.next
    }

    last.next = null                         } breaking cycle
    return start;
}
```

Start of cycle? {proof : doubt session}



$a = ln + l - d$

s_1 : moves = a ← must

s_2 : move = $ln + l - d$

Note: Since both move for same amount of time

ds : distance by slow pointer = $a + d$

df : distance by fast pointer = $a + cl + d$ ↳ // no: of times fast pointer iterates in circle?

$df = 2ds$

$a + cl + d = 2a + 2d$

$cl - d = a$

$a = cl - d \rightarrow cl - d + l - l = cl - l + l - d$

$a = cl - l + l - d$

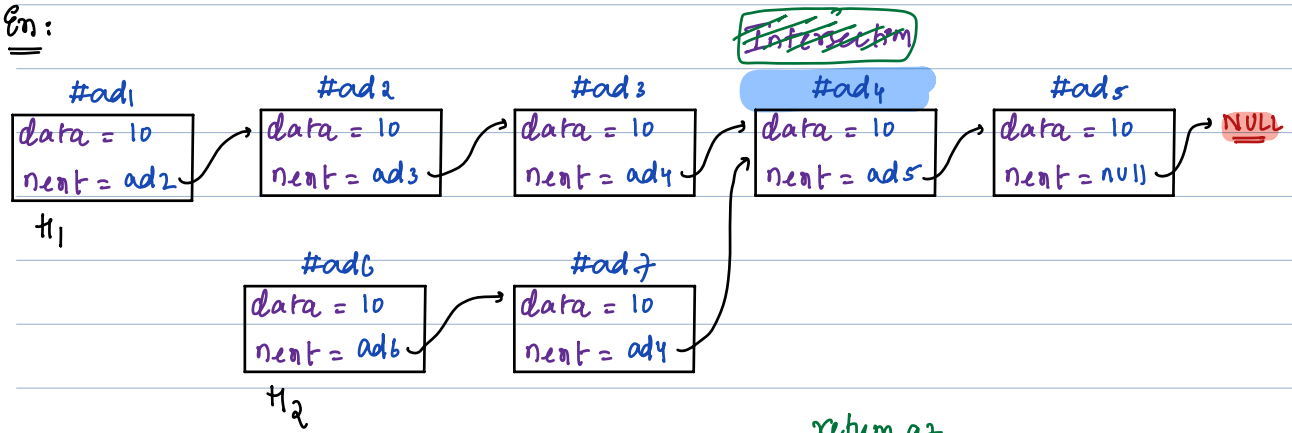
$a = l(c-1) + l - d$ // $c - 1 = n$

$a = ln + l - d$

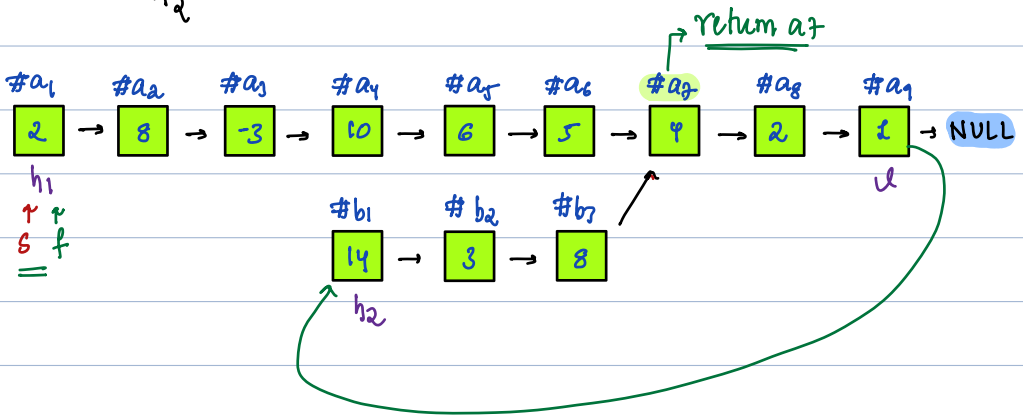
48 Given 2 linked lists return their intersection point?

Note: Cannot make any changes in linked list.

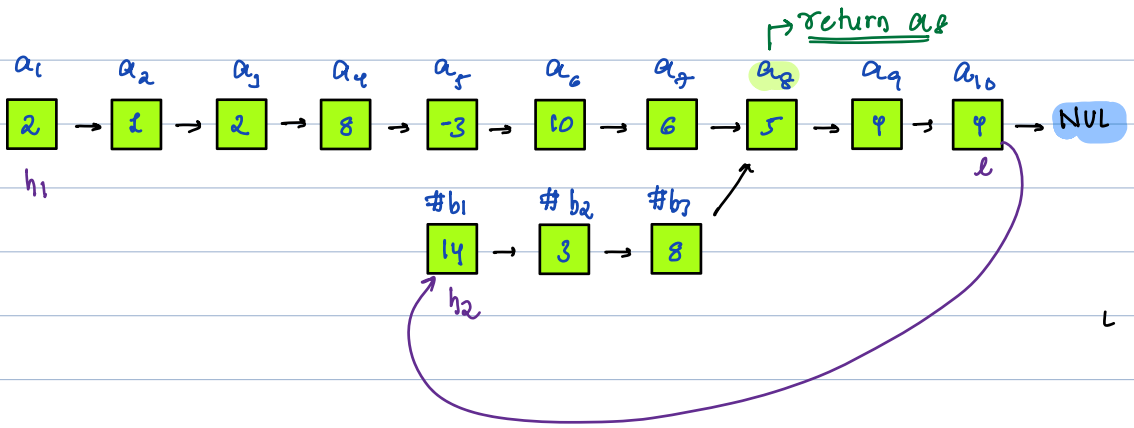
Ex:



Ex1:



Ex2:



```
boolean Intersect( Node h1, Node h2) {   TC: O(N+M)  SC: O(1)

  // Step 1:
     Node t = h1;
     while(t.next != null) { t = t.next }

  // Step 2:
     t.next = h2

     Node s = h1, f = h1;                          // Cycle detection
     boolean isCycle = false;
     while( f != NULL && f.next != NULL) {
        s = s.next
        f = f.next.next
        if(s == f) { isCycle = true; break }
     }

     // Undo change:
        t.next = null
     if ( isCycle == true) { return true }
     else { return false }
}
```