Lecture :- LL-2

Agenda

- Middle of LL
- Merge sort LL
- Detecting cycles
  - Detect
  - find
  - Remove.

Class : 7:08 AM

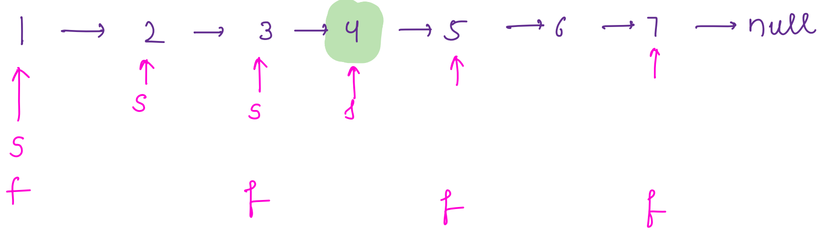
Qu.1 Given a LL, find middle of LL.

Ex: 1 → 2 → 3 → 4 → 5 → 6 → 7 → null

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → null

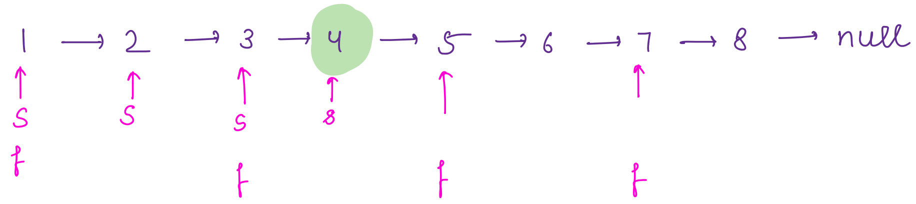
Constraint:- Do it in 1 itr.

Idea: slow and fast pointer



Odd:- when f.next == null,

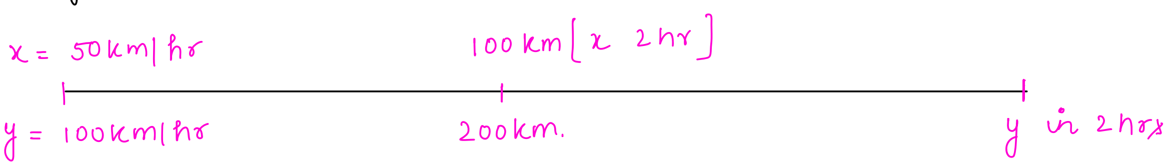
s is at mid



Even: when f.next.next == null

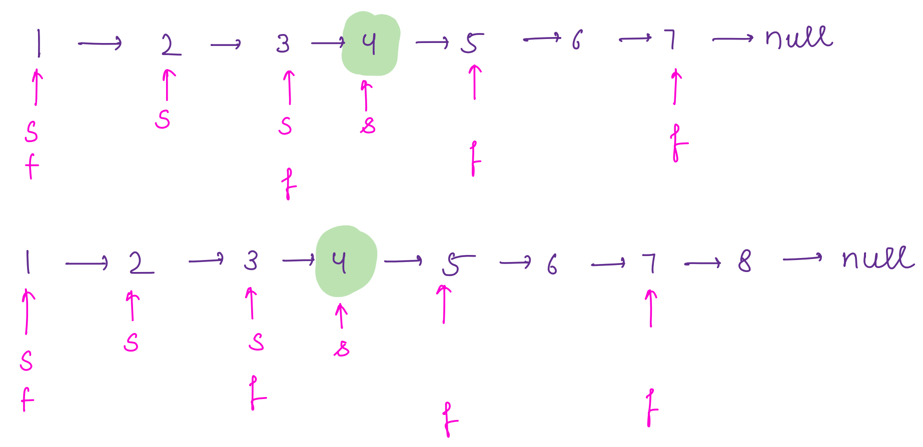
s is at mid.

Logically



```
Node middle( Node h) {
    if( h == null) {
        return null;
    }
    if( h. next == null) {
        return h;
    }
    Node s = h;
    Node f = h;
                            odd                          even
                            ↑                             ↑
    while( f. next != null  &&  f.next . next != null) {
        s = s. next ;
        f = f.next.next;
    }
    return s;
}
```

Dry run:



1 → 2 → 3 → 4 → 5 → 6 → 7 → null
s
f
s
s
f
s
f
f
1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → null
s
f
s
s
f
s
f
f

Qu2   Merge two sorted LLs.

Ex: LL1: 2 → 5 → 9 → 14 → 19 → null
↑
h1

LL2: 3 → 6 → 10 → 11 → 12 → null
↑
h2

Ans: 2 → 3 → 5 → 6 → 9 → 10 → 11 → 12 → 14 → 19 → null

LL1: 2 → 5 → 9 → 14 → 19 → null
↑ h1 ↑ h1 ↑ h1

LL2: 3 → 6 → 10 → 11 → 12 → null
↑ h2 ↑ h2

Node h = head of final LL [h1 = 2]

h = h1. ② , t = h1 ②

```
if( h2.data < h1.data) {
      ~~h~~ t.next = h2;        ② ⟶ ③
      h2 = h2.next;         // 6
      ~~h~~ t = ~~h~~ t.next;       // 3
}

if (h1.data < h2.data) {
      ~~h~~ t.next = h1;        ② ⟶ ③ ⟶ ⑤
      h1 = h1.next;         // h1
      ~~h~~ t = ~~h~~ t.next;       // 5
}
```

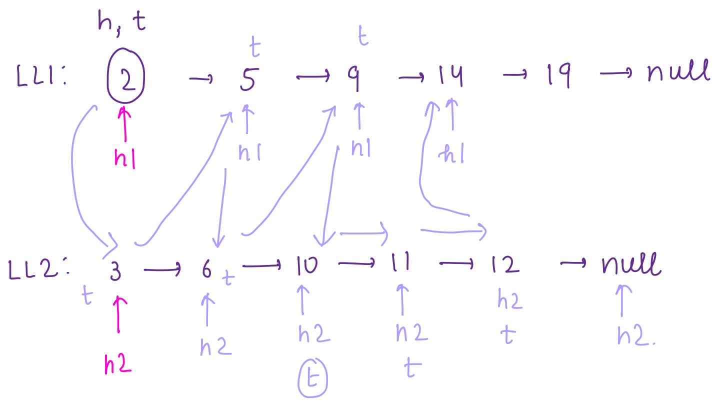
A[] = 1 3 5
      i

B[] = 2 4 6
      j

C[]

```
if (A[i] < B[j]) {
      C[K] = A[i];
      i++;
      K++;
}
if( B[j] < A[i]) {
      C[k] = B[j]
      j++;
      k++;
}
```

Eg 2:



h, t
LL1: 2 → 5 → 9 → 14 → 19 → null
t
t
h1
h1
h1
h1
LL2: 3 → 6 → 10 → 11 → 12 → null
t
t
h2
h2
h2
h2
h2
t
t
t
h2.

```
Node merge( Node h1, Node h2) {
        if( h1 = = null    && h2 = = null) {
              retum  null;
        }
        if  (h1 == null) {
              retum h2;
        }
        if  (h2 == null) {
              retum h1;
        }
        Node  h = null ,  t = null;
              ↑
           head of final LL
        if ( h1. data < h2. data)  {
                h = h1;
                t = h1;
                h1 = h1. next;
        } else {
                h = h2;
                t = h2;
        }       h2 = h2. next;
```

LL1 : null

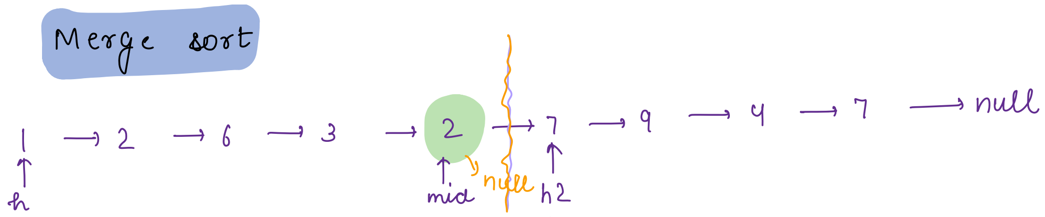
LL2: 2 → 3

```
    while ( h1 != null  &&  h2 != null) {

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

    if ( h1 != null) {
        t.next = h1;                      Remaining nodes.
    } else {
        t.next = h2;
    }

    return h;
}
```

TC: $O(n) \simeq O(m+n)$

SC: $O(1)$

Qu **Merge sort**



Step1: find the middle element.

mid = middle(h).

Part1: [h, mid] → mid. next = null

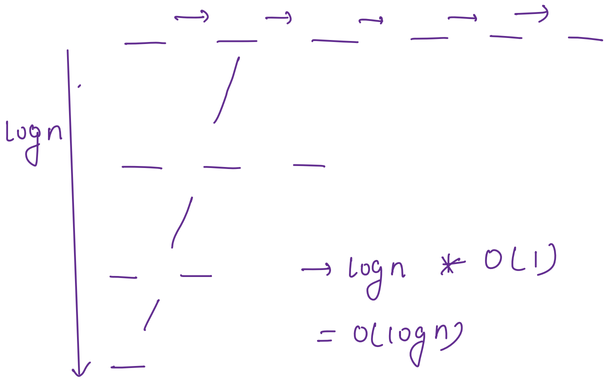
Part2: [h2, null] → h2 = mid. next.

```
Node mergesort( Node h) {
    if( h == null || h.next==null) {
        return h;
    }
    Node mid = middle( h);
     h2 = mid. next   // Part 2
     mid. next = null
     Node t1 = mergesort (h);
     Node t2 = merge sort (h2);
     Node t3 = merge(t1, t2);
     return t3;
}
```

TC: O(nlogn)

SC: O(logn)

SC.

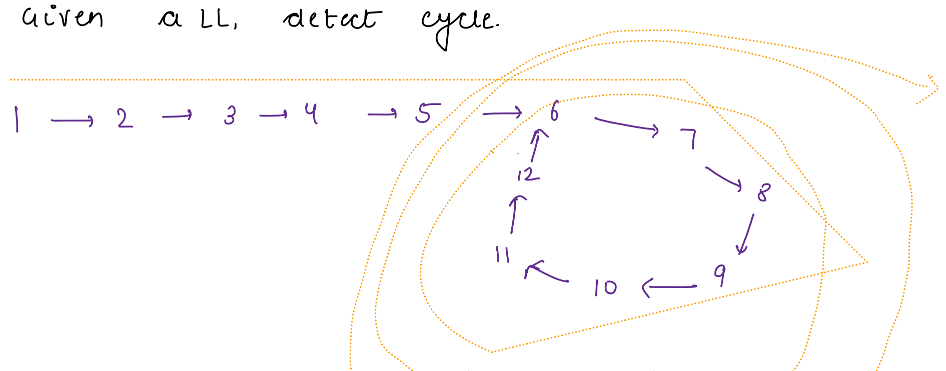


→ logn * O(1)

= O(logn)

Break: 8:25 AM

Qu Given a LL, detect cycle.

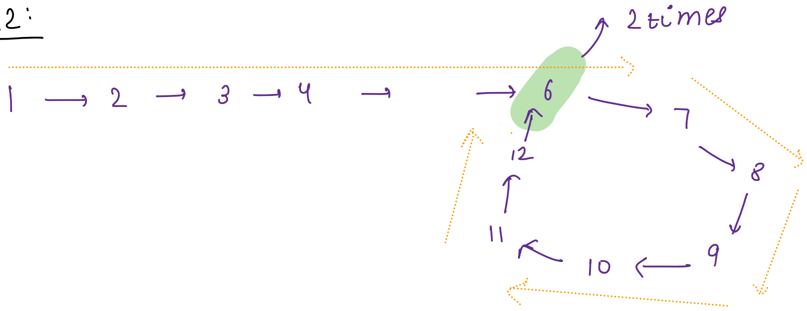


Idea1: Traverse LL [Bad idea]

head == null
[No cycle]

Cycle
[infinite loop]

Idea2:

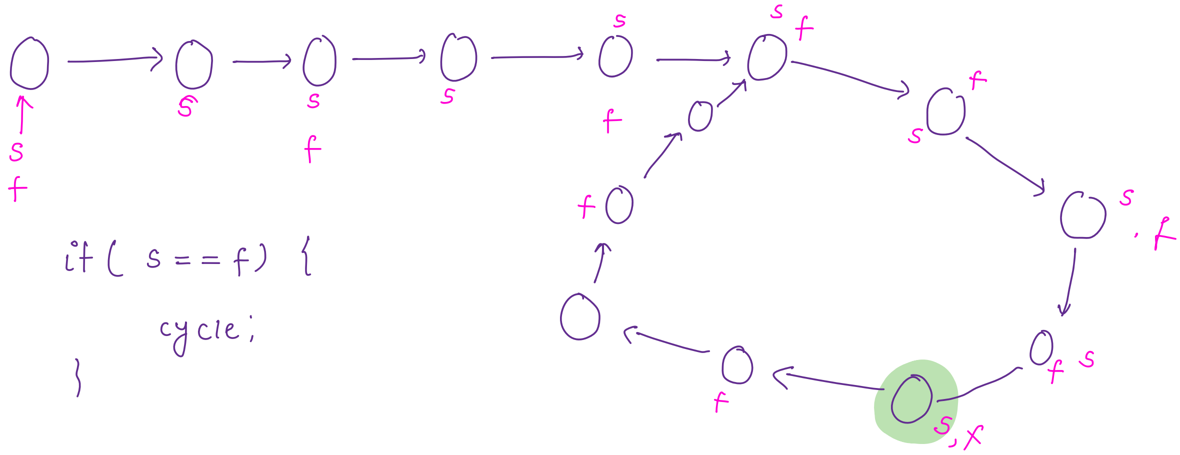


1.) Iterate LL and store address in hashset.

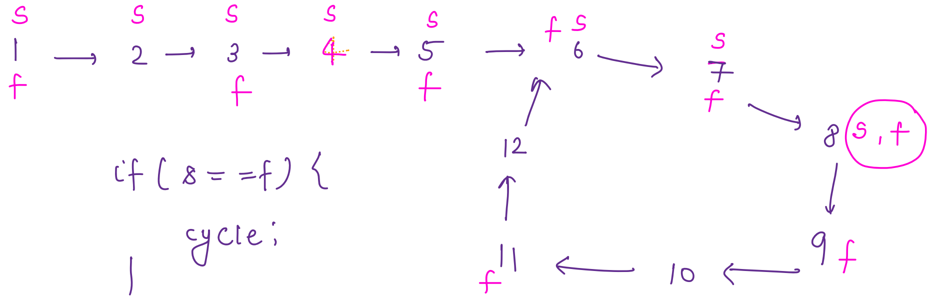
2.) If address is present in hashset , cycle

3.) If you hit null , no cycle.

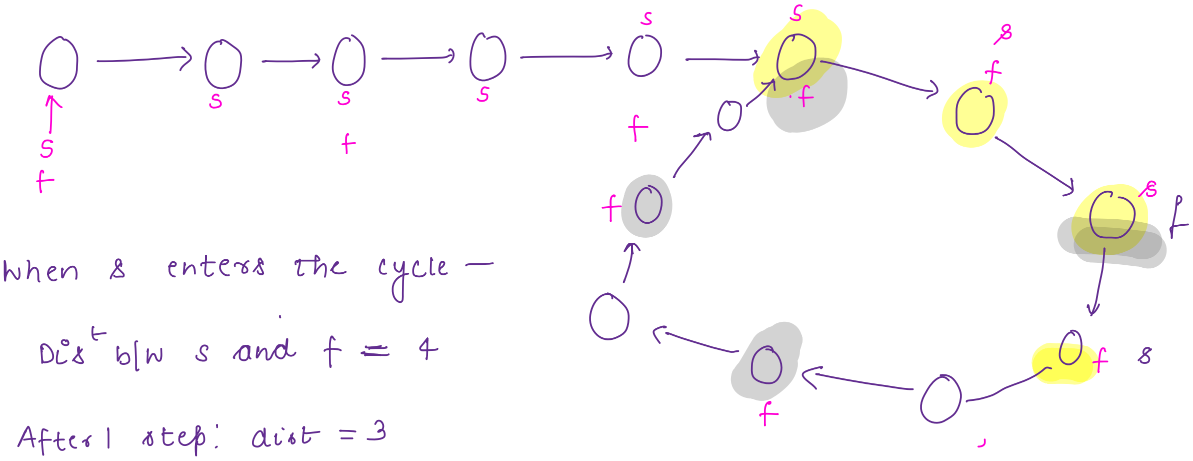
Idea3: slow and fast pointer [flyod algo]



```
if ( s == f) {
      cycle;
}
```



```
if ( s = =f) {
      cycle;
}
```



When s enters the cycle —

Dis^t^ b/w s and f = 4

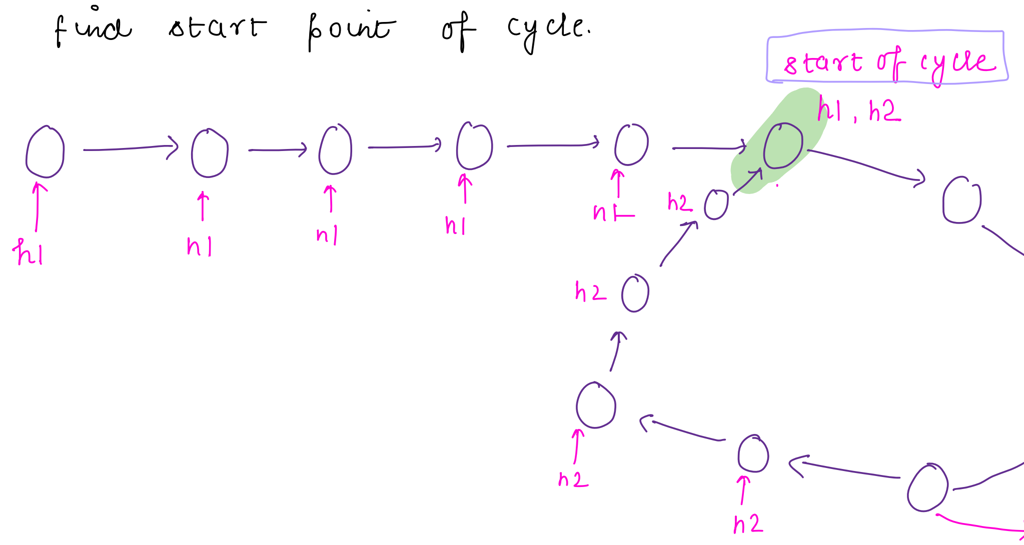
After 1 step: dist = 3

2 step: dist = 2

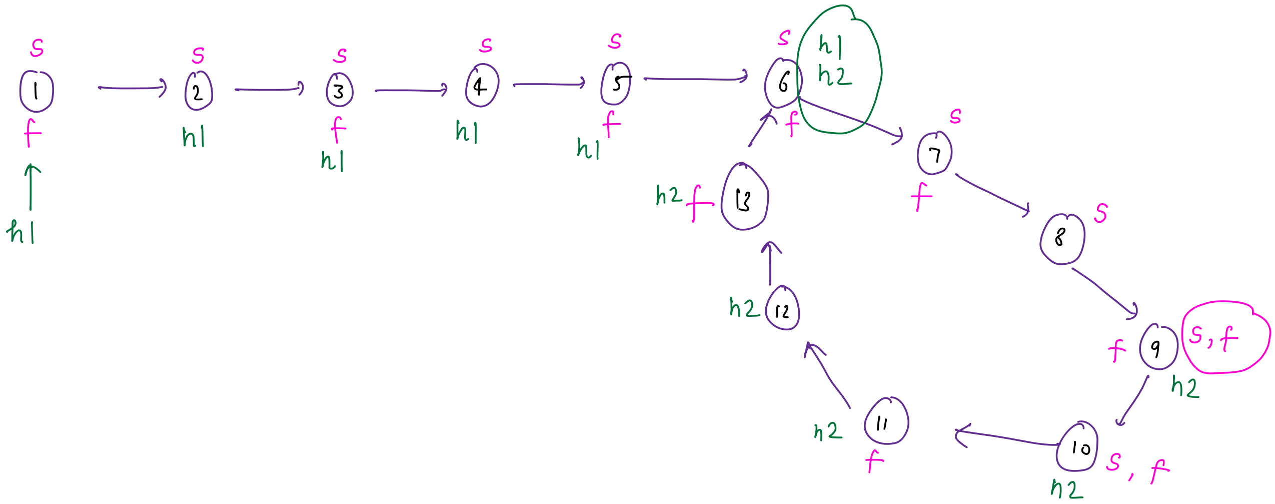
3 step: distance = 1

4 step: dist = 0

* find start point of cycle.



h2 = intersection pt where s & f meet for first time



```
boolean  detectfindRemove( Node h) {

        Node s = h;
        Node f = h;
        boolean iscyclefound = false;
```

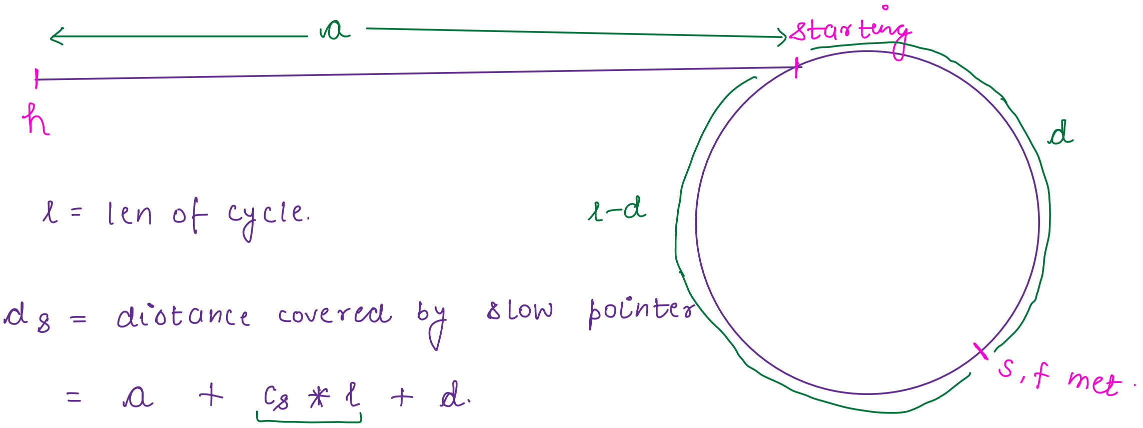
detect cycle.

```
        while ( f != null    &&   f.next != null) {
                s = s.next;
                f = f.next.next;
                if ( s == f) {
                    iscyclefound = true;
                    break;
                }
        }

  if ( ! iscyclefound) {
          return false;
  }
  Node  h1 = h;
  Node  h2 =  s;
  while(  h1  != h2  ) {
          h1 = h1.next;
          h2 = h2.next;
  }
```

[h1 | h2 is your starting point]

```
    Node t = h1;
    while( t.next != h1) {
            t=t.next;
    }
    t.next= null;
    return true;
}
```

Mathematical explanation for start pt.



l = len of cycle.

d_s = distance covered by slow pointer

$$= a + \underbrace{c_s * l}_{\text{no of times you traversed cycle}} + d.$$

$$a = lx + (l-d)$$

d_f = distance covered by fast pointers

$$= a + c_f * l + d$$

$$d_f = 2 * d_s$$

$$a + c_f*l + d = 2(a + c_s*l + d)$$

$$a + c_f*l + d = 2a + 2c_s*l + 2d$$

$$l(c_f - 2c_s) = a + d$$

$$a = l(c_f - 2c_s) - d.$$

↓

Add & subtract l

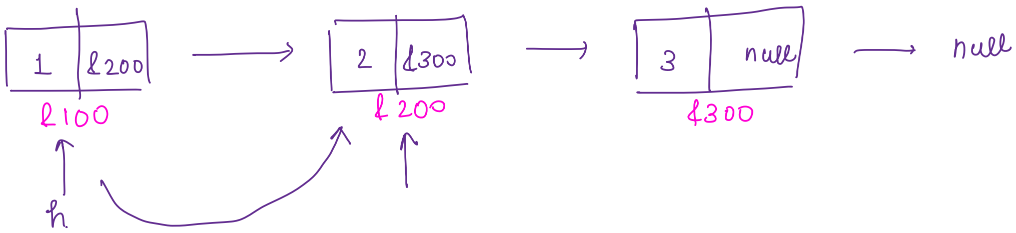
$$a = l(c_f - 2c_s) - d + l - l$$

$$a = l(\underbrace{c_f - 2c_s - 1)}_{x} + (l - d)$$

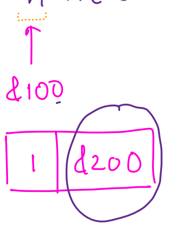
$$a = lx + (l - d)$$

Thankyou ☺

Doubts



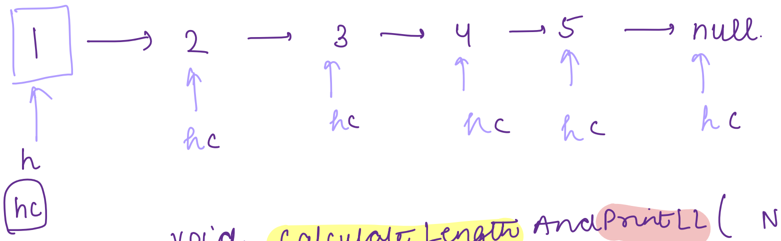
h = h.next



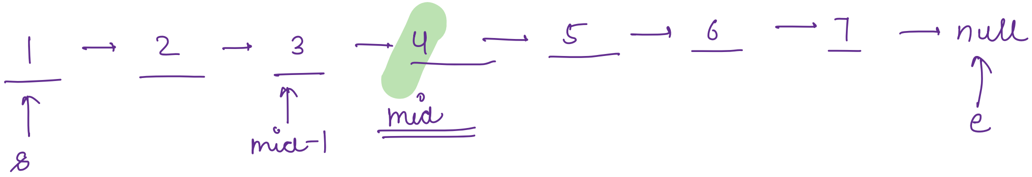
h.next = &200

h = h.next

h = &200



```
void calculateLengthAndPrintLL( Node h ) {
        Node hc = h;
        int len = 0;
        while(hc != null) {
            len++;
            hc= hcnext;
        }
}
```


1
2
3
4
5
6
7
null
s
mid-1
mid
e