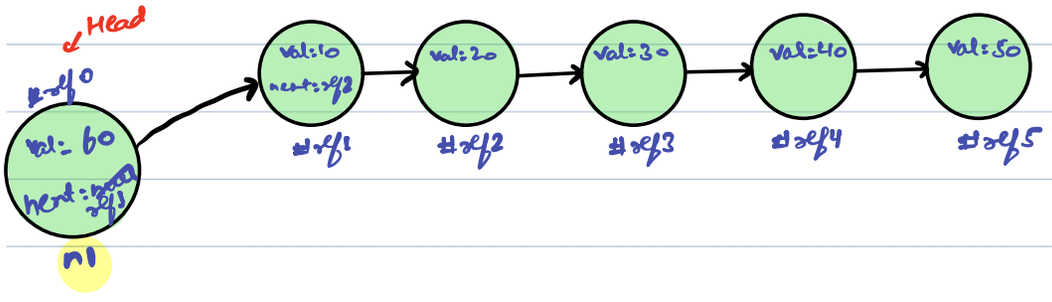
Today's agenda

- insert in a linkedlist
- delete in a linkedlist
- Reverse the linkedlist
- Find mid
- floyd cycle.


```
class Node {
    int val;
    Node next;

    Node (int data) {
        val = data;
    }
}
```

Q) insert in a linkedlist

insert at head
↳ v=60




```
void insertAtStart(Node head, int v){

    Node n1 = new Node(v);

    n1.next = head;

    Head = n1;

}
```

T.C: O(1)

S.C: O(1)

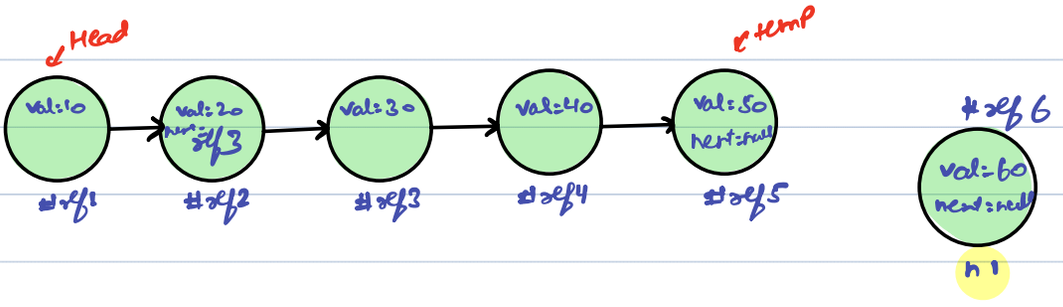
insert after last node

↳ v=60

temp.next == null

↓

ruk jao



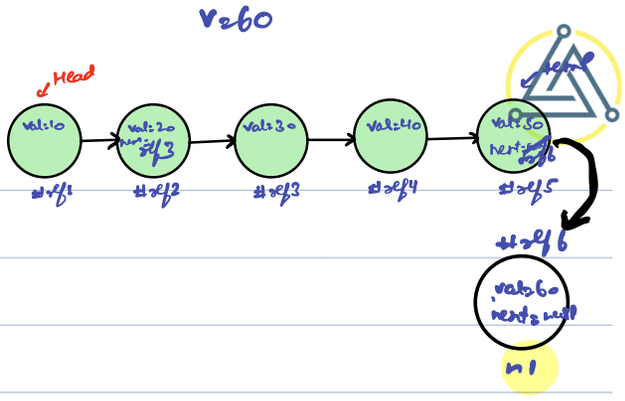

```
void insertatlast (Node head, int v){

    Node n1 = new Node(v);
    Node temp = head;
    while (temp.next != null){
        temp = temp.next;
    }
    temp.next = n1;

}
```

```
void insertatlast (Node head, int v){
    Node n1 = new Node(v);
    Node temp = head;
    while (temp.next != null){
        temp = temp.next;
    }
    temp.next = n1;
}
```

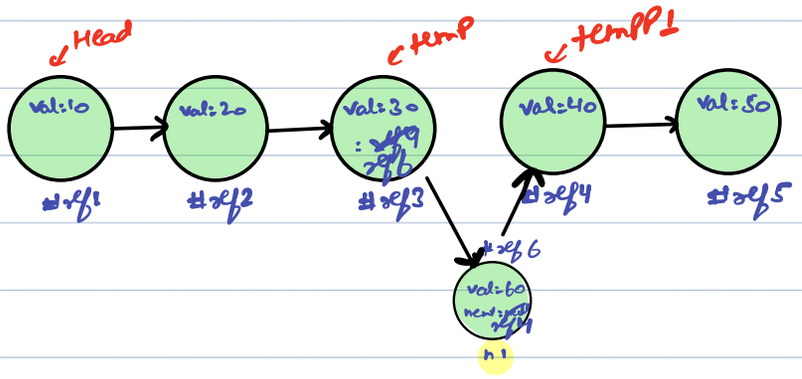
null

temp: null

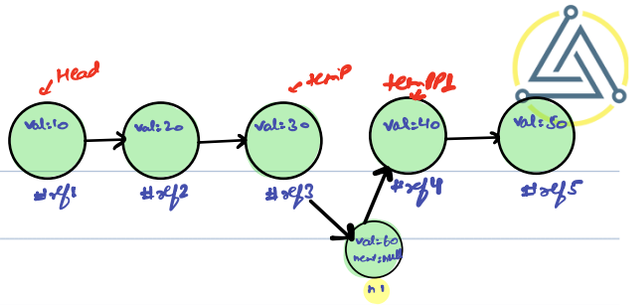
v=60



// insert at index → K=3 , V=60




```
void insertat(Node head, int v , int K){
    Node n1 = new Node (v);
    Node temp = head;
    // K-1 jumps;
    for(int i=1; i<=K-1 ; i++){
        temp = temp.next;
    }
    Node temPP1 = temp.next;
    temp.next = n1;
    n1.next = temPP1;
}
```




```
void insertat(Node head, int v, int K){   // K=0
Node n1 = new Node(v);
Node temp = head;
// K-1 jumps;
for(int i=1; i<=K-1; i++){
    temp = temp.next;
}
Node temPP1 = temp.next;
temp.next = n1;
n1.next = temPP1;
}
```

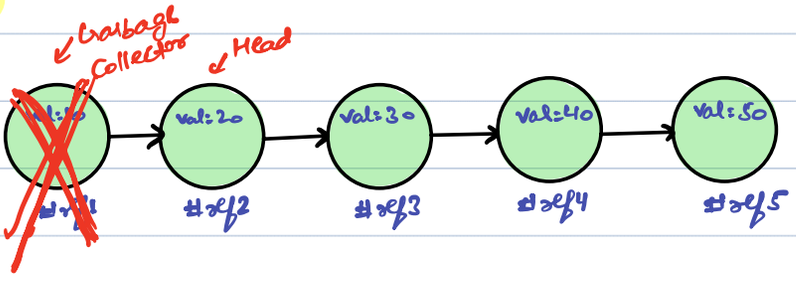
Edgecases

- head == null
- K == 0
- K == sizeof LL (add at last)

→ null Pointer exception

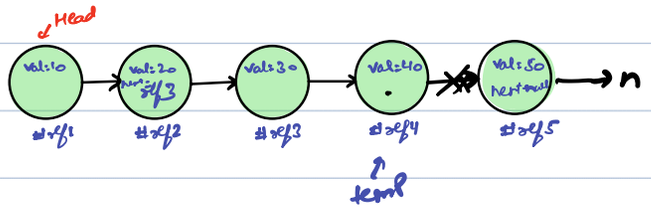
Q) Delete in a linkedlist

→ delete Head



↳ `head = head.next;`

→ delete last




```
deleteLast (Node head){

    Node temp = head;

    while (temp.next.next != null) {
        temp = temp.next;
    }

    temp.next = null

}
```

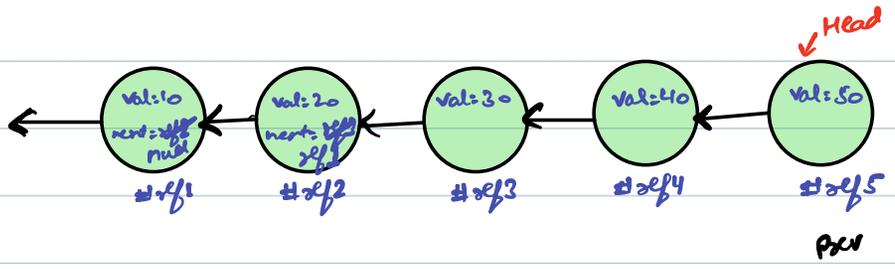
T.C: O(N)

S.C: O(1)

Break till 9:37 PM

Q) Reverse a linkedlist

→ Simple recursion
↓
just to point in reverse.




```
Void reverse (Node head) {

    Node Curr = head;
    Node Prev = null;
    While(Curr != null) {
        Node CurrPl = Curr.next;
        Curr.next = Prev;
        Prev = Curr;
        Curr = CurrPl;
    }

    head = Prev;

}
```

T.C: O(N)

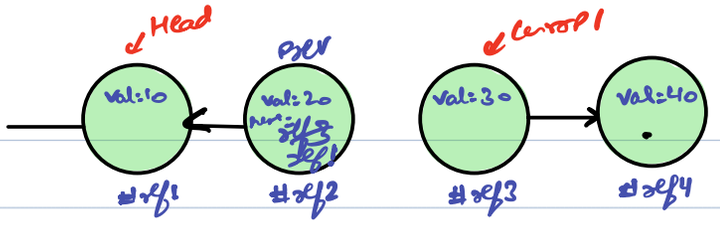
S.C: O(1)

Edgecase

↳ head == null

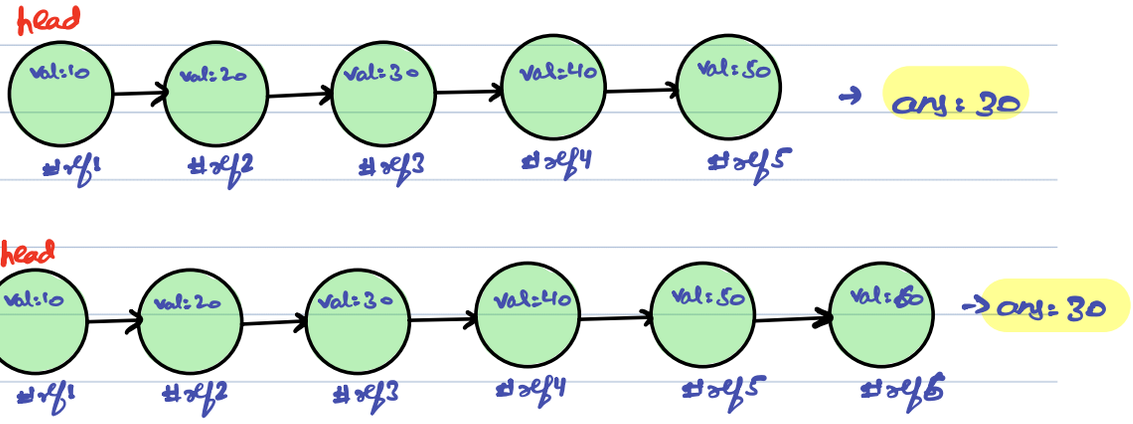
↳ K == 0

↳ K == sizeof LL (add at last)




```
Node Curr = head;
Node Prev = null;
while(Curr != null) {
  Node CurrP1 = Curr.next;
  Curr.next = Prev;
    Prev = Curr;
    Curr = CurrP1;
}

  head = Prev;
```

Q) Given head of a linkedlist, find the mid of it.



//idea 1

↳ iterate and find out size of linkedlist and then in 2nd iteration return $\frac{size}{2}$th element.

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

track = 1 km

//idea 2

↳ do it in single iteration

Jainik

n steps

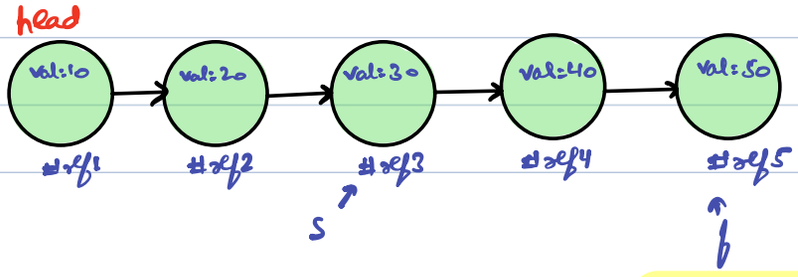
500 m

omneesh

2*n steps

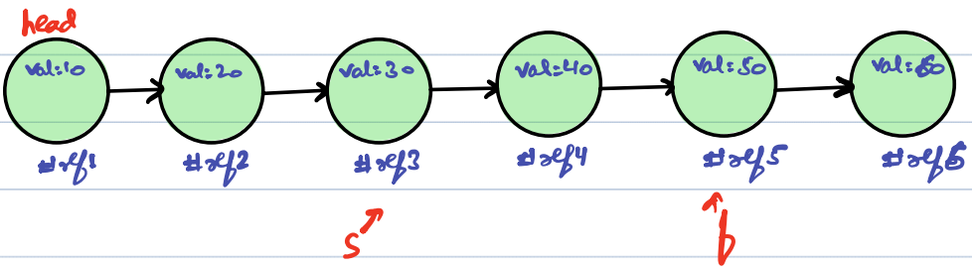
finish line

odd length

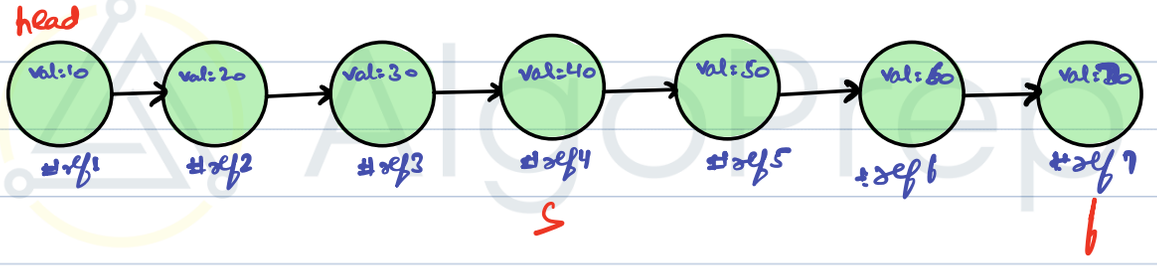


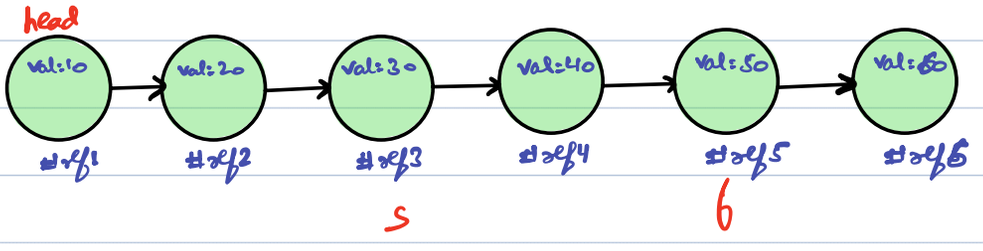
obs: f.next == null → exit, s is our answer

even length



obs: f.next.next == null → exit, s is our answer

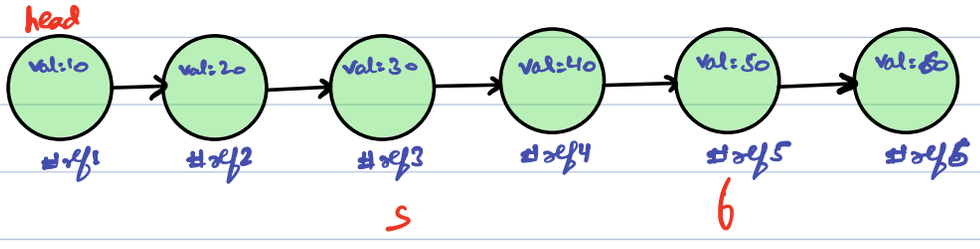




obs: f.next == null → exit

// Pseudo Code

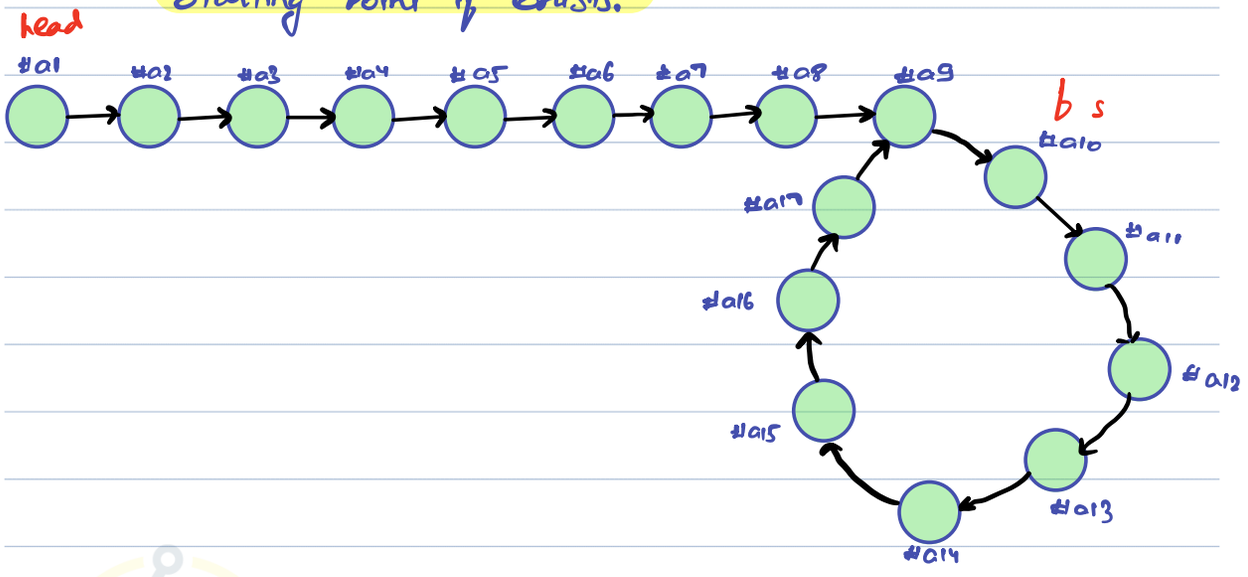
```
Node mid (Node head){

    Node s = head;
    Node f = head;

    while (f.next != null && f.next.next != null)
        s = s.next;
        f = f.next.next;
    }
    return s;
}
```



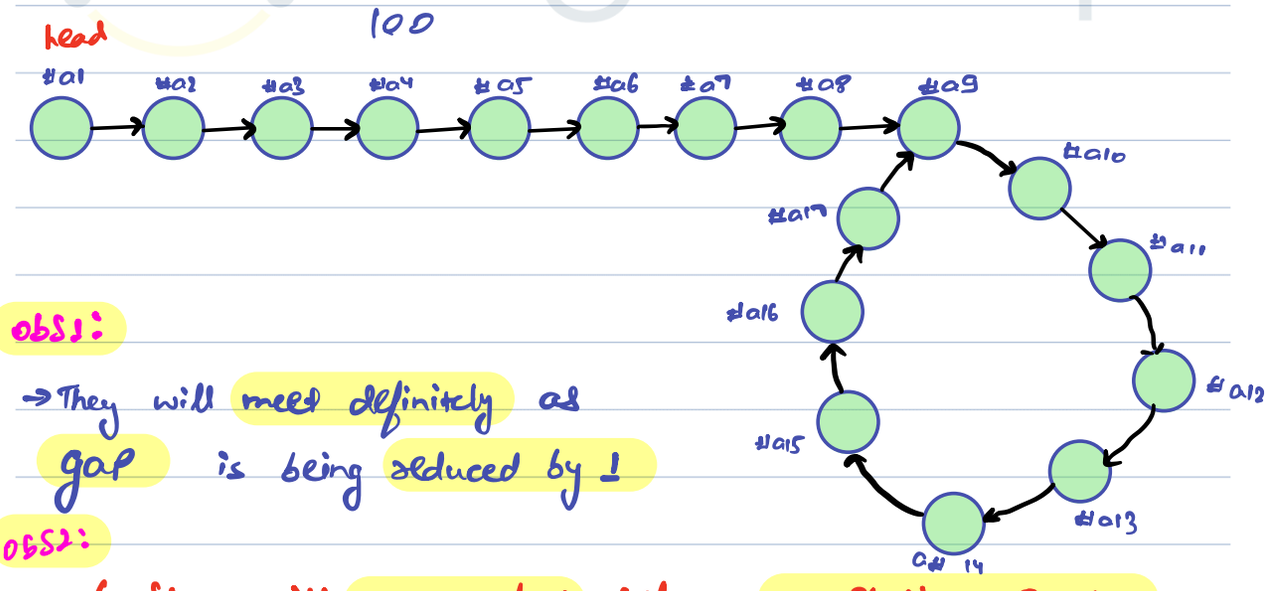
Break till 10:45PM

floyd cycle

Q) Given a linked list, check for cycle & return the starting point if exists.



Ⓑ if s meets f, then we have cycle.



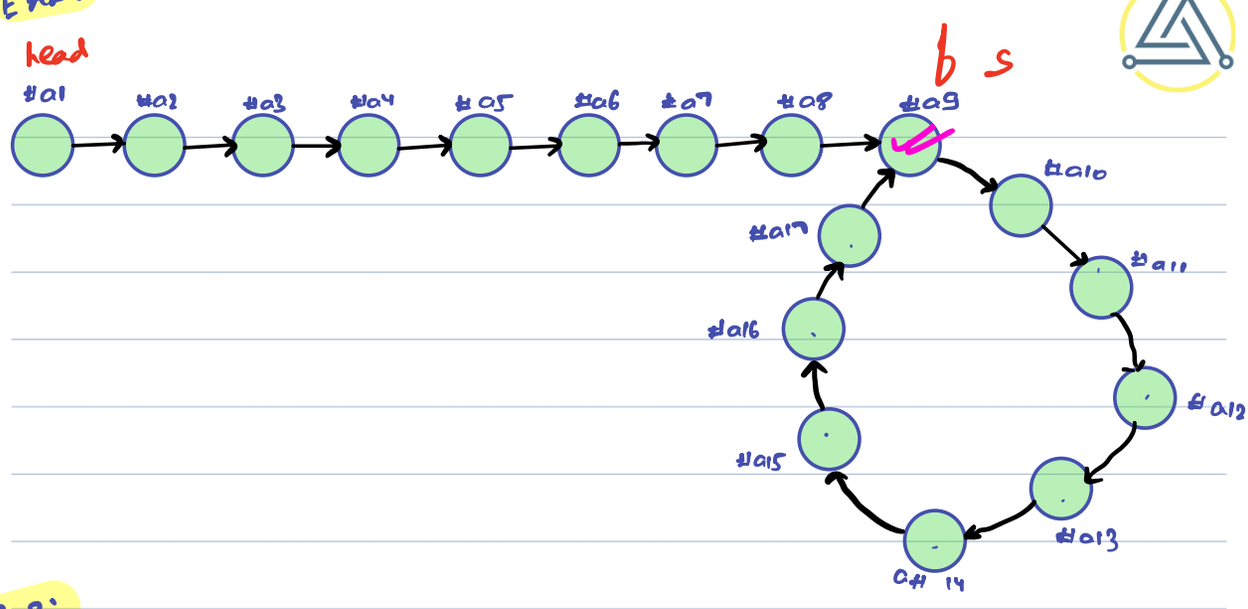
obs1:

→ They will meet definitely as gap is being reduced by 1

obs2:

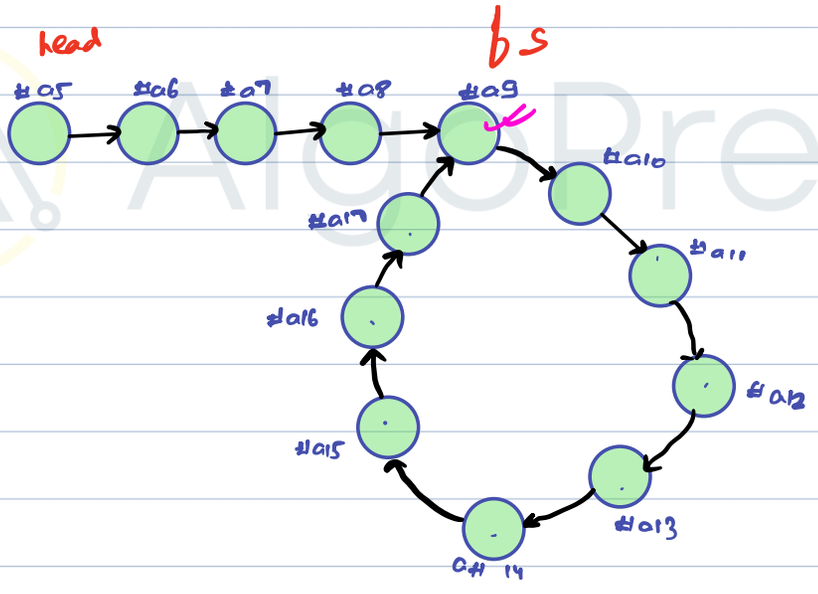
Ⓑ Slow will meet fast before completing cycle.

End:



Ex2:

Chain: 4 Cycle: 9



Steps: Slow & fast Pointers

↳ (i) after some time they will meet.

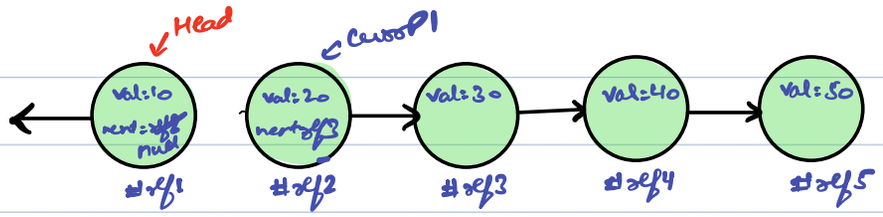
(ii) Pick one Pointer & Put it equal to head.

(iii) Start jumping one step at a time with both s & f, They will meet magically at start Point of cycle.

//Pseudo Code

```
Node removeCycle (Node head){
    Node s = head;
    Node f = head;          // check here

    while (s != f) {
        f = f.next.next;
        s = s.next;
    }                       // check, f == null || f.next == null

    f = head;

    while (f != s){
        s = s.next;
        f = f.next;
    }

    return f;
}
```

WHY ??




```
while (   ) {
  Node CurrP1 = Curr.next;
  Curr.next = Prev;
  Prev = Curr;
  Curr = CurrP1;
}
```

Delete Node

Java Code:

```
Node deleteNode(Node head, int x) {
            if(head==null){
                  return null;
            }
            Node y=head;
            if(x==1&&y.next==null){
                 return null;
            }
            if(x==1&&y.next!=null){
               head=head.next;
               return head;
            }
            Node temp=null;
            for(int i=1;i<x-1;i++){
               y=y.next;
            }
            temp=y.next.next;
            Node del=y.next;
            del.next=null;
            y.next=temp;
            return head;

   }
```

C++ Code:

```
struct Node {
    int data;
    Node* next;
};

Node* deleteNode(Node* head, int x) {
    if (head == nullptr) {
        return nullptr;
    }
    Node* y = head;

    if (x == 1 && y->next == nullptr) {
        delete y;
        return nullptr;
```

```
    }

    if (x == 1 && y->next != nullptr) {
        head = head->next;
        delete y;
        return head;
    }

    Node* temp = nullptr;
    for (int i = 1; i < x - 1; i++) {
        y = y->next;
    }
    temp = y->next->next;
    Node* del = y->next;
    del->next = nullptr;
    y->next = temp;
    delete del;

    return head;
}
```

Python Code:

```
class ListNode:
    def __init__(self, data=0, next=None):
        self.data = data
        self.next = next

def deleteNode(head, x):
    if head is None:
        return None
    y = head

    if x == 1 and y.next is None:
        return None

    if x == 1 and y.next is not None:
        head = head.next
        return head

    temp = None
    for i in range(1, x - 1):
        y = y.next
    temp = y.next.next
    del_node = y.next
```

```
    del_node.next = None
    y.next = temp

    return head
```

Reverse LinkedList

Java Code:

```java
class Solution {
public ListNode reverseList(ListNode head) {
if(head == null){
return null;
}

if(head.next == null){
return head;
}
ListNode prev = null;
ListNode curr = head;
while(curr != null){

ListNode currp1 = curr.next;
curr.next = prev;
prev = curr;
curr = currp1;


}
return prev;
}
}
```

C++ Code:

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if (head == nullptr) {
            return nullptr;
```

```
        }

        if (head->next == nullptr) {
            return head;
        }

        ListNode* prev = nullptr;
        ListNode* curr = head;

        while (curr != nullptr) {
            ListNode* next_node = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next_node;
        }

        return prev;
    }
};
```

Python Code:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        if not head:
            return None

        if not head.next:
            return head

        prev = None
        curr = head

        while curr:
            next_node = curr.next
            curr.next = prev
            prev = curr
            curr = next_node

        return prev
```

Middle of the LinkedList

Java Code:

```
public ListNode middleNode(ListNode head) {
ListNode slow = head, fast = head;
while (fast != null && fast.next != null) {
slow = slow.next;
fast = fast.next.next;
}
return slow;
}
```

C++ Code:

```
Unset
 ListNode* middleNode(ListNode* head) {
    ListNode *slow = head, *fast = head;
    while (fast && fast->next)
      slow = slow->next, fast = fast->next->next;
    return slow;
  }
```

Python Code:

```
Python
def middleNode(self, head):
    slow = fast = head
    while fast and fast.next:
      slow = slow.next
      fast = fast.next.next
    return slow
```

Floyd Cycle

Java Code:

```java
public ListNode detectCycle(ListNode head) {
ListNode slow = head, fast = head;
while (fast != null && fast.next != null) {
slow = slow.next;
fast = fast.next.next;

if (slow == fast) break;
}
if (fast == null || fast.next == null) return null;

fast = head;
while (fast != slow) {
fast = fast.next;
slow = slow.next;
}
return fast;
}
```

C++ Code:

```cpp
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;

            if (slow == fast) break;
        }
        if (fast == nullptr || fast->next == nullptr) return nullptr;
```

```
        fast = head;
        while (fast != slow) {
            fast = fast->next;
            slow = slow->next;
        }
        return fast;
    }
};
```

Python Code:

```
class ListNode:
    def __init__(self, val=0):
        self.val = val
        self.next = None

class Solution:
    def detectCycle(self, head):
        slow = head
        fast = head
        while fast is not None and fast.next is not None:
            slow = slow.next
            fast = fast.next.next

            if slow == fast:
                break

        if fast is None or fast.next is None:
            return None

        fast = head
        while fast != slow:
            fast = fast.next
            slow = slow.next

        return fast
```

Delete Node in a LinkedList_HW

Solution Vid:

https://youtu.be/HReo4x-dJss

Java Code:

```
public void deleteNode(ListNode node) {
      node.val = node.next.val;
      node.next = node.next.next;
}
```

C++ Code:

```
JavaScript
void deleteNode(ListNode* node) {
  auto next = node->next;
  *node = *next;
  delete next;
}
```

Python Code:

```
Unset
def deleteNode(self, node):
  node.val = node.next.val
  node.next = node.next.next
```

Remove Duplicates_HW

Solution Vid:

https://youtu.be/1y8dcee0ixA

Java Code:

```java
public ListNode deleteDuplicates(ListNode head) {
if(head == null){
   return null;
}

ListNode current = head;
while (current.next != null) {
if (current.val == current.next.val) {
ListNode currentp2 = current.next.next;
current.next = currentp2;
} else {
current = current.next;
}
}
return head;

}
```

C++ Code:

```cpp
struct ListNode {
   int val;
   ListNode* next;
   ListNode(int val) : val(val), next(nullptr) {}
};

class Solution {
public:
   ListNode* deleteDuplicates(ListNode* head) {
      ListNode* current = head;
      while (current != nullptr && current->next != nullptr) {
         if (current->val == current->next->val) {
            ListNode* currentp2 = current->next->next;
            delete current->next;
            current->next = currentp2;
         } else {
            current = current->next;
         }
      }
      return head;
   }
```

};

Python Code:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        current = head
        while current is not None and current.next is not None:
            if current.val == current.next.val:
                current.next = current.next.next
            else:
                current = current.next
        return head
```

ReOrder List_HW

Solution Vid:

https://youtu.be/J3JDugg-GpY

Java Code:

```
public void reorderList(ListNode head) {
if(head==null||head.next==null) return;
ListNode s=head;
ListNode f=head;
while(f.next!=null&&f.next.next!=null){
s=s.next;
f=f.next.next;
}
ListNode Prev=null;
ListNode Curr=s.next;
s.next = null;
```

```
while(Curr!=null){
ListNode Currp1 = Curr.next;
Curr.next = Prev;
Prev = Curr;
Curr = Currp1;
}

ListNode left=head;
ListNode right=Prev;
while(right != null){
ListNode leftp1 = left.next;
ListNode rightp1 = right.next;

left.next = right;
right.next = leftp1;
left = leftp1;
right = rightp1;
}
}
```

C++ Code:

```
void reorderList(ListNode* head) {
    if (!head || !head->next) return;

    ListNode* s = head;
    ListNode* f = head;
    while (f->next && f->next->next) {
        s = s->next;
        f = f->next->next;
    }

    ListNode* Prev = nullptr;
    ListNode* Curr = s->next;
    s->next = nullptr;
    while (Curr) {
        ListNode* Currp1 = Curr->next;
        Curr->next = Prev;
        Prev = Curr;
        Curr = Currp1;
    }
```

```
    ListNode* left = head;
    ListNode* right = Prev;
    while (right) {
        ListNode* leftp1 = left->next;
        ListNode* rightp1 = right->next;

        left->next = right;
        right->next = leftp1;

        left = leftp1;
        right = rightp1;
    }
}
```

Python Code:

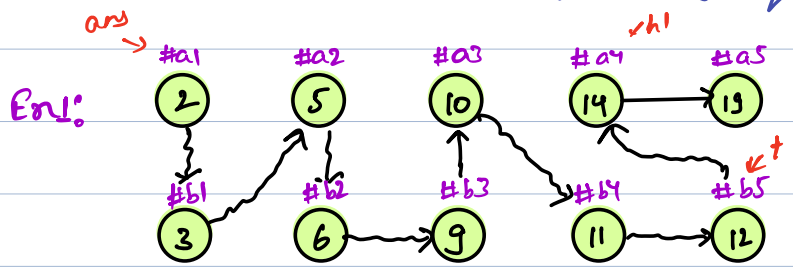
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reorderList(head):
    if not head or not head.next:
        return

    s = head
    f = head
    while f.next and f.next.next:
        s = s.next
        f = f.next.next

    prev = None
    curr = s.next
    s.next = None
    while curr:
        currp1 = curr.next
        curr.next = prev
        prev = curr
        curr = currp1

    left = head
    right = prev
```

```
while right:
    leftp1 = left.next
    rightp1 = right.next

    left.next = right
    right.next = leftp1

    left = leftp1
    right = rightp1
```

Today's agenda

- merge
- merge sort on linkedlist
- Find intersection
- doubly linkedlist
- few basic Problems on DLL

Q) Given 2 sorted linkedlists, merge and get final sorted list.



ans: #a1

↓

return ans;

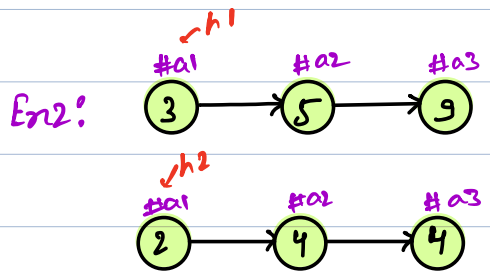
h2

t.next = h2/h1;

t = t.next;

h1/h2 = h1/h2.next;

obs: if h1 or h2 == null, stop comparing



//Pseudo code

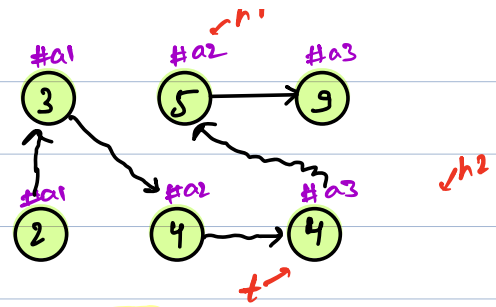
```
Node merge (Node h1, Node h2){
    Node ans = null, Node t = null;
    if (h1.data < h2.data){
        ans = h1; t = h1; h1 = h1.next;
    }
    else{
        ans = h2; t = h2; h2 = h2.next;
    }

    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;
        }
    }
    //add the remaining element
    if (h1 != null) { t.next = h1; }
    if (h2 != null) { t.next = h2; }

    return ans;
}
```

T.C: O(n+m)

S.C: O(1)

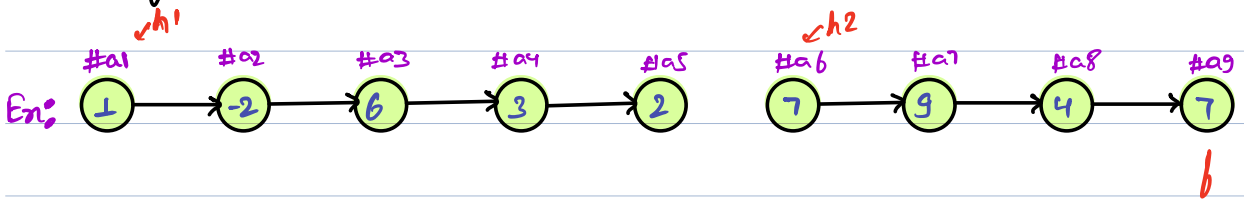
Tracing

```
Node merge (Node h1, Node h2){
    Node ans = null;   Node t = null;
    if (h1.data < h2.data){
        ans = h1;  t = h1;  h1 = h1.next;
    }
    else{
        ans = h2;  t = h2;  h2 = h2.next;
    }

    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;
        }
    }
    //add the remaining element
    if (h1 != null) { t.next = h1; }
    if (h2 != null) { t.next = h2; }

    return ans;
}
```



ans : ~~null~~ #a1

Q) Merge sort on LinkedList



Step1: Calculate mid (slow & fast) & divide into 2 Parts. (Actually split it)

A2 = S.next;

S.next = null;

Step2: use the magic of recursion, call for sorting both halves.

Step3: merge those 2 sorted Part.

// Pseudo code

```
Node mergeSort (Node h1){
    if (h1.next == null) {return h1;}

    Node m = mid (h1);
    Node h2 = m.next;
    m.next = null;

    Node t1 = mergeSort (h1);
    Node t2 = mergeSort (h2);

    Node t3 = merge (t1, t2);
    return t3;
}

Node mid (Node h){
    Node s = h, f = h;

    while (f.next != null && f.next.next != null) {
        s = s.next;
        f = f.next.next;
    }

    return s;
```

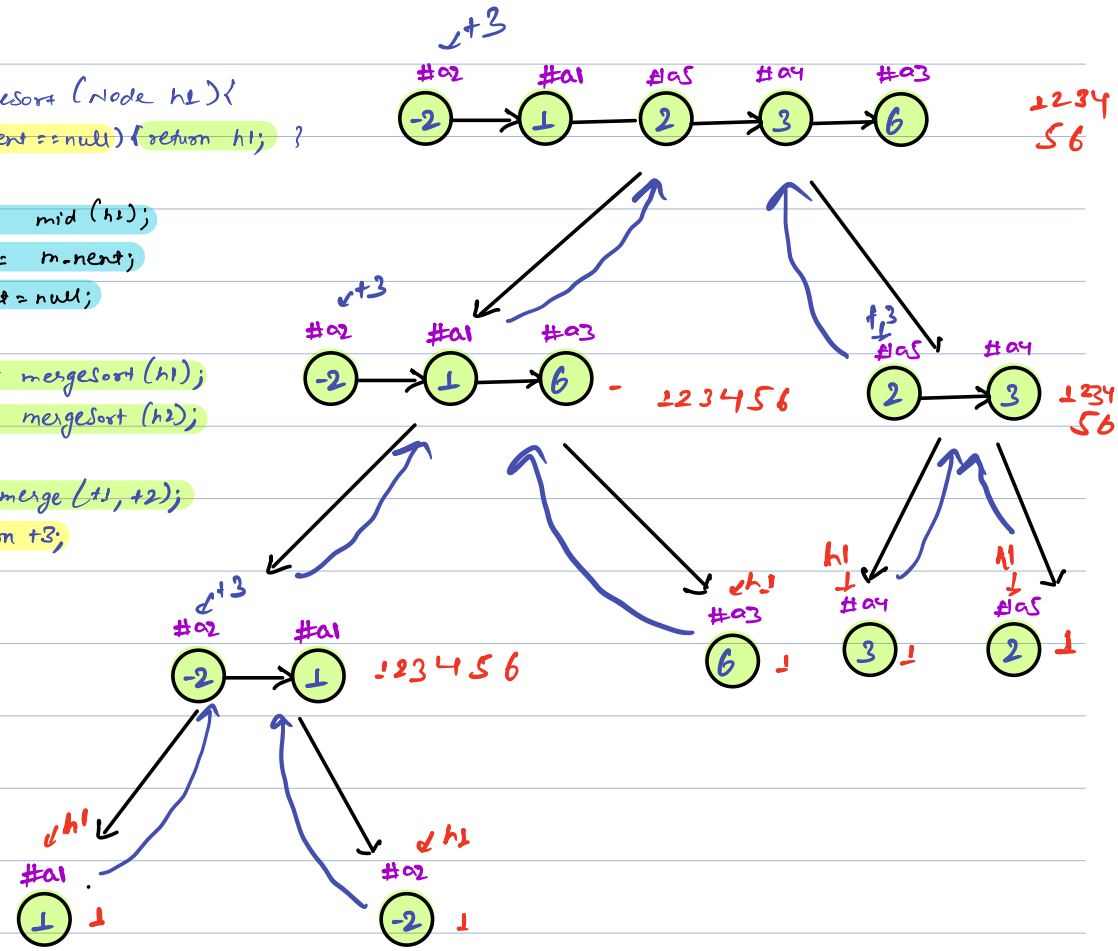
T.C: O(NlogN)

S.C: O(stack space) + O(logn)

13

Tracing

```
Node mergesort (Node h1){
① if (h1.next == null) { return h1; }
② Node m = mid (h1);
   Node h2 = m.next;
   m.next = null;
③ Node t1 = mergesort (h1);
④ Node t2 = mergesort (h2);
⑤ Node t3 = merge (t1, t2);
⑥ return t3;
}
```

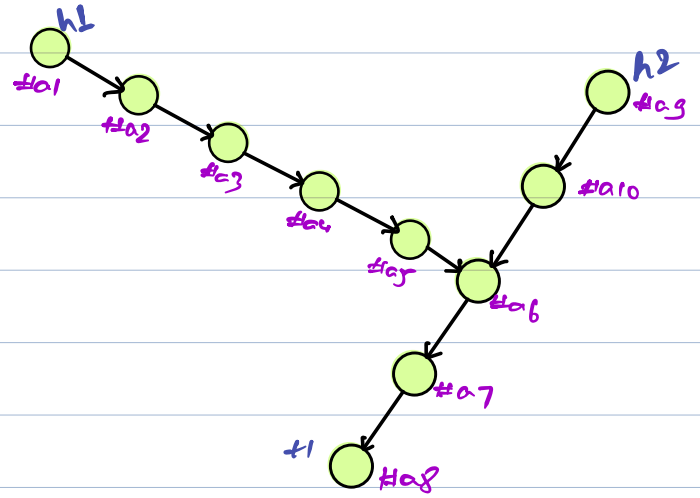


Break till 10:28 Pm

Q) Intersection of a linkedlist

↳ Given the heads of two singly linked-list h1 and h2, return the node at which the two lists intersect.

Ex1:

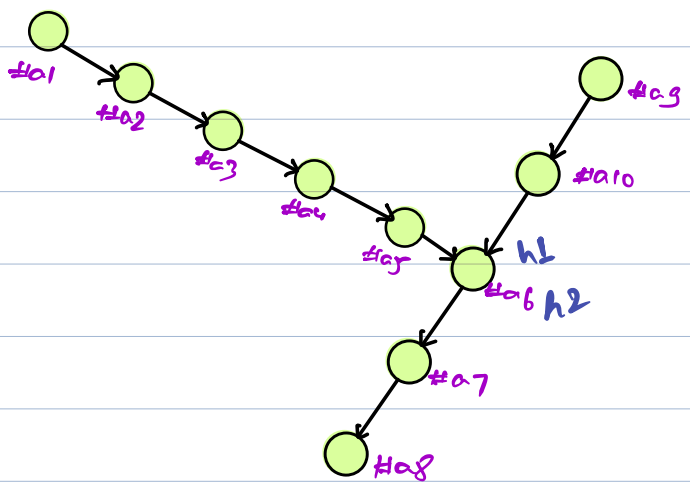


//idea1

↳ t1.next = h1

↳ run floyd cycle

//idea2



Step1: find length of first linkedlist.

size1 = 8

T.C: O(N)

S.C: O(1)

Step2: find length of second linkedlist

size2 = 5

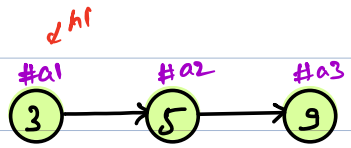
Step3: Calculate the extra node in longer LL.

diff = |size1 - size2| = 3

Step4: iterate "diff" times only in longer LL.

Step5: Start moving the pointer of both the LL, till they match.

// Doubly linked list

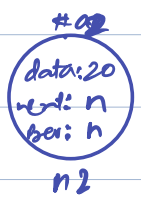


→ Can't iterate backwards

```
Public class Node {
    int data;
    Node next;
    Node prev;
    Node (int val) {
        data : val;
    }
}
```

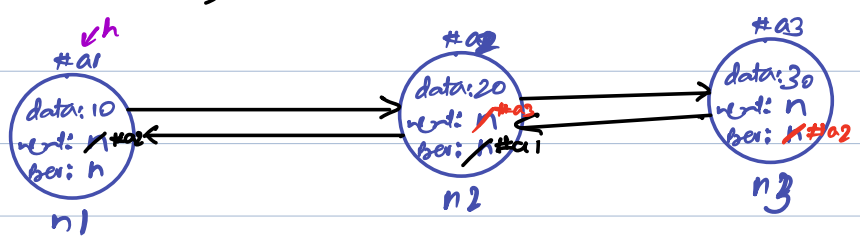
Node n1 = new Node (10);

Node n2 = new Node (20);



→ n1.next = n2;

→ n2.prev = n1;

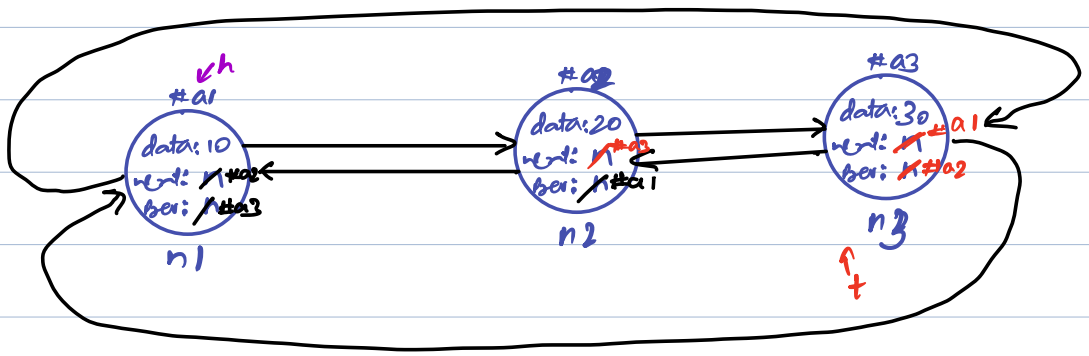


→ Print (n1.next.next.prev.data) → 20

→ Print (n3.prev.prev.data) → 10

doubly linked list → Singly linked list + O(N) space

* Circular doubly linked list



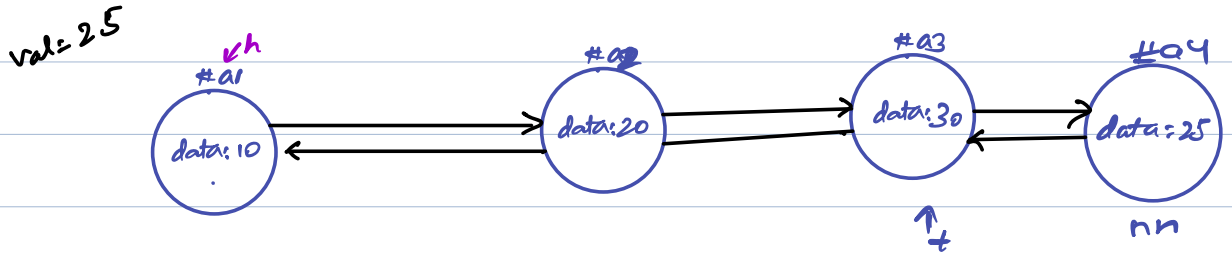
↓

iterate on circular doubly linked list

↓

Check if t.next = h

Q) Given a doubly linked list, add a node with given val at the end.



Step1: iterate till the end of doubly linked list.

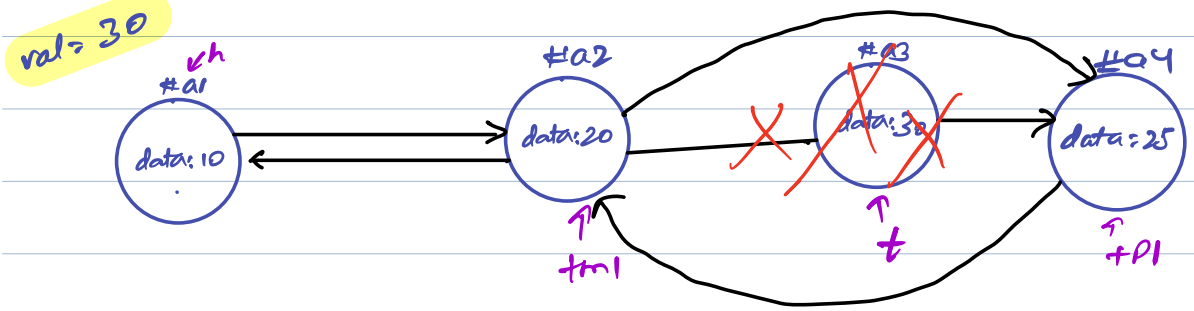
```
while (t.next != null) {
    t = t.next;
}
```

Step2: `Node nn = new Node(25);`

```
t.next = nn;
nn.prev = t;
```

Q) Remove a node with given val from DLL.

↳ if multiple occ, remove any one

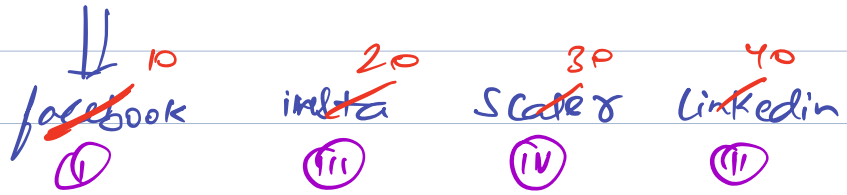


Step1: find the node with given val.

Step2:

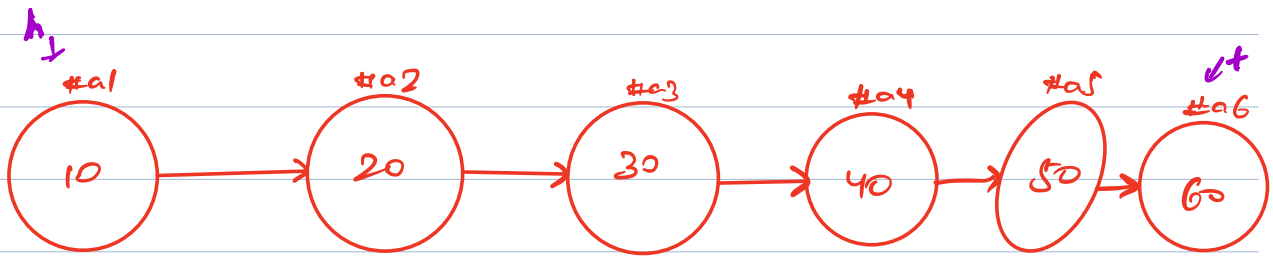
```
Node tm1 = t.prev;
Node tp1 = t.next;

tm1.next = tp1;
tp1.prev = tm1;
```

LRU Cache



Doubt Session

↳ input → { 10, 20, 30, 40, 50, 60 }



Merge two sorted linked lists

Java Code:

```
class LinkedList
{
    //Function to merge two sorted linked list.
    Node sortedMerge(Node h1, Node h2) {
      if(h1 == null) return h2;
      if(h2 == null) return h1;
      Node ans = null , t = null;
      if(h1.data < h2.data){
          ans = h1;
          t = h1;
          h1 = h1.next;
      } else {
          ans = h2;
          t = h2;
          h2 = h2.next;
      }

      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;
      if(h2 != null) t.next = h2;

      return ans;
    }
}
```

C++ Code:

```
#include <iostream>
```

```
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
public:
    // Function to merge two sorted linked lists.
    Node* sortedMerge(Node* h1, Node* h2) {
        if (h1 == nullptr) return h2;
        if (h2 == nullptr) return h1;

        Node* ans = nullptr;
        Node* t = nullptr;

        if (h1->data < h2->data) {
            ans = h1;
            t = h1;
            h1 = h1->next;
        } else {
            ans = h2;
            t = h2;
            h2 = h2->next;
        }

        while (h1 != nullptr && h2 != nullptr) {
            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 != nullptr) t->next = h1;
        if (h2 != nullptr) t->next = h2;

        return ans;
    }
};
```

Python Code:

```
class Node:
    def __init__(self, val):
        self.data = val
        self.next = None

class LinkedList:
    # Function to merge two sorted linked lists.
    def sortedMerge(self, h1, h2):
        if h1 is None:
            return h2
        if h2 is None:
            return h1

        ans = None
        t = None

        if h1.data < h2.data:
            ans = h1
            t = h1
            h1 = h1.next
        else:
            ans = h2
            t = h2
            h2 = h2.next

        while h1 is not None and h2 is not None:
            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 is not None:
            t.next = h1
        if h2 is not None:
            t.next = h2

        return ans
```

Merge Sort for Linked List

Java Code:

```
class Solution
{
    //Function to sort the given linked list using Merge Sort.
    static Node mergeSort(Node h1)
    {
       if(h1.next == null) return h1;
       Node m = mid(h1);
       Node h2 = m.next;
       m.next = null;

       Node t1 = mergeSort(h1);
       Node t2 = mergeSort(h2);
       Node t3 = merge(t1,t2);
       return t3;
    }
    static Node mid(Node h){
        Node s = h,f = h;
        while(f.next != null && f.next.next != null){
            f = f.next.next;
            s = s.next;
        }
        return s;
    }
     static Node merge(Node h1, Node h2) {
      if(h1 == null) return h2;
      if(h2 == null) return h1;
      Node ans = null , t = null;
      if(h1.data < h2.data){
          ans = h1;
          t = h1;
          h1 = h1.next;
      } else {
          ans = h2;
          t = h2;
          h2 = h2.next;
      }

      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;
      if(h2 != null) t.next = h2;

      return ans;
    }
}
```

C++ Code:

```
#include <iostream>

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class Solution {
public:
    // Function to sort the given linked list using Merge Sort.
    Node* mergeSort(Node* head) {
        if (head == nullptr || head->next == nullptr)
            return head;

        Node* m = mid(head);
        Node* h2 = m->next;
        m->next = nullptr;

        Node* t1 = mergeSort(head);
        Node* t2 = mergeSort(h2);
        Node* t3 = merge(t1, t2);
        return t3;
    }

private:
    Node* mid(Node* h) {
        Node* s = h;
        Node* f = h;
        while (f->next != nullptr && f->next->next != nullptr) {
            f = f->next->next;
            s = s->next;
```

```
        }
        return s;
    }

    Node* merge(Node* h1, Node* h2) {
        if (h1 == nullptr) return h2;
        if (h2 == nullptr) return h1;

        Node* ans = nullptr;
        Node* t = nullptr;

        if (h1->data < h2->data) {
            ans = h1;
            t = h1;
            h1 = h1->next;
        } else {
            ans = h2;
            t = h2;
            h2 = h2->next;
        }

        while (h1 != nullptr && h2 != nullptr) {
            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 != nullptr) t->next = h1;
        if (h2 != nullptr) t->next = h2;

        return ans;
    }
};
```

Python Code:

```
class Node:
    def __init__(self, val):
        self.data = val
```

```
        self.next = None

class Solution:
    # Function to sort the given linked list using Merge Sort.
    def mergeSort(self, head):
        if not head or not head.next:
            return head

        m = self.mid(head)
        h2 = m.next
        m.next = None

        t1 = self.mergeSort(head)
        t2 = self.mergeSort(h2)
        t3 = self.merge(t1, t2)
        return t3

    def mid(self, h):
        s = h
        f = h
        while f.next and f.next.next:
            f = f.next.next
            s = s.next
        return s

    def merge(self, h1, h2):
        if not h1:
            return h2
        if not h2:
            return h1

        ans = None
        t = None

        if h1.data < h2.data:
            ans = h1
            t = h1
            h1 = h1.next
        else:
            ans = h2
            t = h2
            h2 = h2.next

        while h1 and h2:
            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:
        t.next = h1
    if h2:
        t.next = h2

    return ans
```

Intersection Point in Y Shaped Linked Lists

Java Code:

```
class Intersect
{
    //Function to find intersection point in Y shaped Linked Lists.
        int intersectPoint(Node head1, Node head2)
        {
     Node p1 = head1 , p2 = head2;
    int s1 = 0, s2 = 0;
    while(p1 != null){
        s1++;
        p1 = p1.next;
    }
     while(p2 != null){
        s2++;
        p2 = p2.next;
    }
    p1 = head1;
    p2 = head2;

    if(s1>s2){
        int diff = s1-s2;
        while(diff>0){
           p1 = p1.next;
            diff--;
        }
    } else {
       int diff = s2-s1;
```

```
            while(diff>0){
               p2 = p2.next;
                diff--;
            }
        }
        while(p1!=null && p2 != null && p1 != p2){
            p1 = p1.next;
            p2 = p2.next;
        }
       if(p1==null) return -1;
       return p1.data;
            }
}
```

C++ Code:

Python Code:

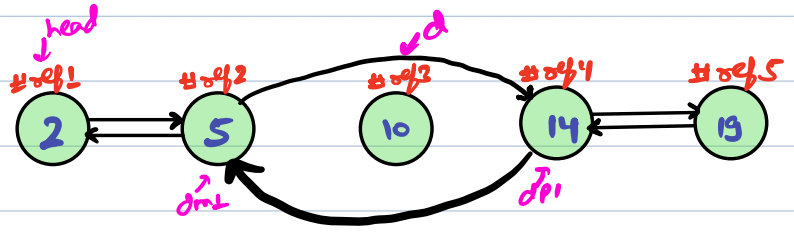
Today's agenda

- Delete a given node in doubly LL.
- Add a node before tail in a doubly LL.
- LRU Cache
- Clone a LL with Random Pointers.

Q) Delete a given Node from doubly linkedlist.

d = #ref3

note: d node can't be head or tail.

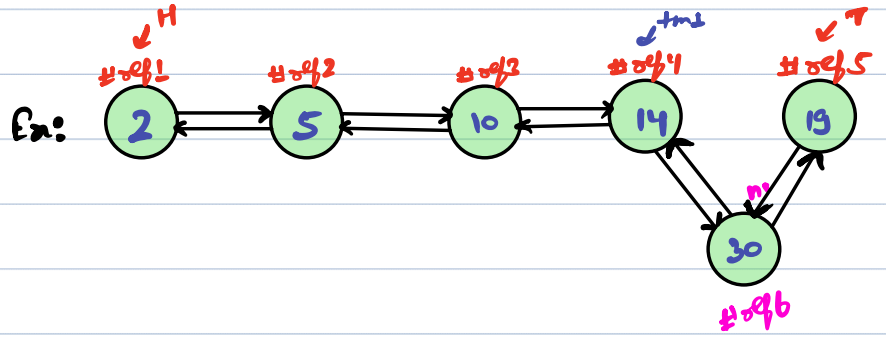



```
Node delete (Node head, Node d){
    Node dm1 = d.prev;
    Node dp1 = d.next;

    dm1.next = dp1;
    dp1.prev = dm1;

    d.next = null;
    d.prev = null;

    return d;
}
```

T.C: O(1)

S.C: O(1)

Q) Add a node before tail in a doubly LinkedList.




```
void add (Node H, Node T, Node n1){
    Node tmp = T.prev;

    tmp.next = n1;
    n1.next = T;

    T.prev = n1;
    n1.prev = tmp;
}
```

T.C: O(1)

S.C: O(1)

3

Q) LRU Cache → last opened app appears first

↳ Create a least recently used Cache.

facebook 1 | Algoprep 2 | instagram 3 | linkedin 4

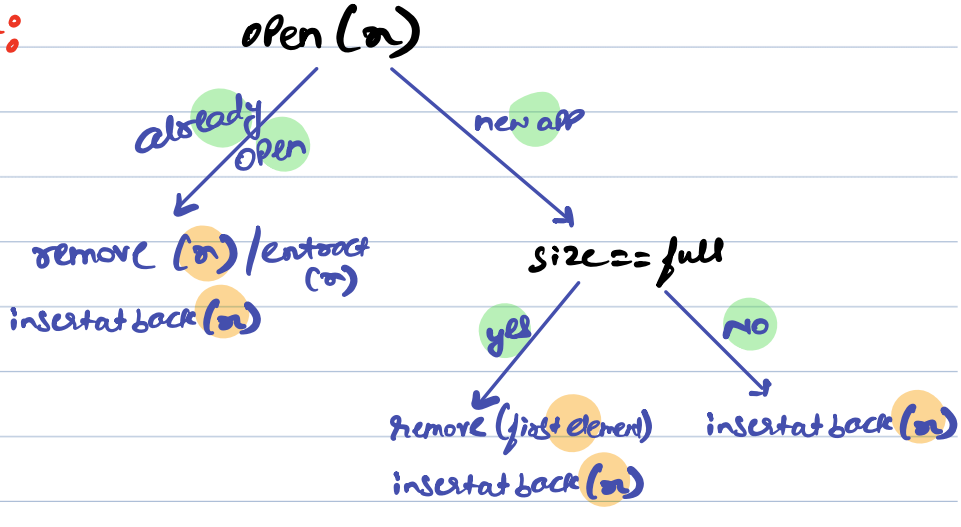
size = 5

En: 7 3 9 2 get() 6 10 get() 14 2

get() → 2 | 10 → get() → 10

old → recent

flow chart:



① HashMap + doubly LL

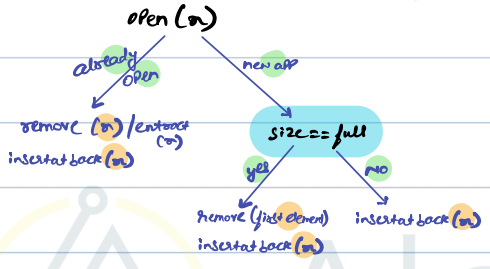
↓ To check whether X is present in cache or not.

↓ order mainting

size = 4

Data: (7,10) (3,20) (9,50) (2,60) get() 6 get() 2

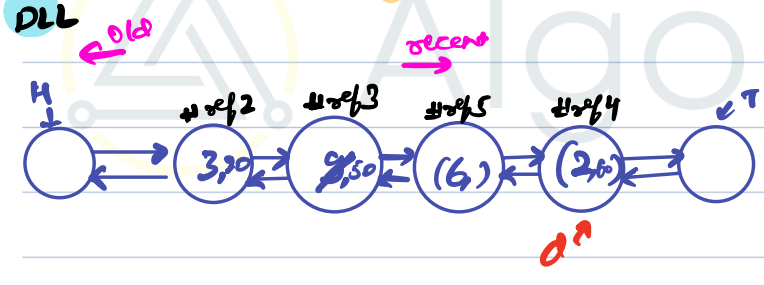
↓ 2 ↓ 6

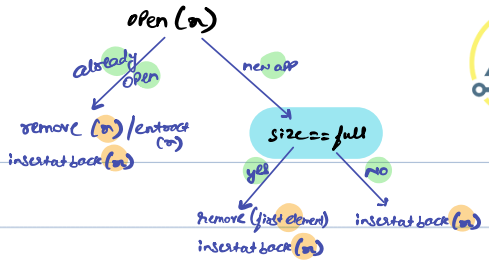


map

Integer	→	node
~~7~~	→	~~ref1~~
3	→	#ref2
9	→	#ref3
2	→	#ref4
6	→	#ref5

DLL





IIPSuedo code

```
Class LRUCache {
    Class Node {
        int key; int val;
        Node next; Node prev;
    }
    HashMap < Integer, Node > hm;
    Node H = new Node();
    Node T = new Node();
    int cap;
    LRUCache ( int K ) {
        cap = K;
        hm = new HashMap<>();
        H.next = T;
        T.prev = H;
    }
    int get (int key){
        if (hm.containsKey (key) == false) { return -1; }
        else {
            int ans = hm.get (key).value;
            Node temp = delete ( H , hm.get (key));
            add (H, T, temp);
            return ans;
        }
    }
    void Put (int key, int value){
        Node res = hm.get(key);
        if ( res == null) {
            if (hmap.size() == cap) {
                hm.remove(H.next.key);
                delete ( H.next);
            }
            Node n1 = new Node()
            n1.key = key;
            n1.val = value;
            hmap.Put (key, n1);
```

O(1) ←

O(1) ←

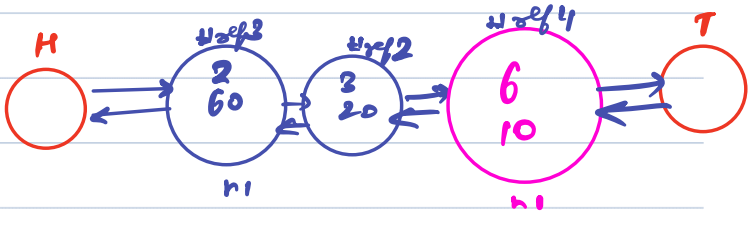
```
        add(n1);
    }
    else { //res != null
        Node temp = remove(H, res);
        temp.val = value;
        hmap.put(key, temp);
        add(temp);
    }
}
```

cap = 3

Data: (7,10) (3,20) (2,60) get(3) (6,100) get(7) (2,150)

```
void Put(int key, int value){
    Node res = hm.get(key);
    if(res == null){
        if(hmap.size() == cap){
            hm.remove(H.next.key);
            delete(H.next);
        }
        Node n1 = new Node()
            n1.key = key;
            n1.val = value;
        hmap.Put(key, n1);
        add(n1);
    }
    else { //res != null
        Node temp = remove(H, res);
        temp.val = value;
        hmap.Put(key, temp);
        add(temp);
    }
}
```

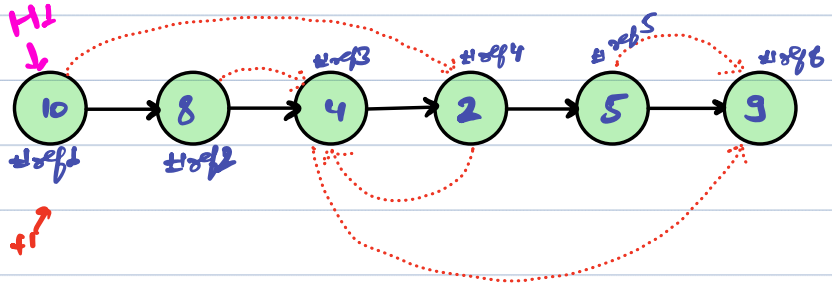
~~7 — #ref1~~
3 — #ref2
2 — #ref3
6 — #ref4



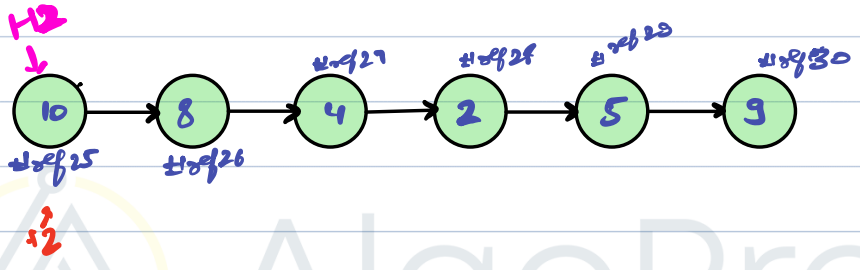

```
int get(int key){
    if(hm.containsKey(key) == false){ return -1; }
    else{
        int ans = hm.get(key).value;
        Node temp = delete(H, hm.get(key));
        add(H, T, temp);
        return ans;
    }
}
```

ans = 20

Q) Clone a LL with random Pointer.




```
Class Node{
  int val;
  Node next;
  Node random;
}
```



//Ideal

HM

Node	Node
#ref1	#ref25
#ref2	#ref26
#ref3	#ref27
#ref4	#ref28

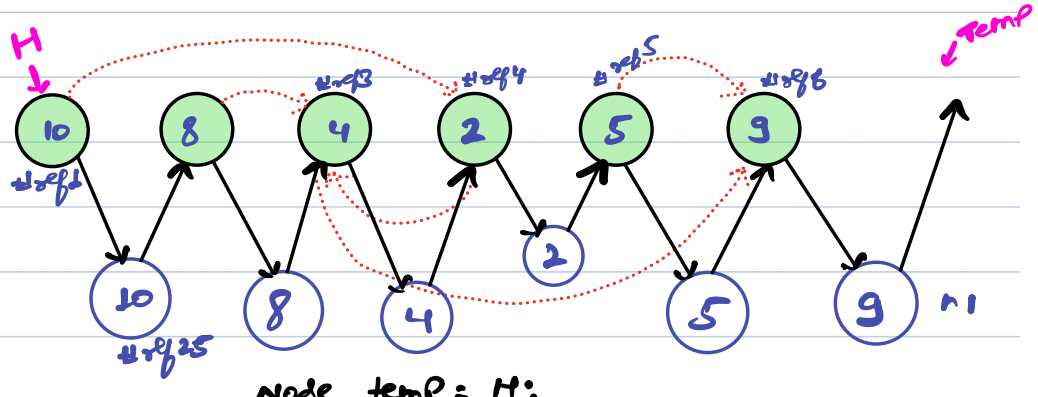
```
t2.random = map.get(t1.random);
t1 = t1.next
t2 = t2.next
```

T.C: O(N)

S.C: O(N)

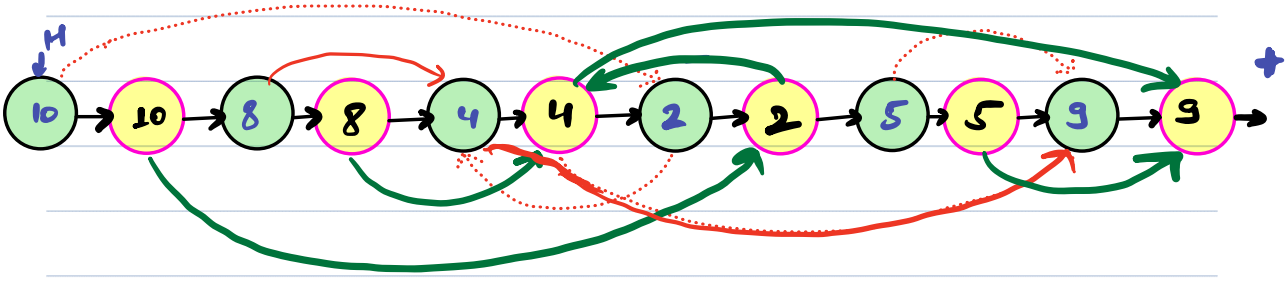
//idea 2

Step 1: insertion



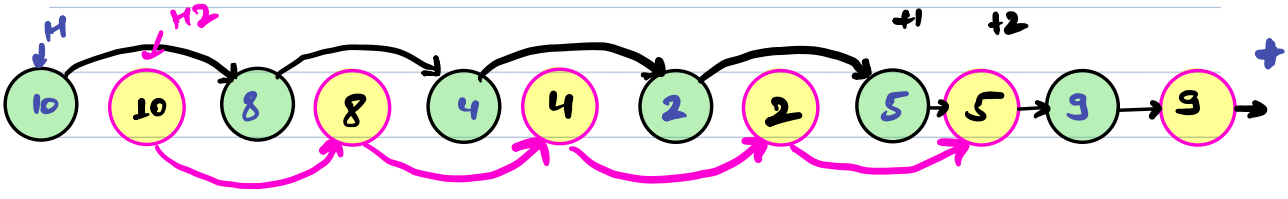

```
Node temp = H;
while (temp != null) {
    Node n1 = new Node(temp.val);
    Node tempP1 = n1.next;
    temp.next = n1;
    n1.next = tempP1;
    temp = tempP1;
}
```

Step2 : Setting the Random Pointer.




```
Node t = H;

while (t != null) {
    if (t.random != null){
        t.next.random = t.random.next;
    }
    t = t.next.next;
}
```

Step3: Segregate the old & new LL




```
Node H2 = H.next;
Node t1 = H;
Node t2 = H.next;
while (t1 != null) {
    Node t1P1 = t1.next.next;
    Node t2P1 = t2.next.next;
    if ( ) {
    }
    H.next = t1P1;
    t2.next = t2P1;
    t1 = t1P1;
    t2 = t2P1;
}
return H2;
```

edge case →

T.C: O(N)

S.C: O(1)

Delete node in Doubly Linked List

Java Code:

```
class Solution
{
    // function returns the head of the linkedlist
    Node deleteNode(Node head,int x)
    {
            Node d = head;
            while(x>1){
                d = d.next;
                x--;
            }
            Node dm1 = d.prev;
            Node dp1 = d.next;
            if(dm1 == null){
                //head node
                head = head.next;
                head.prev = null;
                return head;
            } else if(dp1 == null){
                //tail node
                dm1.next = null;
                d.prev = null;

            } else {
                dm1.next = dp1;
                dp1.prev = dm1;
                d.next = null;
                d.prev = null;
            }
            return head;
    }
}
```

C++ Code:

```
#include <iostream>

struct Node {
    int data;
```

```
    Node* next;
    Node* prev;
    Node(int val) : data(val), next(nullptr), prev(nullptr) {}
};

class Solution {
public:
    Node* deleteNode(Node* head, int x) {
        Node* d = head;
        while (x > 1) {
            d = d->next;
            x--;
        }

        Node* dm1 = d->prev;
        Node* dp1 = d->next;

        if (dm1 == nullptr) {
            // head node
            head = head->next;
            if (head != nullptr) {
                head->prev = nullptr;
            }
            delete d;
            return head;
        } else if (dp1 == nullptr) {
            // tail node
            dm1->next = nullptr;
            d->prev = nullptr;
            delete d;
        } else {
            dm1->next = dp1;
            dp1->prev = dm1;
            d->next = nullptr;
            d->prev = nullptr;
            delete d;
        }

        return head;
    }
};
```

Python Code:

```
class Node:
```

```
    def __init__(self, val):
        self.data = val
        self.next = None
        self.prev = None

class Solution:
    def deleteNode(self, head, x):
        d = head
        while x > 1:
            d = d.next
            x -= 1

        dm1 = d.prev
        dp1 = d.next

        if dm1 is None:
            # head node
            head = head.next
            if head is not None:
                head.prev = None
            del d
            return head
        elif dp1 is None:
            # tail node
            dm1.next = None
            d.prev = None
            del d
        else:
            dm1.next = dp1
            dp1.prev = dm1
            d.next = None
            d.prev = None
            del d

        return head
```

LRU Cache

Java Code:

```
 class Node{
        int key=0;
        int val=0;
```

```
        Node prev = null;
        Node next = null;
        Node(){
        }
    }
class LRUCache {
    HashMap<Integer , Node> hm;
    Node H = new Node();
    Node T= new Node();
    int cap;
    public LRUCache(int capacity) {
        hm = new HashMap<>();
        H.next = T;
        T.prev = H;
        cap = capacity;
    }

    public int get(int key) {
        if(hm.containsKey(key) == false) return -1;
        else {
            int ans = hm.get(key).val;
            Node temp =delete(H , hm.get(key));
            add(H,T,temp);
            return ans;
        }
    }

    public void put(int key, int value) {
        Node res = hm.get(key);
        if(res == null){
            if(hm.size() == cap){
              hm.remove(H.next.key);
              delete(H , H.next);
            }
            Node n1 = new Node();
            n1.key = key;
            n1.val = value;
            hm.put(key , n1);
            add(H,T,n1);
        } else {

            Node temp = delete(H,res);
            temp.val = value;
            add(H,T,temp);
        }
    }

    public void add(Node H , Node T , Node n1){
```

```
        Node tm1 = T.prev;
        tm1.next = n1;
        n1.next = T;

        T.prev = n1;
        n1.prev = tm1;
    }
    public Node delete(Node head , Node d){
        Node dm1 = d.prev;
        Node dp1 = d.next;

        dm1.next = dp1;
        dp1.prev = dm1;

        d.next = null;
        d.prev = null;

        return d;
    }
}
```

C++ Code:

```
#include <iostream>
#include <unordered_map>

class LRUCache {
    struct Node {
        int key = 0;
        int val = 0;
        Node* prev = nullptr;
        Node* next = nullptr;
        Node() {}
    };

    std::unordered_map<int, Node*> hm;
    Node* H = new Node();
    Node* T = new Node();
    int cap;

public:
    LRUCache(int capacity) {
        hm.clear();
        H->next = T;
        T->prev = H;
        cap = capacity;
```

```
    }

    int get(int key) {
        if (hm.find(key) == hm.end()) return -1;
        else {
            int ans = hm[key]->val;
            Node* temp = deleteNode(H, hm[key]);
            addNode(H, T, temp);
            return ans;
        }
    }

    void put(int key, int value) {
        Node* res = hm[key];
        if (res == nullptr) {
            if (hm.size() == cap) {
                hm.erase(H->next->key);
                deleteNode(H, H->next);
            }
            Node* n1 = new Node();
            n1->key = key;
            n1->val = value;
            hm[key] = n1;
            addNode(H, T, n1);
        } else {
            Node* temp = deleteNode(H, res);
            temp->val = value;
            addNode(H, T, temp);
        }
    }

private:
    void addNode(Node* H, Node* T, Node* n1) {
        Node* tm1 = T->prev;
        tm1->next = n1;
        n1->next = T;

        T->prev = n1;
        n1->prev = tm1;
    }

    Node* deleteNode(Node* head, Node* d) {
        Node* dm1 = d->prev;
        Node* dp1 = d->next;

        dm1->next = dp1;
        dp1->prev = dm1;
```

```
        d->next = nullptr;
        d->prev = nullptr;

        return d;
    }
};
```

Python Code:

```
class LRUCache:

    class Node:
        def __init__(self):
            self.key = 0
            self.val = 0
            self.prev = None
            self.next = None

    def __init__(self, capacity):
        self.hm = {}
        self.H = self.Node()
        self.T = self.Node()
        self.H.next = self.T
        self.T.prev = self.H
        self.cap = capacity

    def get(self, key):
        if key not in self.hm:
            return -1
        else:
            ans = self.hm[key].val
            temp = self.deleteNode(self.H, self.hm[key])
            self.addNode(self.H, self.T, temp)
            return ans

    def put(self, key, value):
        res = self.hm.get(key)
        if res is None:
            if len(self.hm) == self.cap:
                del self.hm[self.H.next.key]
                self.deleteNode(self.H, self.H.next)
            n1 = self.Node()
            n1.key = key
            n1.val = value
            self.hm[key] = n1
            self.addNode(self.H, self.T, n1)
```

```
        else:
            temp = self.deleteNode(self.H, res)
            temp.val = value
            self.addNode(self.H, self.T, temp)

    def addNode(self, H, T, n1):
        tm1 = T.prev
        tm1.next = n1
        n1.next = T

        T.prev = n1
        n1.prev = tm1

    def deleteNode(self, head, d):
        dm1 = d.prev
        dp1 = d.next

        dm1.next = dp1
        dp1.prev = dm1

        d.next = None
        d.prev = None

        return d
```

Copy List with Random Pointer

Java Code:

```
class Solution {
    public Node copyRandomList(Node head) {
        if(head == null) return null;
        Node temp = head;
        while(temp != null){
            Node n1 = new Node(temp.val);
            Node tempp1 = temp.next;
            temp.next = n1;
            n1.next = tempp1;
            temp = tempp1;
        }

        Node t = head;
```

```
        while(t != null){
            if(t.random != null){
                t.next.random = t.random.next;
            }
            t = t.next.next;
        }
        Node H2 = head.next;
        Node t1 = head;
        Node t2 = head.next;
        while(t1 != null ){
            Node t1p1 = t1.next.next;
            Node t2p1 = null;
            if(t2.next != null) {
                t2p1  = t2.next.next;
            }
            t1.next = t1p1;
            t2.next = t2p1;
            t1 = t1p1;
            t2 = t2p1;
        }
        return H2;
    }
}
```

C++ Code:

```
#include <unordered_map>

class Node {
public:
    int val;
    Node* next;
    Node* random;

    Node(int _val) {
        val = _val;
        next = nullptr;
        random = nullptr;
    }
};

class Solution {
public:
    Node* copyRandomList(Node* head) {
        if (head == nullptr) return nullptr;

        Node* temp = head;
```

```
        while (temp != nullptr) {
            Node* n1 = new Node(temp->val);
            Node* tempp1 = temp->next;
            temp->next = n1;
            n1->next = tempp1;
            temp = tempp1;
        }

        Node* t = head;
        while (t != nullptr) {
            if (t->random != nullptr) {
                t->next->random = t->random->next;
            }
            t = t->next->next;
        }

        Node* H2 = head->next;
        Node* t1 = head;
        Node* t2 = head->next;
        while (t1 != nullptr) {
            Node* t1p1 = t1->next->next;
            Node* t2p1 = (t2->next != nullptr) ? t2->next->next : nullptr;
            t1->next = t1p1;
            t2->next = t2p1;
            t1 = t1p1;
            t2 = t2p1;
        }

        return H2;
    }
};
```

Python Code:

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None
        self.random = None

class Solution(object):
    def copyRandomList(self, head):
        if not head:
            return None

        temp = head
```

```
while temp:
    n1 = Node(temp.val)
    tempp1 = temp.next
    temp.next = n1
    n1.next = tempp1
    temp = tempp1

t = head
while t:
    if t.random:
        t.next.random = t.random.next
    t = t.next.next

H2 = head.next
t1 = head
t2 = head.next
while t1:
    t1p1 = t1.next.next if t1.next else None
    t2p1 = t2.next.next if t2.next else None
    t1.next = t1p1
    t2.next = t2p1
    t1 = t1p1
    t2 = t2p1

return H2
```

Today's agenda

- Flatten binary tree to LinkedList
- Convert a binary tree to a circular DLL

BT

```
Class Node {
    int val;
    Node left;
    Node right;
}
```

LL

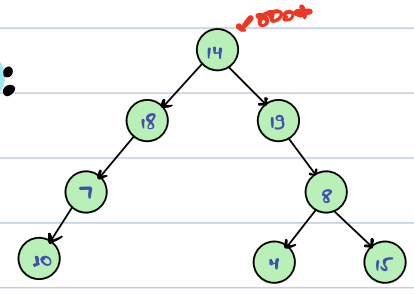
```
Class Node {
    int val;
    Node next;
}
```

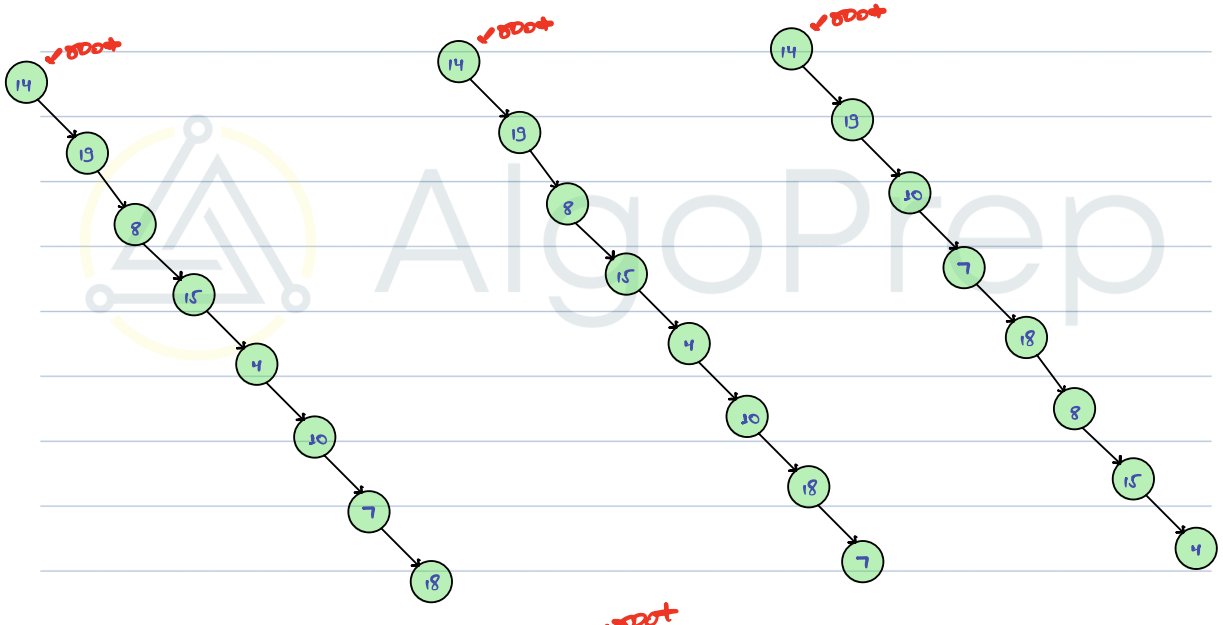
Q) Flatten Binary tree to LinkedList

↳ you have to convert Binary tree to linkedList, "right" should be considered next and left should be null.

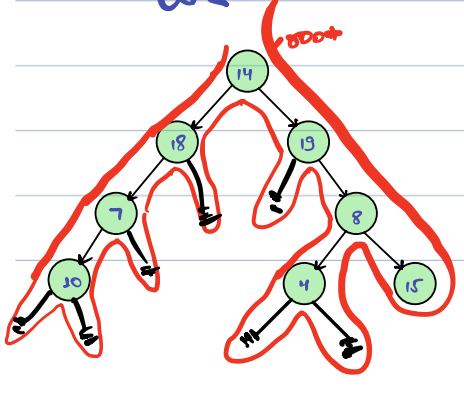
↳ Preorder of tree.

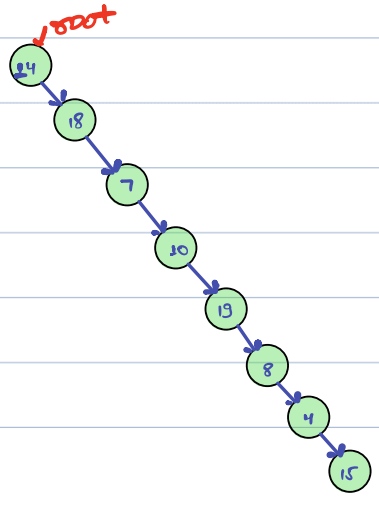
Ex:

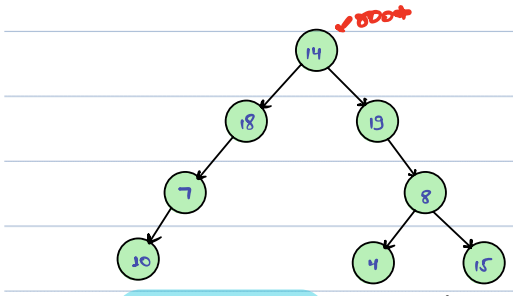


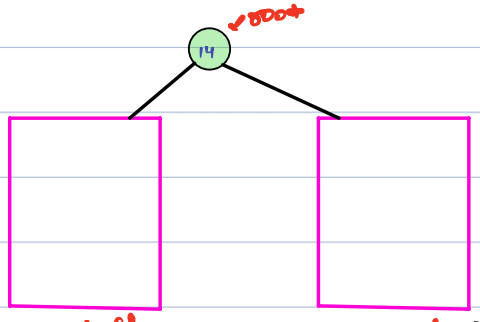


Ans:









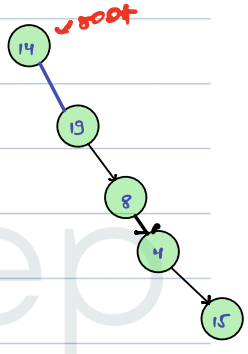
Null
Null
Non-Null
Non-Null

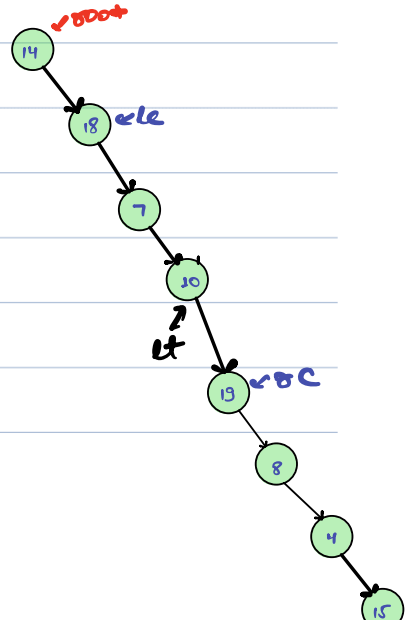
Null → return root
Non-Null → return root
Null
Non-Null

```
Node flattenMain(Node root){
    flatten(root);
    return root;
}
```

```
Node flatten(Node root){
    if(root == null){ return null; }

    Node lt = flatten(root.left);
    Node rt = flatten(root.right);
    if(lt == null && rt == null){
        return root;
    }
    else if(lt == null && rt != null){
        return rt;
    }
    else if(lt != null && rt == null){
        Node lc = root.left;
        root.right = lc;
        root.left = null;
        return lt;
    }
    else{
        Node lc = root.left;
        Node rc = root.right;
        root.right = lc;
        root.left = null;
        lt.right = rc;
        return rt;
    }
}
```

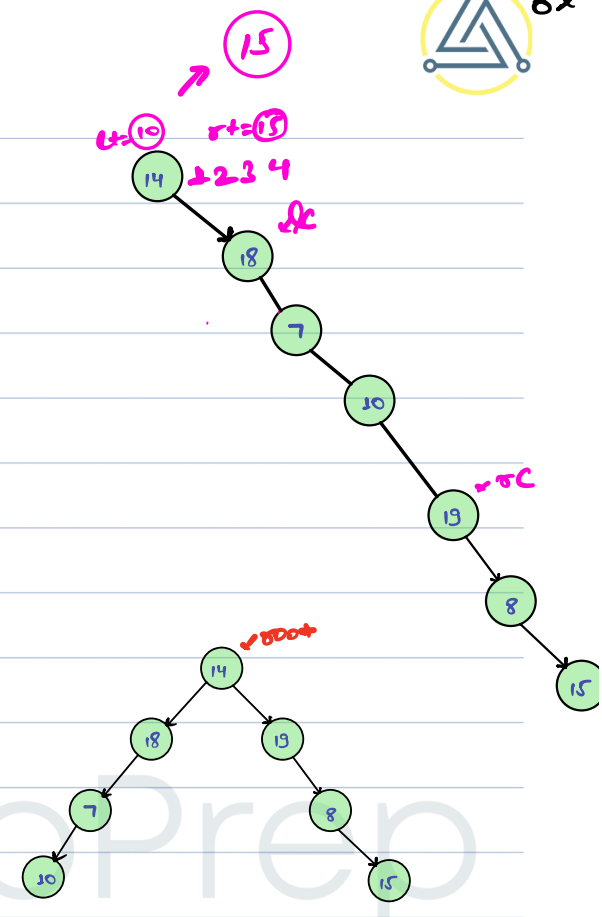
T.C: O(N)
S.C: O(H)



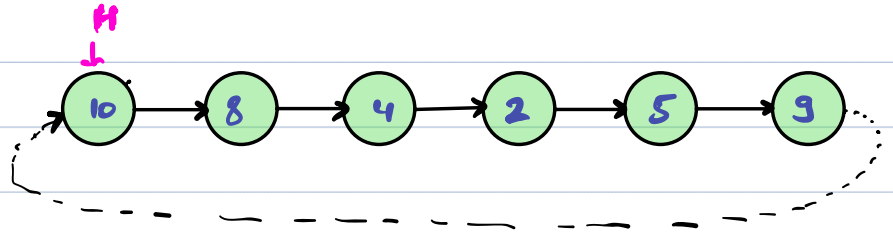


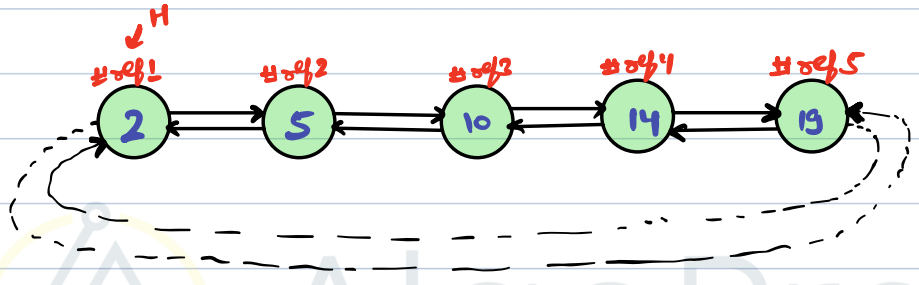
tracing

```
Node flatten (Node root){
① if(root == null){ return null; }
② Node lt= flatten (root.left);
③ Node rt= flatten (root.right);
   if( lt== null && rt== null){
        return root;
   }
   else if ( lt== null && rt!= null){
        return rt;
   }
④  else if ( lt!=null && rt!=null){
        Node lc= root.left;
        root.right= lc;
        root.left = null;
        return lt;
   }
   else {
        Node lc= root.left;
        Node rc= root.right;
        root.right= lc;
        root.left=null;
        lc.right= rc;
        return rt;
   }
}
```

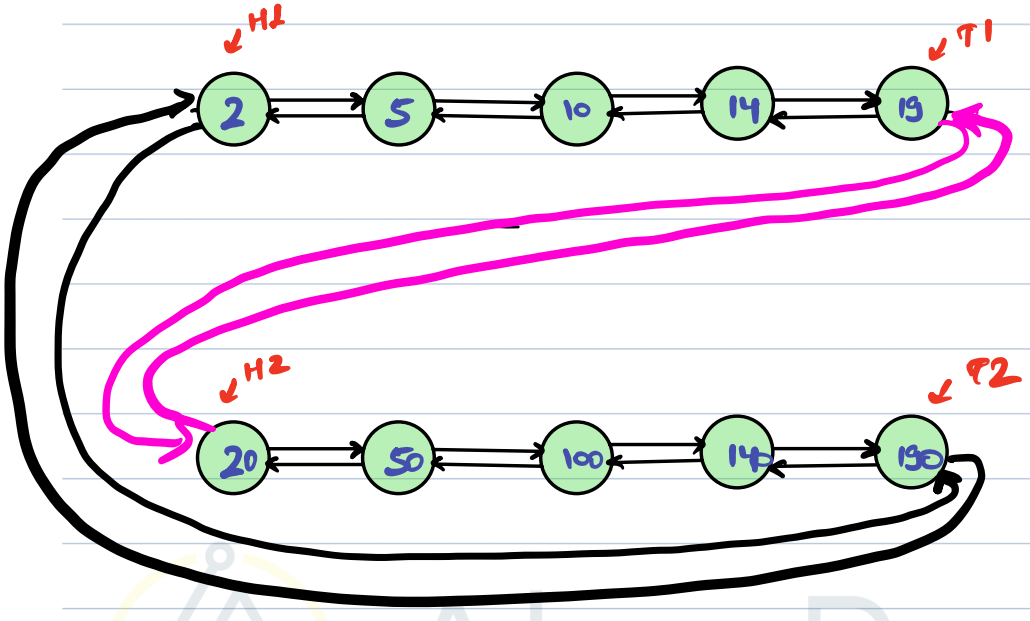


//Circular LinkedList





Q) Combine given two circular doubly LinkedList.




```
Node Concatenate (Node H1, Node H2){

    Node T1 = H1.Prev;
    Node T2 = H2.Prev;
    T1.next = H2;
    H2.Prev = T1;
    T2.next = H1;
    H1.Prev = T2;

    Return H1;
}
```

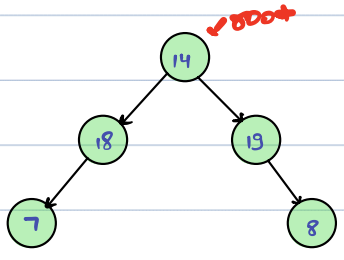
BT

```
Class Node {
    int val;
    Node Right;
    Node left;
```

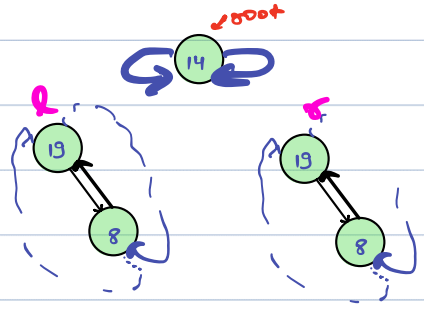
LL

```
Class Node {
    int val;
    Node next;
    Node Prev;
```

Q) Convert Binary tree to a circular DLL



↳ follow Inorder: → LNR




```
Node btreeDLL(Node root){
    if(root==null){return null;}

    Node l = btreeDLL(root.left);
    Node r = btreeDLL(root.right);

    if(l==null && r==null){
        root.left = root.right = root;
        return root;
    }
    else if(l==null && r!=null){
        root.left = root.right = root;
        Concatenate(root, r);
        return root;
    }
    else if(l!=null && r==null){
        root.left = root.right = root;
        Concatenate(l, root);
        return l;
    }
    else{
        root.left = root.right = root;
```

```
    }
    Concatenate(l, root);
    Concatenate(l, r);
    return l;
}
```

Flatten Binary Tree to Linked List

Java Code:

```
class Solution {
    public void flatten(TreeNode root) {
        flattn(root);
    }
    public TreeNode flattn(TreeNode root){
        if(root == null) return null;

        TreeNode lt = flattn(root.left);
        TreeNode rt = flattn(root.right);

        if(lt == null && rt == null) return root;
        else if(lt == null && rt != null) return rt;
        else if(lt != null && rt == null){
            TreeNode lc = root.left;
            root.right = lc;
            root.left = null;
            return lt;
        } else {
            TreeNode lc = root.left;
            TreeNode rc = root.right;
            root.right = lc;
            root.left = null;
            lt.right = rc;
            return rt;
        }

    }
}
```

C++ Code:

```
#include <vector>

class Solution {
public:
    double findMedianSortedArrays(std::vector<int>& nums1, std::vector<int>& nums2) {
        if (nums1.size() > nums2.size())
            return findMedianSortedArrays(nums2, nums1);

        int n1 = nums1.size();
```

```
        int n2 = nums2.size();
        int lo = 0, hi = n1;
        int t = n1 + n2;

        while (lo <= hi) {
            int m1 = (lo + hi) / 2;
            int m2 = (t + 1) / 2 - m1;

            int am = (m1 == n1) ? INT_MAX : nums1[m1];
            int am1 = (m1 == 0) ? INT_MIN : nums1[m1 - 1];
            int bm = (m2 == n2) ? INT_MAX : nums2[m2];
            int bm1 = (m2 == 0) ? INT_MIN : nums2[m2 - 1];

            if (am1 <= bm && bm1 <= am) {
                if (t % 2 != 0) {
                    return std::max(am1, bm1);
                } else {
                    double lmax = std::max(am1, bm1);
                    double rmax = std::min(am, bm);
                    return (lmax + rmax) / 2.0;
                }
            } else if (am1 > bm) {
                hi = m1 - 1;
            } else if (bm1 > am) {
                lo = m1 + 1;
            }
        }

        return 0.0;
    }
};
```

Python Code:

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        if len(nums1) > len(nums2):
            return self.findMedianSortedArrays(nums2, nums1)

        n1 = len(nums1)
        n2 = len(nums2)
        lo, hi = 0, n1
        t = n1 + n2

        while lo <= hi:
            m1 = (lo + hi) // 2
```

```
            m2 = (t + 1) // 2 - m1

            am = nums1[m1] if m1 < n1 else float('inf')
            am1 = nums1[m1 - 1] if m1 > 0 else float('-inf')
            bm = nums2[m2] if m2 < n2 else float('inf')
            bm1 = nums2[m2 - 1] if m2 > 0 else float('-inf')

            if am1 <= bm and bm1 <= am:
                if t % 2 != 0:
                    return max(am1, bm1)
                else:
                    lmax = max(am1, bm1)
                    rmax = min(am, bm)
                    return (lmax + rmax) / 2.0
            elif am1 > bm:
                hi = m1 - 1
            elif bm1 > am:
                lo = m1 + 1

        return 0.0
```

Binary Tree to CDLL

Java Code:

```
class Solution
{
    //Function to convert binary tree into circular doubly linked list.
    Node concatenate(Node H1 , Node H2){
        Node T1 = H1.left;
        Node T2 = H2.left;
        T1.right = H2;
        H2.left = T1;
        T2.right = H1;
        H1.left = T2;
        return H1;
    }
    Node bTreeToClist(Node root)
    {
       if(root == null) return null;
       Node l = bTreeToClist(root.left);
       Node r = bTreeToClist(root.right);
       if(l == null && r==null){
           root.left = root.right = root;
```

```
            return root;
        } else if(l == null && r != null){
            root.left = root.right = root;
            concatenate(root , r);
            return root;

        } else if(l != null && r==null){
            root.left = root.right = root;
            concatenate(l,root);
            return l;
        } else {
            root.left = root.right = root;
            concatenate(l,root);
            concatenate(l,r);
            return l;
        }
    }

}
```

C++ Code:

```
#include <iostream>

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    // Function to convert binary tree into circular doubly linked list.
    Node* bTreeToCList(Node* root) {
        if (!root)
            return nullptr;

        Node* l = bTreeToCList(root->left);
        Node* r = bTreeToCList(root->right);

        if (!l && !r) {
            root->left = root->right = root;
            return root;
        } else if (!l && r) {
```

```
            root->left = root->right = root;
            concatenate(root, r);
            return root;
        } else if (l && !r) {
            root->left = root->right = root;
            concatenate(l, root);
            return l;
        } else {
            root->left = root->right = root;
            concatenate(l, root);
            concatenate(l, r);
            return l;
        }
    }

private:
    Node* concatenate(Node* H1, Node* H2) {
        Node* T1 = H1->left;
        Node* T2 = H2->left;
        T1->right = H2;
        H2->left = T1;
        T2->right = H1;
        H1->left = T2;
        return H1;
    }
};
```

Python Code:

```
class Node:
    def __init__(self, val):
        self.data = val
        self.left = None
        self.right = None

class Solution:

    # Function to convert binary tree into circular doubly linked list.
    def bTreeToClist(self, root):
        if not root:
            return None

        def concatenate(H1, H2):
            T1 = H1.left
            T2 = H2.left
            T1.right = H2
```

```
    H2.left = T1
    T2.right = H1
    H1.left = T2
    return H1

l = self.bTreeToClist(root.left)
r = self.bTreeToClist(root.right)

if not l and not r:
    root.left = root.right = root
    return root
elif not l and r:
    root.left = root.right = root
    return concatenate(root, r)
elif l and not r:
    root.left = root.right = root
    return concatenate(l, root)
else:
    root.left = root.right = root
    return concatenate(concatenate(l, root), r)
```