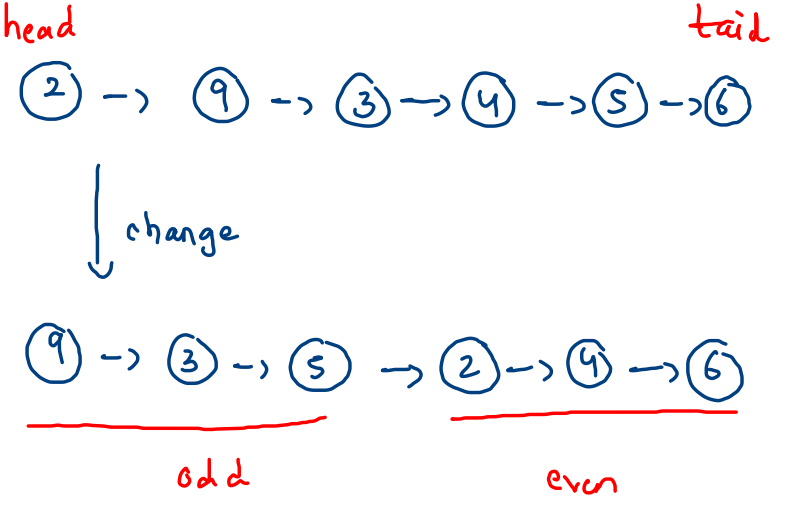
Odd Even Linked List

1. You are given a partially written LinkedList class.
2. You are required to complete the body of oddEven function. The function is expected to tweak the list such that all odd values are followed by all even values. The relative order of elements should not change. Also, take care of the cases when there are no odd or no even elements. Make sure to properly set head, tail and size as the function will be tested by calling addFirst and addLast.
3. Input and Output is managed for you.



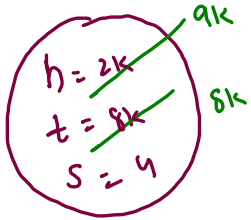
T: O(n)

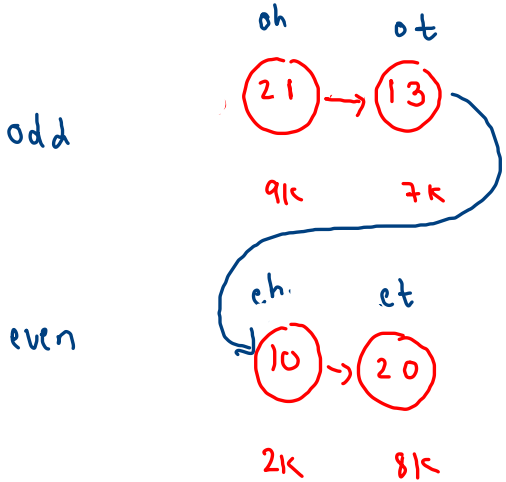
S: O(1)

O(1) space

without using new Node( )

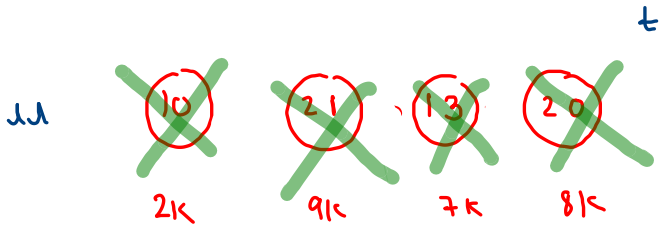
```
addLast (Node node) {
    if (s==0) head = tail = node;
    else
        tail.next = node;
        tail = node;
}
```

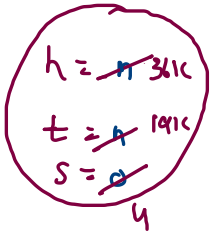


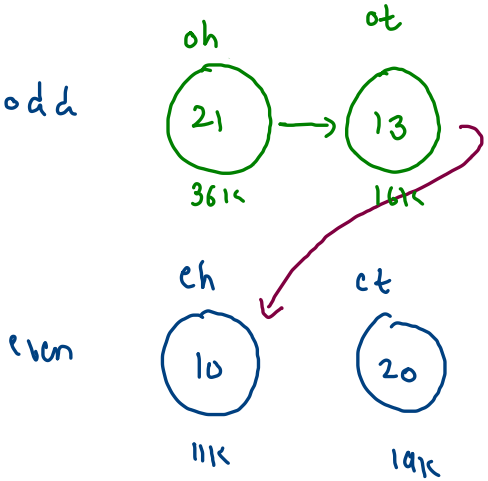



```
even.aL(c);
odd.aL(c);
```

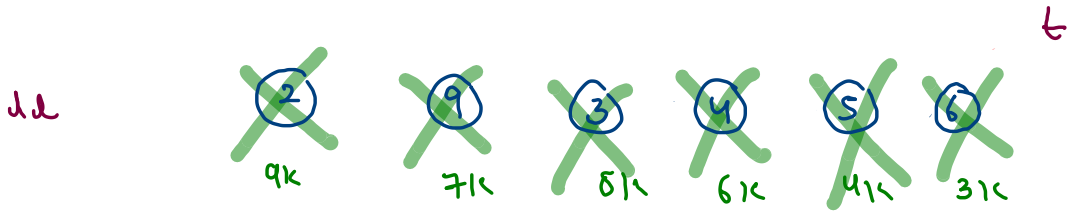
new Node( )



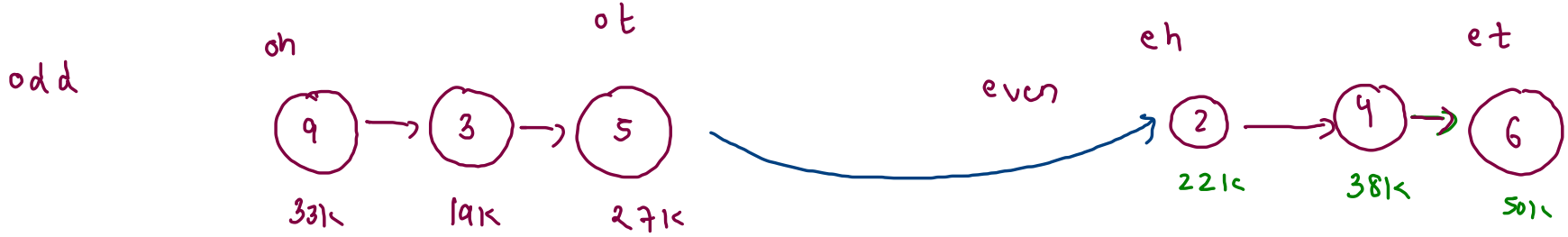


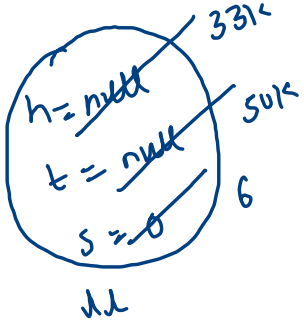



```
ll.rF();
if (t.data%2==0)
    even.aL(t.data);
else
    odd.aL(t.data);
```



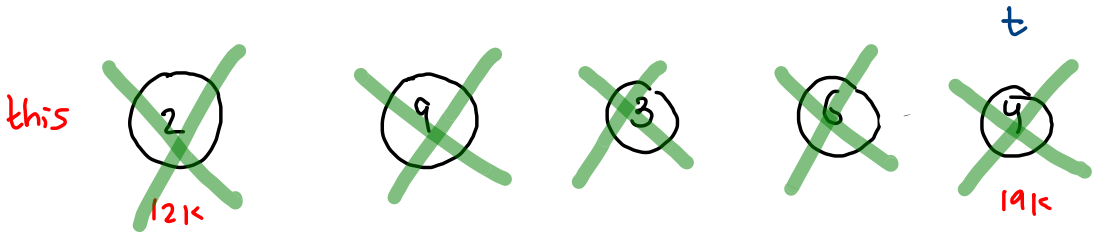

```
ll.xF ()
if (t.data -> even) even.al(t.data)
else    odd.al (t.data);
```

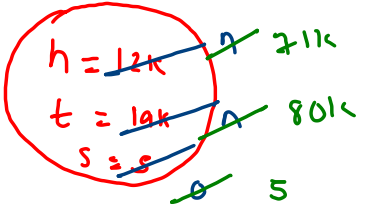


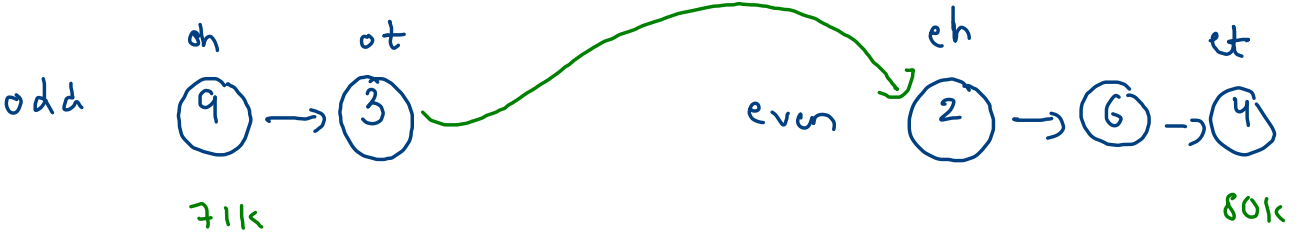



```
ot.next = eh.
ll. head = oh;
ll. tail = et;
```

```java
public void oddEven(){
    LinkedList odd = new LinkedList();
    LinkedList even = new LinkedList();


    while(this.size != 0) {

        Node temp = this.head;
        this.removeFirst();

        if(temp.data % 2 == 0) {
            //temp.data is even
            even.addLast(temp.data);
        }
        else {
            //temp.data is odd
            odd.addLast(temp.data);
        }

    }

    if(odd.size != 0 && even.size != 0) {
        //both odd and even are present
        odd.tail.next = even.head;

        this.head = odd.head;
        this.tail = even.tail;
        this.size = odd.size + even.size;
    }
    else if(odd.size != 0) {
        //only odd is present
        this.head = odd.head;
        this.tail = odd.tail;
        this.size = odd.size;
    }
    else if(even.size != 0) {
        //only even is present
        this.head = even.head;
        this.tail = even.tail;
        this.size = even.size;
    }

}
```

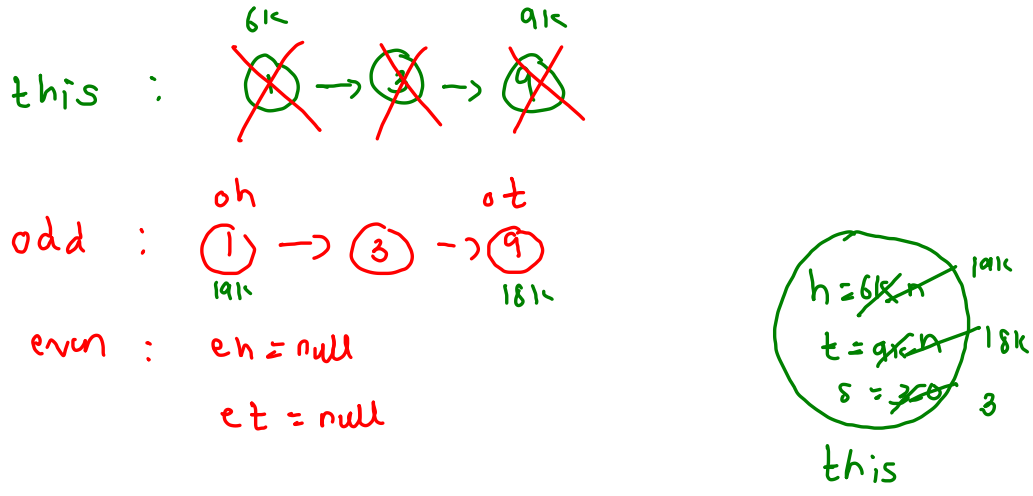




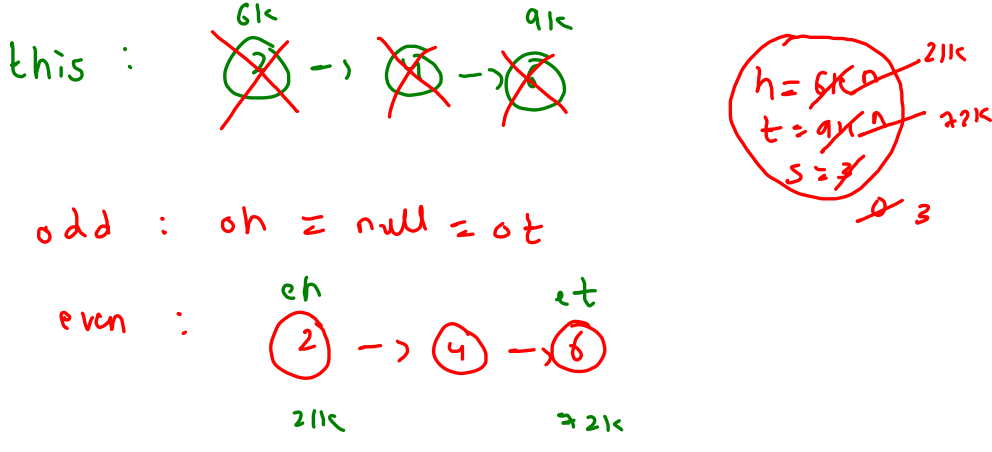


```java
public void oddEven(){
    LinkedList odd = new LinkedList();
    LinkedList even = new LinkedList();


    while(this.size != 0) {

        Node temp = this.head;
        this.removeFirst();

        if(temp.data % 2 == 0) {
            //temp.data is even
            even.addLast(temp.data);
        }
        else {
            //temp.data is odd
            odd.addLast(temp.data);
        }

    }

    if(odd.size != 0 && even.size != 0) {
        //both odd and even are present
        odd.tail.next = even.head;

        this.head = odd.head;
        this.tail = even.tail;
        this.size = odd.size + even.size;
    }
    else if(odd.size != 0) {
        //only odd is present
        this.head = odd.head;
        this.tail = odd.tail;
        this.size = odd.size;
    }
    else if(even.size != 0) {
        //only even is present
        this.head = even.head;
        this.tail = even.tail;
        this.size = even.size;
    }

}
```

only odd



only even



Intersection Point Of Linked Lists

$$k = l1 - x = l2 - y$$

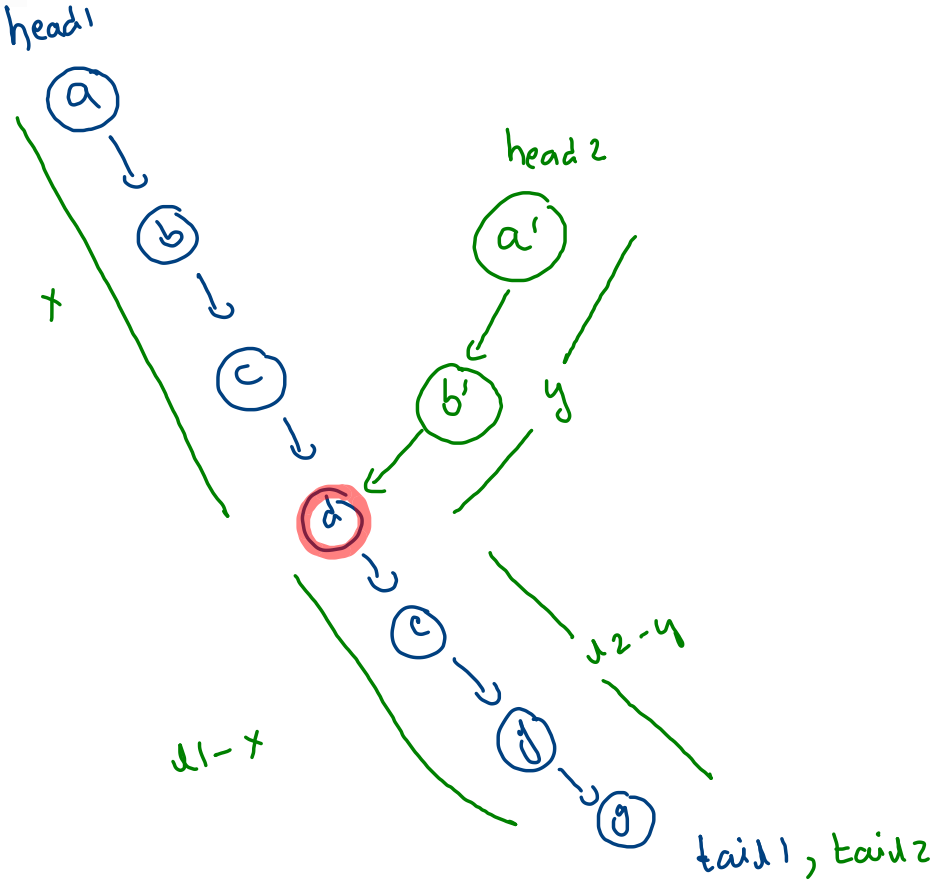
$$l1 = x + (l1 - x)$$

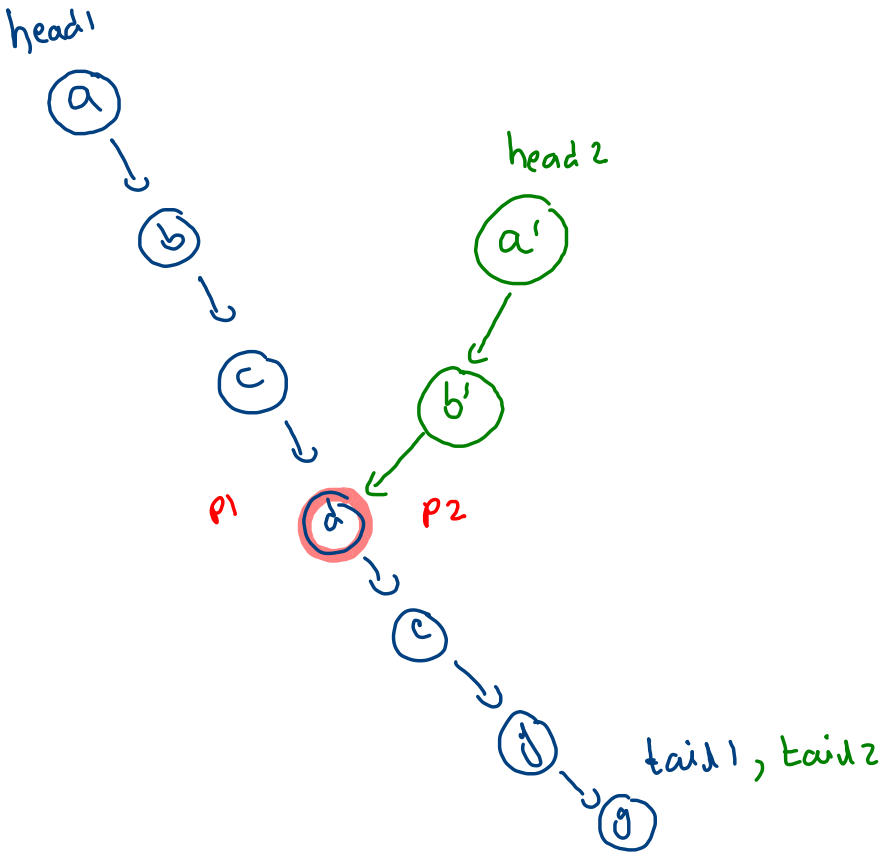
$$l2 = y + (l2 - y)$$

$$l1 = x + k \quad -(i)$$

$$l2 = y + k \quad -(ii)$$

$$\boxed{l1 - l2 = x - y}$$

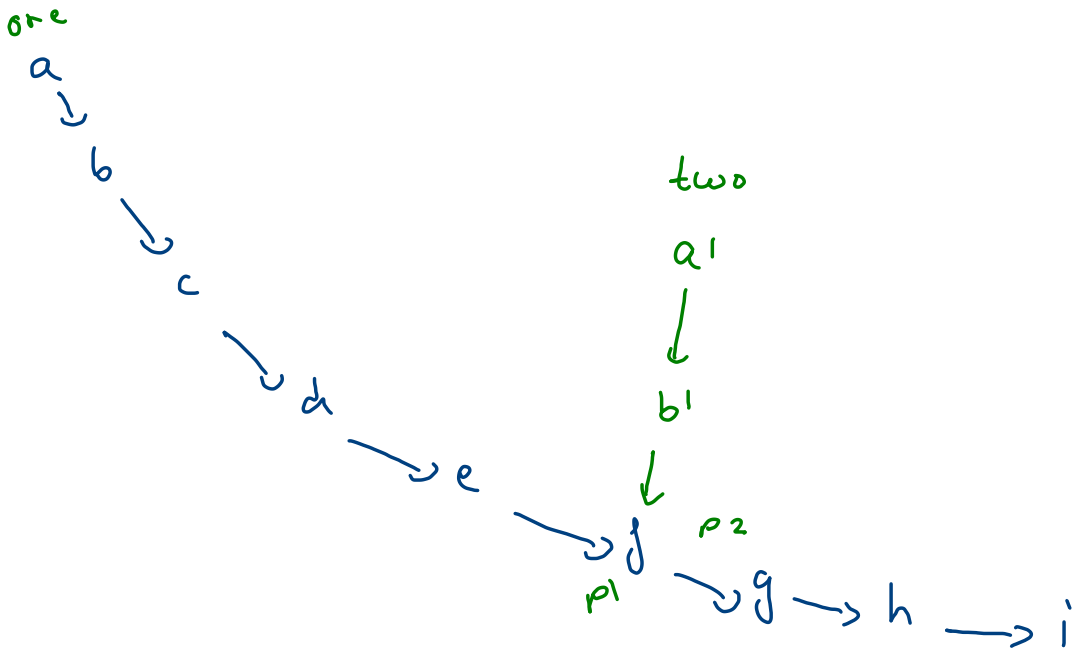




gap = d1 - d2

(x-y) = 7 - 6 = 1

```java
int l1 = one.size;
int l2 = two.size;

int gap = 0;

Node p1 = null; //p1 will point to longer ll
Node p2 = null; //p2 will point to shorted ll

if(l1 > l2) {
    p1 = one.head;
    p2 = two.head;
    gap = l1-l2;
}
else {
    p1 = two.head;
    p2 = one.head;
    gap = l2-l1;
}

//travel longer ll gap times
while(gap-- > 0) {
    p1 = p1.next;
}

//travel simultaneously in both ll
while(p1 != p2) {
    p1 = p1.next;
    p2 = p2.next;
}

if(p1 == null) {
    //no intersection point
    return -1;
}
else {
    return p1.data;
}
```



$l1 = 9$

$l2 = 6$

gap = ~~3~~ ~~2~~ ~~1~~ 0

ans: f

```java
int l1 = one.size;
int l2 = two.size;

int gap = 0;

Node p1 = null; //p1 will point to longer ll
Node p2 = null; //p2 will point to shorted ll

if(l1 > l2) {
    p1 = one.head;
    p2 = two.head;
    gap = l1-l2;
}
else {
    p1 = two.head;
    p2 = one.head;
    gap = l2-l1;
}

//travel longer ll gap times
while(gap-- > 0) {
    p1 = p1.next;
}

//travel simultaneously in both ll
while(p1 != p2) {
    p1 = p1.next;
    p2 = p2.next;
}

if(p1 == null) {
    //no intersection point
    return -1;
}
else {
    return p1.data;
}
```

one

a -> b -> c -> d -> e

p2

l1 = 5

two

a' -> b' -> c' -> d' -> e' -> g' -> g

l2 = 7

p1

gap = 2

Remove Duplicates In A Sorted Linked List

2 2 2 3 3 5 5 5 5 5

2 → 2 → 2 → 3 → 3 → 4 → 5 → 5 → 5 → 5

↓ change

2 → 3 → 4 → 5

this

~~2~~ → ~~2~~ → ~~2~~ → ~~3~~ → ~~3~~ → ~~4~~ → ~~5~~ → ~~5~~ → ~~5~~ → ~~5~~

9k 15k

ans

2 → 3 → 4 → 5

12k 19k

h = ~~9k~~ ~~n~~ 12k

t = ~~15k~~ ~~n~~ 19k

s = ~~10~~ ~~0~~ 4

this

h = 12k

t = 19k

s = 4

ans

```java
public void removeDuplicates(){
    LinkedList ans = new LinkedList();

    while(this.size != 0) {
        int data = this.head.data;
        this.removeFirst();

        if(ans.size == 0 || ans.tail.data != data) {
            ans.addLast(data);
        }
    }

    this.head = ans.head;
    this.tail = ans.tail;
    this.size = ans.size;
}
```

K Reverse In Linked List

$k = 3$

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h$

$c \rightarrow b \rightarrow a \rightarrow f \rightarrow e \rightarrow d \rightarrow g \rightarrow h$

$O(1)$: S

$O(n)$: T

this $\cancel{a} \to \cancel{b} \to \cancel{c} \to \cancel{d} \to \cancel{e} \to \cancel{f} \to \cancel{g} \to \cancel{h}$

k = 3

curr ch, ct

curr -> ck group

ans -> maintain overall ans

at

$f \to e \to d \to g \to h$

at.next = ch;

at = ct;

ans $c \to b \to a$

ah

```java
public void kReverse(int k) {
    LinkedList ans = new LinkedList(); //to store overall ans

    while(this.size >= k) {
        LinkedList curr = new LinkedList();

        //work on a single group of k nodes
        for(int i=0; i < k;i++) {
            int data = this.head.data;
            this.removeFirst();
            curr.addFirst(data);
        }

        //settle curr in overall ans
        if(ans.size == 0) {
            //no connection required
            ans.head = curr.head;
            ans.tail = curr.tail;
            ans.size = curr.size;
        }
        else {
            //connection required
            ans.tail.next = curr.head;
            ans.tail = curr.tail;
            ans.size += curr.size;
        }
    }

    while(this.size != 0) {
        int data = this.head.data;
        this.removeFirst();
        ans.addLast(data);
    }


    this.head = ans.head;
    this.tail = ans.tail;
    this.size = ans.size;

}
```

K = 4

this ~~a~~ → ~~b~~ → ~~c~~ → ~~d~~ → ~~e~~ → ~~f~~ → ~~g~~ → ~~h~~ → ~~i~~ → ~~j~~

at

h → g → f → e → i → j

h = ~~a~~ ah

t = ~~a~~ at

s = ~~0~~ 10

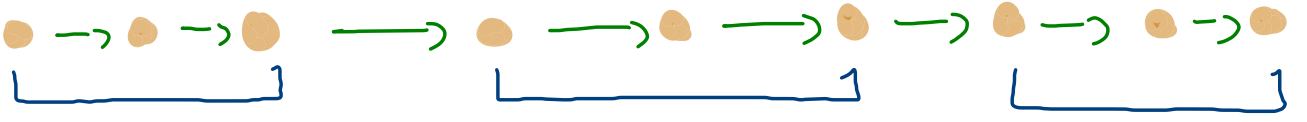
this

ans d → c → b → a

ah

```java
public void kReverse(int k) {
    LinkedList ans = new LinkedList(); //to store overall ans

    while(this.size >= k) {
        LinkedList curr = new LinkedList();

        //work on a single group of k nodes
        for(int i=0; i < k;i++) {
            int data = this.head.data;
            this.removeFirst();
            curr.addFirst(data);
        }

        //settle curr in overall ans
        if(ans.size == 0) {
            //no connection required
            ans.head = curr.head;
            ans.tail = curr.tail;
            ans.size = curr.size;
        }
        else {
            //connection required
            ans.tail.next = curr.head;
            ans.tail = curr.tail;
            ans.size += curr.size;
        }
    }

    while(this.size != 0) {
        int data = this.head.data;
        this.removeFirst();
        ans.addLast(data);
    }


    this.head = ans.head;
    this.tail = ans.tail;
    this.size = ans.size;

}
```

$k = 3$



$$\frac{n}{k} * k = O(n)$$

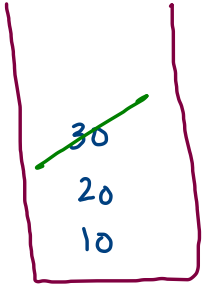
Analysis:

Java (DLL)	Self LL (SLL)
aF, rF -> O(1)	aF, rF -> O(1)
aL, rL -> O(1)	rL -> O(n)
	aL -> O(1)

LL to Stack

Stack
- pop O(1)
- push O(1)
- top O(1)

```java
LinkedList<Integer> list;

public LLToStackAdapter() {
  list = new LinkedList<>();
}


int size() {
  // write your code here
}

void push(int val) {
  // write your code here
}

int pop() {
  // write your code here
}

int top() {
  // write your code here
}
```

dev	Client
~~30~~ -> 20 -> 10	

rF

aF

gF

St.push(10)

St.push(20)

St.push(30)

St.pop()

St.top()

Linked List To Queue Adapter

```
int size() {
  // write your code here
}

void add(int val) {
  // write your code here
}

int remove() {
  // write your code here
}

int peek() {
  // write your code here
}
```

dev

client

~~10~~ → 20 → 30

~~10~~	20	30

rF

aL

oF

q.add(10)

q.add(20)

q.add(30)

q.front()

q.remove()