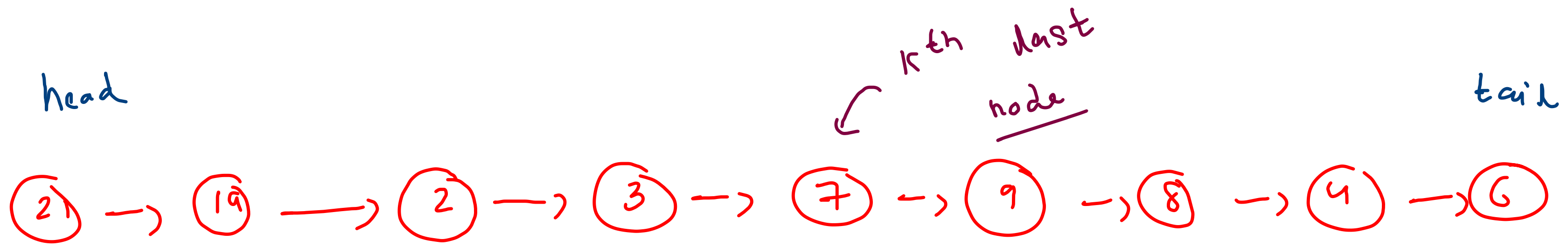


S

j

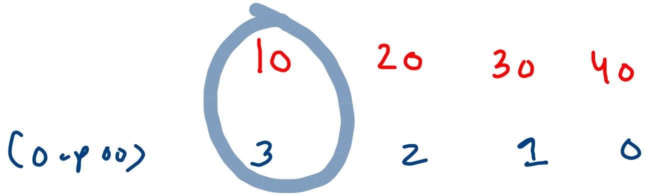
addLast 10
getFirst
addLast 20
addLast 30
getFirst
getLast
getAt 1
addLast 40
kthFromEn
d 3

```
10
10
30
20
10
40
20
```

k = 4

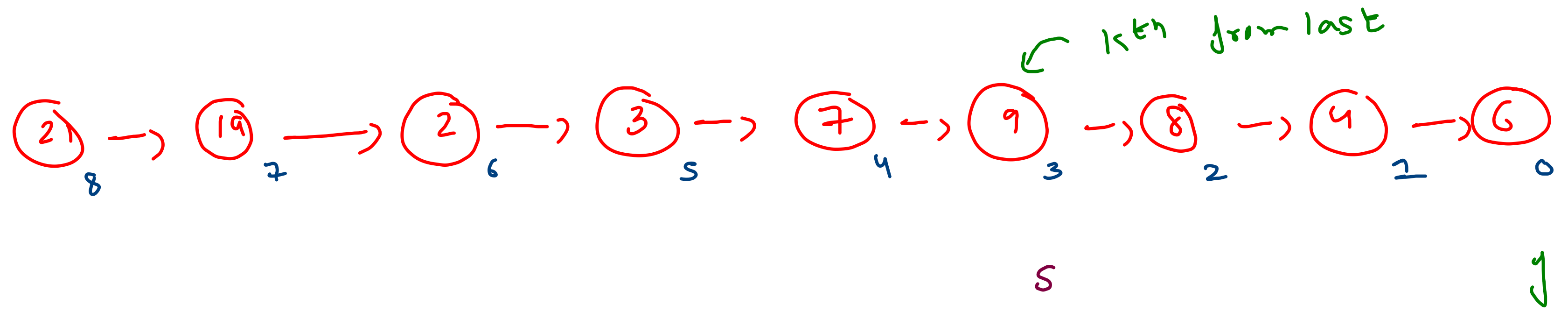
(i) reverse ×

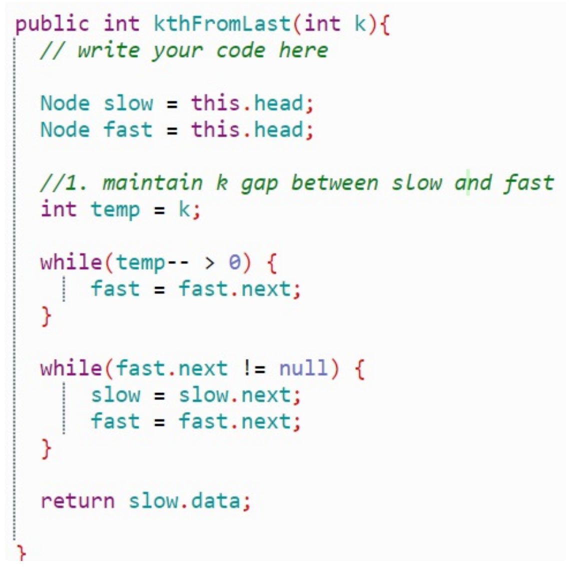
(ii) size ×



T: O(n), S: O(1)

k posn is 0 based





```
public int kthFromLast(int k){
  // write your code here

  Node slow = this.head;
  Node fast = this.head;

  //1. maintain k gap between slow and fast
  int temp = k;

  while(temp-- > 0) {
      fast = fast.next;
  }

  while(fast.next != null) {
      slow = slow.next;
      fast = fast.next;
  }

  return slow.data;

}
```

temp = ~~3~~ ~~2~~ ~~1~~ 0

k = 3

fast.next == null

ll : 2 -> 3 -> 3 -> 5 -> 5 -> 5 -> 6 -> 7 -> 7

5k 2k

al : 2

1

```
while (   ) {
node <-- ll.rF();
    if (true) {
        al.aL(node);
    }
}
```

2

```
while (   ) {
    int data = ll.head.data;
    ll.rF();
    if (true) {
        al.aL(data);
    }
}
```

T : o(n)

S : o(1)

ll :

head　　　　　　　　　　tail

al : 2 → 3 → 5 → 6 → 7

21k　　　　　　　　　　　74k

n nodes

```
while(      ) {   2
    int data = ll.head.data;
    ll.rF();

    if ( true ) {
        al.aL(data);
    }
}
```

T : O(n)

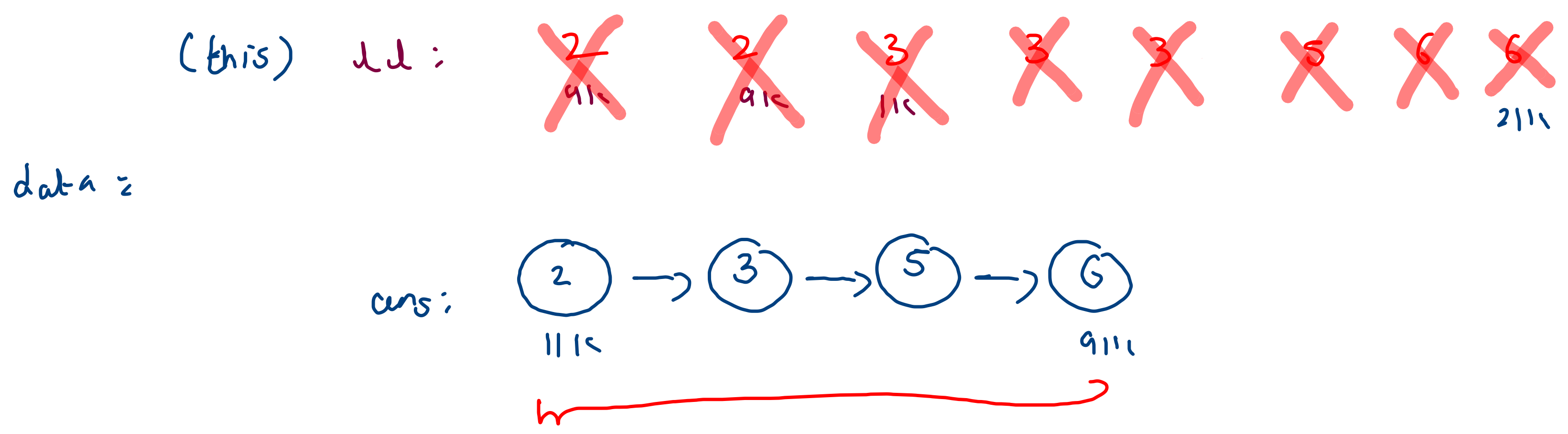
S : O(1)

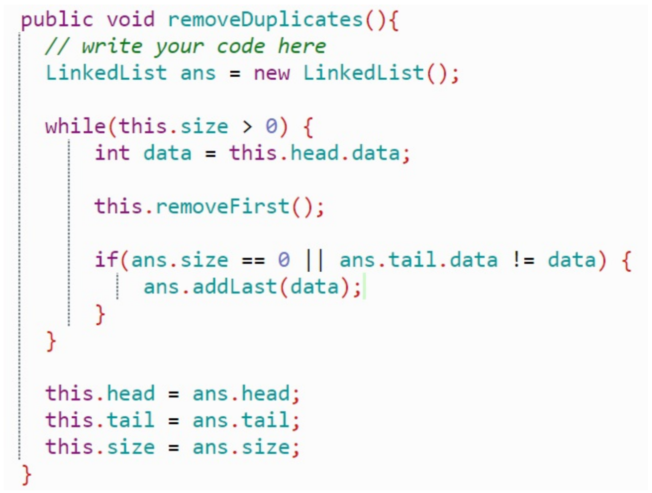
h = 21k
t = 74k
s = 5

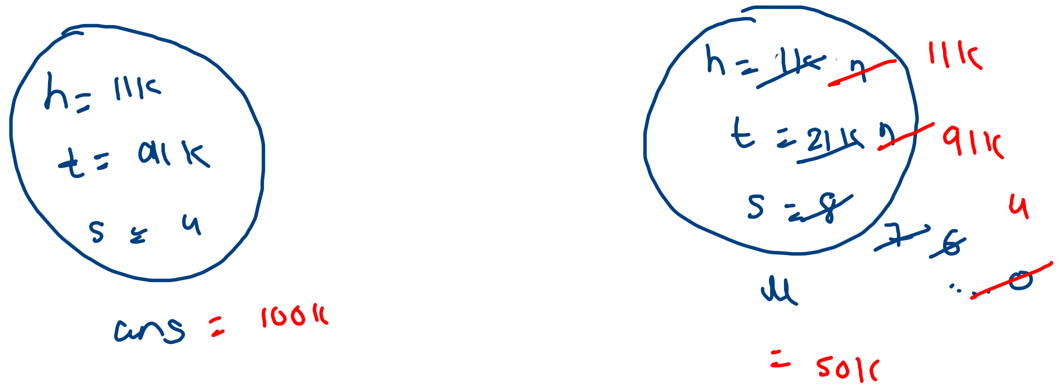
al

h = 21k
t = 74k
s = 5

ll




```
public void removeDuplicates(){
  // write your code here
  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;
}
```



this ~~17~~ ~~18~~ ~~19~~ ~~31~~ ~~30~~ ~~60~~ ~~33~~

$$\begin{tikzcd}
\text{od} & \overset{oh}{\underset{4k}{17}} \arrow[r] & 19 \arrow[r] & 31 \arrow[r] & \overset{ot}{\underset{2k}{33}} \arrow[dlll, bend left] \\
\text{ed} & \overset{eh}{\underset{5k}{18}} \arrow[r] & 30 \arrow[r] & \overset{et}{\underset{7k}{60}} &
\end{tikzcd}$$

ot.next = eh

S : O(1)

T : O(n)

h = 4k
t = 2k
s = 4

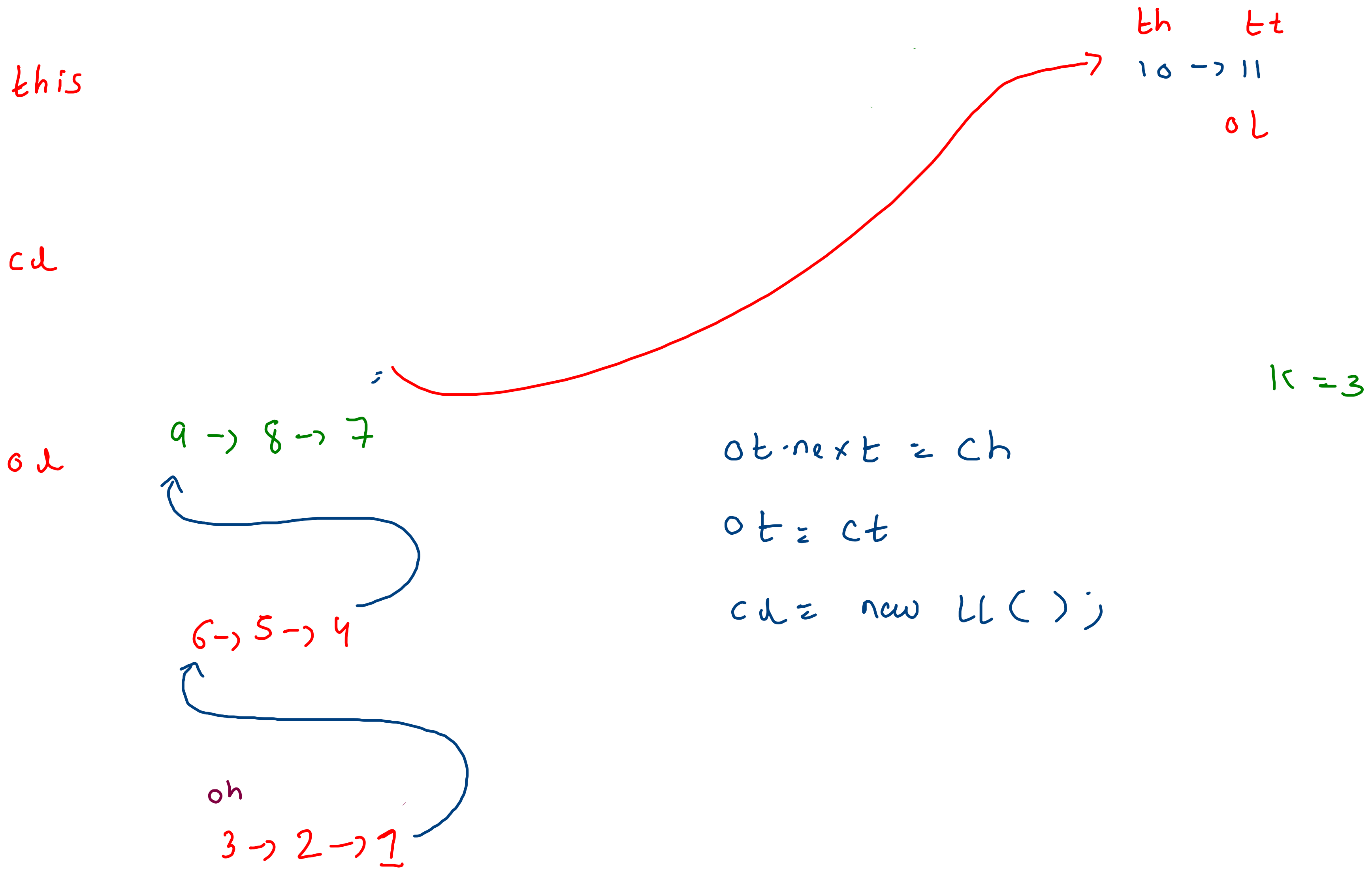
od

h = 5k
t : 7k
s = 3

ed

h = ~~n~~ 4k
t = ~~n~~ 7k
S = ~~s~~ 7

this



```
ot.next = ch
ot = ct
cd = new LL();
```

th tt

this $\cancel{1} \to \cancel{2} \to \cancel{3} \to \cancel{4} \to \cancel{5} \to \cancel{6} \to \cancel{7} \to \cancel{8} \to \cancel{9} \to \cancel{10} \to \cancel{11}$

oh

od

$$\begin{tikzcd} 4 \arrow[r] & 3 \arrow[r] & 2 \arrow[r] & 1 \arrow[dlll, bend left] \\ 8 \arrow[r] & 7 \arrow[r] & 6 \arrow[r] & 5 \arrow[d] & \text{ot} \\ & & & 9 \arrow[r] & 10 \arrow[r] & 11 \end{tikzcd}$$

ot.next = ch

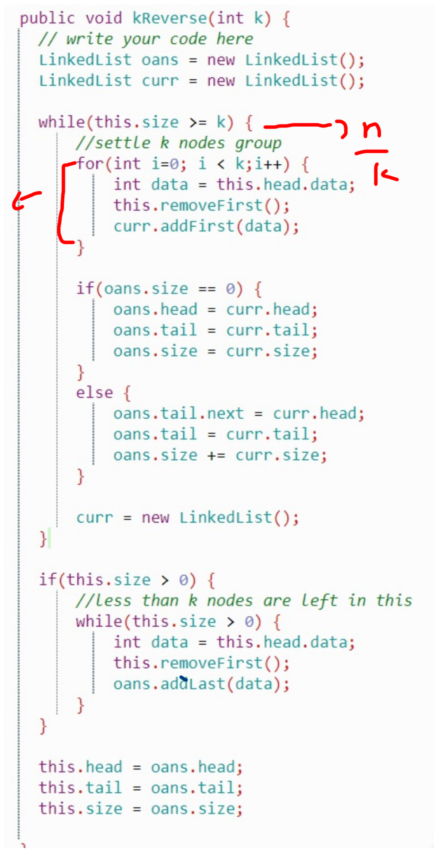
ot = ct

cd = new LL()

K = 4

cd

this $\cancel{1} \to \cancel{2} \to \cancel{3} \to \cancel{4} \to \cancel{5} \to \cancel{6} \to \cancel{7} \to \cancel{8} \to \cancel{9} \to \cancel{10} \to \cancel{11}$



```java
public void kReverse(int k) {
  // write your code here
  LinkedList oans = new LinkedList();
  LinkedList curr = new LinkedList();

  while(this.size >= k) {
    //settle k nodes group
    for(int i=0; i < k;i++) {
      int data = this.head.data;
      this.removeFirst();
      curr.addFirst(data);
    }

    if(oans.size == 0) {
      oans.head = curr.head;
      oans.tail = curr.tail;
      oans.size = curr.size;
    }
    else {
      oans.tail.next = curr.head;
      oans.tail = curr.tail;
      oans.size += curr.size;
    }

    curr = new LinkedList();
  }

  if(this.size > 0) {
    //less than k nodes are left in this
    while(this.size > 0) {
      int data = this.head.data;
      this.removeFirst();
      oans.addLast(data);
    }
  }

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

$\to \frac{n}{k}$

$k \leftarrow$

oh

oans
$$\begin{tikzcd}
4 \arrow[r] & 3 \arrow[r] & 2 \arrow[r] & 1 \arrow[dlll, bend right] \\
8 \arrow[r] & 7 \arrow[r] & 6 \arrow[r] & 5 \arrow[d] \\
 & & & 9 \arrow[r] & 10 \arrow[r] & 11
\end{tikzcd}$$

ot

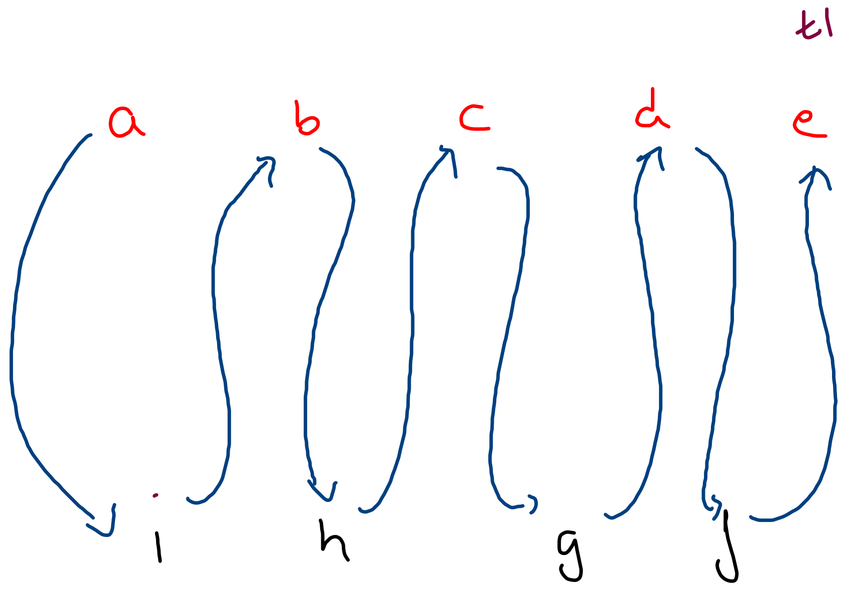
curr

$k = 4$

$T \to \frac{n}{k} * k$

$= n$

ans·tail = t1;



backup

t1n = t1.next

t2n = t2.next

links

t1.next = t2

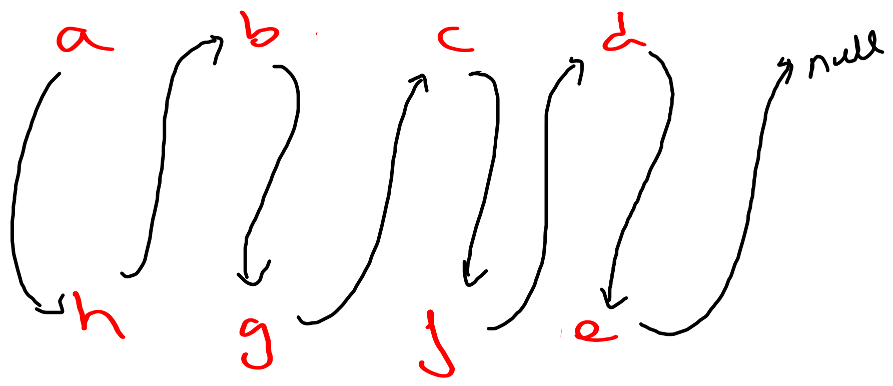
t2.next = t1n

move :

t1 = t1n

t2 = t2n

tail to be managed



backup

t1n = t1.next

t2n = t2.next

links

t1.next = t2

t2.next = t1n

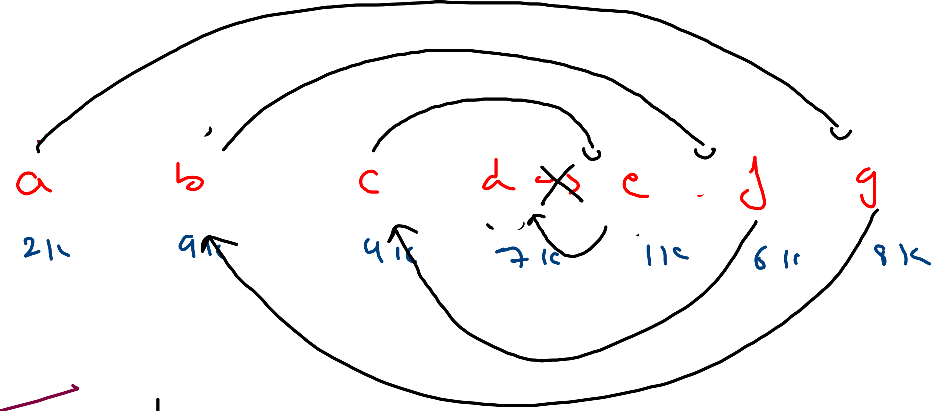
move :

t1 = t1n

t2 = t2n

Recursion

reverse X



~~hi = null~~

~~hi = 8k , lev = 7~~

~~hi = 6k , lev = 6~~

~~hi = 1k , lev = 5~~ hi.next = null

hi = 7k , lev = 4 → tail = hi

hi = 4k , lev = 3

hi = 9k , lev = 2

hi = 2k , lev = 1

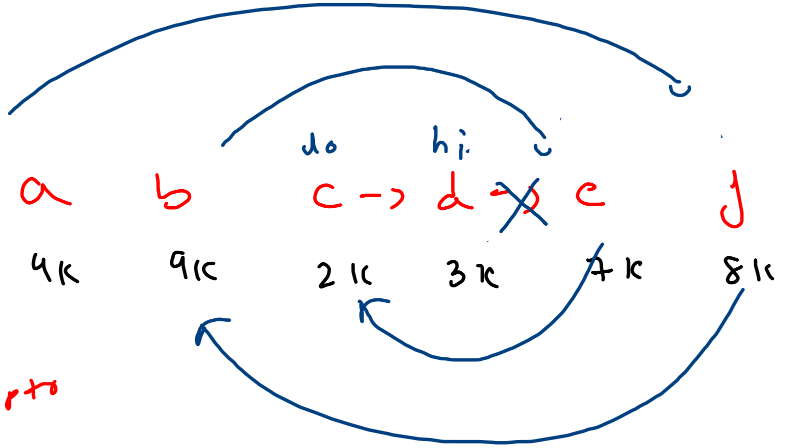
lon = lo.next

lo.next = hi

hi.next = lon

lo = 7k

$$\text{tail} \rightarrow \left(\frac{\text{size}}{2} + 1 \right) = 4$$



~~hi = null~~	~~d: 7~~
~~hi = 8k~~	~~d: 6~~
~~hi = 7k~~	~~d: 5~~
hi = 3k	d = 4
hi = 2k	d = 3
hi = 9k	d = 2
hi = 4k	d: 1

} swap ptr

$\longrightarrow$ hi.next = null

tail = hi

don = do.next

do.next = hi

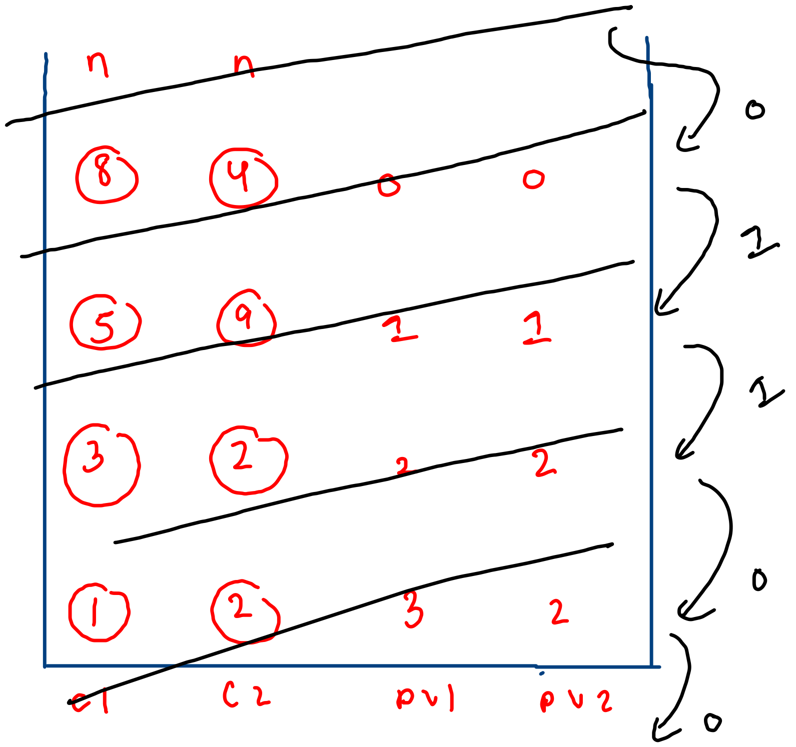
hi.next = don

do = 2k

tail -> $\left(\frac{size}{2} + 1\right)$

$l1$ $\qquad 1_3 \to 3_2 \to 5_1 \to 8_0$

$l2$ $\qquad 2_2 \to 9_1 \to 4_0$



$pv1 = l1.size() - 1$

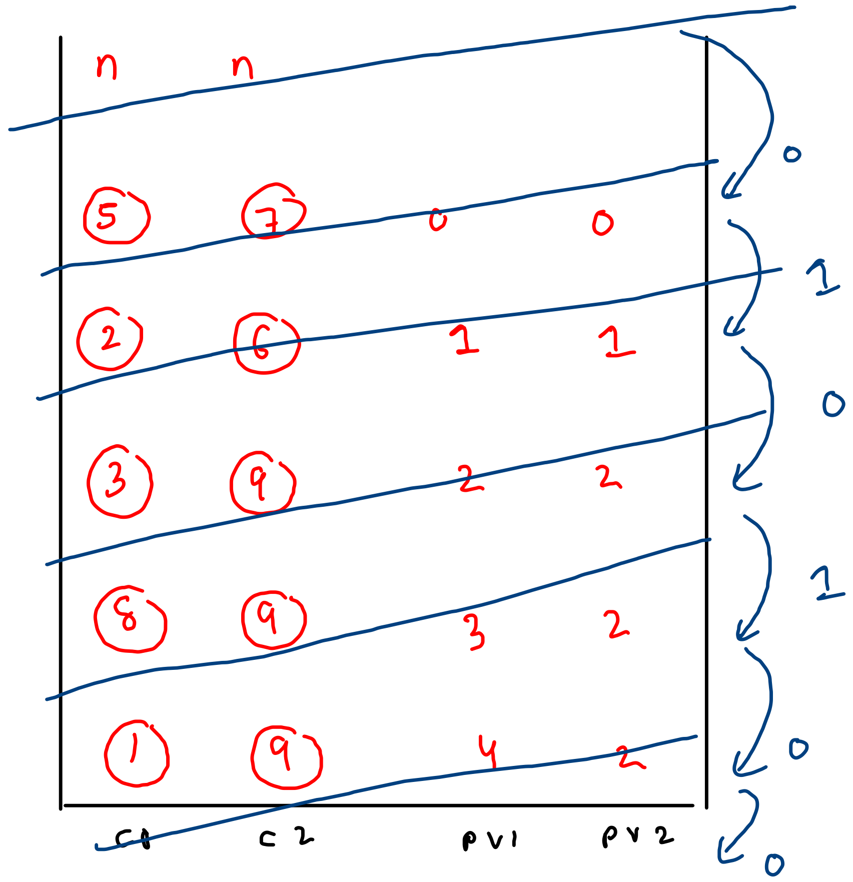
$pv2 = l2.size() - 1$

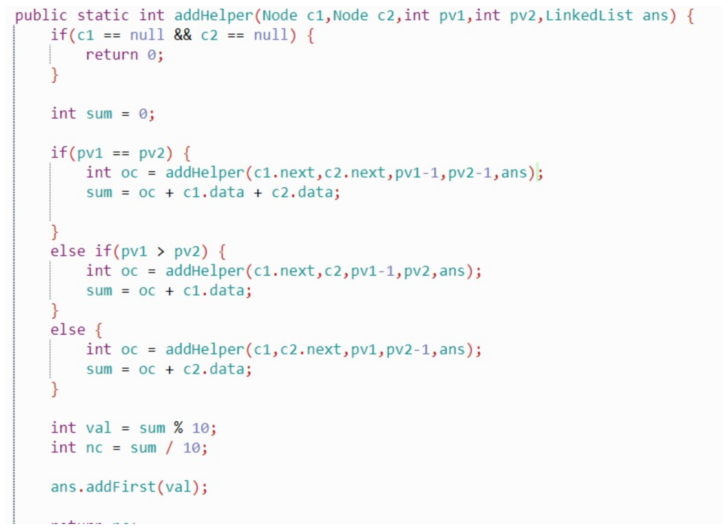
$$1 \to 6 \to 5 \to 2$$

$1_4 \rightarrow 8_3 \rightarrow 3_2 \rightarrow 2_1 \rightarrow 5_0$

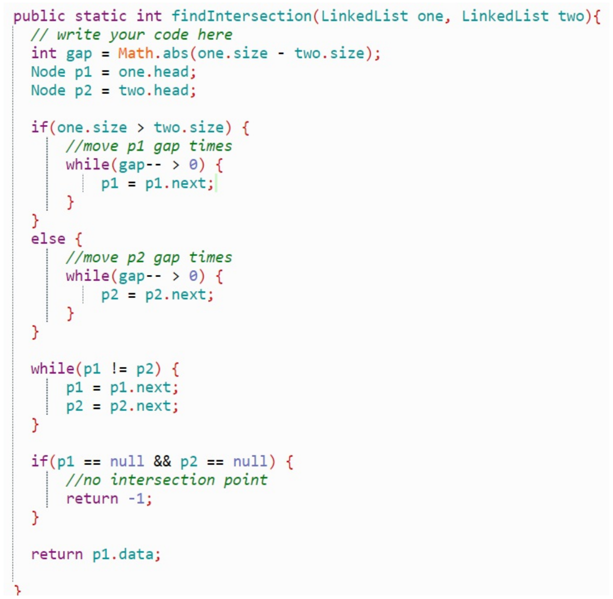
$9_2 \rightarrow 6_1 \rightarrow 7_0$

$1 \rightarrow 9 \rightarrow 2 \rightarrow 9 \rightarrow 2$





```
public static int addHelper(Node c1,Node c2,int pv1,int pv2,LinkedList ans) {
    if(c1 == null && c2 == null) {
        return 0;
    }

    int sum = 0;

    if(pv1 == pv2) {
        int oc = addHelper(c1.next,c2.next,pv1-1,pv2-1,ans);
        sum = oc + c1.data + c2.data;

    }
    else if(pv1 > pv2) {
        int oc = addHelper(c1.next,c2,pv1-1,pv2,ans);
        sum = oc + c1.data;
    }
    else {
        int oc = addHelper(c1,c2.next,pv1,pv2-1,ans);
        sum = oc + c2.data;
    }

    int val = sum % 10;
    int nc = sum / 10;

    ans.addFirst(val);
```



```java
public static int findIntersection(LinkedList one, LinkedList two){
  // write your code here
  int gap = Math.abs(one.size - two.size);
  Node p1 = one.head;
  Node p2 = two.head;

  if(one.size > two.size) {
    //move p1 gap times
    while(gap-- > 0) {
      p1 = p1.next;
    }
  }
  else {
    //move p2 gap times
    while(gap-- > 0) {
      p2 = p2.next;
    }
  }

  while(p1 != p2) {
    p1 = p1.next;
    p2 = p2.next;
  }

  if(p1 == null && p2 == null) {
    //no intersection point
    return -1;
  }

  return p1.data;

}
```

$$\begin{tikzcd}
 & & & & & & p2 & & & & \\
 & & & & & & p1 & & & & \\
a \arrow[r] & b \arrow[r] & c \arrow[r] & d \arrow[r] & e \arrow[r] & f \arrow[r] & g \arrow[r] & h \arrow[r] & i \arrow[r] & j \arrow[r] & k \\
 & & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[ur] & & & & &
\end{tikzcd}$$

gap = 11 - 9 = 2