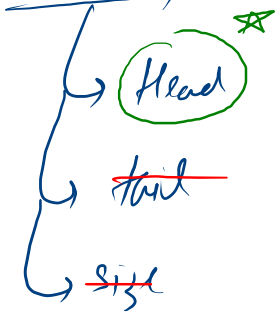
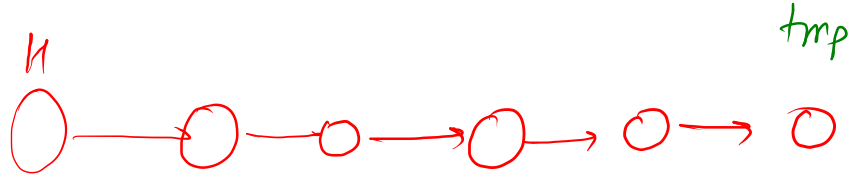
LL → Create

LL → Concepts

Prepcodes → H
T
S

① Summary





tmp.next != null

- Reverse A Linked List (data Iterative)
- Reverse Linked List (pointer Iterative)
- Linked List To Stack Adapter
- Linked List To Queue Adapter
- Kth Node From End Of Linked List
- Mid Of Linked List
- Merge Two Sorted Linked Lists
- Merge Sort A Linked List
- Remove Duplicates In A Sorted Linked List
- Odd Even Linked List
- K Reverse In Linked List
- Display Reverse (recursive) - Linked List
- Reverse Linked List (pointer - Recursive)
- Is Linked List A Palindrome?
- Fold A Linked List
- Add Two Linked Lists
- Intersection Point Of Linked Lists

Kth Node From End of linked list

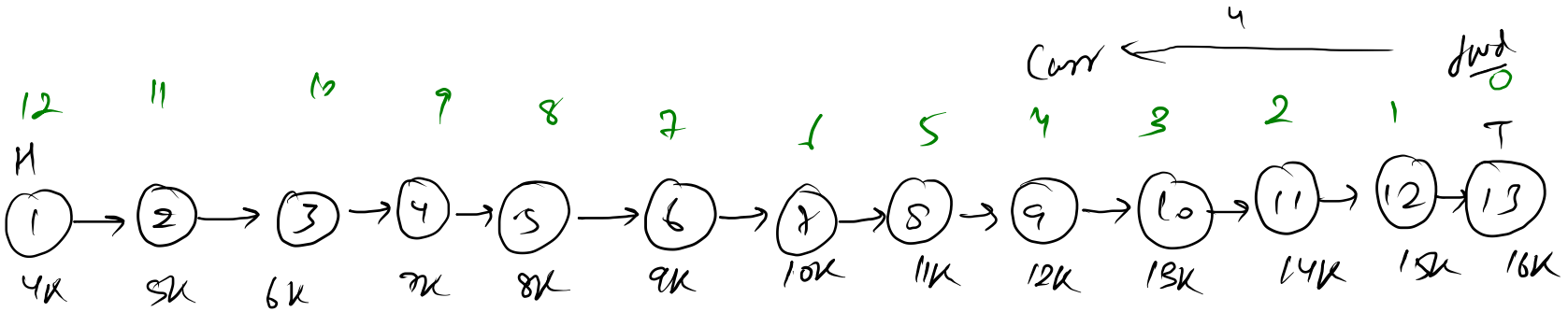
5 min

K = 4 2 1 0

head = 4K

tail = 16K

Size = 13



Doubt

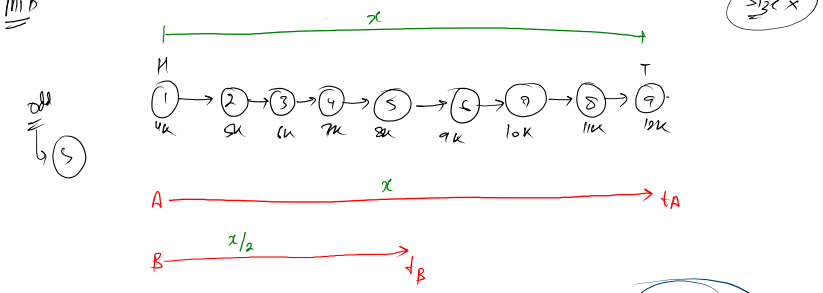
fwd.next != null

Curr.data → ans

```
public int kthFromLast(int k) {
  Node curr, fwd;
  curr = fwd = head;

  while (k != 0) {
    fwd = fwd.next;
    k--;
  }

  while (fwd.next != null) {
    fwd = fwd.next;
    curr = curr.next;
  }

  return curr.data;
}
```

Mid

Size x

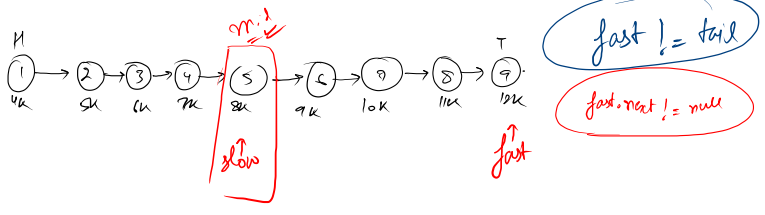


$S_A = \frac{x}{t_A}$ $S_B = \frac{x}{2t_B}$ $t_A == t_B$

$t_A = \frac{x}{S_A}$ $t_B = \frac{x}{2S_B}$

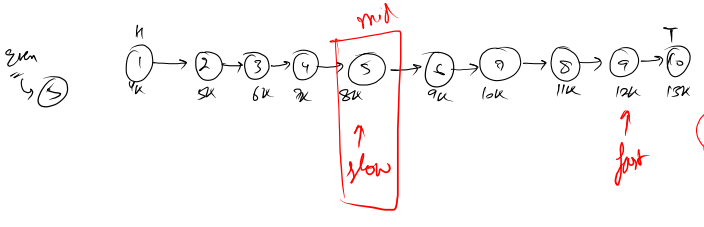
$\frac{x}{S_A} = \frac{x}{2S_B} \Rightarrow S_A = 2 \cdot S_B$

f s



fast != tail

fast.next != null

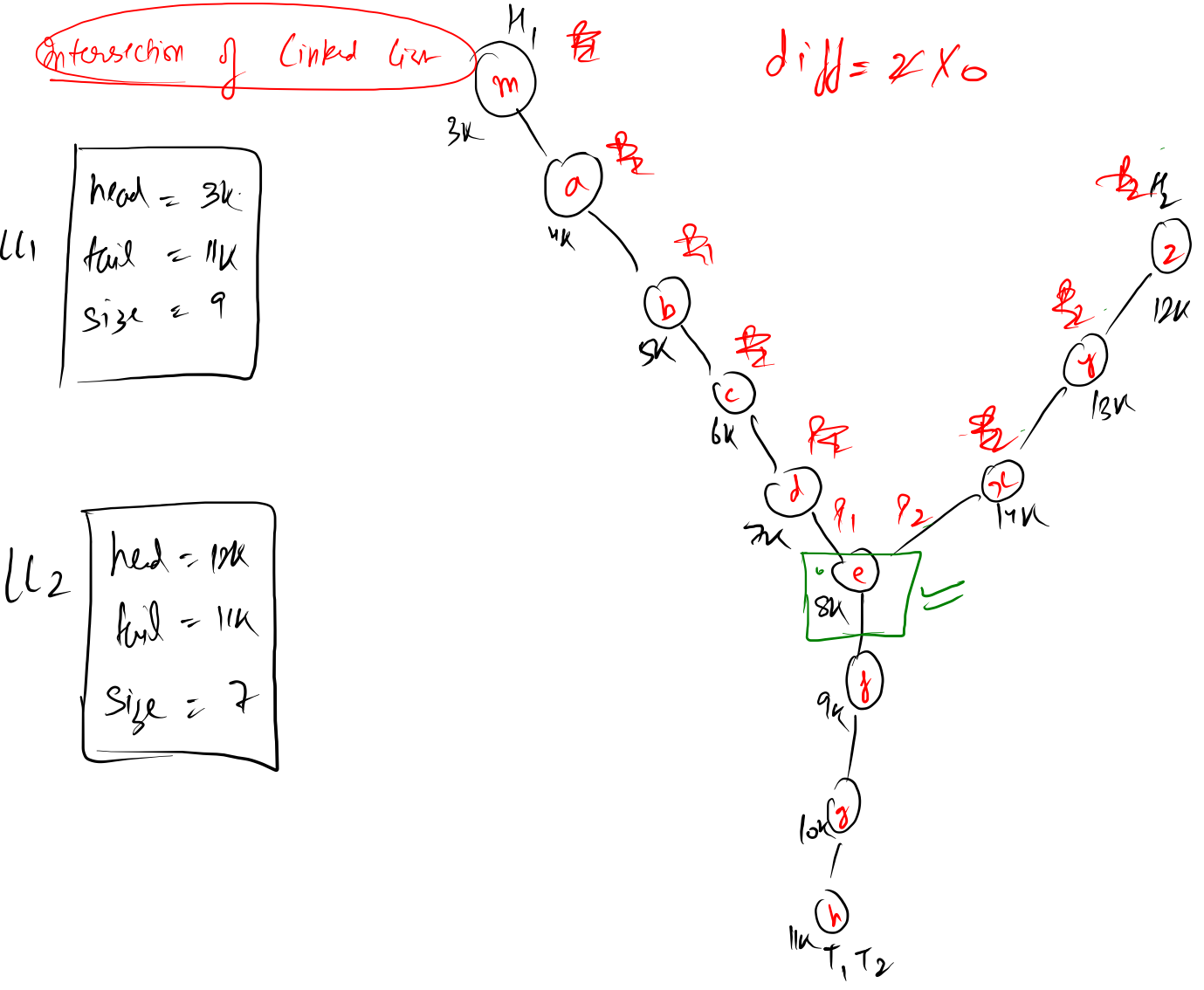


fast.next != tail

fast.next.next != null

```
public int mid(){
    Node fast , slow;
    fast = slow = this.head;

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

    return slow.data;
}
```

Intersection of Linked List

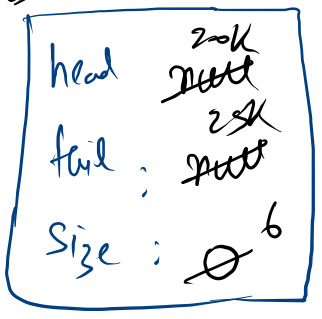



```java
public static int findIntersection(LinkedList one, LinkedList two) {
  Node p1 = one.head, p2 = two.head;

  if (one.size > two.size) {
    int diff = one.size - two.size;
    while (diff > 0) {
      p1 = p1.next;
      diff--;
    }
  } else if (one.size < two.size) {
    int diff = two.size - one.size;
    while (diff > 0) {
      p2 = p2.next;
      diff--;
    }
  }

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

  return p1.data;
}
```

1. Sorted
2. Duplicate values

remove all duplicates

this

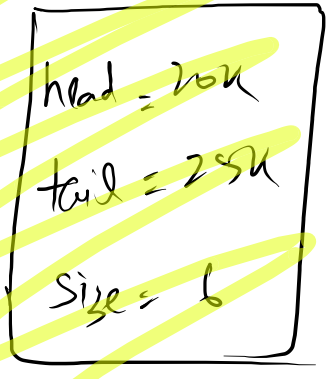


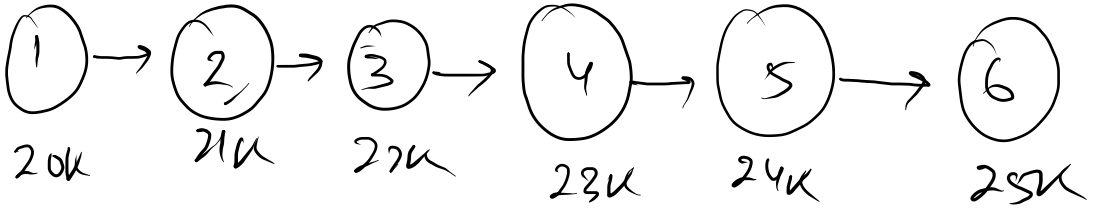

```java
public void removeDuplicates(){
    // this -> original input
    LinkedList ll = new LinkedList();
    ll.addLast(this.getFirst());

    while(this.size != 0){
        if(this.getFirst() == ll.getLast())
            this.removeFirst();
        }else{
            ll.addLast(this.getFirst());
        }
    }

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

T.C. => O(n)

S.C. => O(1)

ll



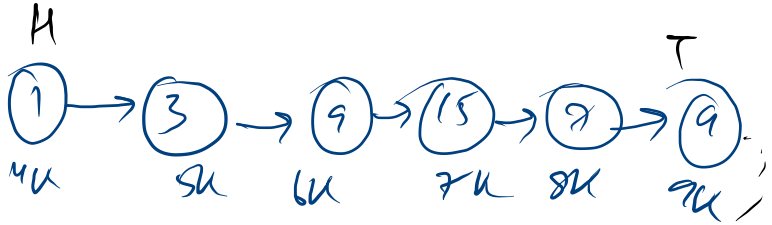


this

head = ~~null~~ 4K
tail = ~~null~~ 15K
size = ~~0~~ 12

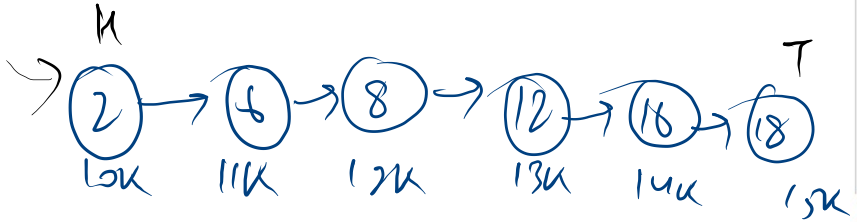
odd

head = 4K
tail = 9K
size = 6

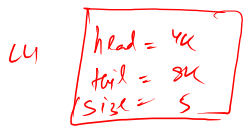


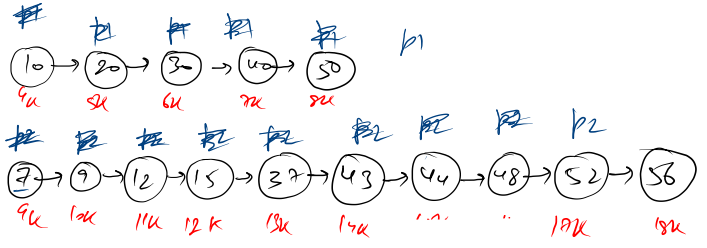
Even

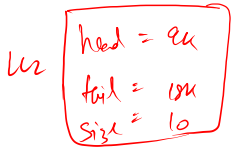
head = 10K
tail = 18K
size = 6

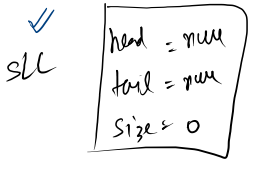


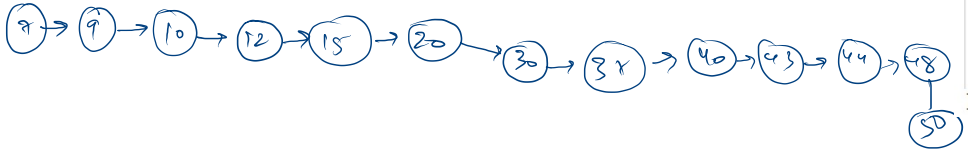

```java
public void oddEven(){
    LinkedList odd = new LinkedList() , even = new LinkedList();

    while(this.size != 0){
        int vl = this.getFirst();
        this.removeFirst();

        if(vl % 2 == 0){
            // even
            even.addLast(vl);
        }else{
            // odd
            odd.addLast(vl);
        }
    }
    if(odd.size != 0 && even.size != 0){
        // odd & even elements exists
        odd.tail.next = even.head;
        this.head = odd.head;
        this.tail = even.tail;
        this.size = odd.size + even.size;
    }else if(even.size != 0){
        // only even elements exists
        this.head = even.head;
        this.tail = even.tail;
        this.size = even.size;
    }else if(odd.size != 0){
        // only odd elements exists
        this.head = odd.head;
        this.tail = odd.tail;
        this.size = odd.size;
    }
}
```





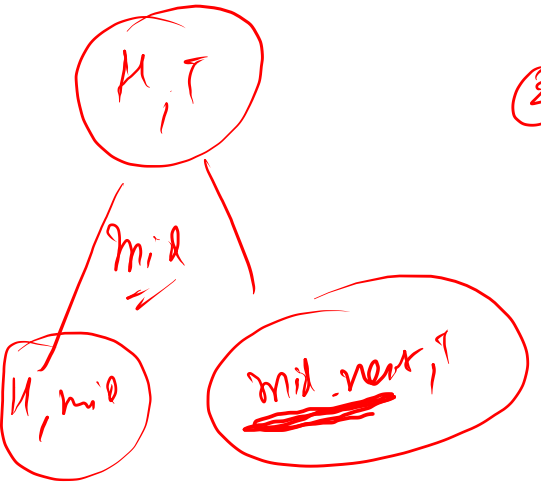


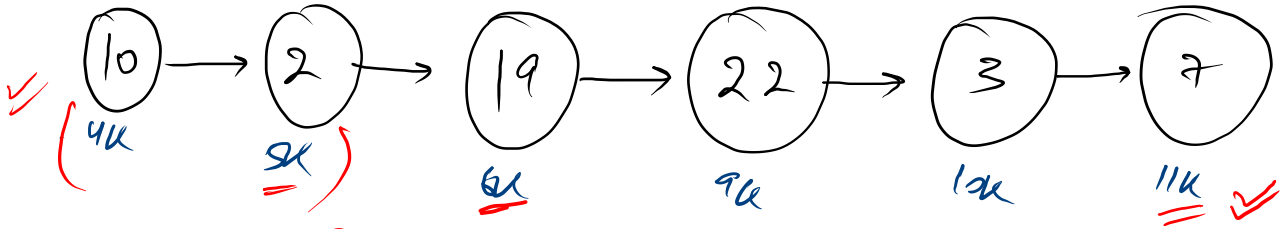


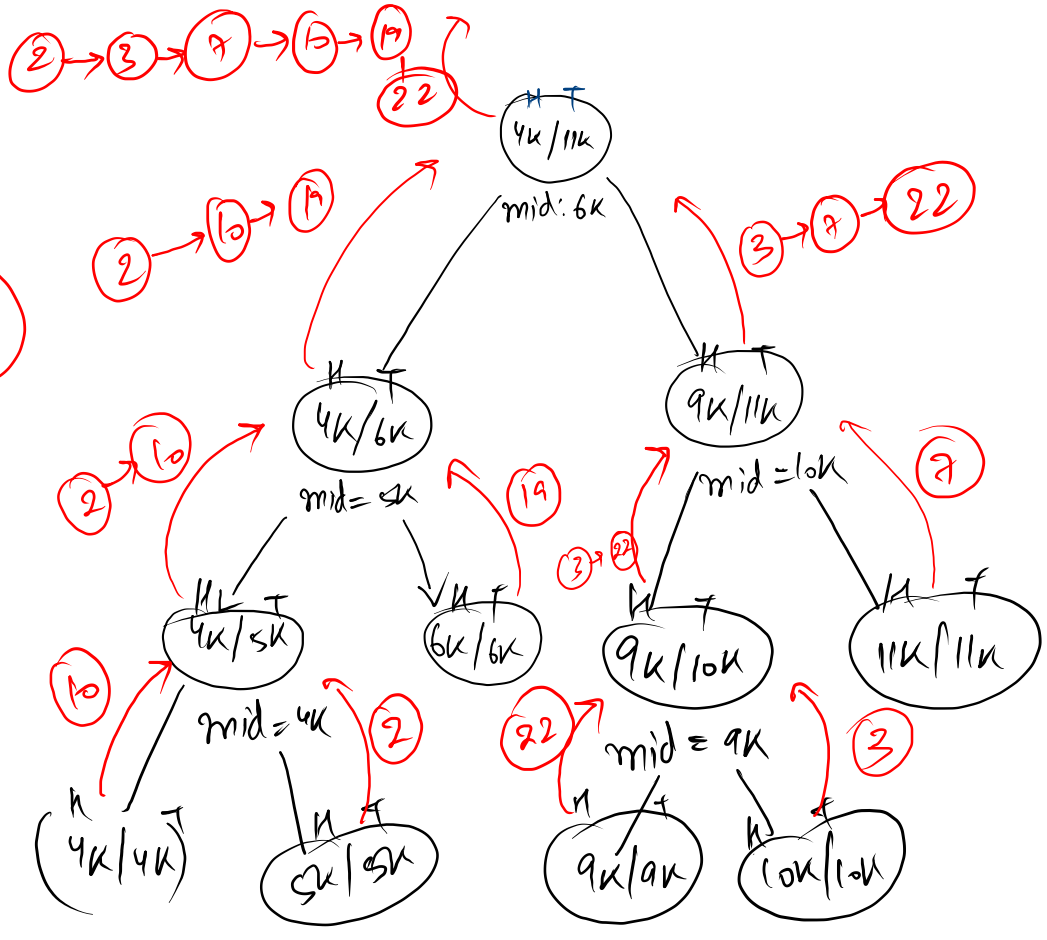


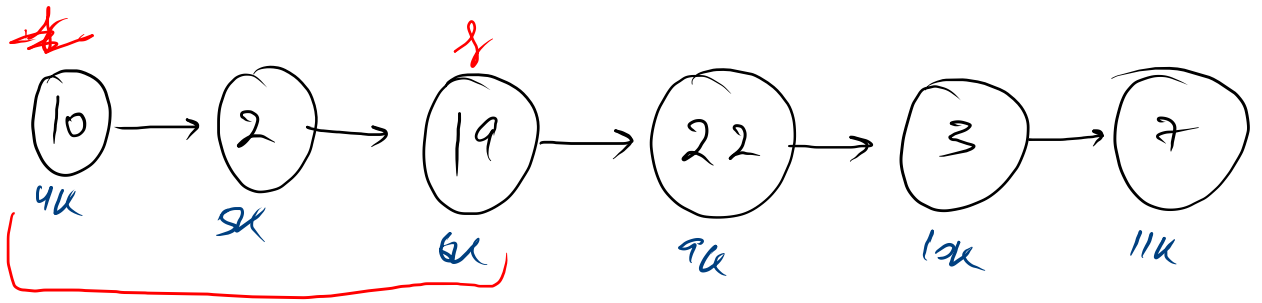
```java
public static LinkedList mergeTwoSortedLists(LinkedList l1, LinkedList l2) {
    LinkedList sll = new LinkedList();

    Node p1 = l1.head , p2 = l2.head;

    while(p1 != null && p2 != null){
        if(p1.data <= p2.data){
            sll.addLast(p1.data);
            p1 = p1.next;
        }else{
            sll.addLast(p2.data);
            p2 = p2.next;
        }
    }

    while(p1 != null){
        sll.addLast(p1.data);
        p1 = p1.next;
    }

    while(p2 != null){
        sll.addLast(p2.data);
        p2 = p2.next;
    }

    return sll;
}
```

6


H, T
mid
H, mid
mid_next, T


10
2
19
22
3
7
4K
8K
6K
9K
10K
11K


4K/11K
mid: 6K
4K/6K
mid= 5K
9K/11K
mid =10K
4K/5K
6K/6K
9K/10K
11K/11K
mid= 4K
mid = 9K
4K/4K
5K/5K
9K/9K
10K/10K




```java
public static Node midNode(Node head,Node tail) {
  Node f = head;
  Node s = head;

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

  return s;
}

public static LinkedList mergeSort(Node head, Node tail){
  if(head == tail){
      LinkedList b = new LinkedList();
      b.addLast(head.data);
      return b;
  }

  Node mid = midNode(head,tail);
  LinkedList left = mergeSort(head,mid);
  LinkedList right = mergeSort(mid.next,tail);

  return mergeTwoSortedLists(left,right);
}
```