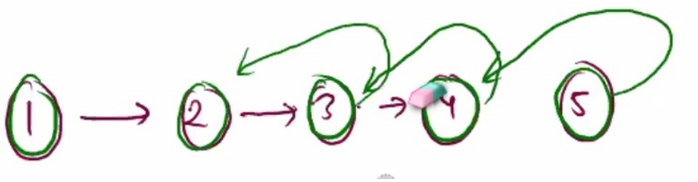
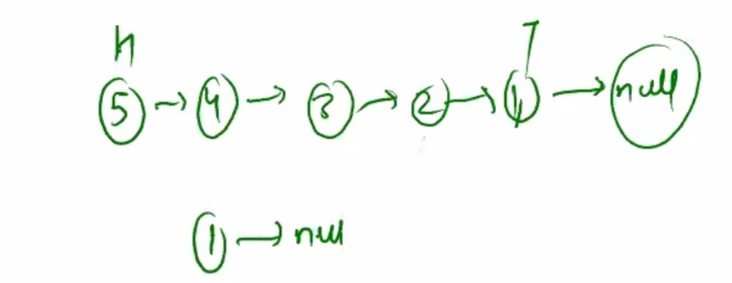
Lecture 39

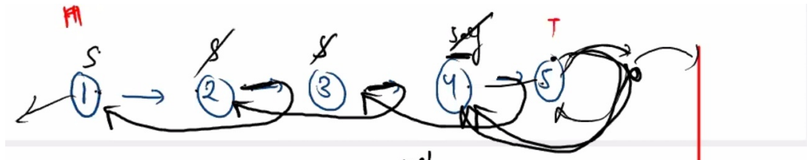
Display Reverse (recursive) - Linked List

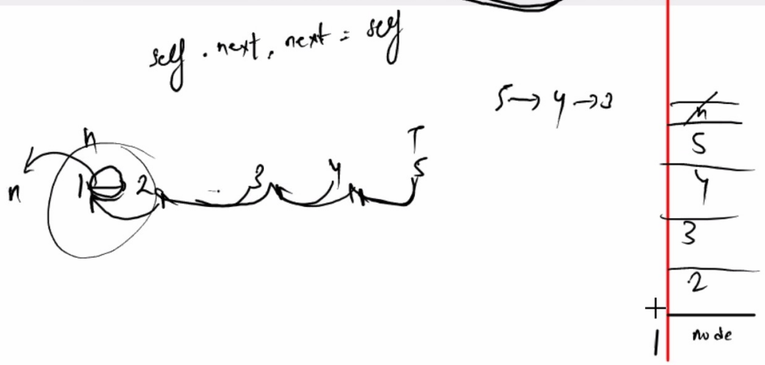
```java
private void displayReverseHelper(Node node) {
    if (node == null) {
        return; // base case
    }
    displayReverseHelper(node.next); // recursive call
    System.out.print(node.data + " "); // print the data of the node
}
```

Reverse Linked List (pointer - Recursive)



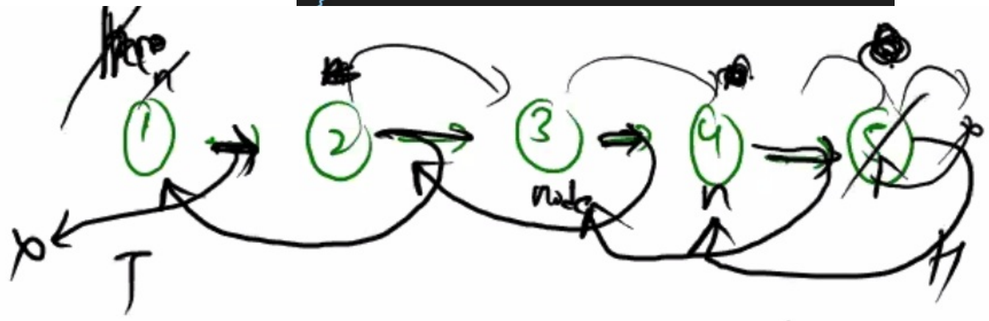







```
private void reversePRHelper(Node node){
    if(node == null)
        return;
    reversePRHelper(node.next);
    if(node != tail){
        node.next.next = node;
    }
}

public void reversePR(){
    // write your code here
    reversePRHelper(head);
    head.next = null;
    Node tmp = head;
    head = tail;
    tail = tmp;
  }
}
```

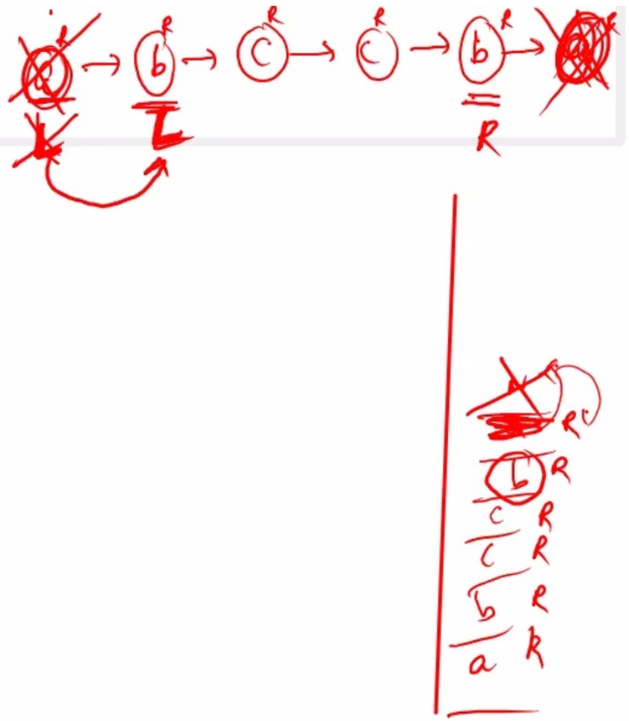
```
// write your code here
private void reversePRHelper(Node node) {
    if (node == null)
        return; // base case
    reversePRHelper(node.next); // recursive call
    if (node ≠ tail) { // if not the last node
        node.next.next = node; // reverse the link
    }
}

// write your code here
public void reversePR() {
    reversePRHelper(head);
    head.next = null; // remove the last link (cycle break)
    Node temp = head;
    head = tail;
    tail = temp;
}
```



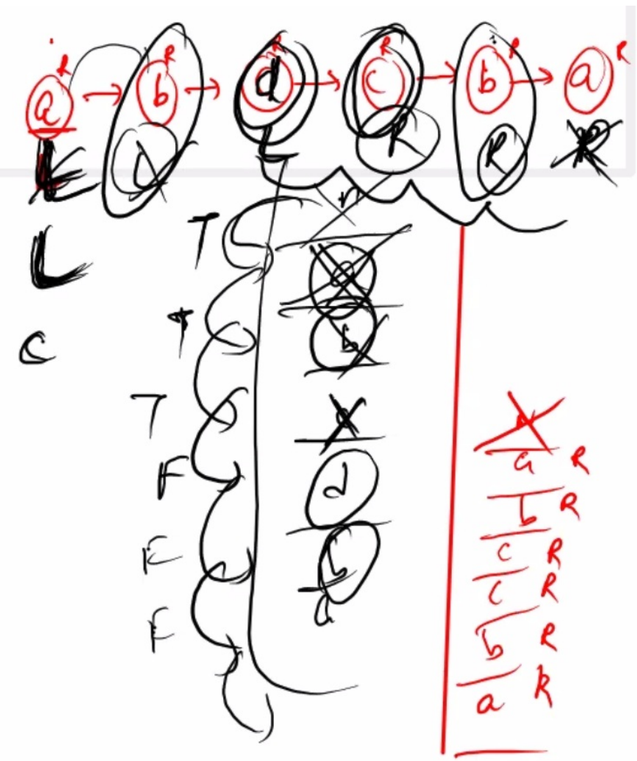
t = h
n = T
T = t

Is Linked List A Palindrome? (recursive)

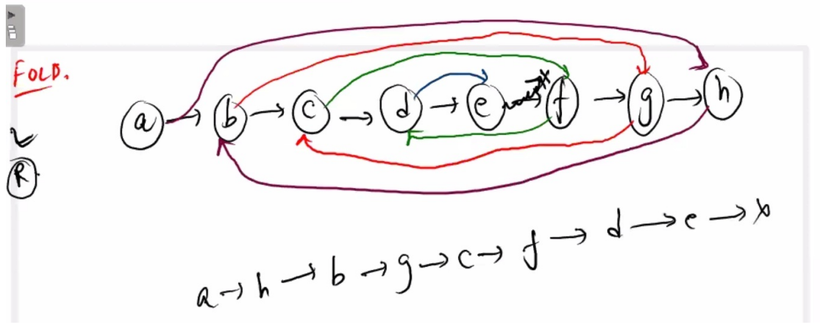
```
Node pleft;
private boolean palindromeHelper(Node right){
    if(right == null){
        return true;
    }

    boolean recAns = palindromeHelper(right.next);
    if(recAns == false){
        return false;
    }
    if(pleft.data != right.data){
        return false;
    }
    else{
        //same
        pleft = pleft.next;
        return true;
    }

}
public boolean IsPalindrome() {
  pleft = this.head;
  return palindromeHelper(head);
}
```

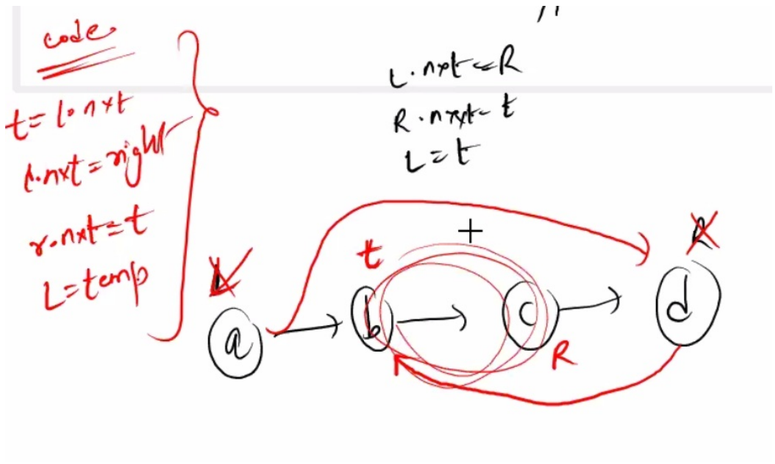


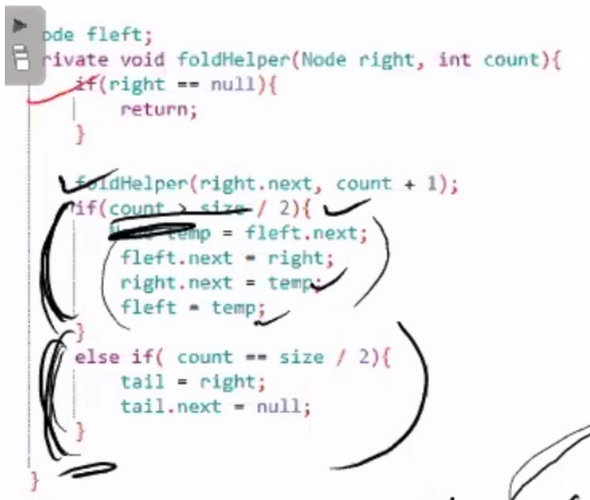

```
Node pleft;
private boolean palindromeHelper(Node right){
    if(right == null){
        return true;
    }

    boolean recAns = palindromeHelper(right.next);
    if(recAns == false){
        return false;
    }
    if(pleft.data != right.data){
        return false;
    }
    else{
        //same
        pleft = pleft.next;
        return true;
    }

}
public boolean IsPalindrome() {
  pleft = this.head;
  return palindromeHelper(head);
}
```



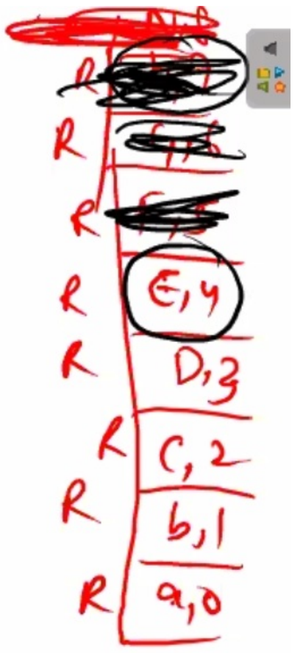
Fold A Linked List (recursive)



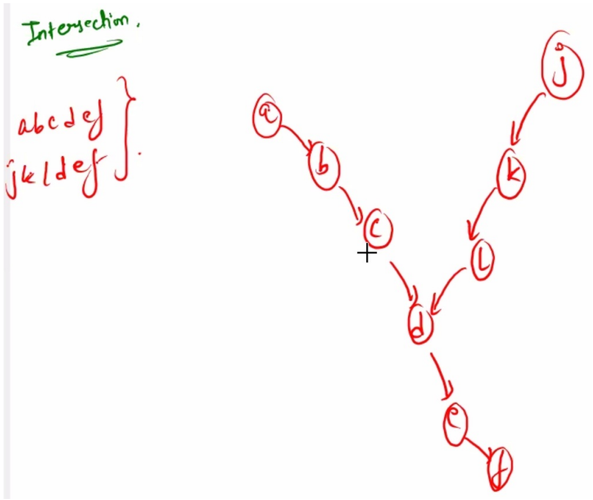




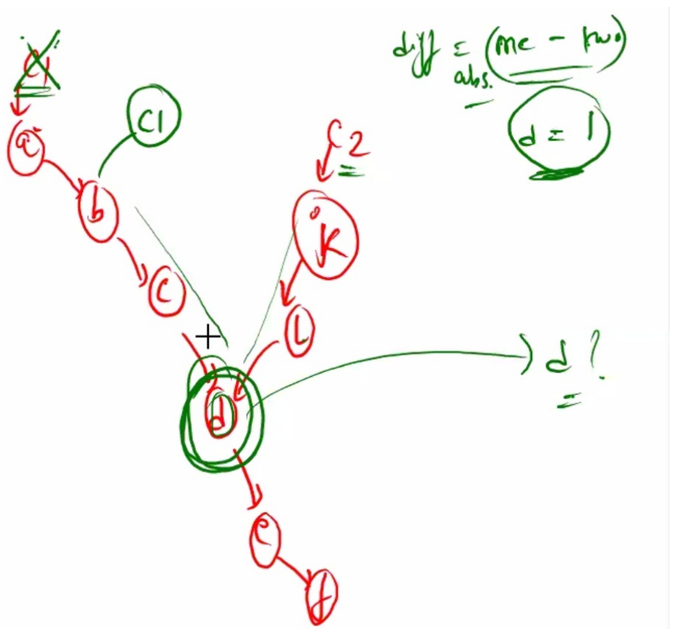
```
ode fleft;
rivate void foldHelper(Node right, int count){
    if(right == null){
        return;
    }

    foldHelper(right.next, count + 1);
    if(count > size / 2){
        temp = fleft.next;
        fleft.next = right;
        right.next = temp;
        fleft = temp;
    }
    else if( count == size / 2){
        tail = right;
        tail.next = null;
    }
}
```

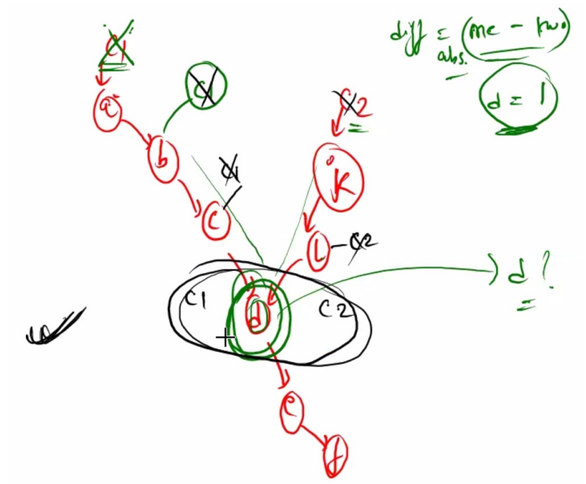
Intersection Point Of Linked Lists



start w/ same diffrence



same data then, return!



h/w : Add Linked List, revise recursion (backtracking)