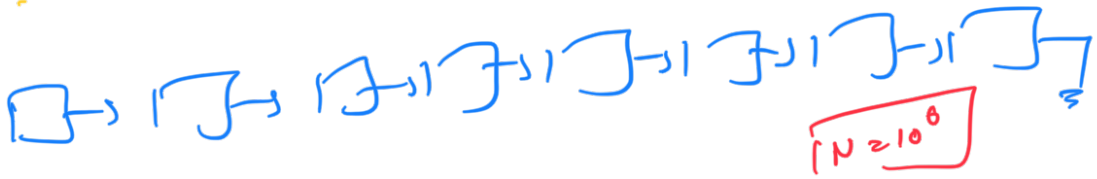
Linked List -3

Quiz:



From Start T.C: 8 ⇒ O(1) ↓ O(n)

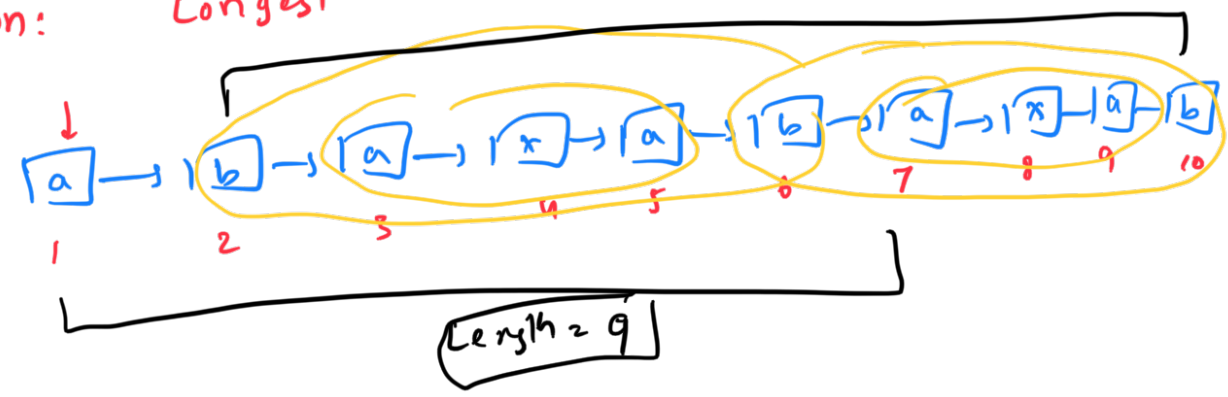
From End :

13%

curr.data = curr.next.data. → NULL → ERROR

curr.next = curr.next.next

Question: Longest Palindromic Sub linked list



String:

S = a b a x a b a x a b

left

Brute force:

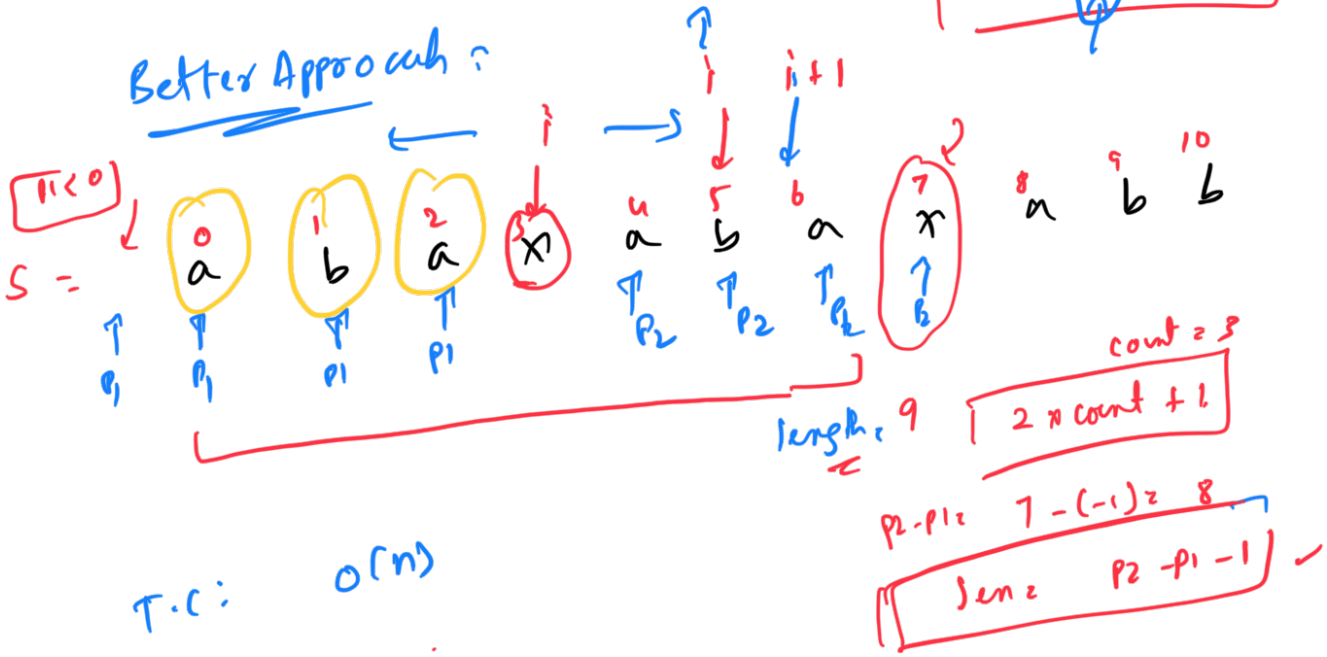
substrings

check all

T.C: $O(n^2) \times O(n)$ => $O(n^3)$

a b c x c b a

Better Approach:



T.C: $O(n)$

p1 = i-1, p2 = i+1

```
for (i = 0; i < n; i++) {
    p1 = i-1; p2 = i+1
    // odd length palindrom
    while (p1 >= 0 && p2 < n && s[p1] == s[p2])
        p1--;
        p2++;
        ans = max(ans, p2-p1-1);
}
```

O(n) [ ]

T.C: $O(n^2)$

// Even Length palindrom

p1 = i, p2 = i+1;

T.C: $O(n \times n) = O(n^2)$

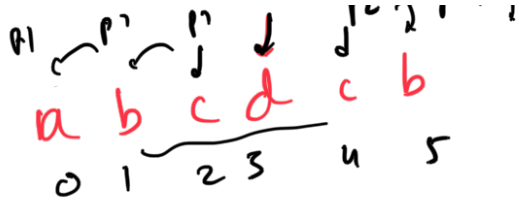
S.C: $O(1)$

p1 = 0
p2 = 2

p1 = 0
p2 = 6

i

DP:



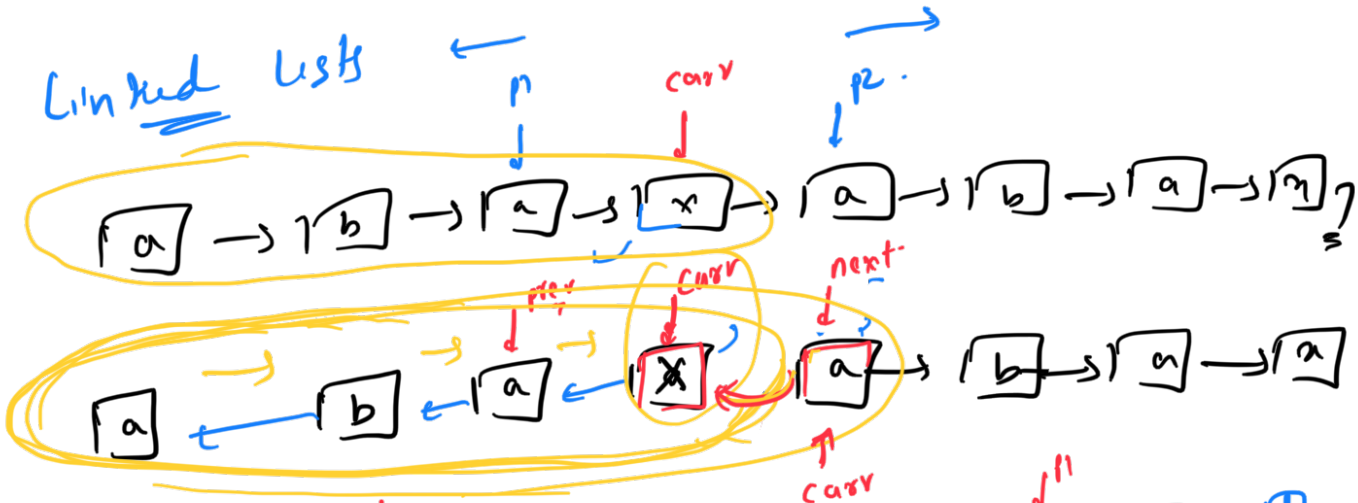
T.C : O(n)
S.C : O(n²)

p1 = -1, p2 = 1

p1 = 0, p2 = 1
p1 = -1, p2 = 2

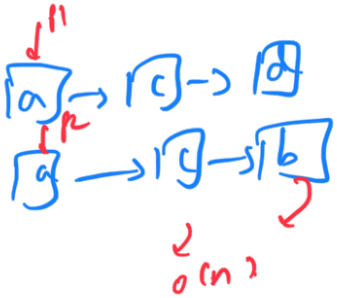
T.C : O(n²)
S.C : O(1)

Linked Lists



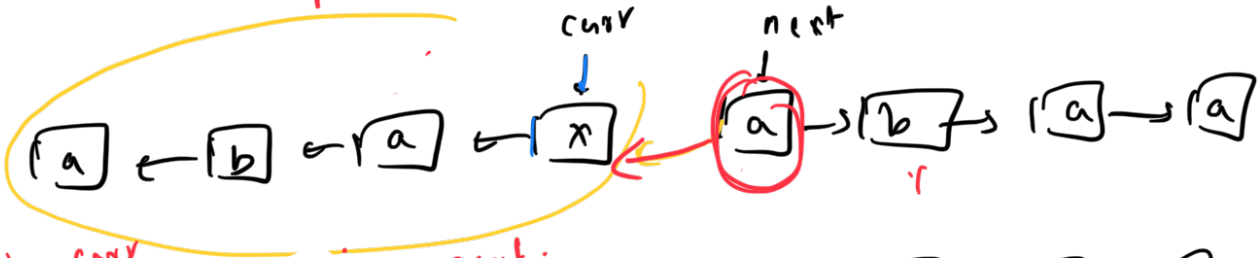
odd Length :

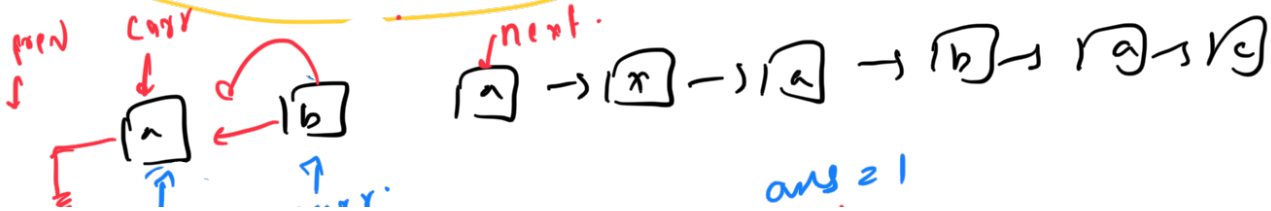
p1 = prev
p2 = next.



Even

p1 = curr
p2 = next





ans = 1

prev = NULL, curr = head.

// odd Length
p1 = prev (NULL) p2 = next

// Even Length
p1 = curr
p2 = next

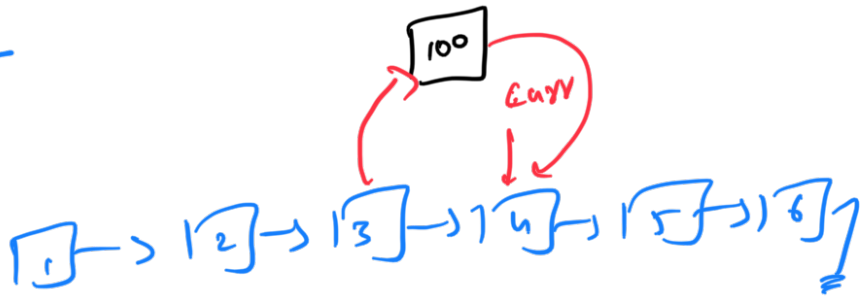
```
int maxPalindrome(Node head) {
    int ans = 0;
    curr = head, prev = NULL;
    while (curr != null) {
        next = curr.next;
        curr.next = prev;

        // odd length
        p1 = prev, p2 = next.
        c1 = No. of common Nodes.
        ans = max(ans, 2*(c1+1));

        // Even length
        p1 = curr, p2 = next
        c2 = No of common Nodes
        ans = max(ans, 2*c2)

        prev = curr;
        curr = next;
    }
```

O(n) O(n)

T.C: $O(n^2)$
S.C: $O(1)$

Doubly Linked Lists

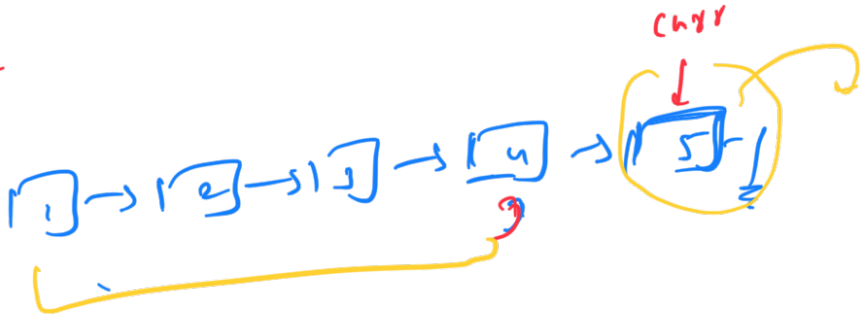
Scenario 1



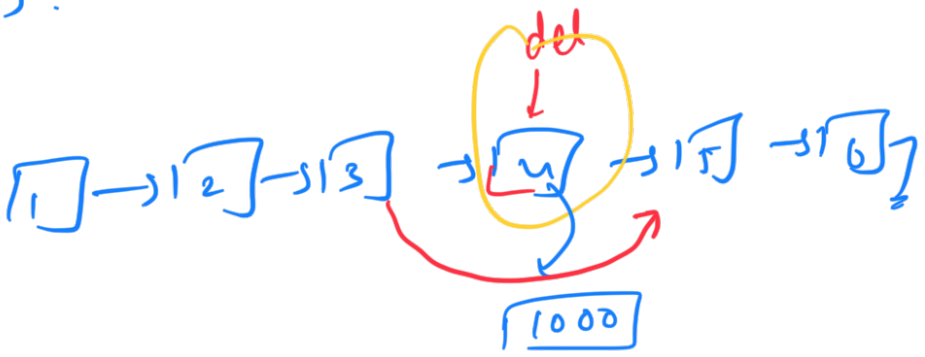
newNode.next = curr;

Prev Node ? ✗

Scenario 2:



scenario 3:



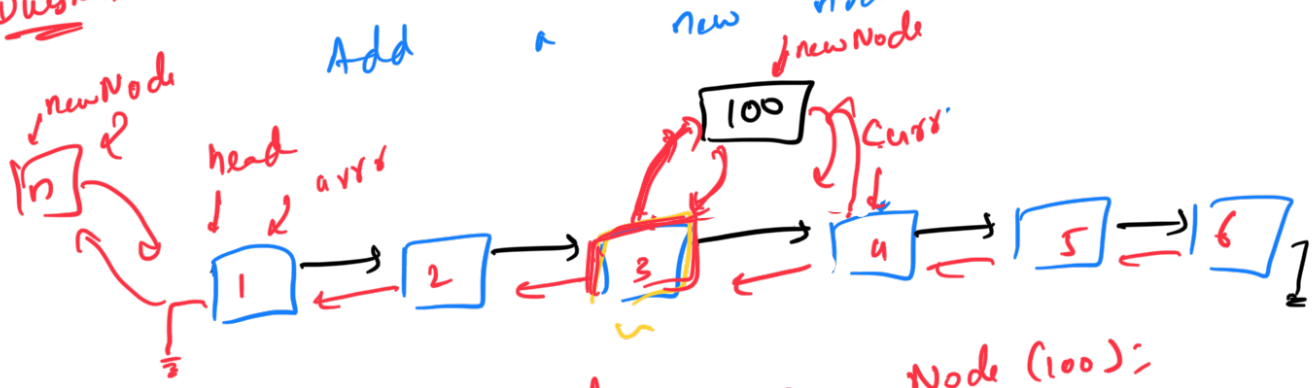
prev

Doubly Linked lists

```
class Node {
    int data;
    Node next;
    Node prev;
```

}

Question: Given a DLL and a curr pointer, Add a new node before this node



curr = head

```
newNode = new Node(100);
newNode.next = curr;
curr.prev.next = newNode;
newNode.prev = curr.prev;
curr.prev = newNode;

if(head == curr) {
    newNode.next = head;
    head.prev = newNode;
    head = newNode;
}
```

Question: Given a DLL and a point del, Delete that node from DLL

if(head == curr)

```
(curr.prev).next = curr.next;
(curr.next).prev = curr.prev;

if(curr == head){
    head = head.next;
    head.prev = NULL;
```

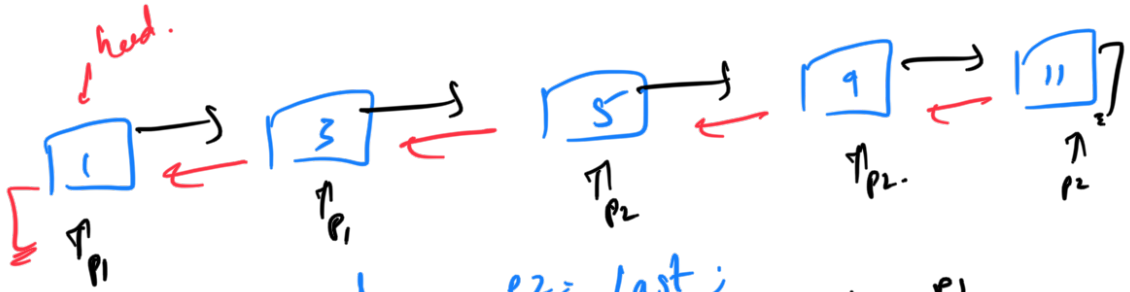
head.next

}

T.C: O(1)

Question: Given a sorted doubly LL, check if there is a pair of node whose sum = k

k = 8



p1 = head, p2 = last;

count = 0;

while (p1 != p2 && p2.next != p1) {

if(p1.data + p2.data == k) {

count ++;

p1 = p1.next;

p2 = p2.prev;

}

else if (p1.data + p2.data < k)

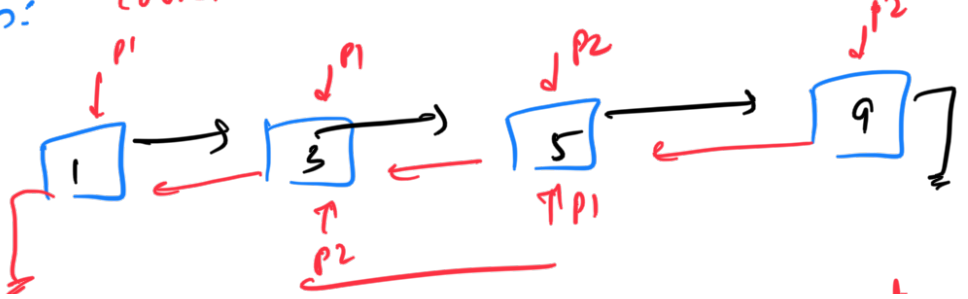
p1 = p1.next;

else

p2 = p2.prev;

}

Extensions: Count No of pairs k≥8



p2.next = p1

count = 1

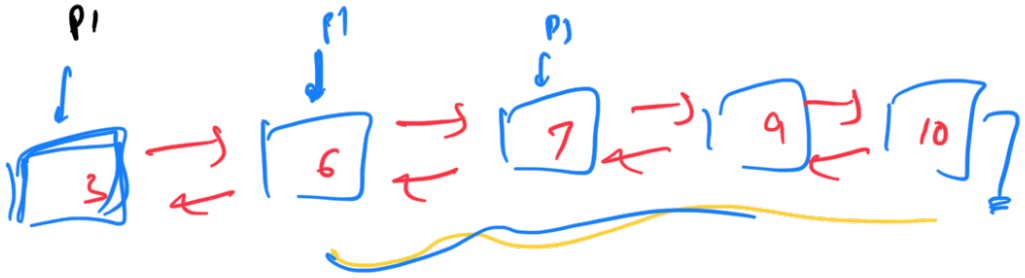
T.C : O(n)

S.C : O(1)

Question : Three nodes a, b, c

a.data + b.data . c.data = k

k = 15



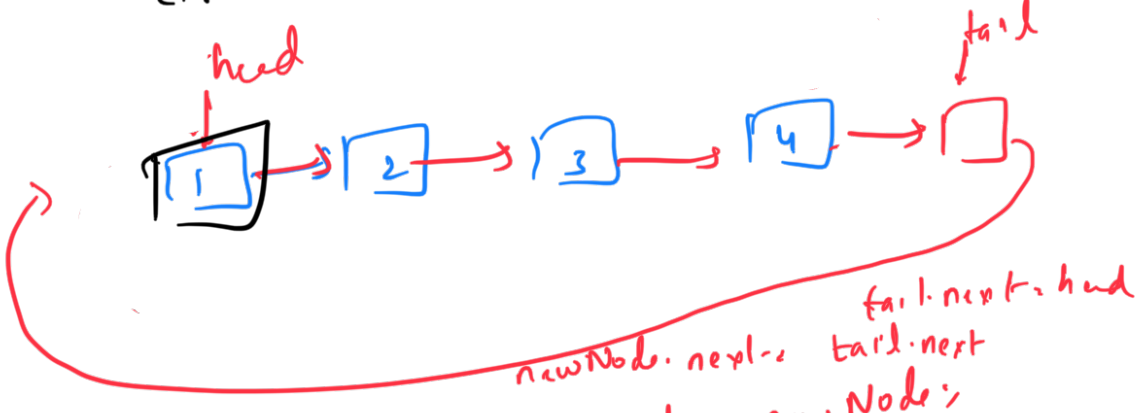
15-3

T.C : $O(n^2)$

S.C : O(1)

Types of Linked Lists

1) Singly LL
2) Doubly LL
3) Circular LL
4) Circular Doubly LL



tail.next = head

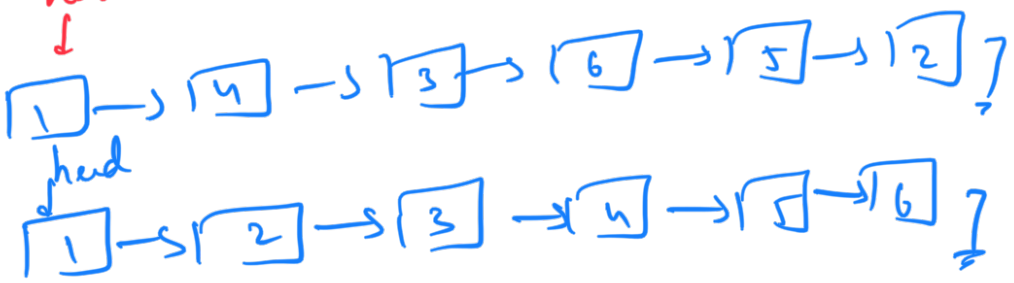
newNode.next = tail.next

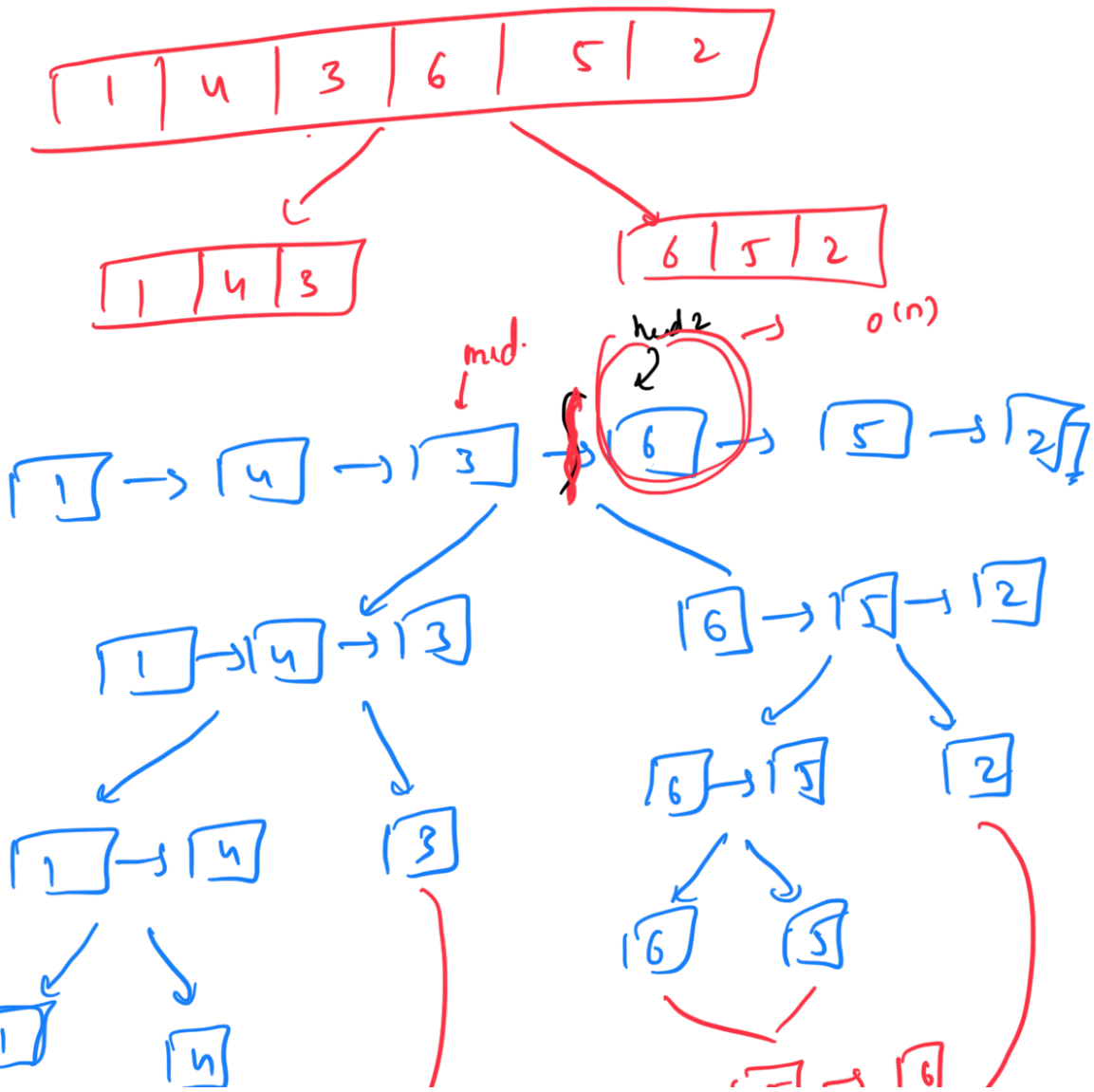
tail.next = newNode;

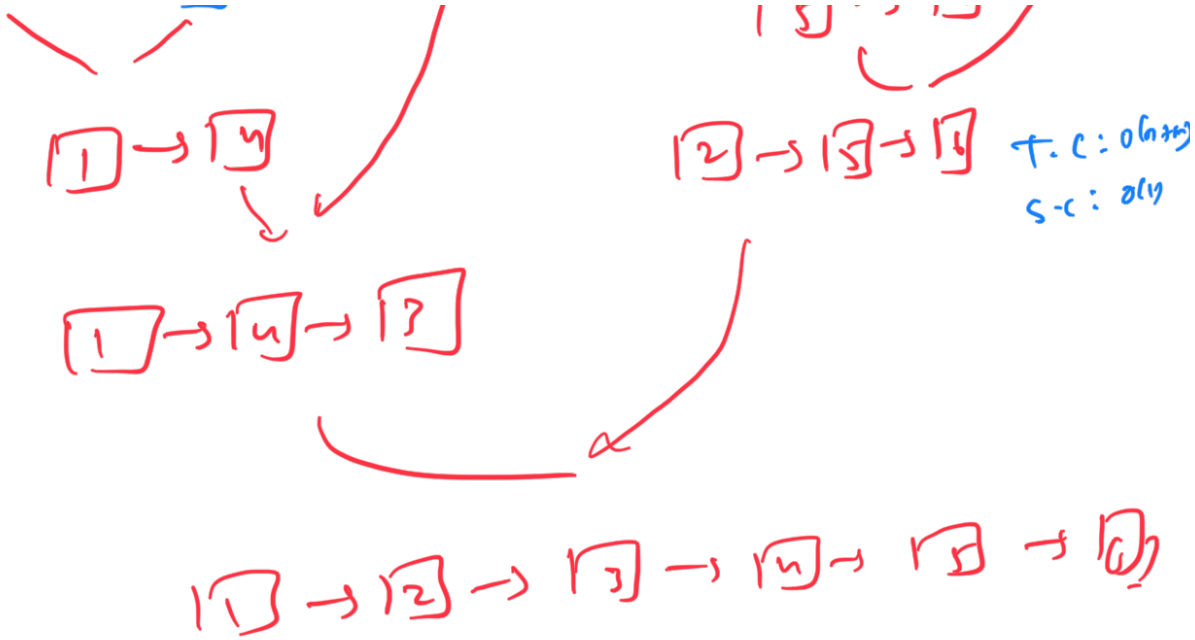
tail = newNode.

head.next = newNode

Question: Sort the Linked List using Merge Sort.







Recursion :

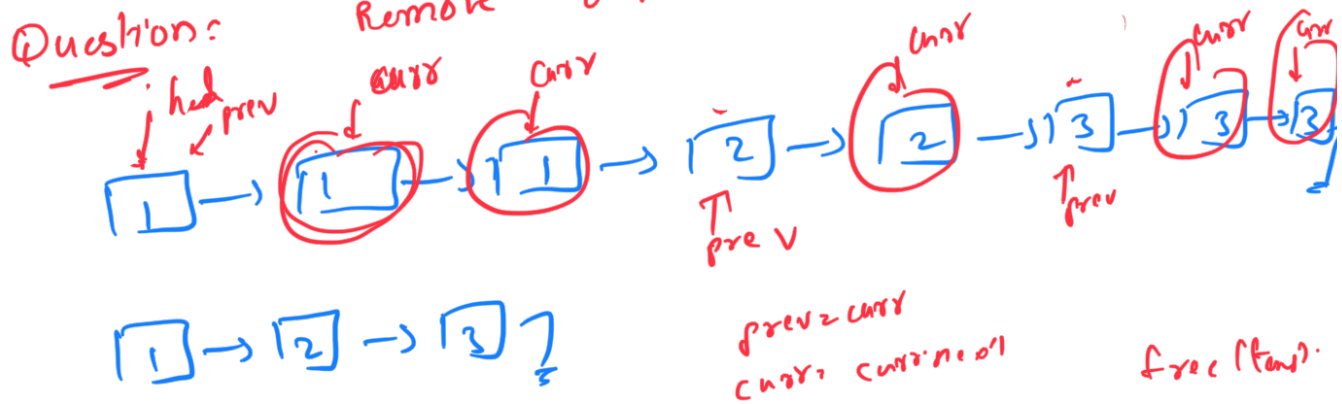
```
Node    mergeSort(head) {
        if(head == NULL || head.next == NULL)
            return head;

        middle = findMiddle(head);          O(n)
        head2 = middle.next;                O(1)
        middle.next = null;                 O(1)

        p1 = mergeSort(head);               → T(n/2)
        p2 = mergeSort(head);               → T(n/2)

        newHead = merge(p1, p2);            → T(N)

        return newHead;
    }
```

$T(N) = O(N) + 2T(\frac{N}{2}) + O(N)$

$T(n) = 2T(\frac{n}{2}) + O(n)$

T.C : O(n log n)

Question: Remove duplicates from a sorted list

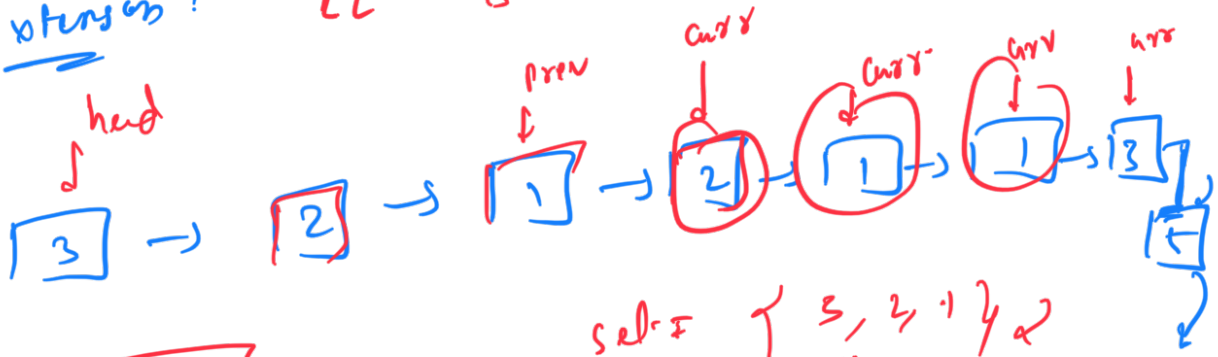


prev = curr
curr = curr.next

free (temp).

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

Extension: LL is not sorted



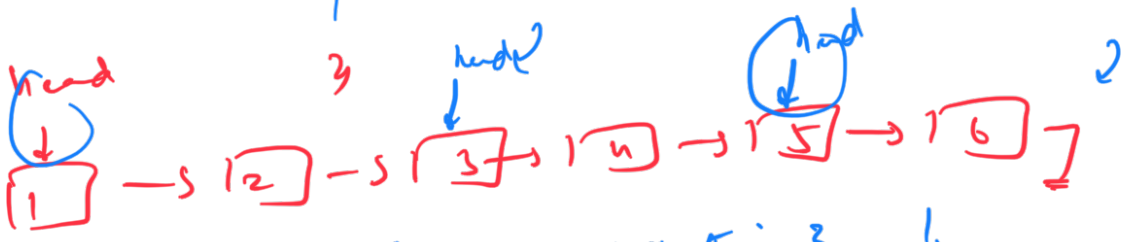
Hashset

T.C: O(n)
S.C: O(n) (Hashset)

Indentation ✗
CPP code

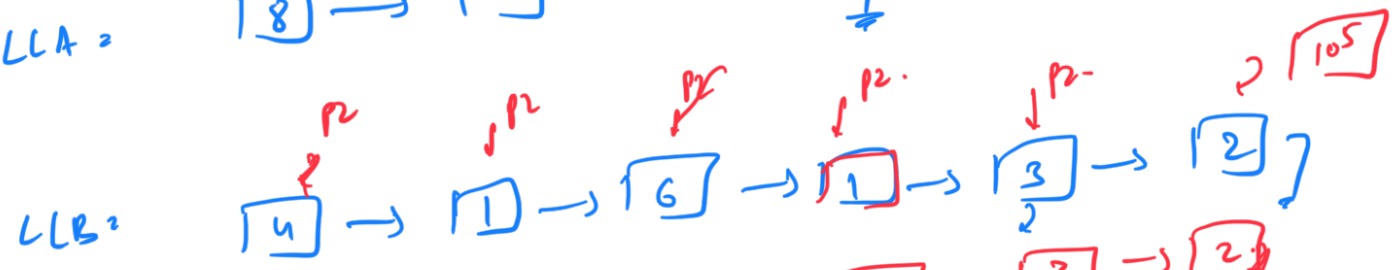
```
void fun (start) {
    if (start == NULL)
        return;

    print (start.data);
    if (start.next != NULL)
        fun (start.next.next)
```

print (start.data);



Question: Add 2 numbers as a list

carry = 0



LL: 2 → 2 → 1 → 1 → 3 → 2

2 3 1 6 1 4

LL = 2 → 2 → 1 → 2 → 3 → 2

2 3 1 6 1 4
5 0 8
2 3 2 1 2 2

A = 2 3 1 6 1 4
B = 5 0 8
2 2

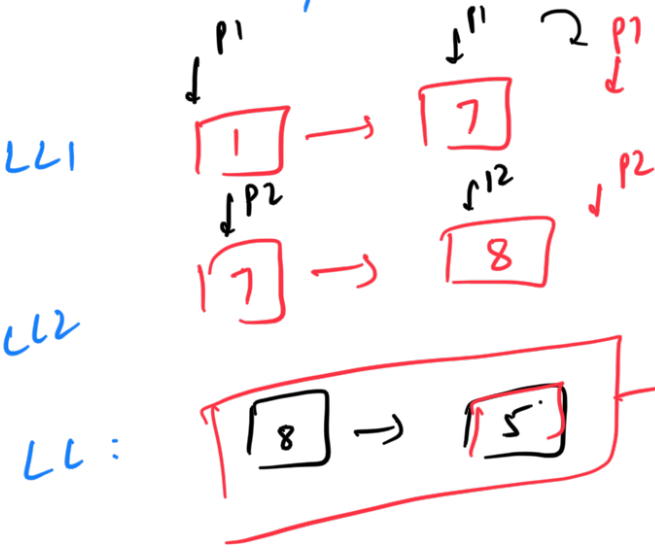
12%.10

```
addNumbers ( Node A, Node B) {
    p1 = A,  p2 = B ;   carry = 0;
{   while( p1 != NULL || p2 != NULL) {
        sum = 0;
        if(p1 != NULL)
        sum += p1.data
```

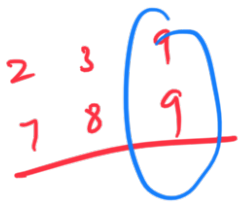
```
if(p2 != NULL) {
    sum += p2.data;
    sum += carry;
    newNode = new Node();
    newNode.data = sum % 10;
    carry = sum/10;
    insertAtEnd(newNode);
}
if(p1 != NULL) p1 = p1.next;
if(p2 != NULL) p2 = p2.next;
}
return head;
```

sum of:

carry = 0 _ 1



T.C: O(max(N, M))



carry > 1

a+1

sum = 10

c0 % 10 =

sum = 0

carry = 1.

S.C: O(1)

```
Node* mergeSort(Node* head){
      if(head == NULL || head->next == NULL) return head;

      Node* middle = middle(head);
      Node* head2 = middle->next;
      middle->next = NULL;
      Node* A = mergeSort(head);
      Node* B = mergeSort(head2);
      Node* newHead = merge(A, B);

return newHead;
}
```

```
Node* addNumbers(Node* A, Node* B)
      sum = 0, carry = 0;
      Node* head = NULL, tail = NULL;

      while(A!= NULL || B!=NULL || carry > 0){
            sum = 0;
            if(A != NULL ) sum += A->data;
            if(B != NULL) sum += B->data;
            sum += carry;
            carry = sum/10;
            sum = sum%10;
            Node* newNode = new Node(sum);
            if(tail == NULL){
                  head = tail = newNode;
            }
            else{
                  tail->next = newNode;
                  tail = newNode;
            }
            if(A != NULL ) A = A->next;
            if(B != NULL ) B = B->next;

      return head;
}
```

Question: Longest palindrome in a linked list

```
int longestCommonList(Node *p1, Node *p2){
    int len = 0;
    while (p1 != null && p2 != null && p1->data == p2->data){
           len++;
           a = a.next;
           b = b.next;
     }
    return len;
}


int maxPalindrome(Node *head){
    int result = 0;
    Node *prev = NULL, *curr = head;

     while (curr){
        next = curr->next;
        curr->next = prev;

           // Odd length
        result = max(result, 2*countCommon(prev, next)+1);
           // Even Length
        result = max(result, 2*countCommon(curr, next));

        prev = curr;
        curr = next;
    }
    return result;
}
```