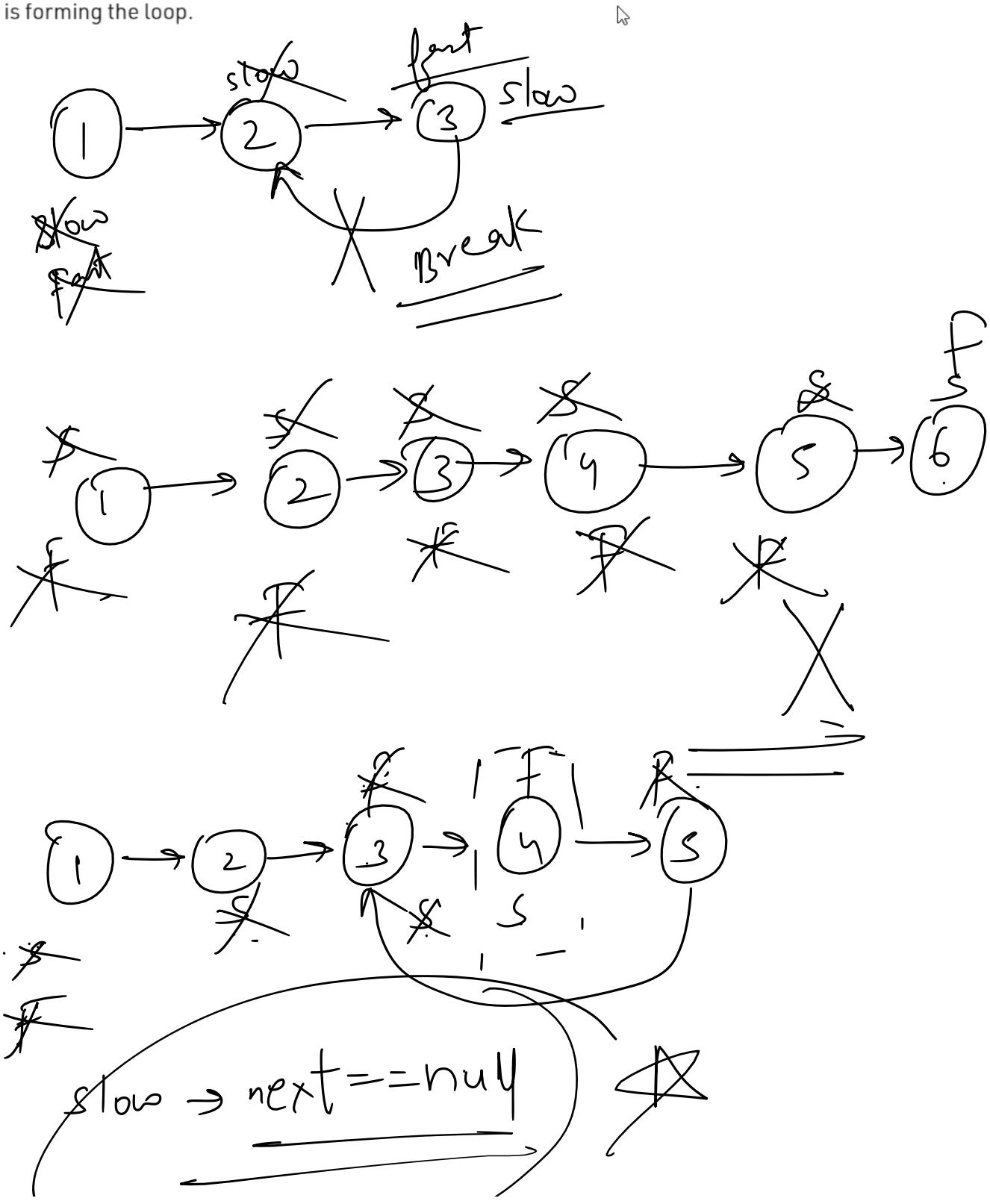
Day 16

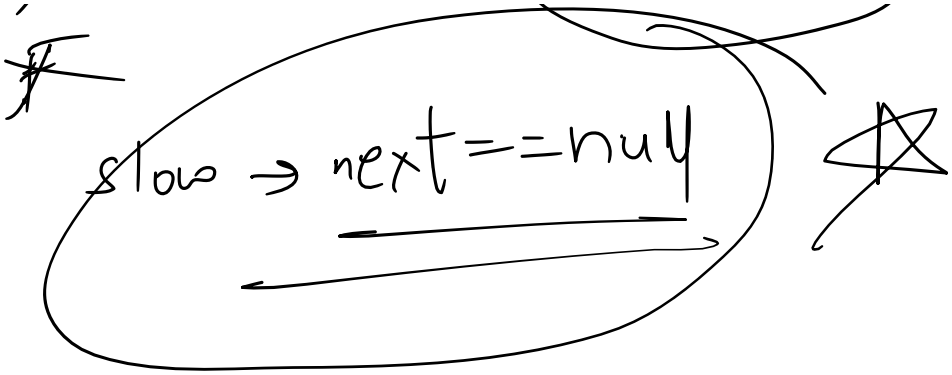
28 November 2022 21:00

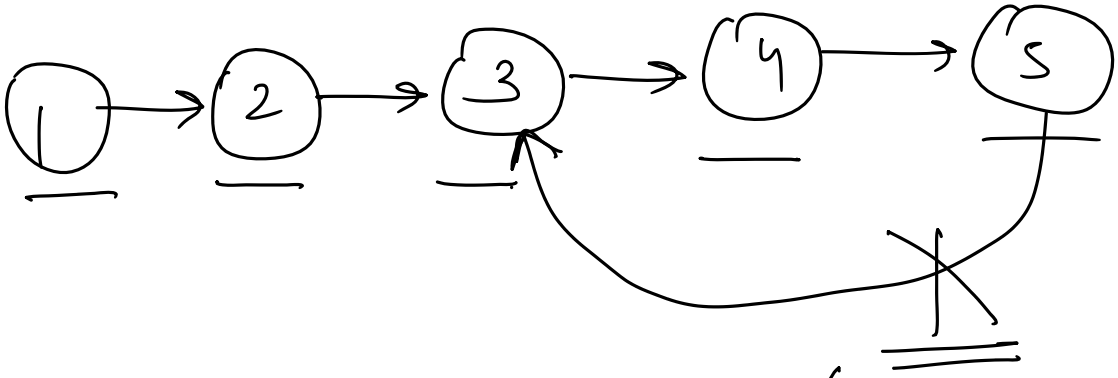
Given a linked list of **N** nodes such that it may contain a loop.

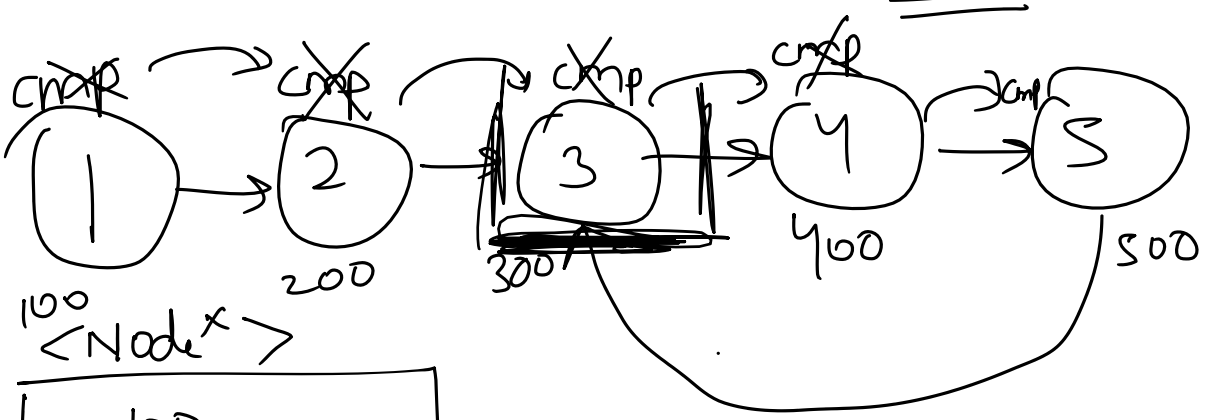
A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.

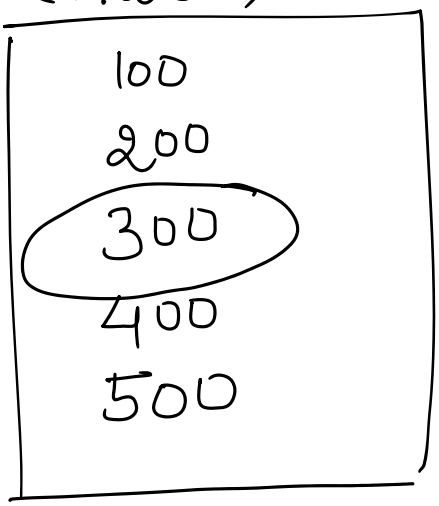
Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.



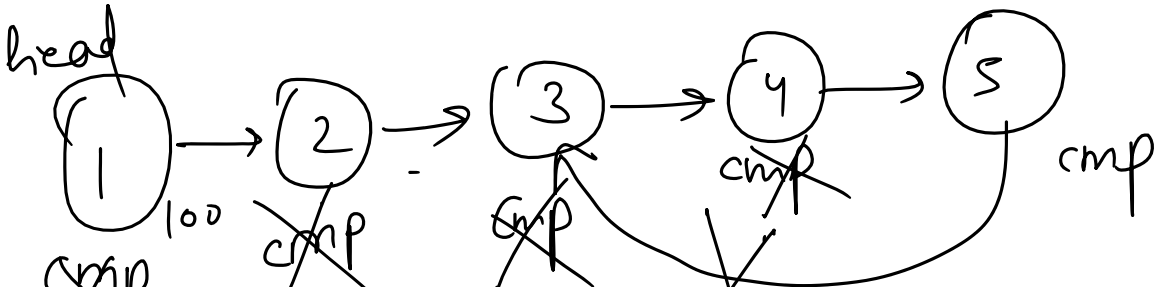

slow → next == null

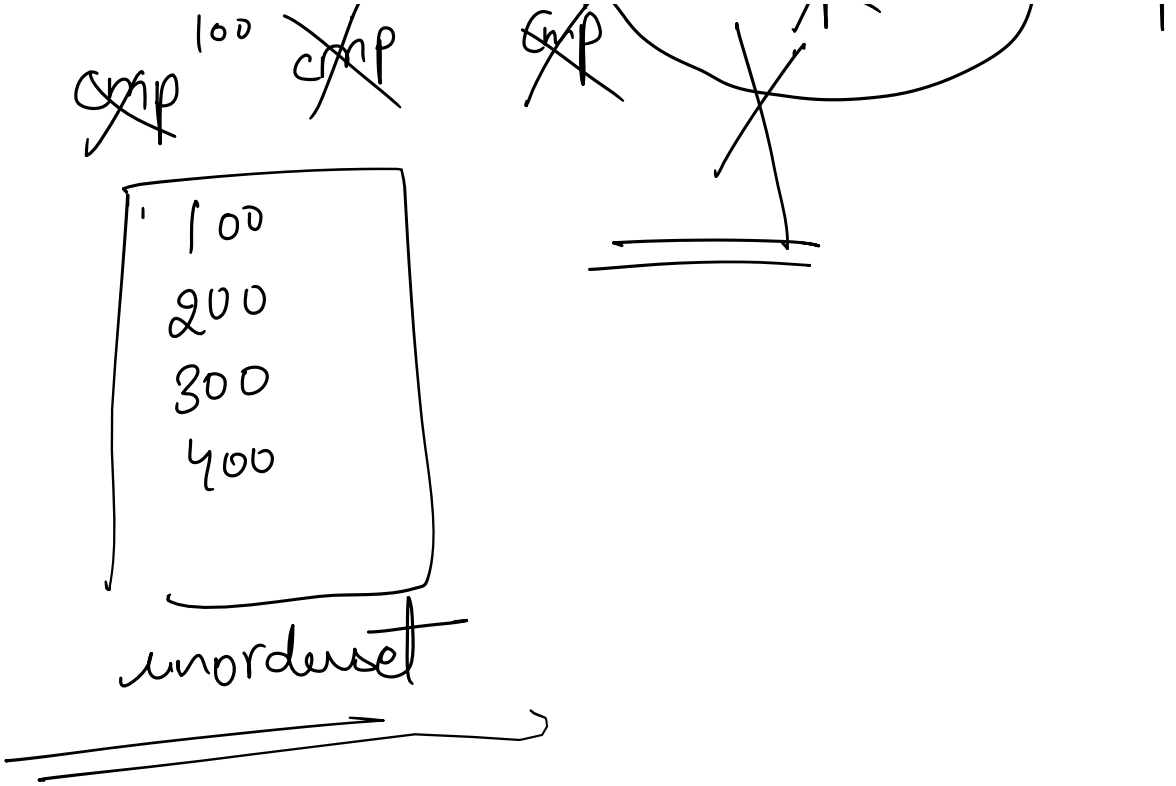

1
2
3
4
5


cmp
1
2
3
4
5
100
200
300
400
500


100
<Node x>
100
200
300
400
500

set

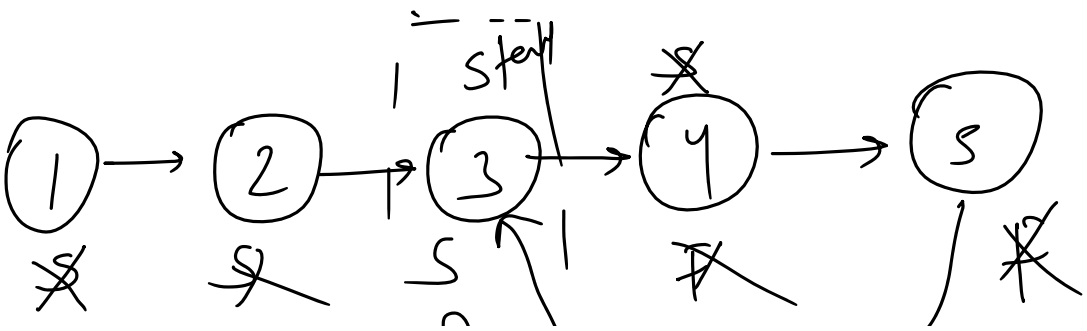

head
1
2
3
4
5
100
cmp

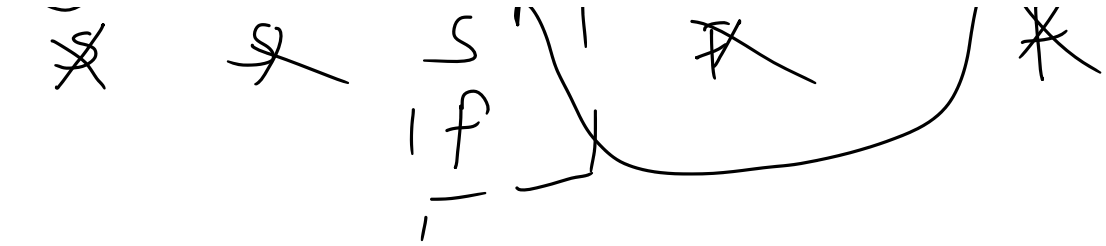


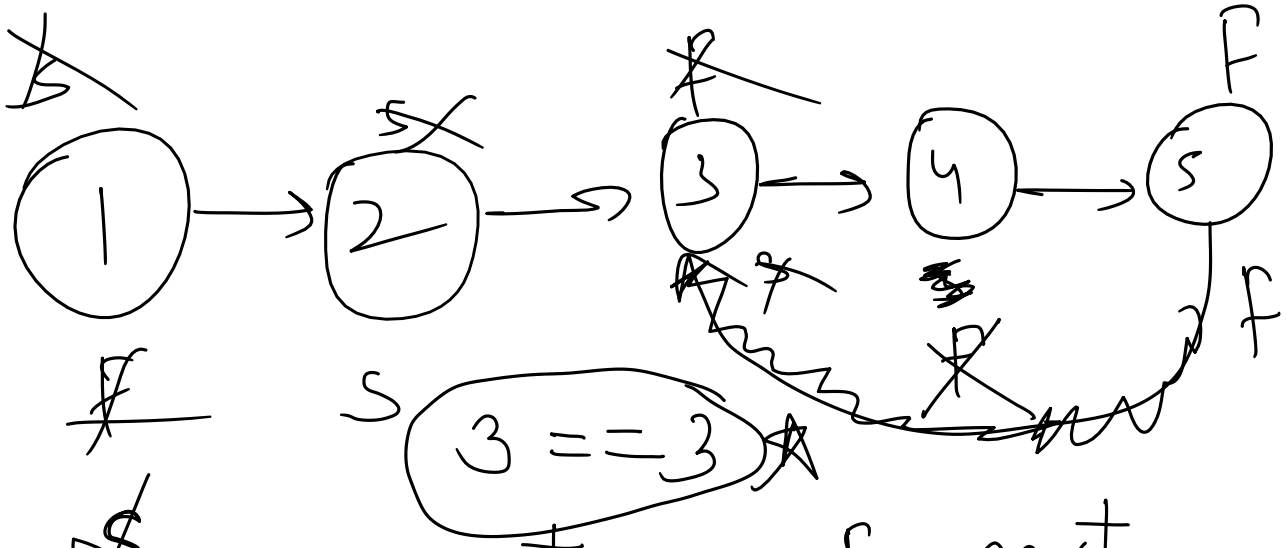

```cpp
void removeLoop(Node* head)
{
    Node * slow = head;
    Node * fast = head;
    bool flag = false;
    do{
        slow = slow->next;
        fast = fast->next->next;
        if(slow == fast){
            flag = true;
            break;
        }
    }while(fast!=NULL && fast->next != NULL);
    if(!flag){
        return;
    }
    unordered_set<Node *> um;
    Node * temp = head;
    while(true){
        if(um.find(temp->next)!=um.end()){
            temp->next = NULL;
            return;
        }
        um.insert(temp);
        temp = temp->next;
    }
}
```

T.C - O(N)

S.C - O(N)



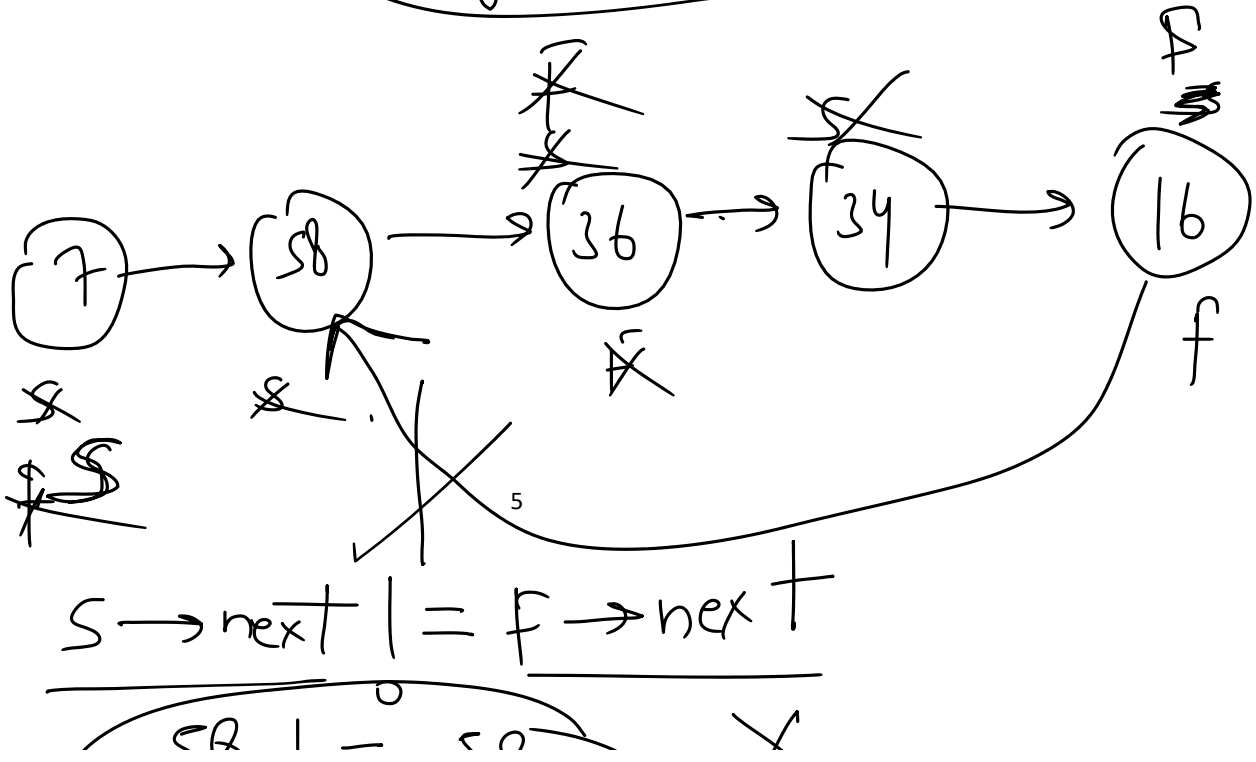




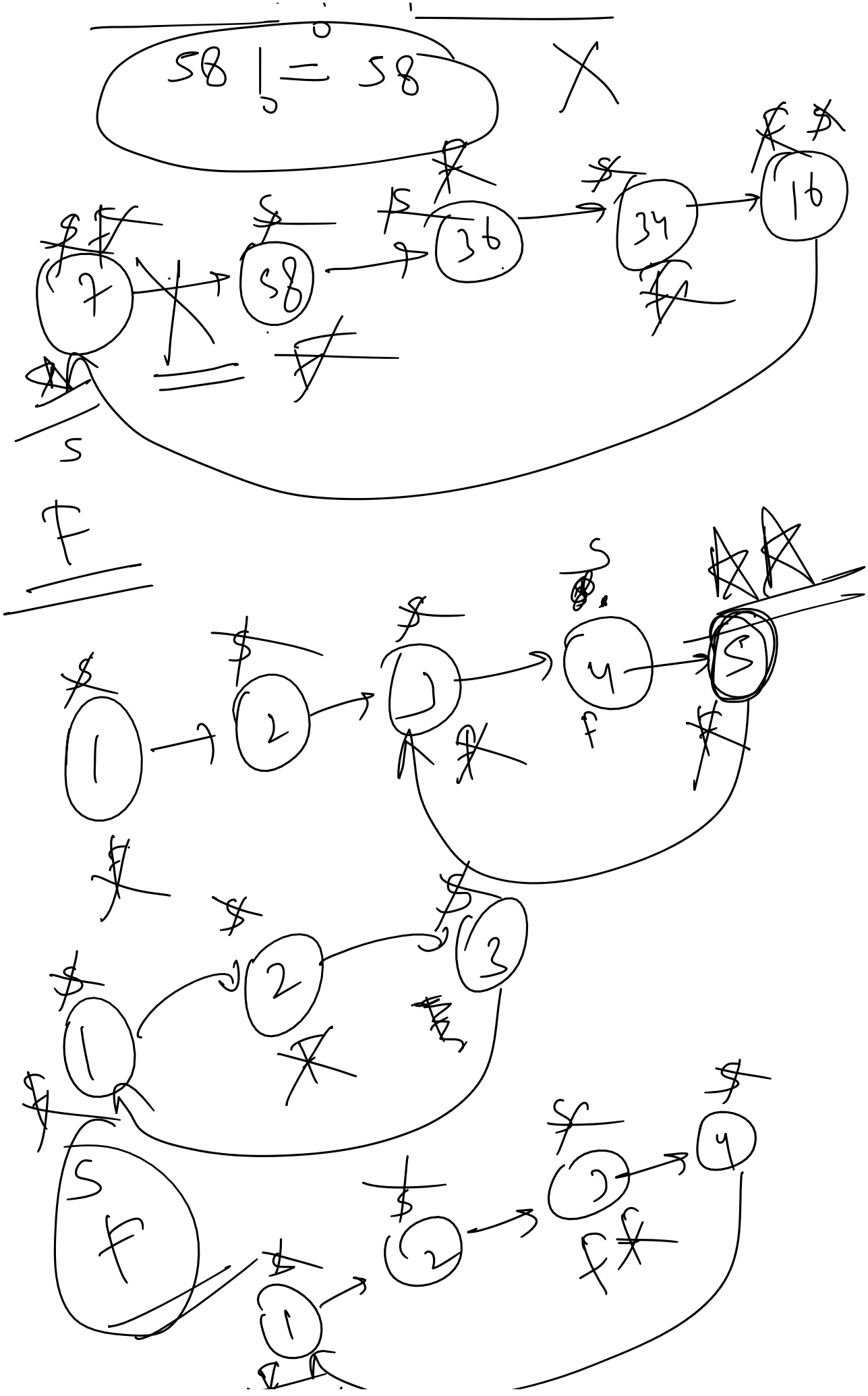
s->next == f->next

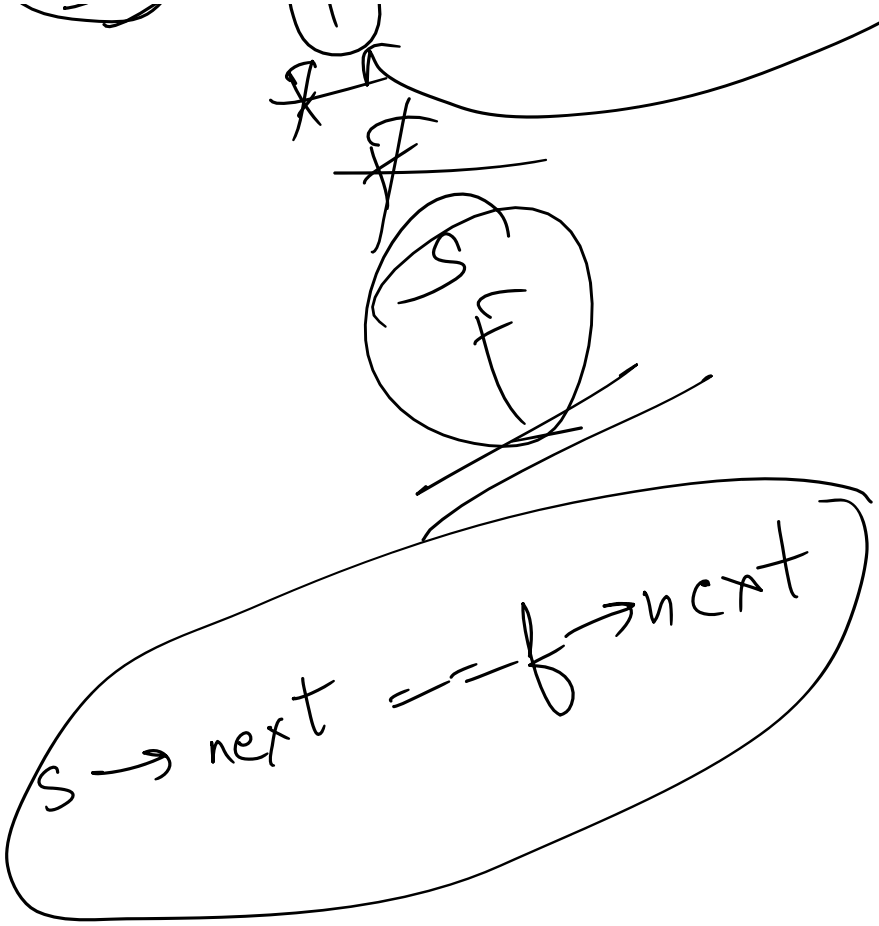
2 == 5

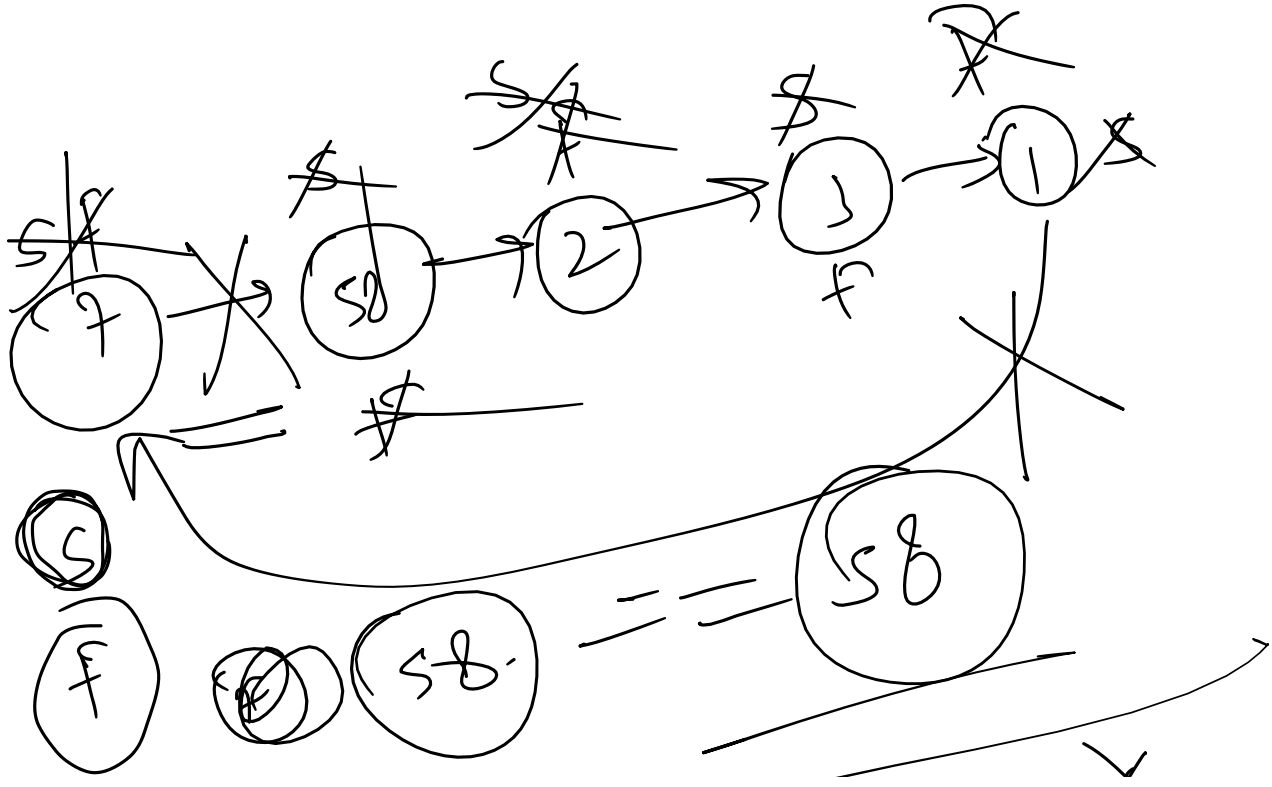
f->next = NULL

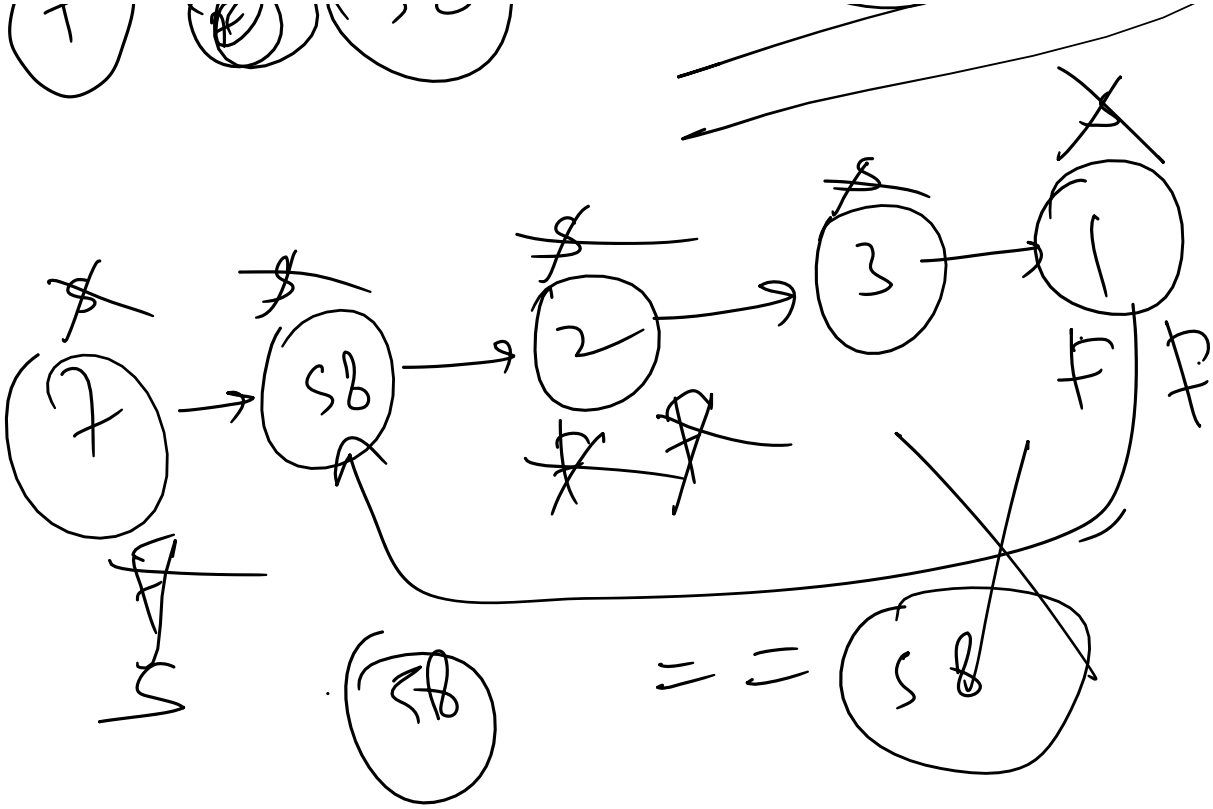


s->next != f->next




s -> next == f -> next





```
void removeLoop(Node* head)
{
    Node * slow = head;
    Node * fast = head;
    bool flag = false;
    do{
        slow = slow->next;
        fast = fast->next->next;
        if(slow == fast){
            flag = true;
            break;
        }
    }while(fast!=NULL && fast->next != NULL);
    if(!flag){
        return;
    }
    slow = head;
    while(slow!= fast){
       slow = slow->next;
       fast = fast->next;
    }
    while(fast->next != slow){
        fast = fast->next;
    }
    fast->next = NULL;
}
```

T.C - O(N)

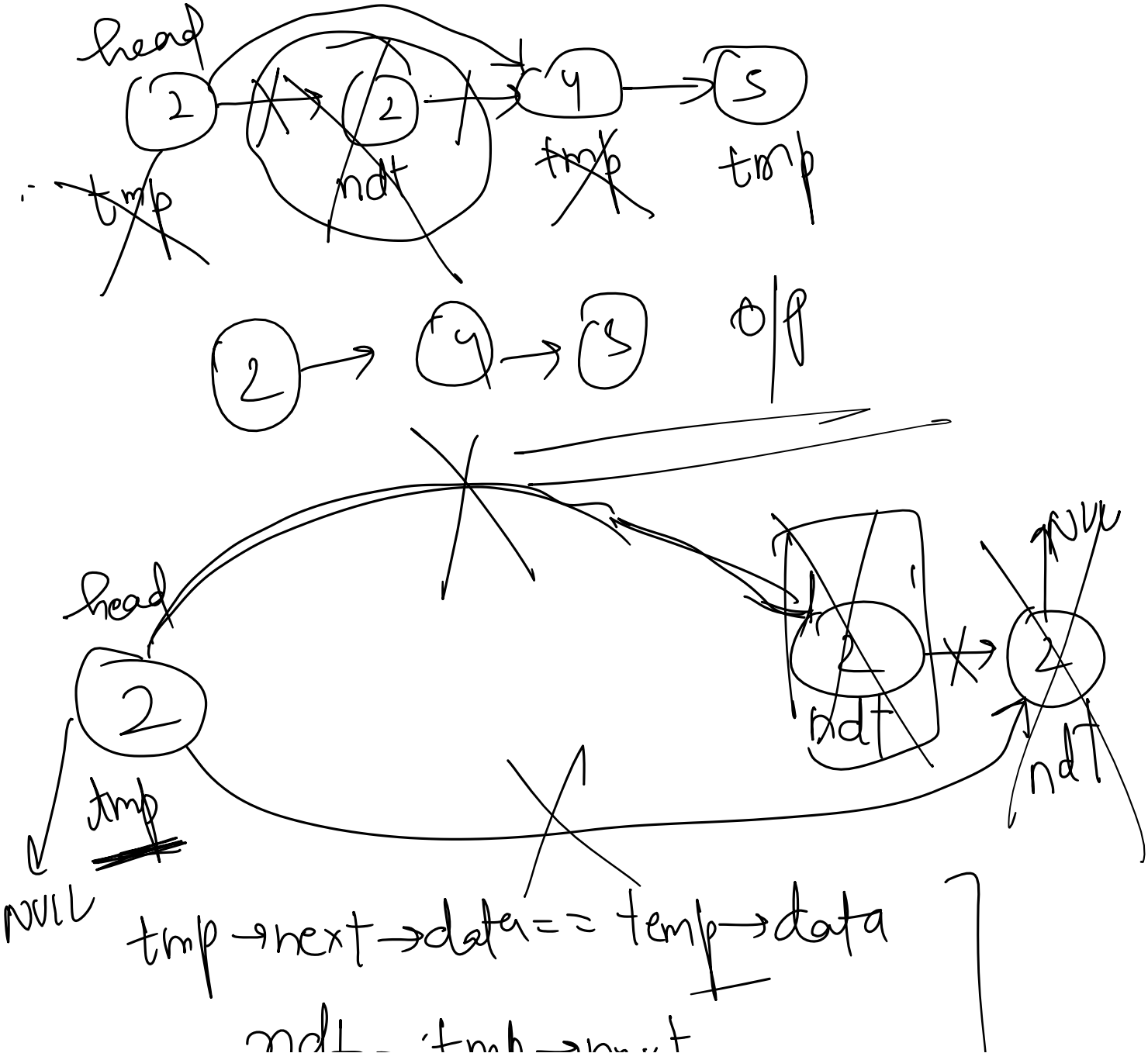
S.C - O(1)

Given a singly linked list consisting of **N** nodes. The task is to remove duplicates (nodes with duplicate values) from the given list (if exists).

Note: Try not to use extra space. Expected time complexity is **O(N)**. The nodes are arranged in a **sorted** way.

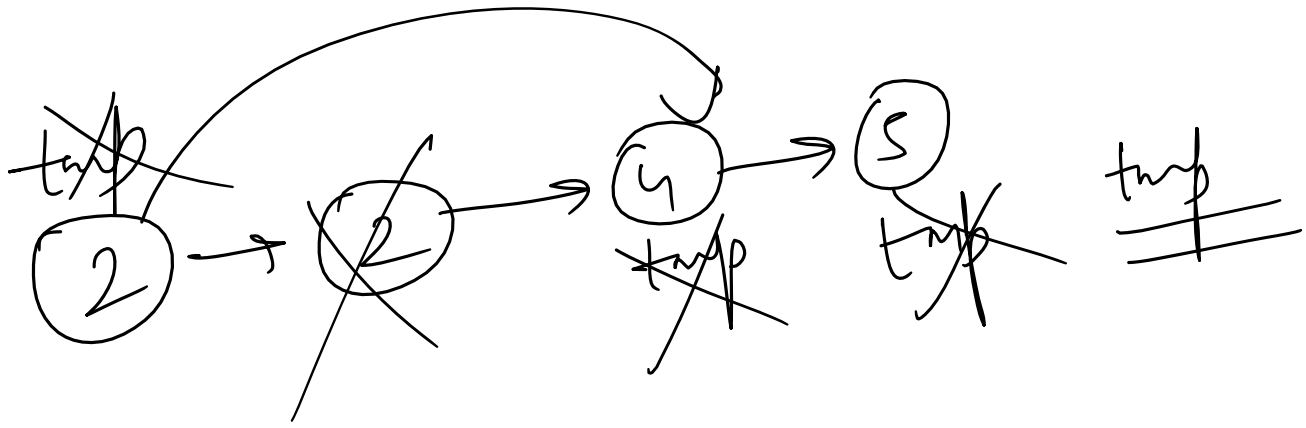
Example 1:

Input:
LinkedList: 2->2->4->5
Output: 2 4 5
Explanation: In the given linked list 2 ->2 -> 4-> 5, only 2 occurs more than 1 time.



```
ndt = tmp->next
tmp->next = ndt->next
ndt->next = NULL
delete ndt
```

Space X T.C - O(N)



```
//Function to remove duplicates from sorted lin
Node *removeDuplicates(Node *head)
{
  Node * tmp = head;
  while(tmp->next!= NULL){
      if(tmp->next->data == tmp->data){
          Node * ndt = tmp->next;
          tmp->next = ndt->next;
          ndt->next = NULL;
          delete ndt;
      }else{
           tmp = tmp->next;
      }
  }
  return head;
}
```