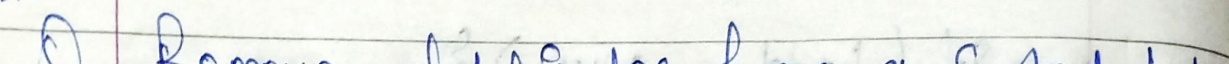
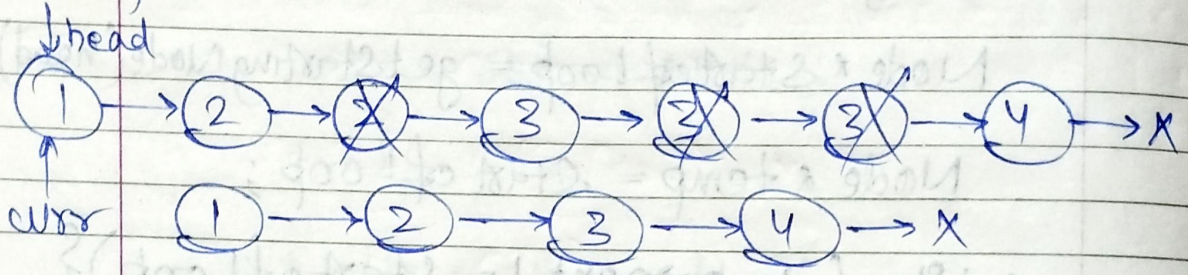
Page No.
Date

Q. Remove duplicates from a Sorted LL



Eg:-



Approach:

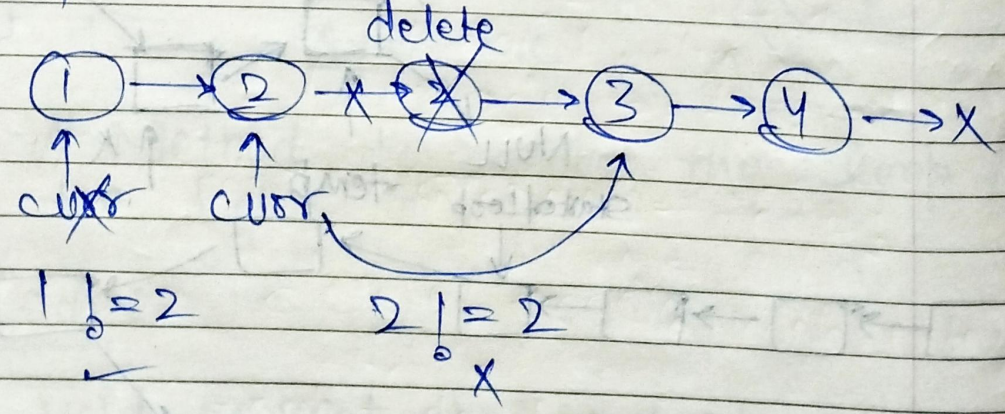
```
Node * curr = head;
while ( curr -> data != curr -> next -> data)
```

If not then

```
curr -> next = curr -> next -> next;
```

and delete the duplicated data.

Eg:..



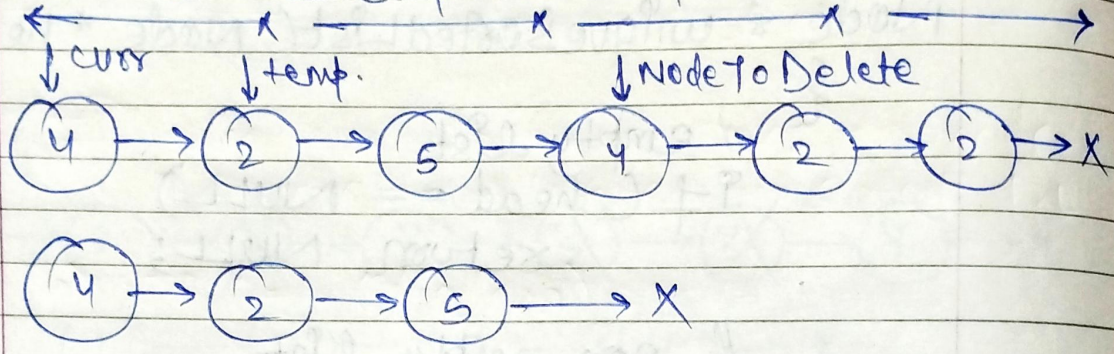
Page No.
Date

Code :

```
Node * uniqueSortedList( Node * head)
    {
        // empty list
        if ( head == NULL)
            return NULL;

        // non-empty list
        Node* curr= head;

    while ( curr != NULL) {

    if( (curr->next != NULL) &&

    curr->data == curr->next->data) {

    Node* next_next= curr->next->next;
    Node * nodeToDelete= curr->next;
        delete( nodeToDelete);
    curr->next= next_next;
    }
    else{ // not equal
        curr= curr->next;
        }
    }
    return head;
}
```

*Imp

Q Remove the duplicate from unsorted LL



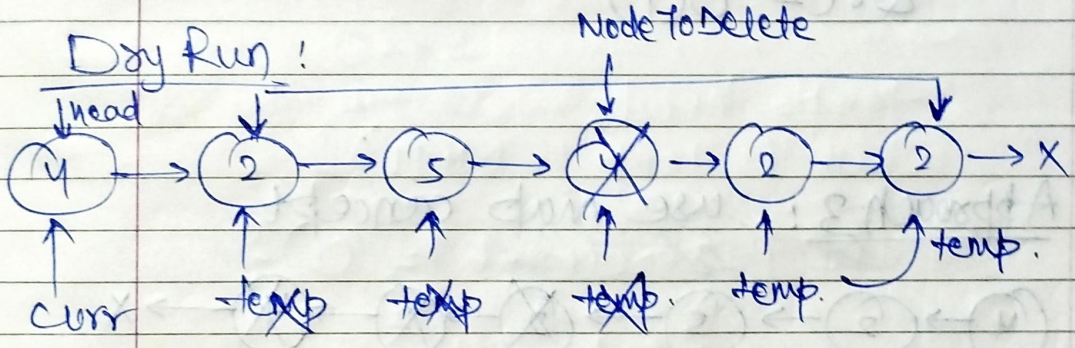
Eg:

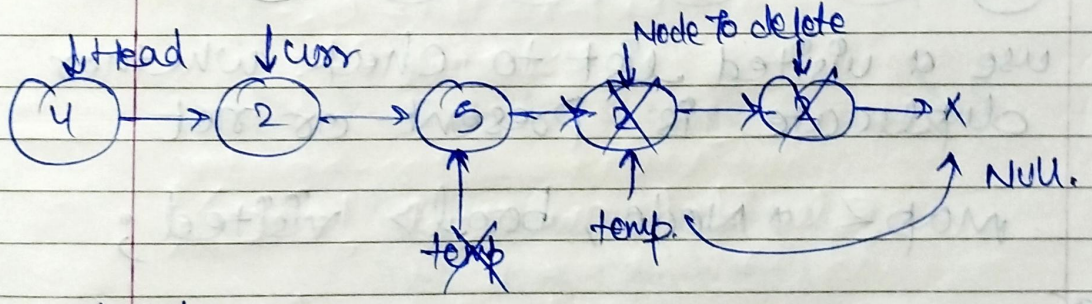
Approach:- (1) use the loop for the traversal till the end and compare each value, if equal then delete that node.

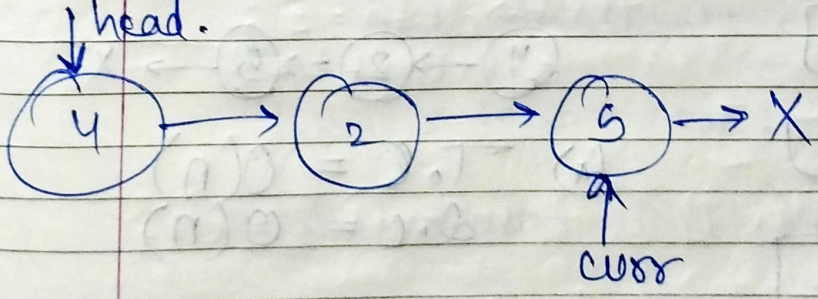
```
Node * curr = head.
while (curr != NULL){
    Node * temp = curr -> next;
    while( temp != NULL){
        // equal value
        if (curr->data == temp->data){
            Node * NodeToDelete = temp;
            delete( NodeToDelete);
            temp = temp -> next;
        }
```

```
else{ // Not equal values
        temp = temp->next;
    }
}
return head;
}
```

Dry Run :







(*) T.C = $O(n^2)$, S.C = $O(1)$

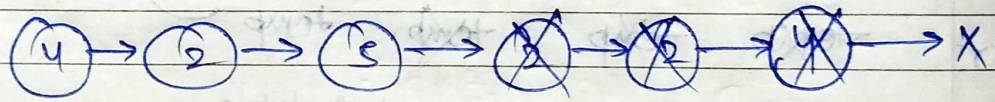
Approach 2 : (I) first sort this unsorted linked list

(II) use the previous question code // Remove duplicate in sorted LL

(*) T.C = O(nlogn)

S.C = O(1)

Approach 3 : use map concept.



use a visited list to check whether duplicate is present or not

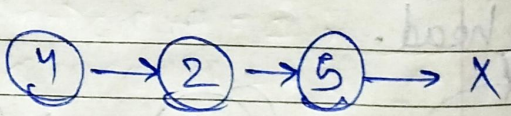
map< *Node, bool> visited;

curr → [4 | true]

curr → [2 | true]

curr → [5 | true]

curr → [2 | true] → already visited

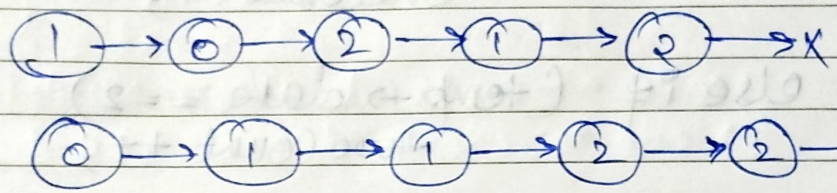


(*) T.C = O(n)

S.C = O(n)

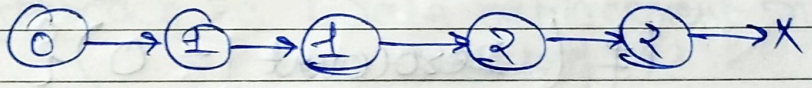
imp (*****)

Q Sort 0's, 1's and 2's in the linked list

Eg:- 

→→ Approach 1: count the no. of 0's, 1's and 2's and then place them accordingly

0 → 1
1 → 2
2 → 2



Code :-

```
Node * SortList (Node * head) {

    int zeroCount = 0;
    int oneCount = 0;
    int twoCount = 0;

    Node * temp = head;

    while (temp != NULL) {

        if (temp -> data == 0)
            zeroCount ++;
```

```
else if ( temp->data == 1)
        onecount++;

else if (temp->data == 2)
        twocount++;

    temp= temp->next;

}

temp= head;
while ( temp != NULL) {
  if ( zerocount != 0) {
      temp->data = 0;
       zerocount--;
    }

else if ( onecount != 0){
      temp->data = 1;
       onecount--;
   }

else if ( twocount != 0) {
      temp->data = 2;
       twocount--;
```

```
        temp = temp->next;
    }
    return head;
}
```

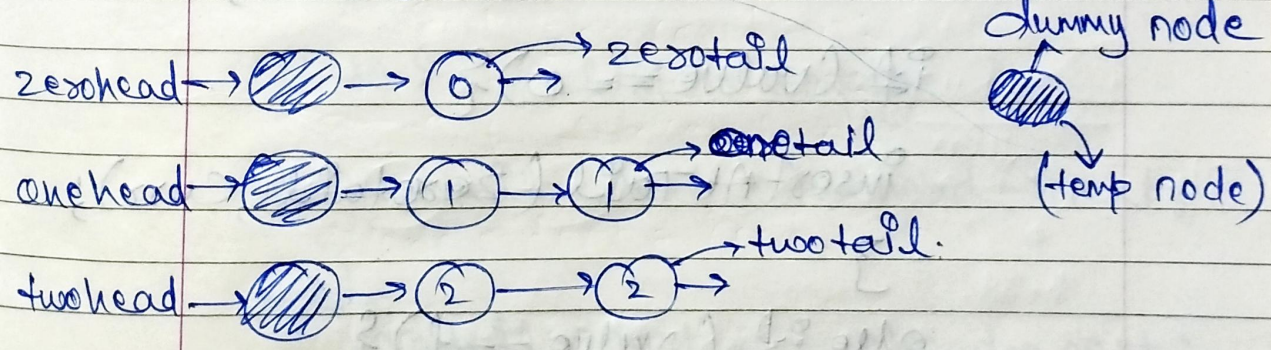
(*) T.C = O(n)

S.C = O(1)

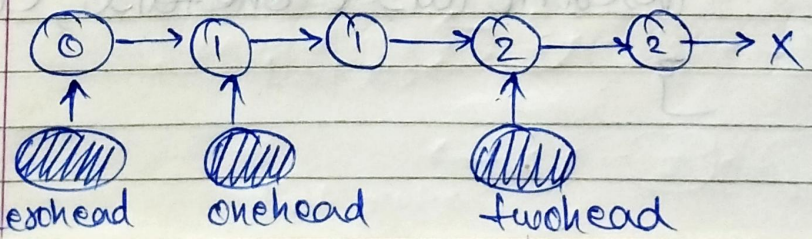
→ Approach 2: Data replacement not allowed.

Approach: Change the links

Create 3 list containing their req^n element as.



Merge Sorted list



Code :-

```
Node * SortList (Node *head){

    Node *zerohead = new Node(-1);
    Node *zerotail = zerohead;

    Node *onehead = new Node (-1);
    Node * onetail = onehead;

    Node *twohead = new Node (-1);
    Node * twotail = twohead;

  Node *curr= head;

  // create separate list
  while ( curr != NULL) {
      int value= curr->data;

    if (value == 0){

     insertAtTail( zerotail, curr);
      }
    else if ( value == 1) {
     insertAtTail( onetail, curr);
      }
```

```
else if (value == 2){
    insertAtTail(twotail, curr);
    }
    curr = curr->next;
}
    // merge 3 sublist
// 1's list is not empty
if(onehead->next != NULL){
    zerotail->next = onehead->next;
    }
else{ // list is empty
    zerotail->next = twohead->next;
    }
    onetail->next = twohead->next;
    twotail->next = NULL;
```

// Setup the head.

```
head = zerohead->next;
```

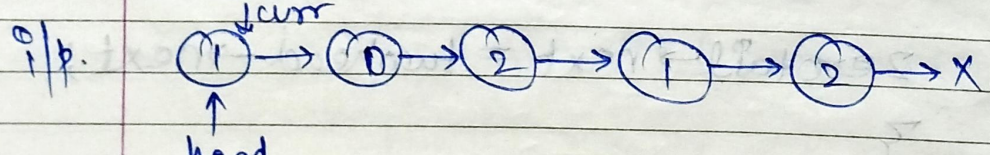
// delete dummy node

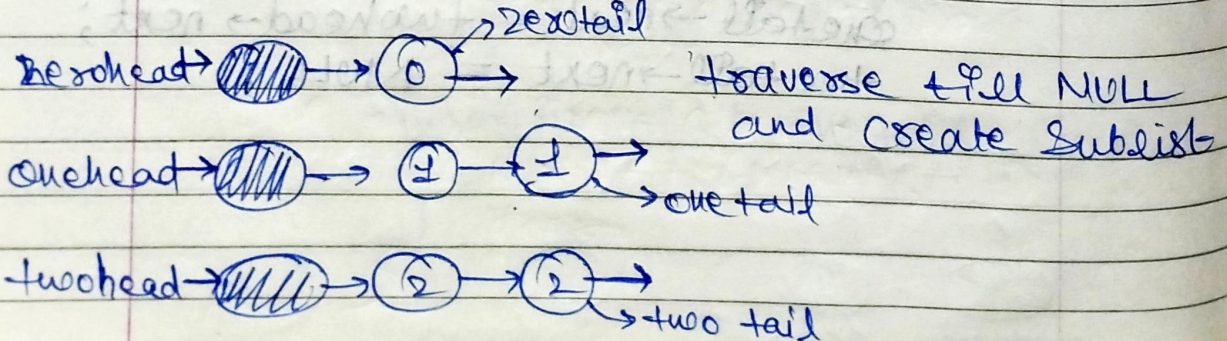
```
delete zerohead;
delete onehead;
delete twohead;
```

// use to create Sublist of 0's, 1's & 2's

```
void insertAtTail (Node* &tail, Node* curr)
{
    tail->next = curr;
    tail = curr;
}
```

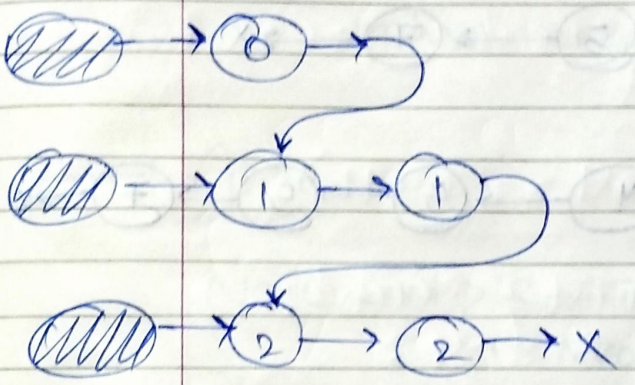
Dry Run:

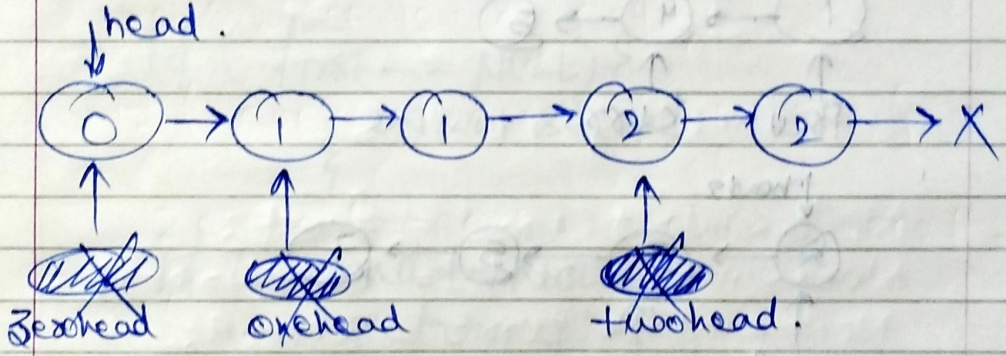




After Sublist creation, Merge these sorted list

head = zeroHead->next;



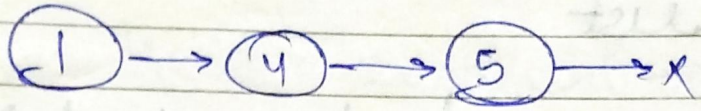


Now delete all the dummy pointer

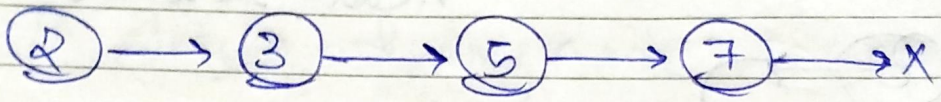
(A) Time complexity = O(n)

S.C = O(1)

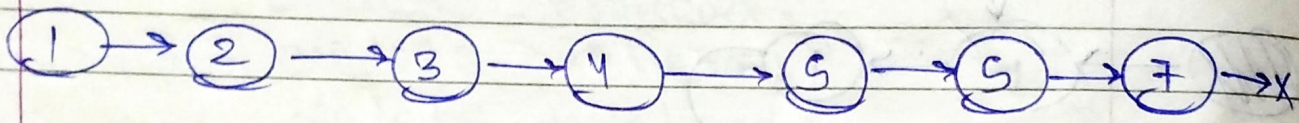
Eg:



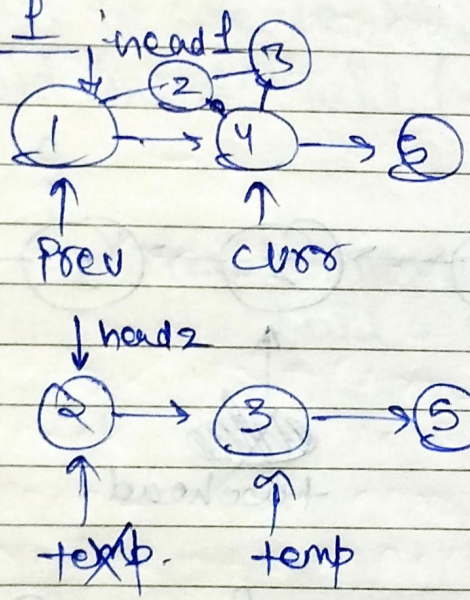
I/P



O/P



Approach 1



if (prev→data ≤ temp→data ≤ curr→data)

then push temp→data in between.

and then increase temp.

else { prev = curr;

curr = curr→next;

}

```
if( head1 == NULL) // head1 empty
    return head2;

if ( head2 == NULL)  // head2 empty.
    return head1;
```

→ Code :-

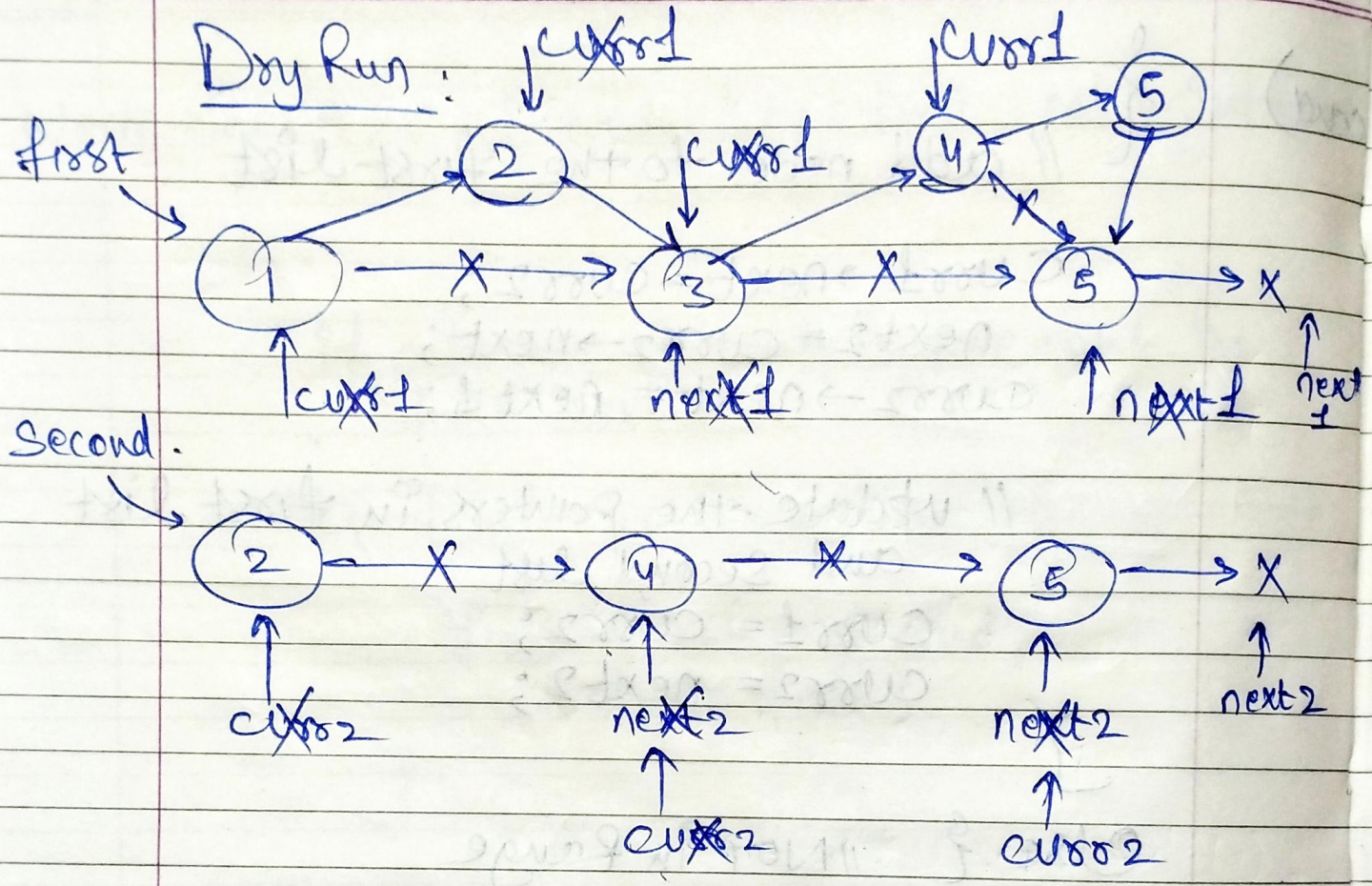
```
Node<int> SortTwoLists (Node<int>* first,
                        Node<int>* second) {

    if ( first == NULL){
            return second;
        }
    if ( second == NULL){
            return first;
        }
    if( first->data <= second->data){
            return Solve ( first, second);
        }
    else{
            return Solve (second, first);
        }
}
```

```
Node<int>* Solve( Node<int>* first, Node<int>* second)
{
// if only one element is present in
   first list
if( first->next == NULL){
    first->next = second;
    return first;
}
// Creating Pointers
Node <int> * curr1 = first;
Node<int> * next1 = curr1->next;
Node<int> * curr2 = second;
Node<int> * next2 = curr2->next;

// Checking the condition & traversing
while( next1 != NULL && curr2 != NULL)
{
if ((curr1->data <= curr2->data) &&
    (curr2->data <= next1->data))
}
```

(cont)

```
{
    // add node to the first list
    curr1->next = curr2;
    next2 = curr2->next;
    curr2->next = next1;

    // update the pointers in first list
       and second list
    curr1 = curr2;
    curr2 = next2;
}
else {  // Not in range
    curr1 = next1;
    next1 = next1->next;

// first list is at the end, then
   merge to second.
if (next1 == NULL) {
    curr1->next = curr2;
    return first;
}
}
```

Dry Run :


curr1
curr1
curr1
first
2
4
5
1
3
5
curr1
next1
next1
next1
Second.
2
4
5
curr2
next2
next2
next2
curr2
curr2