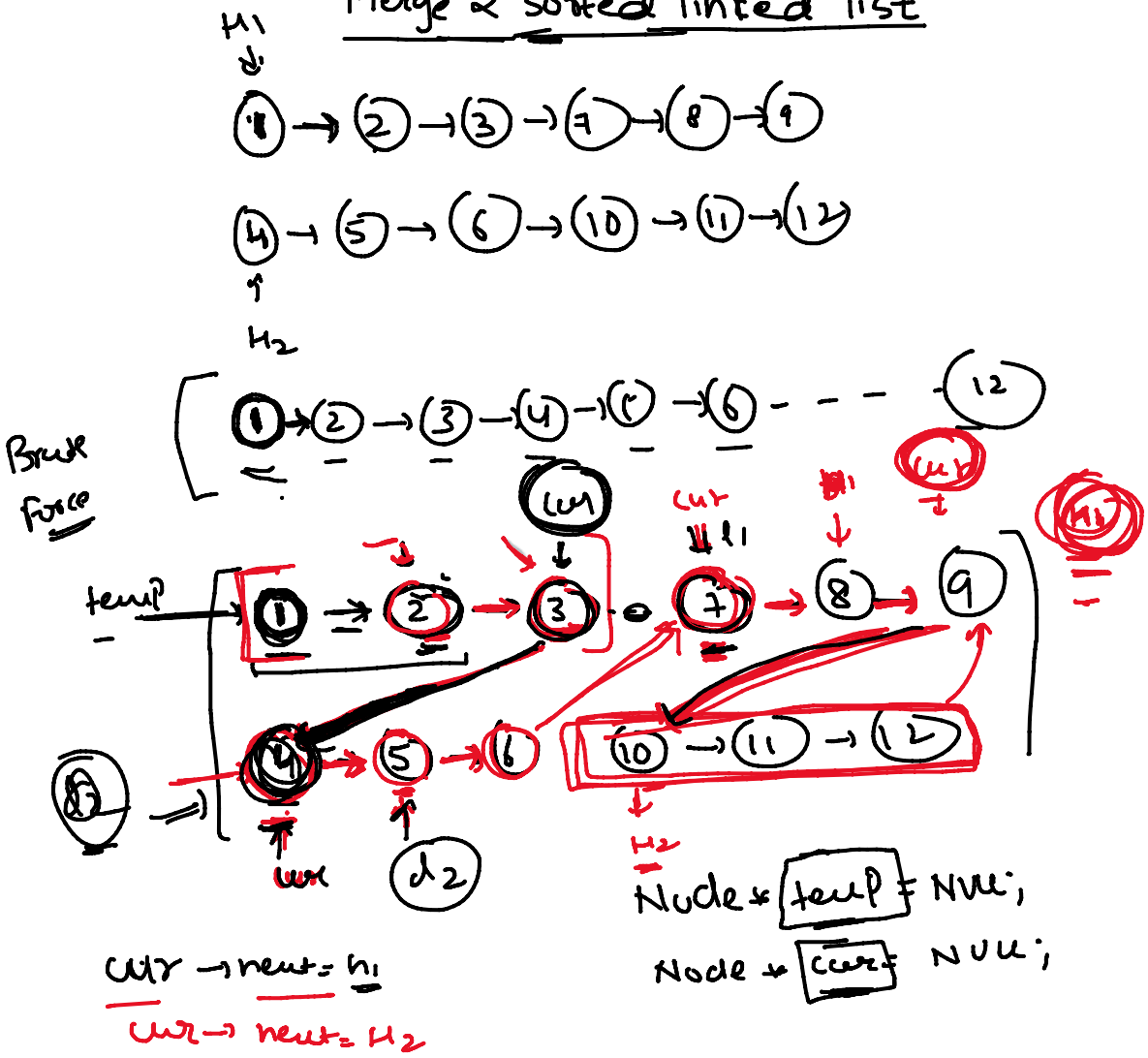
CLASS-53

Merge 2 sorted linked list



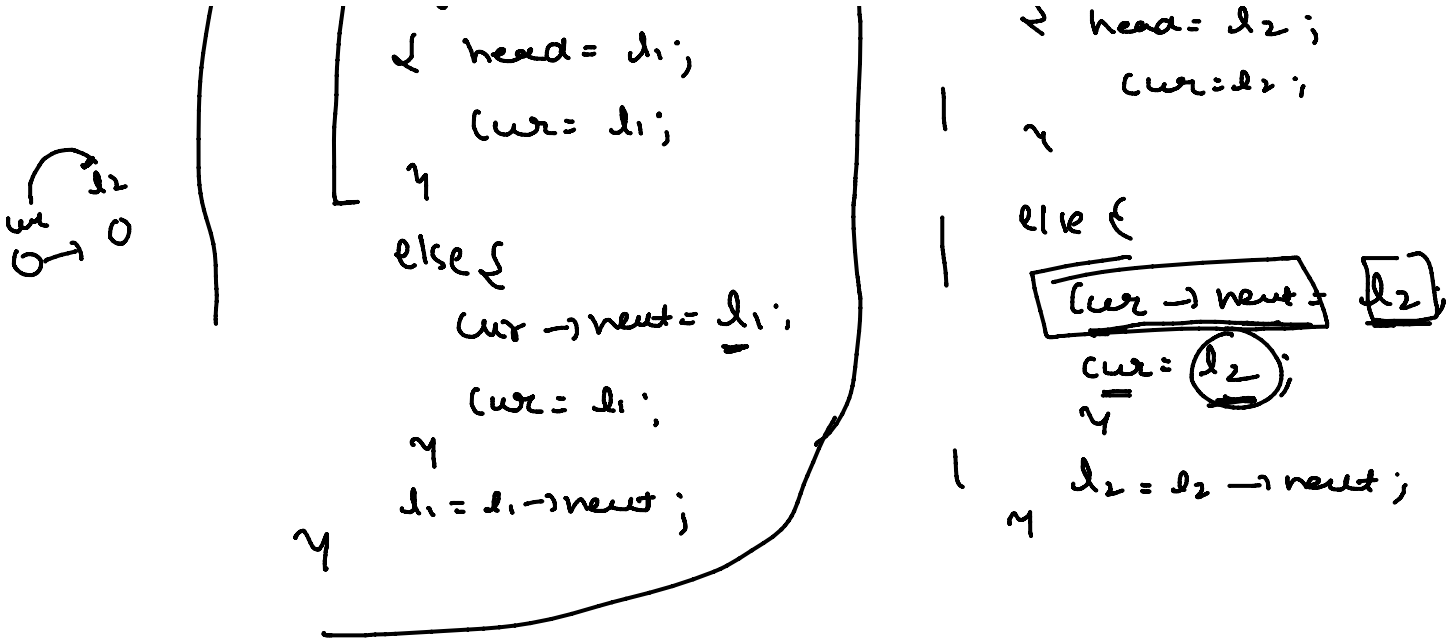
: Merge (l1 , l2)

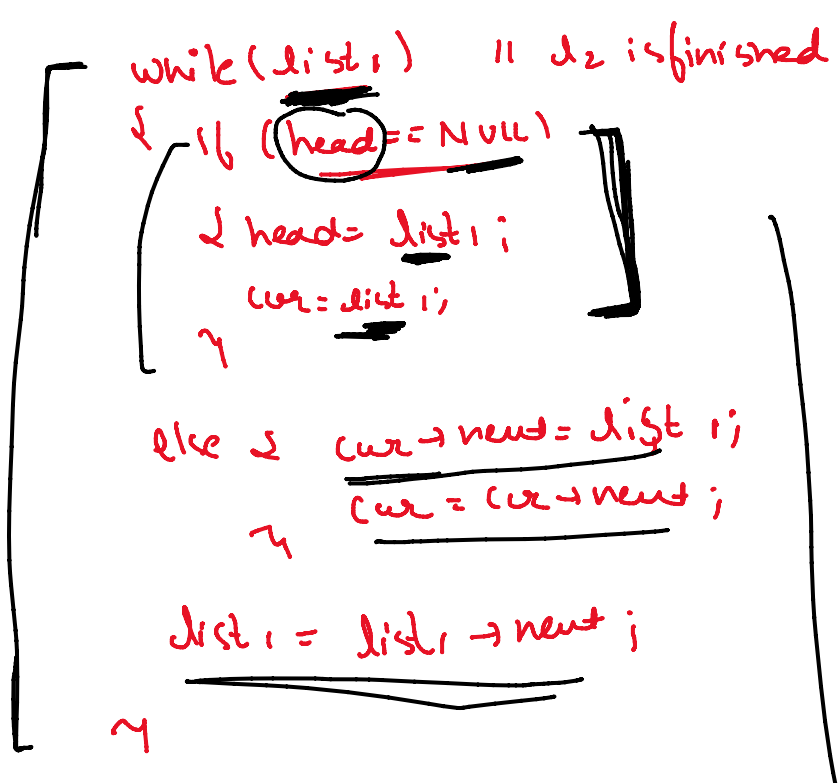
{ Node * head = NULL;

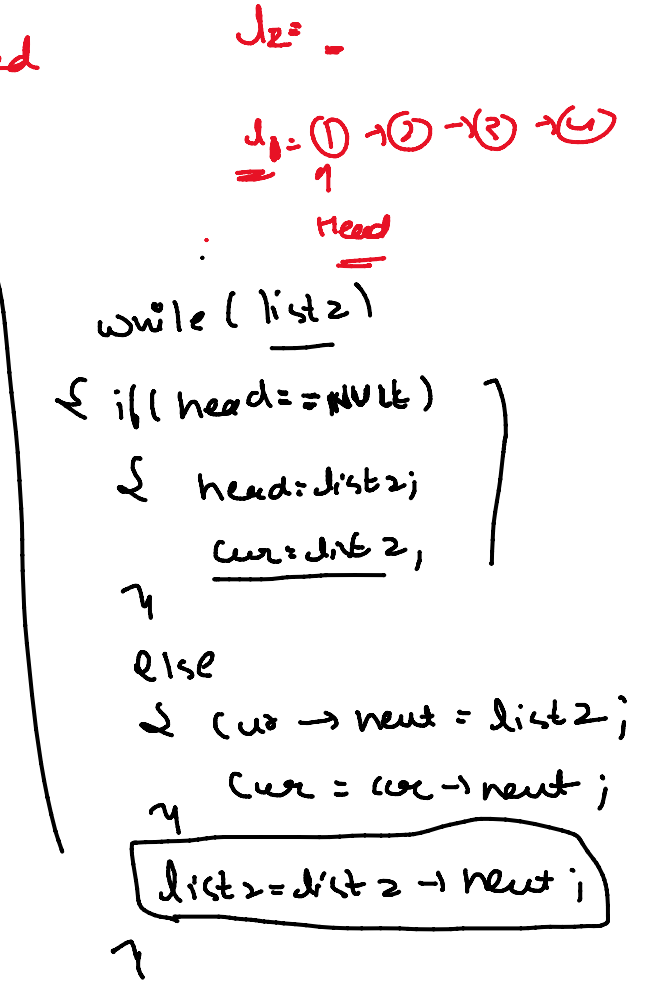
Node * cur = NULL;

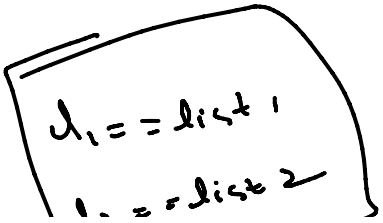
while (l1 != NULL and l2 != NULL)

{ if (l1 → val < l2 → val) { if (head == NULL) { head = l1; } } else if (l2 → val < l1 → val) { if (head == NULL) { head = l2; cur = l2;


head = l1;
cur = l1;
else {
cur → next = l1;
cur = l1;
l1 = l1 → next;
head = l2;
cur = l2;
else {
cur → next = l2
cur = l2;
l2 = l2 → next;
l2
cur
0


while (list1) // l2 is finished
if (head == NULL)
head = list1;
cur = list1;
else cur → next = list1;
cur = cur → next;
list1 = list1 → next;

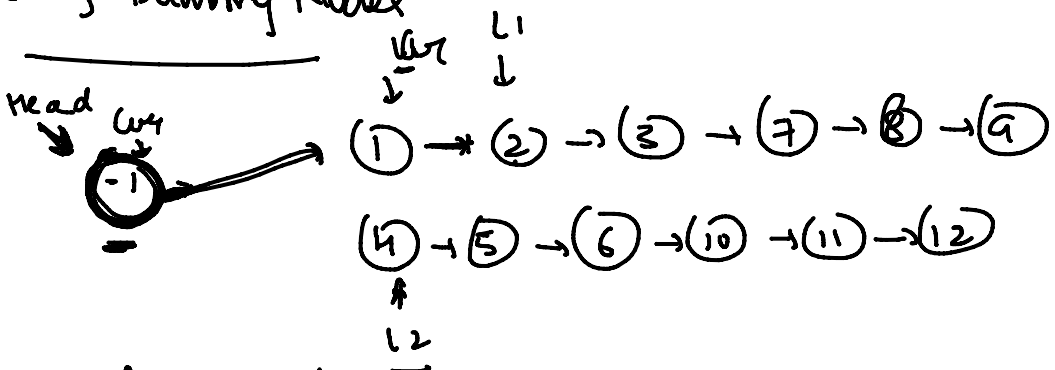

l2 =
l1 = 1 → 2 → 3 → 4
head
while (list2)
if (head == NULL)
head = list2;
cur = list2;
else
cur → next = list2;
cur = cur → next;
list2 = list2 → next;

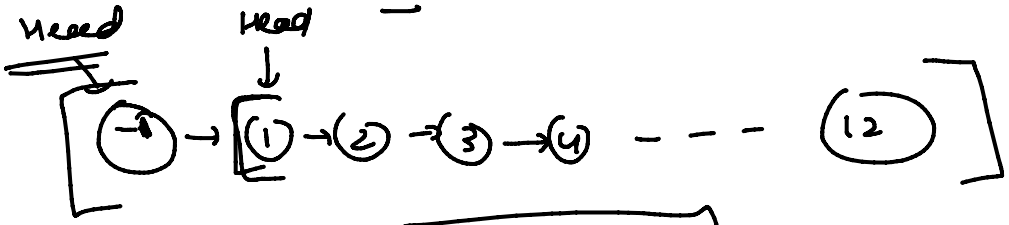

l1 = = list1
l2 = = list2

l1 = list1

l2 = list2

Using Dummy Node



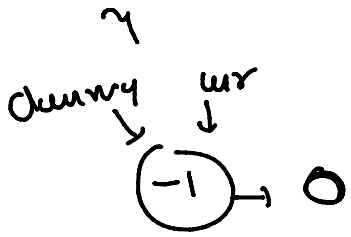


Return Head → next

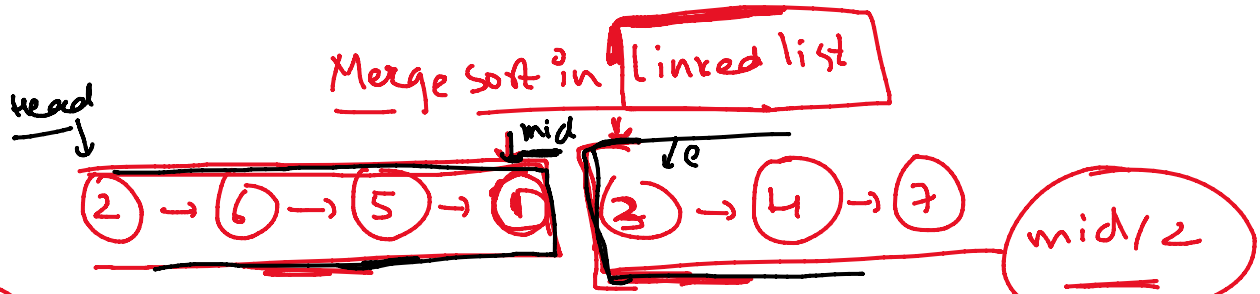
```
Node * dummy = new Node(0);

Node * cur = dummy;

while (list1 and list2)
{
    if (list1 -> val < list2 -> val)
    {
        cur -> next = list1;
        cur = list1;
        list1 = list1 -> next;
    }
    else
    {
        cur -> next = list2;
        cur = list2;
        list2 = list2 -> next;
    }
}

while (list1)

while (list2)
{ cur -> next = list2;
```

```
while (list1)
{
    cur->next = list1;
    cur = list1;
    list1 = list1->next;
}
```

```
while (list2)
{
    cur->next = list2;
    cur = list2;
    list2 = list2->next;
}
return dummy->next;
```

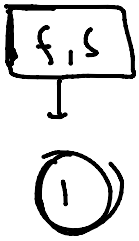


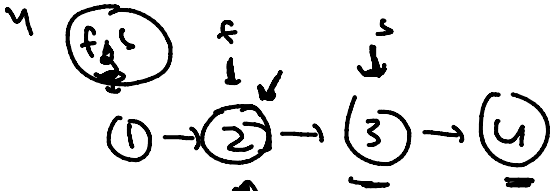
Merge sort in linked list

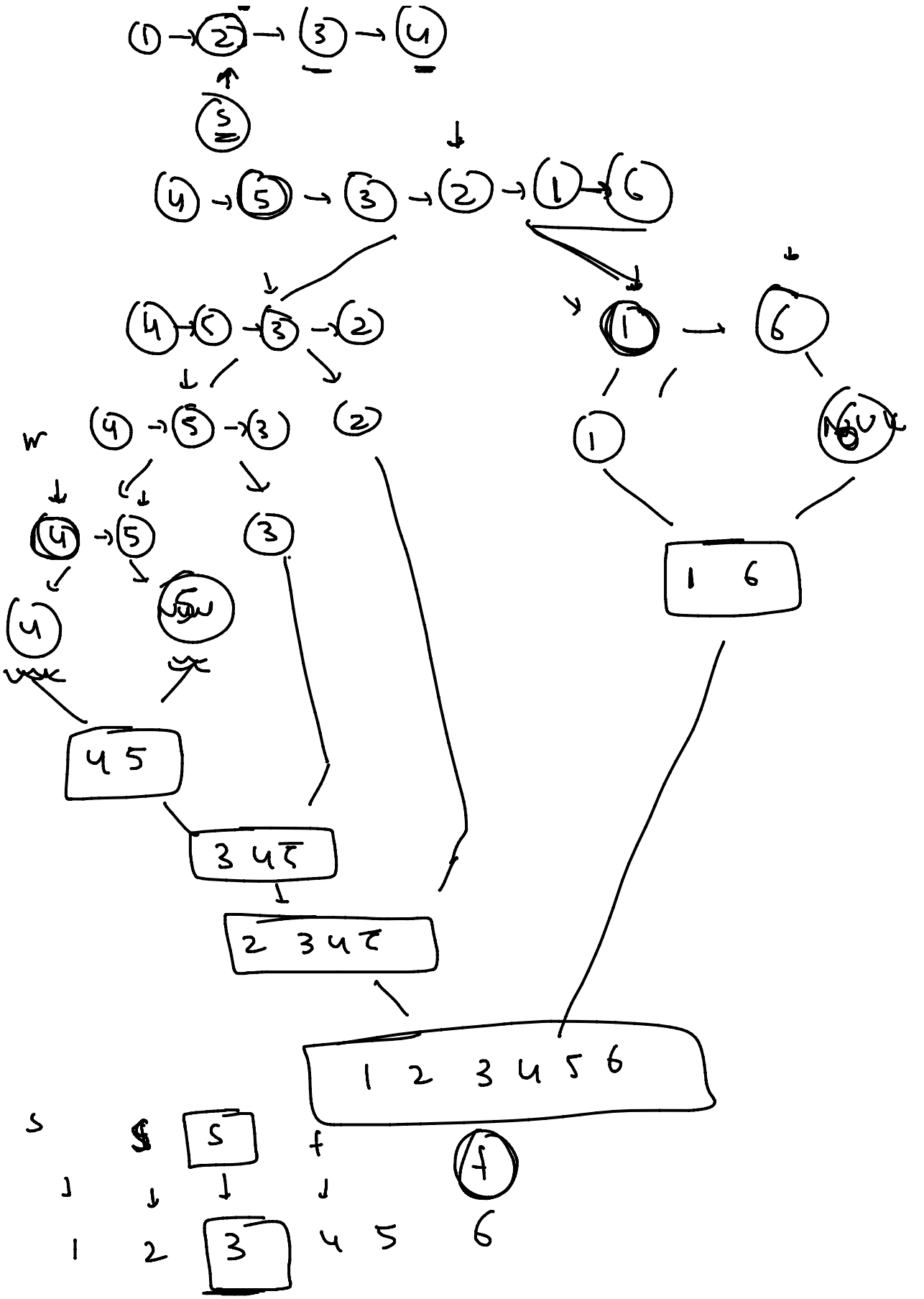


Mid value

```
Node* mergesort (Node* head)
{
    if (head == NULL or head->next == NULL)
        return head;

    Node* mid = middle (head);
    Node* e = mid->next;
    mid->next = NULL;
    head = mergesort (head);
    e = mergesort (e);
    merge (head, e);
}
```

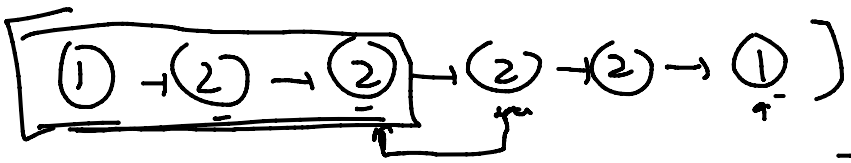






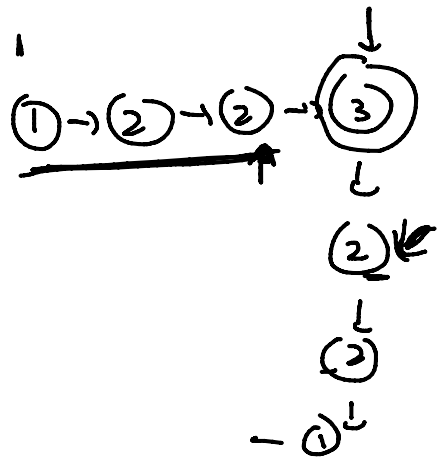
Check if linkedlist is Pallindrome

Check if linked list is Palindrome

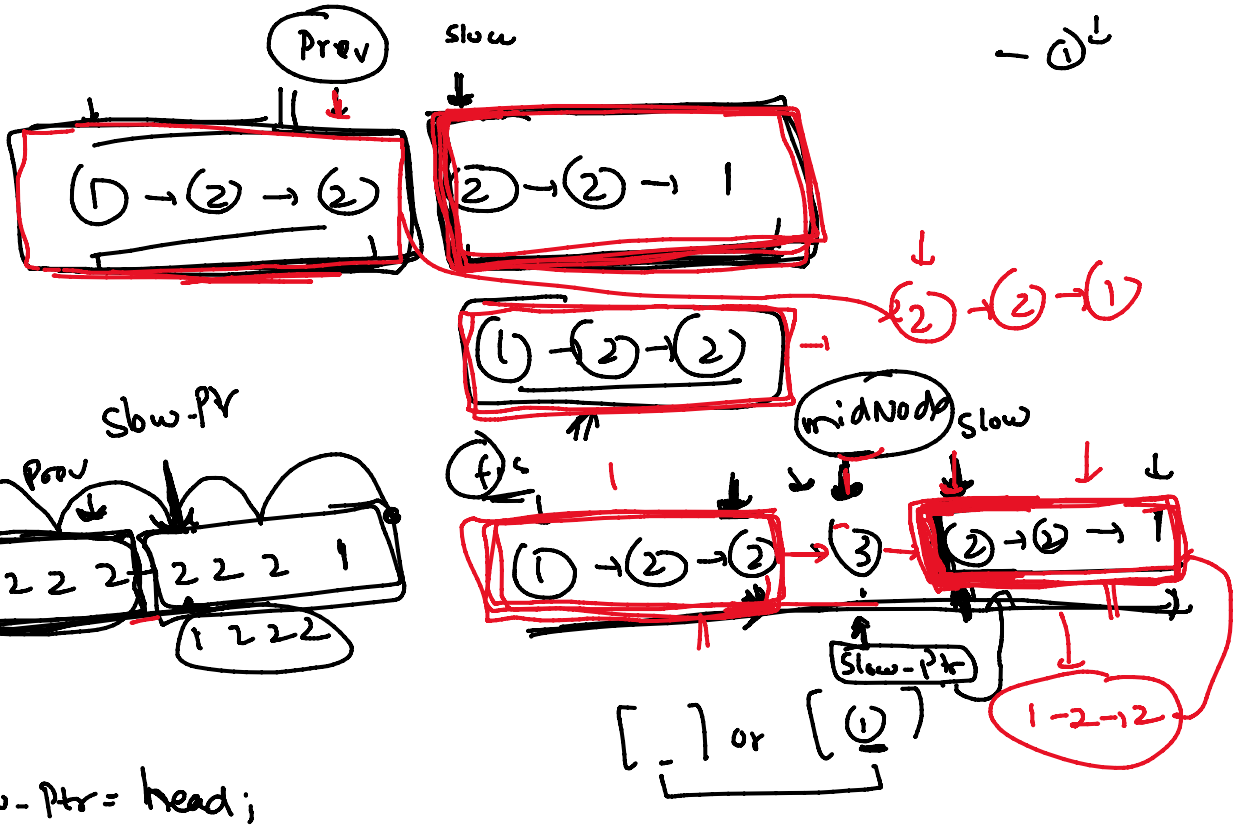


Brute Force → vector<int> v = { 1, 2, 2, 2, 2, 1 }

Palindrome using Stack



3rd Approach




```
Node* slow_ptr = head;
Node* fast_ptr = head;
Node* midNode = NULL;
if (head != NULL and head->next != NULL)
```

```
if (head != NULL and head->next != NULL)
    Node* Prev_of_slow = NULL;
{   while ( fast_ptr != NULL and fast_ptr->next != NULL)
    {   fast_ptr = fast_ptr->next->next;
        slow_ptr = slow_ptr->next;
        Prev_of_slow = slow_ptr;
    }

    if( fast_ptr != NULL)
    {  // linked list is of odd length
       midNode = slow_ptr;
       slow_ptr = slow_ptr->next;
    }

    second_half = slow_ptr;
    Prev_of_slow ->next = NULL;
    if(midNode)
      midNode->next = NULL;

[ second_half = reverse ( second_half)

bool res = Compare ( head, secondhalf )

{   return res;
}

    secondHalf = reverse (second_half);
    [Prev_of_slow->next = second_half;]

For ever {  for odd  if (midNode)
                     { Prev_of_slow->next = midNode;
```

4th → Recursion

Inbuilt stack

① → ② → ② → ② → ①

By reference

bool Palindrome (Node** left, Node* right)

{ if (right == NULL)
return true;

bool isP = Palindrome (left, right->next);

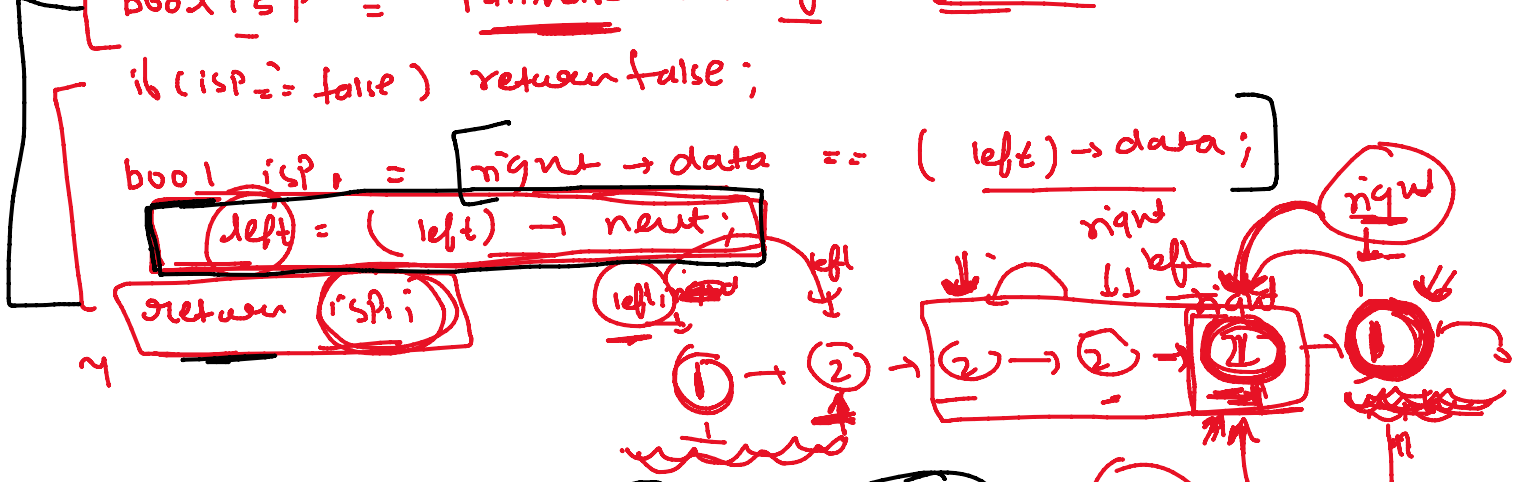
if (isP == false) return false;

bool isP = right->data == (left)->data;

left = (left)->next;

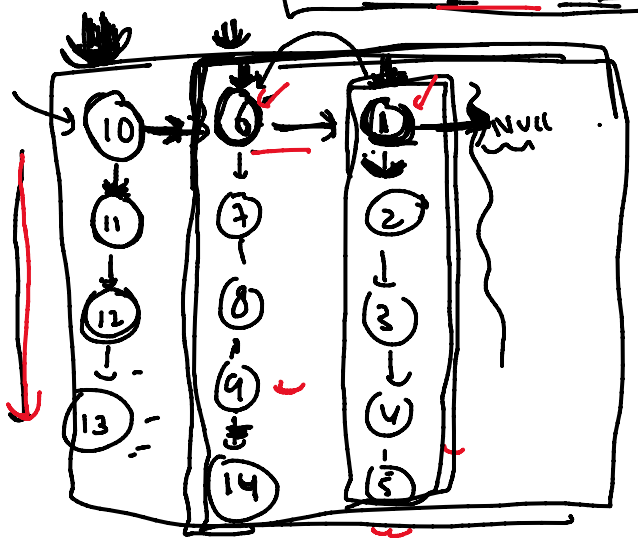
return (isP);

}

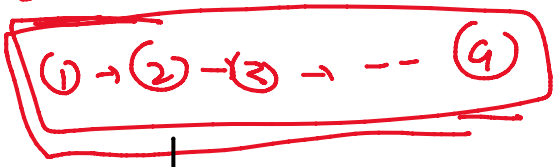


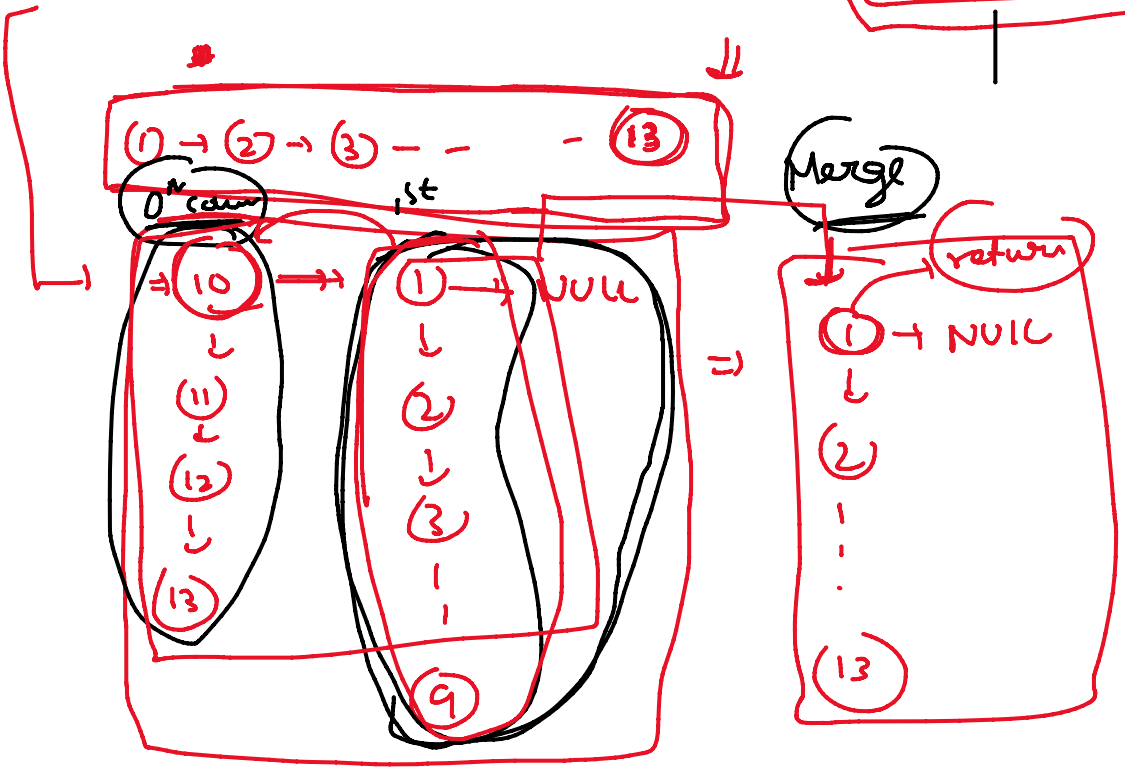
Global variable

Flattening a linked list



Node
data
next
right





Problem → Flatten a linked list from column 0

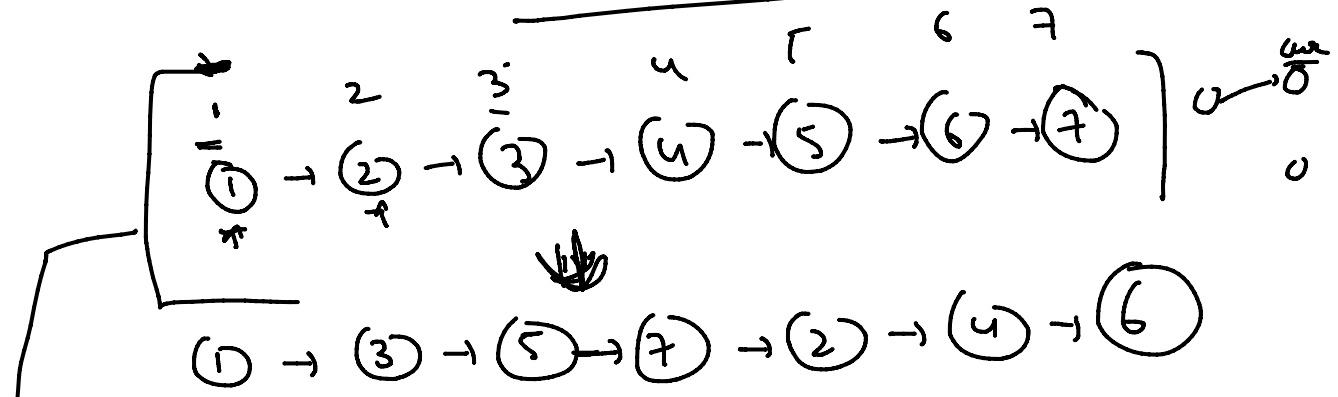
SubProblem → " " " " column 1

Flatten (column) = flatten (column+1) + merge (column, column+1)

flatten (column) = flatten (column.next) + merge (column, column.next)

```
Node *flatten ( Node *root )
{   if ( root == NULL or root->right == NULL )
        return root;

    root->right = flatten ( root->right );
    root = merge ( root, root->right );

    return root;
}
```

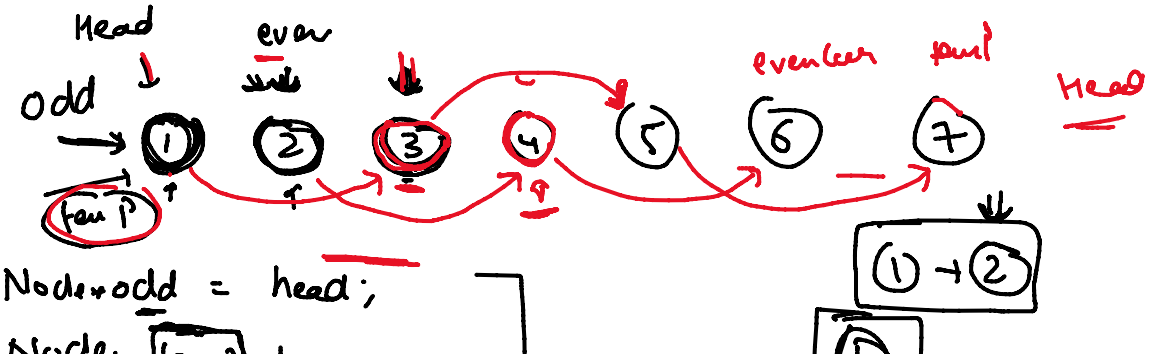
Odd Even LL



vector <int> odd = [1, 3, 5, 7]

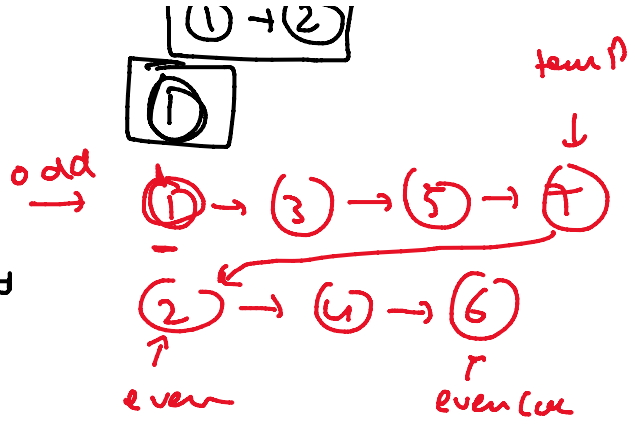
vector <int> even = [2, 4, 6]

Brute force



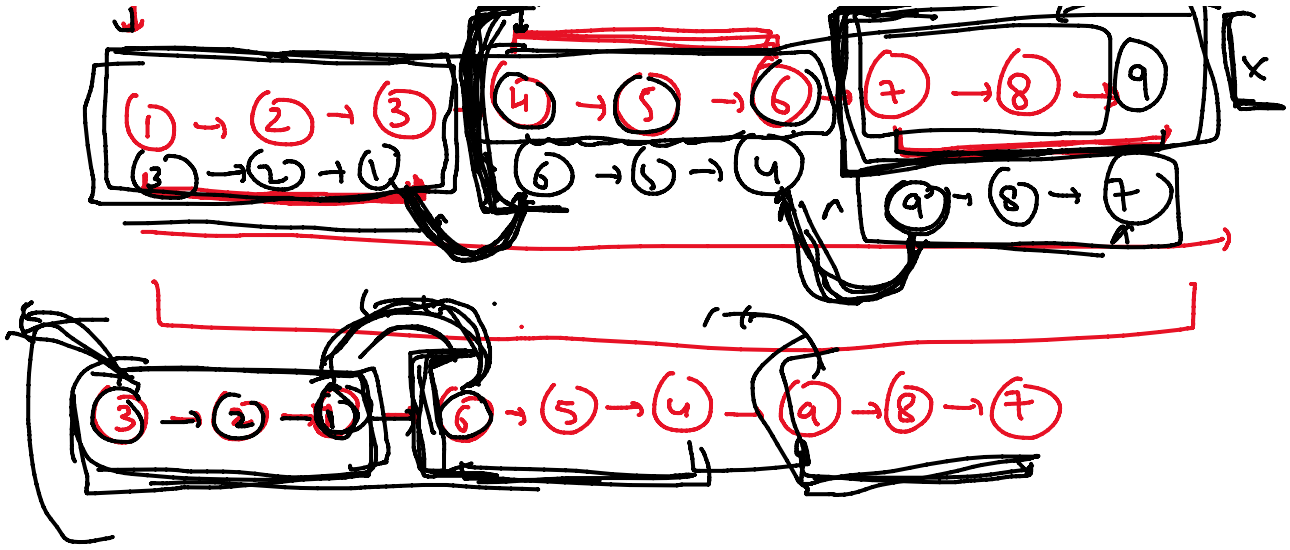

```
Node *odd = head;
```

```
Node *odd = head;
Node *temp = head
Node *even = head->next
Node *evenCur = ~~Nothing~~ head->next
```




```
int i=3;
head = head->next->next;
while (head)
{
    if (i%2)
    {
        temp->next = head;
        temp = head;
    }
    else
    {
        evenCur->next = head;
        evenCur = head;
    }
    head = head->next; i++;
}
if(evenCur)
    evenCur->next = NULL;
temp->next = even;
return odd;
```

Reverse a linked list in group of K



Rec ⇒ Work + SubProblem + Connection

Reverse k length

k

Reverse (root)

reverse k Nodes

root

Connection

Reverse (root + k)

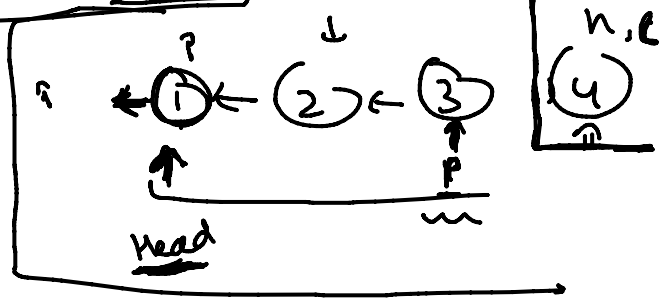
reverse k Nodes start from root + k + 1

reverse (root + 2k)

root + k → root + 2k

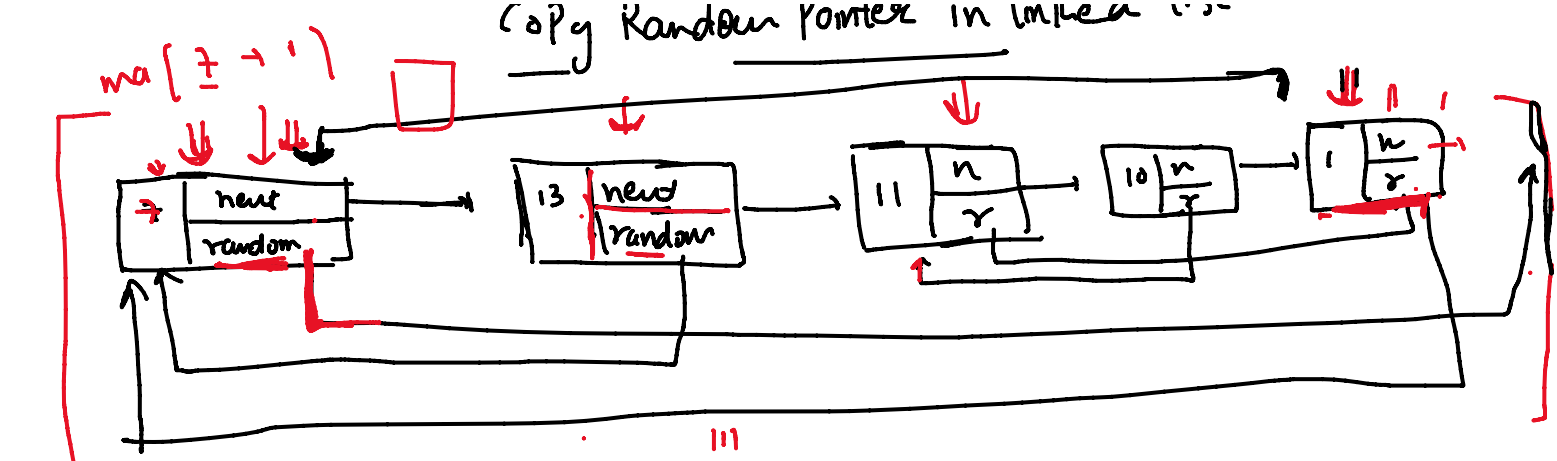
n = count

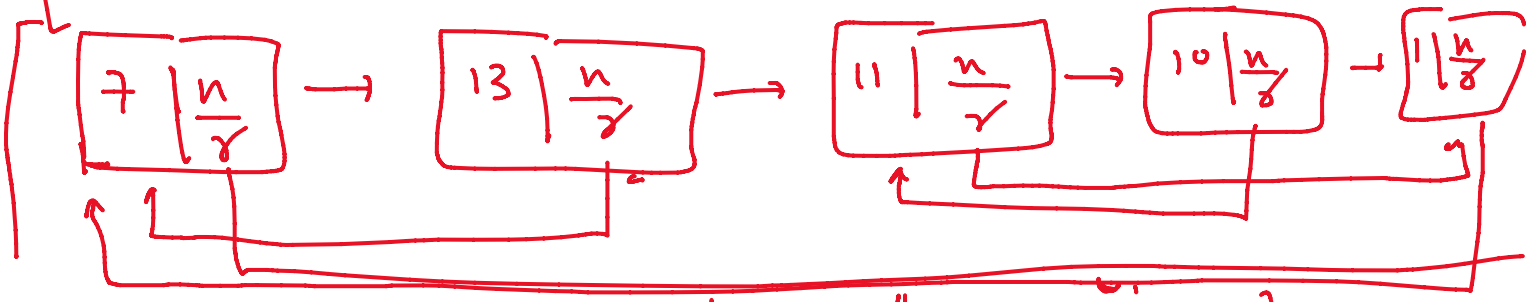
3

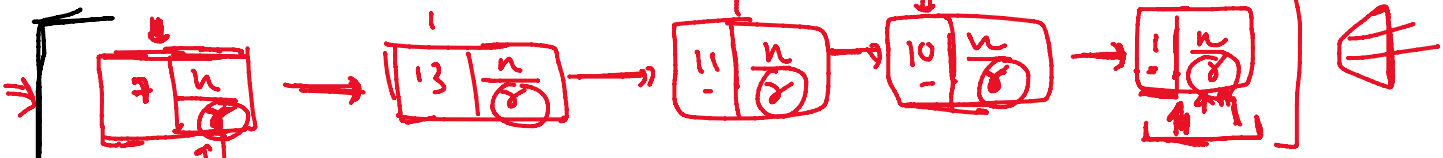


Copy Random Pointer in linked list

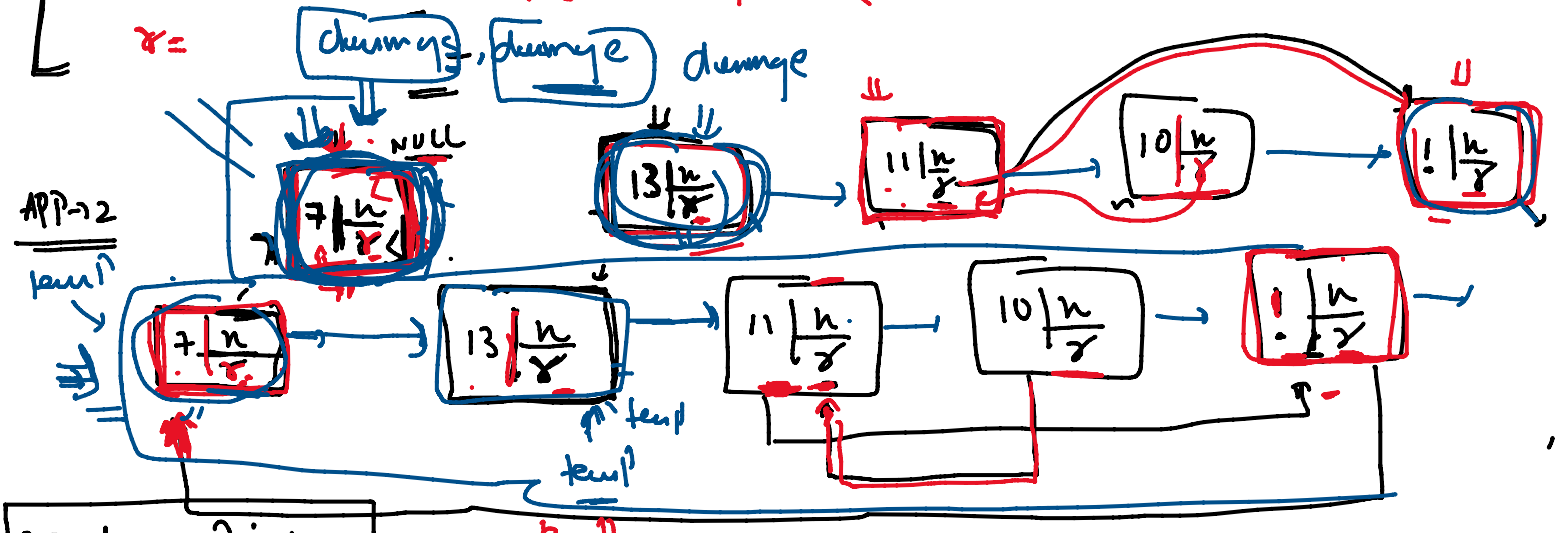
Copy Random Pointer in Linked List

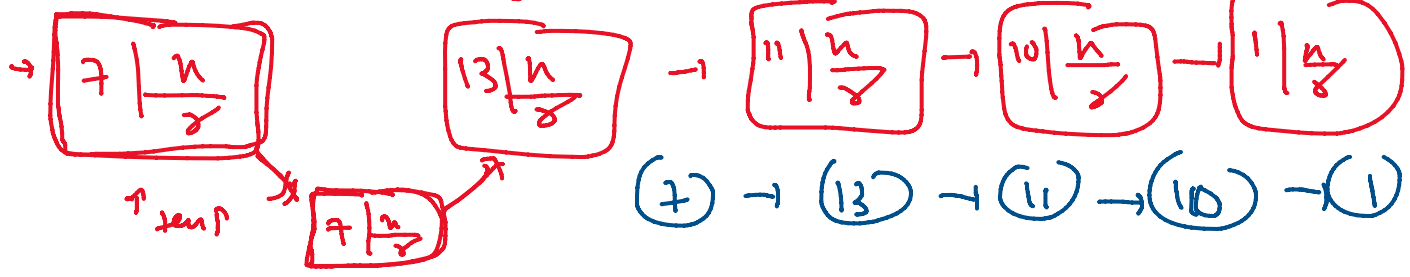






T.C ↑ Impossible





3rd step