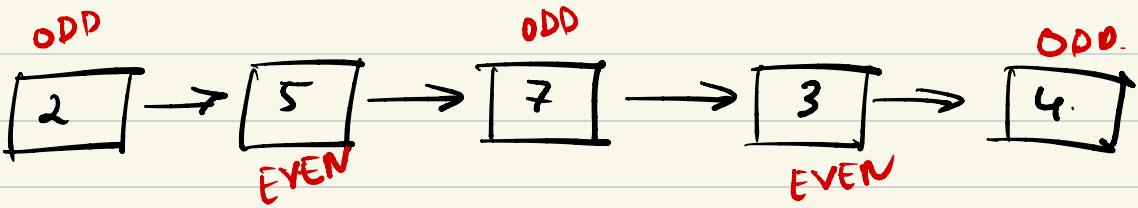
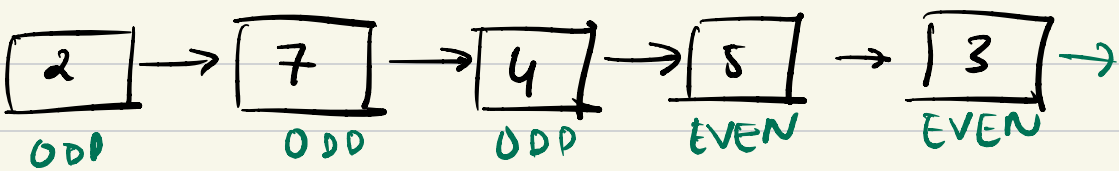
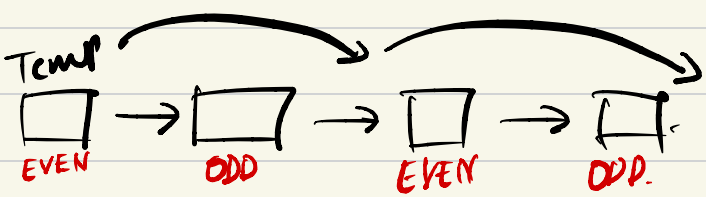
SEGREGATE ODD AND EVEN NODES IN LL



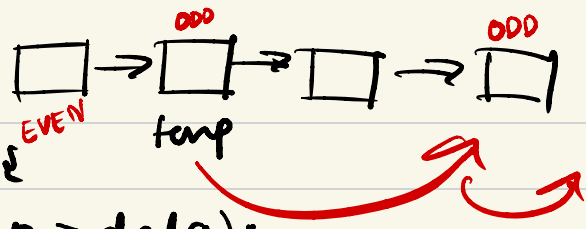
OUTPUT:



BRUTEFORCE:




```
temp = head;
vector<int> arr; // to store ans.
while (temp != nullptr && temp->next != nullptr)
    arr.push_back(temp->data)
    temp = temp->next->next;
}
if (temp) { => if last Node remains.
        arr.push_back(temp->data)
```



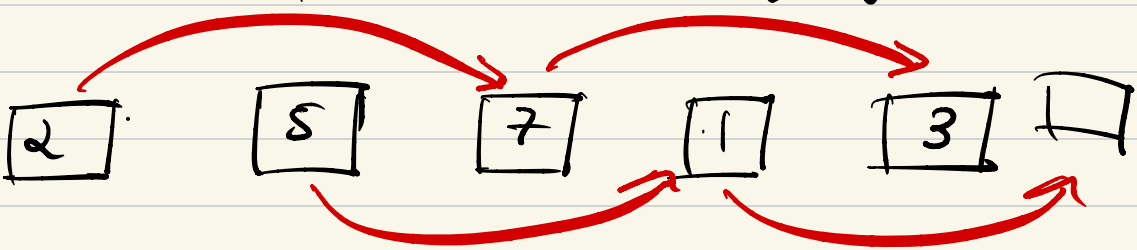

```
temp = head -> next;
while (temp != nullptr){
    arr.push_back(temp -> data);
    temp = temp -> next;
}
if (temp){
    arr.push_back(temp -> data);
}
```

// now we cant return arr so we transfer all values from arr to original list.

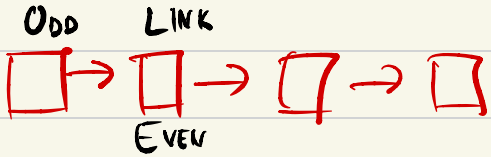
```
temp = head;
int i = 0;
while (temp != nullptr){
    temp -> data = arr[i];
    i++;
    temp = temp -> next;
}
return head;
}
```

OPTIMAL :

// changing links.

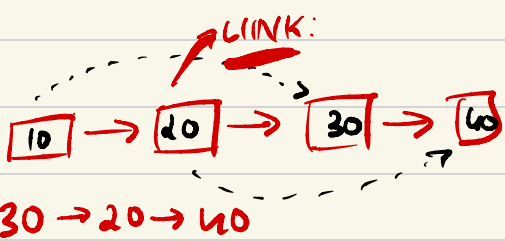



```
ListNode* oddEven (ListNode* head){

if(head==nullptr || head->next!=nullptr){
    return head; }
```



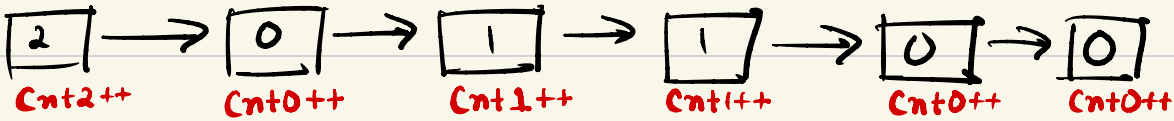

```
ListNode* odd = head;
ListNode* even = head->next;
ListNode* link = head->next;

while (even!=nullptr && even->next!=nullptr){
    odd->next = odd->next->next;
    even->next = even->next->next;
    odd = odd->next;
    even = even->next; }
odd->next = link;
return head; }
```

// even will always be ahead

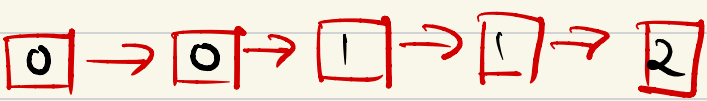


BRUTE SORT AN LL OF 0, 1 and 2's

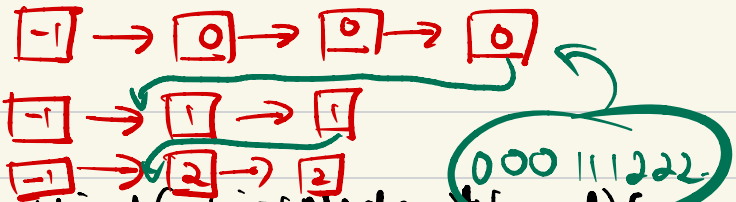
if (temp->data == 0||1||2).



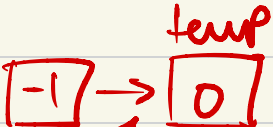

```
ListNode* Sort012(ListNode* head){
   int cnt1=0, cnt2=0, cnt3=0;
   ListNode* temp = head;
       while(temp != nullptr){
        if(temp->data == 0) cnt0++;
       elseif (temp->data ==1) cnt1++;
       else cnt2++;
        temp= temp->next;
   }
 temp  = head;
 while( temp != nullptr){
  if(cnt0) temp->data= 0 cnt0--;
  if(cnt1) temp->data= 1 cnt1--;
  if(cnt2) temp->data= 2 cnt2--;
  temp= temp->next;
}
return head;
```



OPTIMAL:

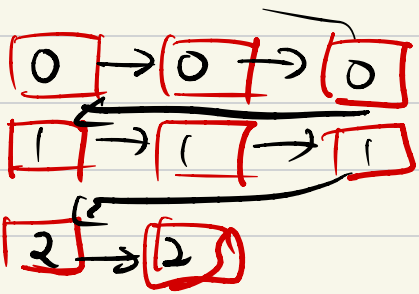



```
ListNode* SortList(ListNode *head){

    if(head==NULLPtr || head->next==nullptr)
        return head; }
//creating DummmyNodes at start.
ListNode* ZeroHead= new ListNode(-1);
ListNode* OneHead= new ListNode(-1);
ListNode* TwoHead= new ListNode(-1);
//making 3 different LL.
ListNode* zero= ZeroHead;
ListNode* one = OneHead;
ListNode* Two = TwoHead;
ListNode  Temp = head;
```




```
while (Temp != nullptr){
    if(temp->val == 0){
        zero->next = temp;
        zero = temp;
    } // based on data appending to
respective linked list.
```

```
else if (temp -> data == 1){
    one -> next = temp;
    one = temp;
 }
else if (temp -> data == 2){
    two -> next = temp;
    two = temp;
 }
 temp = temp -> next;
}
```



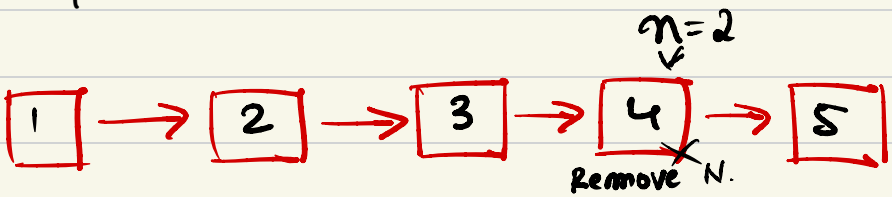
// connecting 000 → 111 → 222.

```
zero -> next = (oneHead -> next) ? oneHead -> next :
                    twoHead Next;
one -> next = twoHead -> next;
two -> next = nullptr;

delete zeroHead, deleteoneHead, delet twoHed

return newHead;
}.
```

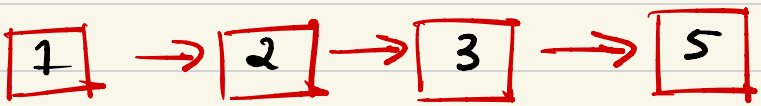
Remove nth node from back of linked-list

Given the head of a single linked list and an integer N remove the node from the back of the linked list. The value of N will always be less than equal to the number of notes in the list.

//input



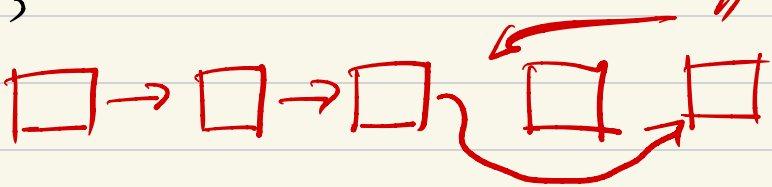
//output



Bruteforce

```
ListNode* RemoveNth(ListNode*head, int n){
    if(head == nullptr){ //if empty LL
        return nullptr; } return NULL.

int cnt=0;
ListNode* temp = head;

while(temp != nullptr){
    cnt++; // counting total no of elements
    temp = temp->next;
}
if(cnt == n){ //if only single Node.
    ListNode* newNode = head->next;
    delete(head);
    return newNode; }

int pos= cnt-n;
temp = head;
```

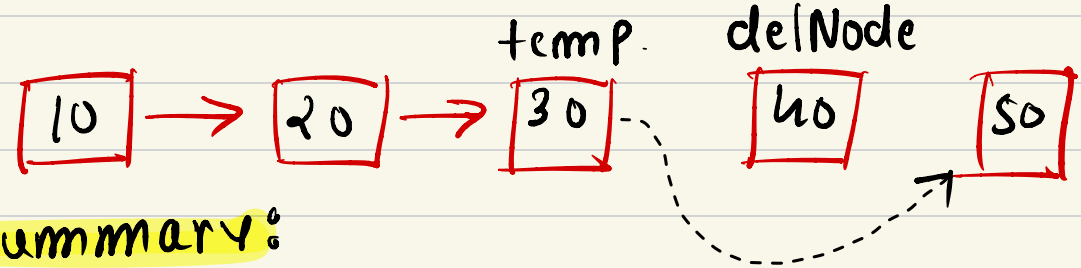
cnt = 5 { 5-2 = 3 //
n = 2.



```
while (temp != nullptr){
  pos--;
  if (pos==0){
      break;
  }
  temp = temp->next;
}
```

→ Node to be deleted.

```
ListNode* delNode = temp->next;
temp->next = temp->next->next;
delete (delNode);
return head;
}
```

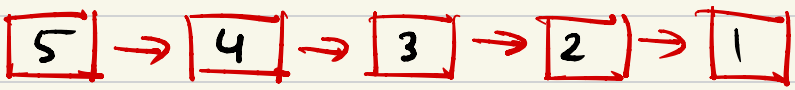


Summary:

- Count number of Nodes.
- if single Node then create one & delhead
- pos = count - n, & move to n.
- now link to delNode next & del delNode.

Reverse a linked-list:

// input: 

// output: 


```
ListNode* ReverseLL(ListNode* head){
    ListNode* temp = head;
    vector<int> ans{};
    //Pushing all values of LL to array.
    while(temp != nullptr){
        ans.push_back(temp->data);
        temp = temp->next;
    } int i = 0;  temp = head;   // reversing.
    reverse(ans.begin(), ans.end());
    while(temp != nullptr){
        temp->data = ans[i]; //again storing back to original LinkedList.
        i++;
        temp = temp->next;
    }
    return head;
}
```