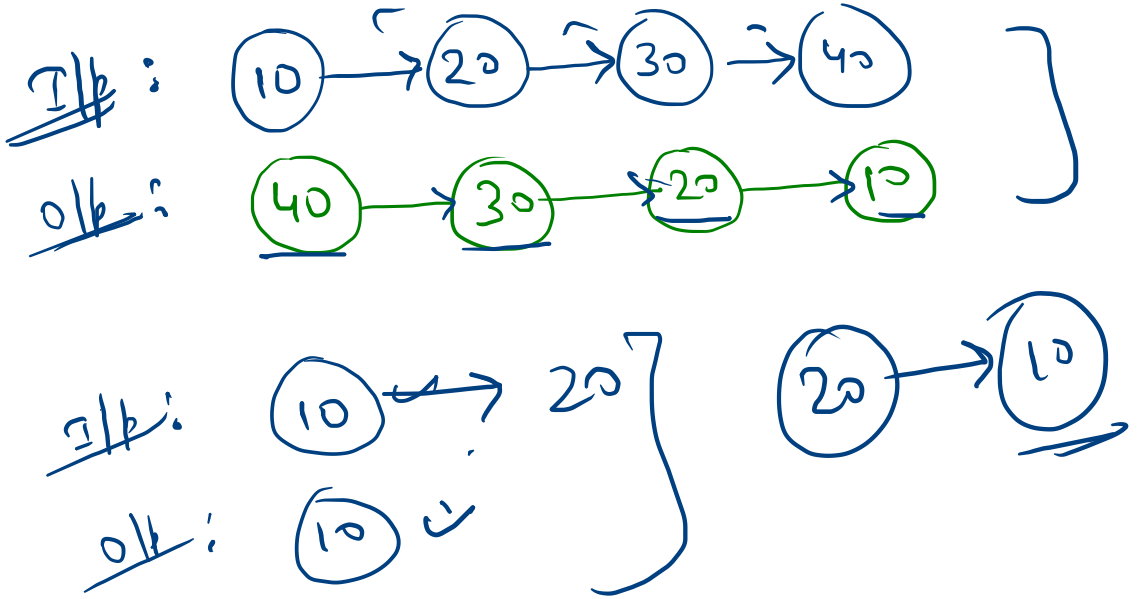
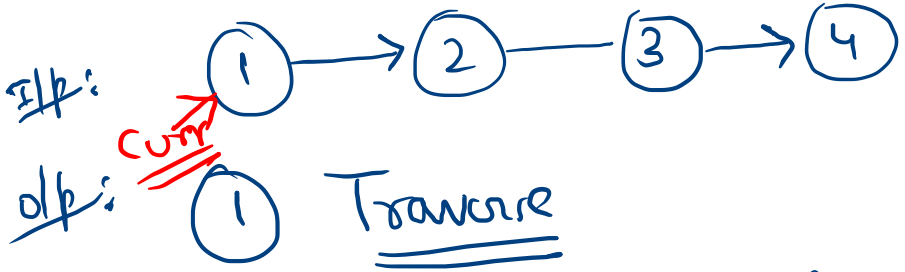
[Reverse a Linked List]

(Iterative method).



NAIVE SOLUTION

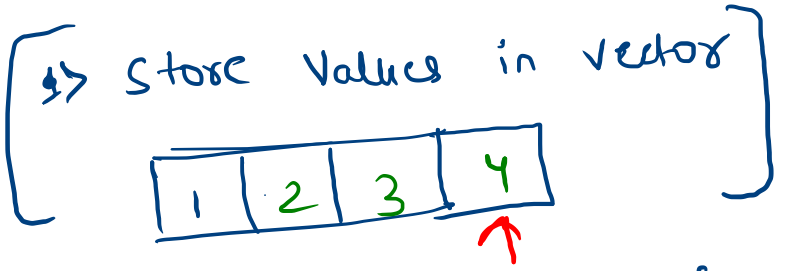


1) Traverse

data → store → vector

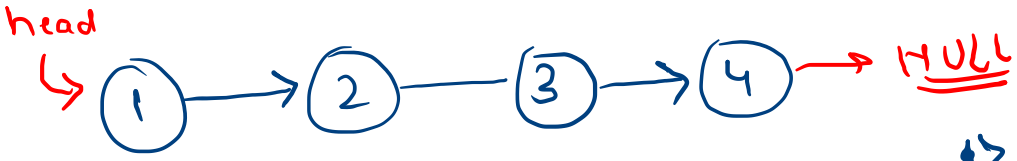
2) Traverse

curr → data = ans.back();



2) Re-assign the values from back of vector

NAIVE SOLUTION



I/p:

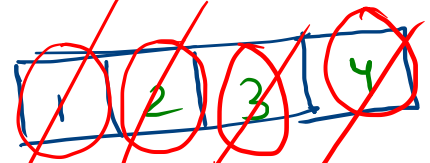
O/p:

Vector <int> data;

```
for( Node *curr = head ; curr != NULL ; curr = curr->next)
{
    data.push_back ( curr->val);
}
```

```
for( Node *curr = head ; curr != NULL ; (curr = curr->next)
{
    curr->val = data.back();
    data.pop_back();
}
```

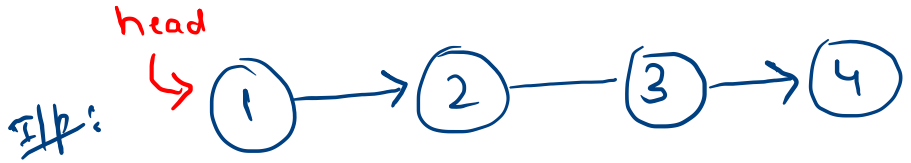
1) Store Values in vector



= data

2) Re-assign the values from back of Vector

NAIVE SOLUTION

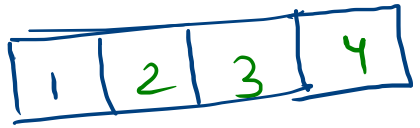


I/p:

O/p:

```
vector <int> data;
```

1) Store Values in vector



2) Re-assign the values from back of vector

O(H)

```
for(Node *curr = head; curr != NULL; curr = curr->next)
{
    data.push_back(curr->val);
}
```

O(H)

```
for(Node *curr = head; curr != NULL; curr = curr->next)
{
    curr->val = data.back();
    data.pop_back();
}
```

Problem

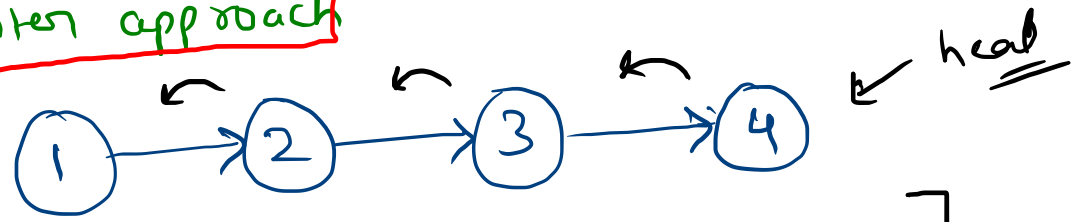
Time: O(H)

Aux space: O(H)

2nd Approach

⇒ 3 pointer approach

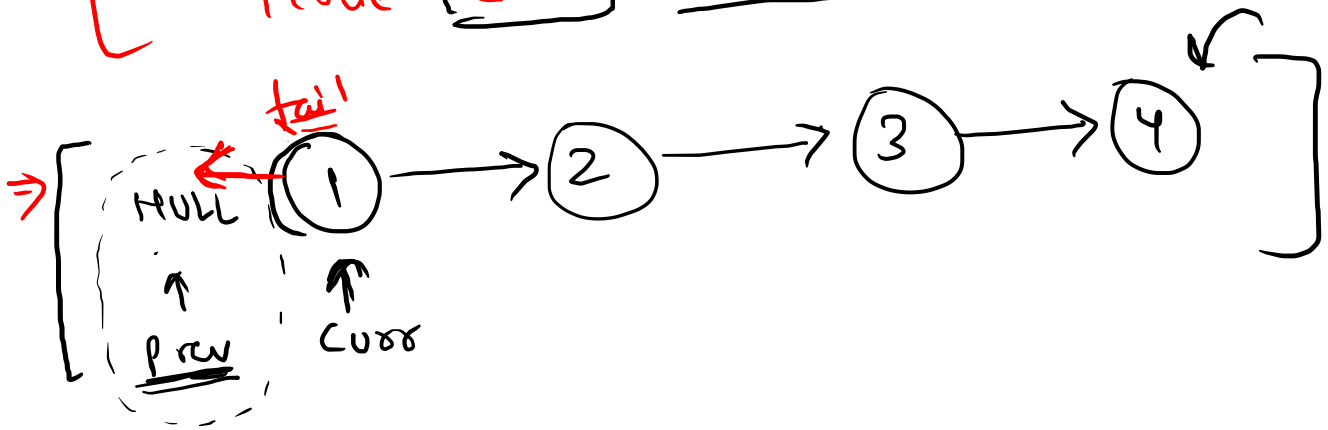
Ilb:



Node * prev
* curr
* temp

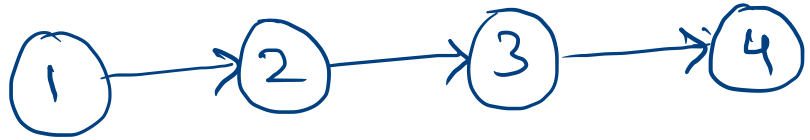
•> our mission: [To change the direction of Linkage]

[•> Node* prev = NULL;
Node* Curr = head;]



2nd Approach ⇒ 3 pointer approach

Ex:

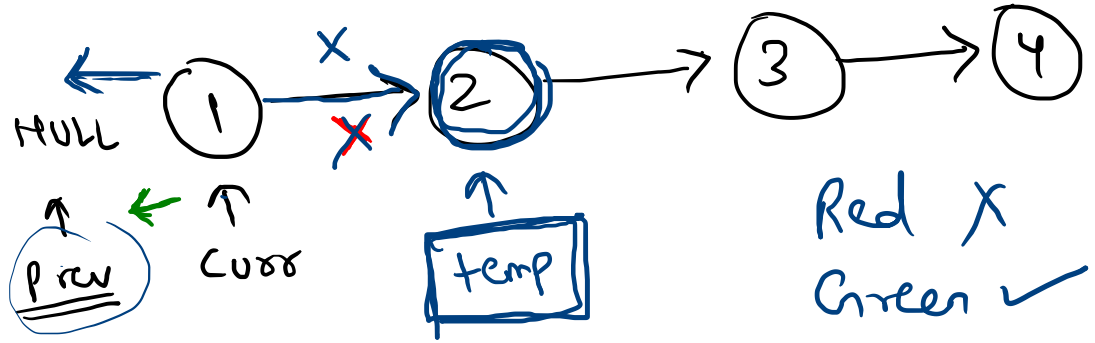


-> our mission: To change the direction of Linkage

```
Node * prev
     * curr
     * temp
```

```
-> Node* prev = NULL;
   Node* curr = head;
```

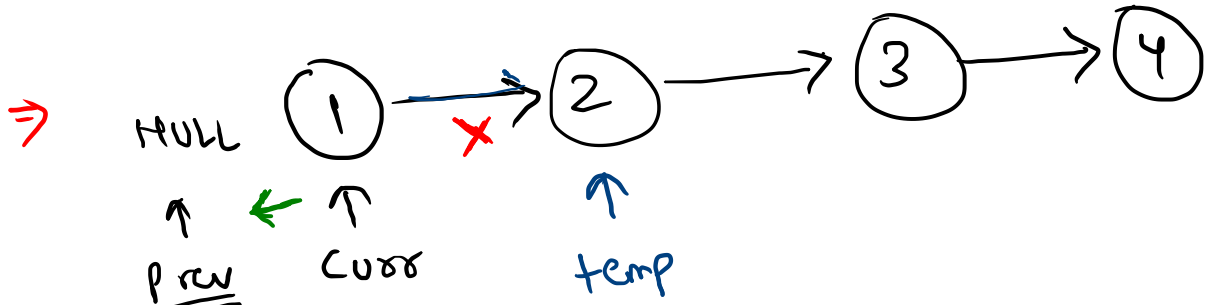
⇒



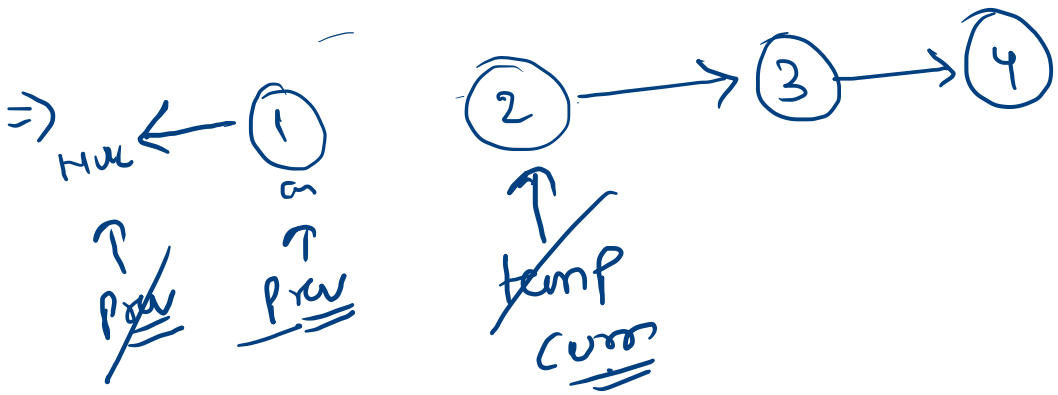
Red X
Green ✓

curr->next = prev X
First store curr->next

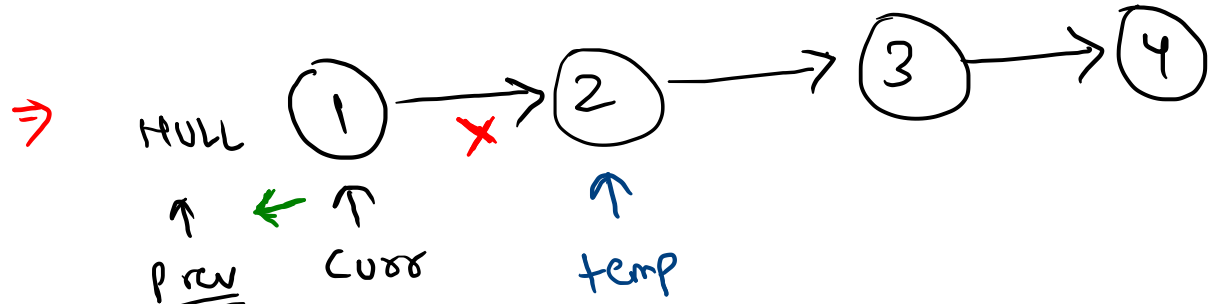
Steps:




```
temp = curr->next;
curr->next = prev;
prev = curr
curr = temp
```

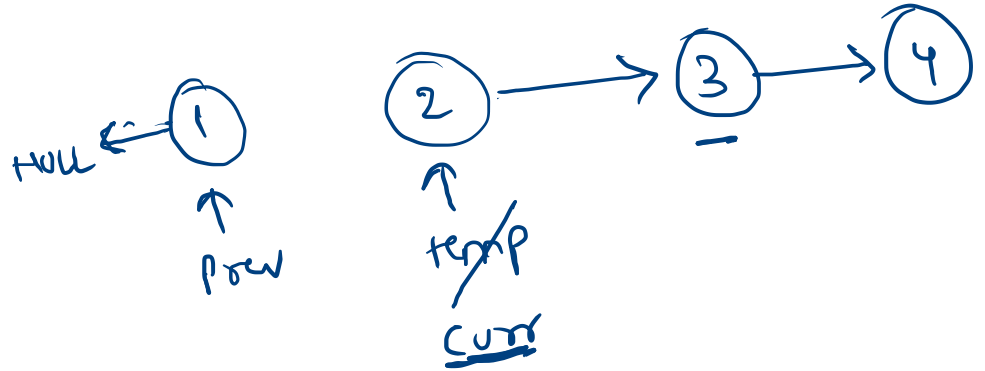


Steps:



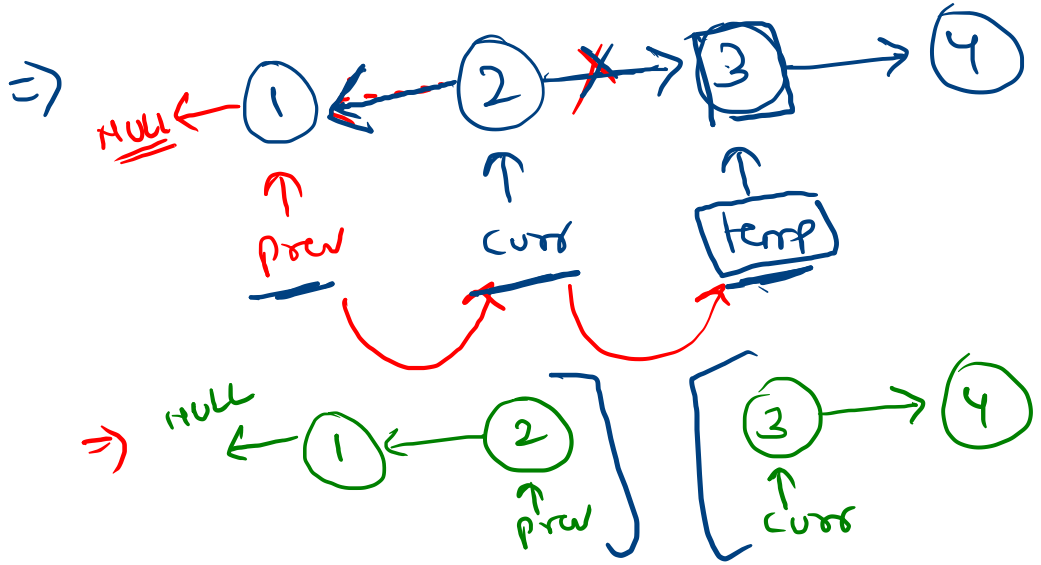

```
temp = curr -> next;
curr -> next = prev;
prev = curr
curr = temp
```

Step

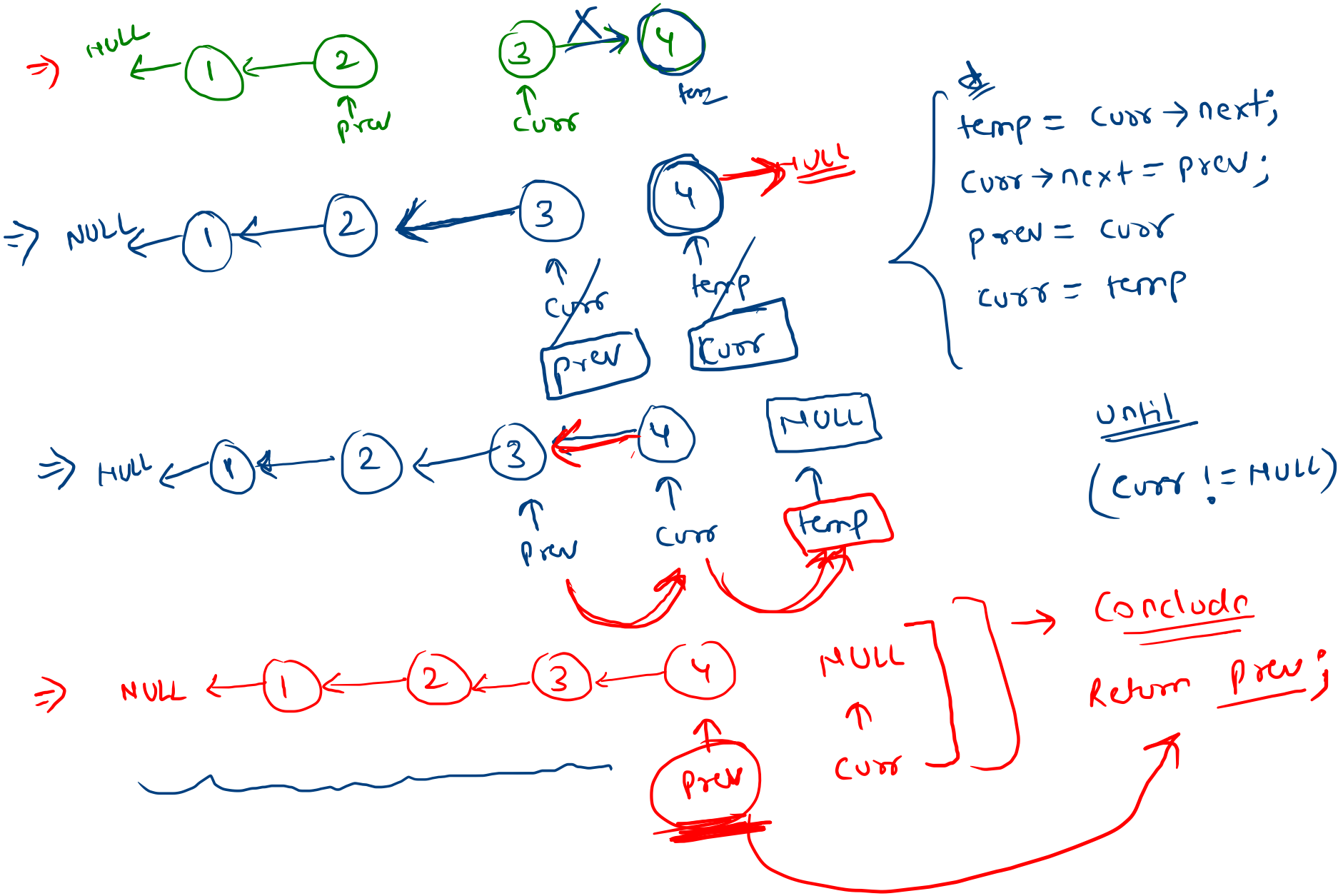


until

(curr != NULL)



Steps:


NULL
1
2
3
4
prev
curr
temp
X
temp = curr → next;
curr → next = prev;
prev = curr
curr = temp
until
(curr != NULL)
Conclude
Return prev;